UNIVERSIDAD AUTONOMA
DE MADRID

Automatización de la Generación de Cursos Tutores para Software Interactivo

TESIS DOCTORAL

Autor: Federico García Salvador

Director: Roberto Moriyón Salomón

Escuela Técnica Superior de Informática

Memoria presentada para optar al grado de Doctor en Ingeniería Informática

Con cariño a mis hermanos,
a mi padre y a mis tres madres.

Fede.

Agradecimientos

Parece la parte más fácil de todo el documento y...vaya, es la más difícil. Y lo es porque citar a toda la gente que ha estado a mi alrededor apoyándome durante los últmos años es imposible, y dejar fuera a una parte es injusto. De modo que creo que como mal menor voy a ser tan breve como pueda y citar a cuantas menos personas, mejor.

Quiero agradecer a R.Moriyón el gran apoyo que me ha prestado para la elaboración de este trabajo. No sólo he recibido de él un apoyo técnico, sino, además, un apoyo humano que valoro por encima de los resultados concretos que este trabajo pueda aportar.

Igualmente quiero expresar mi especial gratitud por el interés que mostró y las sugerencias que recibí de J.Contreras, a quien hoy tengo que recriminar que tenga que escribir esto ;-) También agradezco los valiosos comentarios de los profesores P.Rodríguez, M.Alfonseca, P.Castells y P.Saiz.

De igual manera quiero agradecer a mis ya ex-compañeros los momentos de ordenadores y café que hemos disfrutado juntos. Hago mención especial de Rosa, Ruth, Juan, Jony y Álvaro, sin olvidar a la larga lista de becarios que hemos desfilado por los laboratorios del Departamento.

También destacar la calidad humana de todos los integrantes del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid, tanto del personal docente como del personal de administración y servicios. Desde aquí les animo a que sigan haciendo crecer la escuela que yo vi nacer.

Finalmente, y no por ello menos importante, mi agradecimiento a muchas otras personas, la mayoría de las cuáles aún siguen muy cerca de mí, por contagiarme de este cosquilleo raro llamado *vida*. No los voy a citar, pero ellos saben quiénes son, y también saben que estoy orgulloso de tenerles por amigos. ¡Ya lo celebraremos!

Secciones

Lista de Figuras

1 Introducción

Las aplicaciones actuales encierran una complejidad que hace muy pocos años era impensable. Con estos sistemas se pueden llevar a cabo en muchos casos cientos de acciones. Si a este hecho se añade el que a menudo los efectos de dichas acciones son dependientes del contexto en que se realicen, nos encontramos ante sistemas interactivos de una flexibilidad y una riqueza funcional asombrosas. Esta complejidad es una consecuencia directa de las capacidades ofrecidas por las modernas GUIs (*Graphic User Interface)*, basadas en el paradigma WIMP –*Windows, Icons, Menus, Pointer*- y que hacen uso intensivo de técnicas de manipulación directa. Si a esto le añadimos la riqueza de interacción que los nuevos dispositivos de entrada (como sistemas de tratamiento de lenguaje natural, de seguimiento ocular, de entrada tridimensional o nuevos clientes de acceso ligeros como *handhelds, palmtops*, y teléfonos móviles) son capaces de proporcionar, así como las salidas de naturaleza cada vez más sofisticada que las aplicaciones interactivas manejan (gráficos tridimensionales, animaciones, o salidas adaptadas a discapacitados visuales), difícilmente podemos hacernos una idea de la riqueza de las aplicaciones que dispondremos en un futuro próximo.

Sin embargo, y como posteriormente veremos, estos sistemas interactivos nunca vienen acompañados de herramientas de enseñanza lo suficientemente potentes como para convertirse en sistemas de guía útiles para los usuarios. Esta falta de equilibrio entre la potencia de los sistemas interactivos y la potencia de las herramientas encargadas de enseñar a utilizarlos provoca que sólo usuarios muy avanzados consigan aprovechar buena parte de las capacidades que dichas aplicaciones ponen a su disposición. Por si éso fuera poco, el problema se agrava más aún cuando son usuarios noveles los que se enfrentan a los sistemas interactivos actuales.

La falta de herramientas lo suficientemente potentes como para convertirse en sistemas de guía útiles para los usuarios se debe, fundamentalmente, a la falta de integración existente entre las aplicaciones y las herramientas de enseñanza, así como al enfoque empleado por estos sistemas para proporcionar explicaciones a los usuarios. En la práctica totalidad de los casos de aplicaciones comerciales, los sistemas de guía se desarrollan de manera completamente independiente al desarrollo de las aplicaciones. Esta independencia acarrea altos costes de desarrollo y de mantenimiento, ya que los cambios que sufran las aplicaciones deberían ocasionar cambios en la herramienta de guía, a fin de mantenerse la consistencia entre la aplicación y las instrucciones proporcionadas a los usuarios. Una excepción a la independencia citada son los entornos de ayuda orientados a *widgets* que soportan generalmente los entornos de desarrollo de interfaces (IDEs) como Visual Basic [Microsoft91a] o Visual C++ [Microsoft91b]. Sin embargo, estos y otros muchos sistemas sufren terriblemente el segundo de los grandes problemas de los entornos de guía, que es la pobreza semántica de las indicaciones que proporcionan. El hecho es que normalmente la información que se aporta a los usuarios está relacionada con características directamente perceptibles de las aplicaciones o con acciones de bajo nivel, de manera que los usuarios encuentran dificultades a la hora de asociar este tipo de información con las tareas que ellos quieren realizar. La consecuencia de todo esto es que, en ningún caso, estos sistemas de guía permiten a los usuarios noveles el aprendizaje sistemático de los sistemas interactivos.

En esta tesis se propone una técnica para el diseño de cursos para la formación de usuarios de aplicaciones interactivas. Con estos cursos, los usuarios aprenderán a utilizar sus sistemas interactivos mediante el uso de los propios sistemas, sin necesidad de que los diseñadores tengan

que crear una réplica de los mismos. De este modo, la barrera de la independencia entre la aplicación y el entorno de guía se elimina, permitiéndose una mayor integración (y, como consecuencia, una mayor interacción) entre ambos sistemas y obteniéndose una reducción drástica en los costes de desarrollo del sistema de guía. Además de esta integración, las indicaciones que se les dan a los usuarios van enfocadas a ayudarles en la realización de las tareas que ellos desean realizar, y no solo a la explicación de los distintos elementos de la interfaz gráfica o a cuestiones más propias del diseño de bajo nivel de la aplicación. Además, y dado que son precisamente los usuarios noveles quienes más sufren las deficiencias de los sistemas actuales de guía, se hace énfasis en ellos.

Como solución se proporciona un entorno de gestión de cursos interactivos que permite el seguimiento directo y automático de las actividades que realizan los alumnos durante la ejecución de los cursos. Gracias a este seguimiento de la actividad de los usuarios, los cursos basados en la técnica propuesta aportan a los usuarios finales, los alumnos, la ventaja fundamental de poder recibir retroalimentación (*feedback*) a partir del seguimiento de sus actividades. Este entorno, CACTUS, ofrece un marco integrado para el diseño de cursos tutores para la enseñanza de aplicaciones interactivas, así como el soporte necesario para permitir al sistema el seguimiento de la actividad de los alumnos durante la ejecución de dichos cursos tutores.

Desde el punto de vista del diseño de los cursos tutores, el sistema ofrece un marco para el desarrollo de unidades pedagógicas, que permite a los diseñadores la creación y modificación de sesiones de enseñanza de manera completamente interactiva, mediante el uso de técnicas de programación visual y de Programación por Demostración [Cypher93]. Estas técnicas permiten la especificación de los cursos tutores mediante el uso de la propia aplicación para la que se quiere desarrollar el curso tutor. Gracias a ello, la interfaz del sistema permite a los diseñadores establecer los contenidos de los cursos tutores sin necesidad de codificación, simplemente proporcionando al sistema ejemplos de lo que se quiere que los alumnos aprendan a hacer.

La interfaz de los cursos tutores se asimila a la interfaz a la que los usuarios más noveles están acostumbrados a utilizar para el aprendizaje: los libros. Así, podemos tratar a los cursos tutores como si fueran libros interactivos cuyo objetivo es la enseñanza de una materia un tanto especial, como es una determinada aplicación interactiva. Los usuarios utilizan estos libros interactivos de la misma manera en que utilizan los libros de texto convencionales, con la distinción de que a la vez que leen los contenidos de los libros, deberán practicar con la aplicación que están aprendiendo bajo el control y supervisión del curso tutor, que le guiará y corregirá durante la realización de las tareas encomendadas.

Los cursos tutores producidos con esta herramienta utilizan como soporte la tecnología de desarrollo basada en modelos declarativos de interfaces, que aporta beneficios en el desarrollo de aplicaciones, como la automatización parcial de diseño de aplicaciones, validación de soluciones, prototipado de aplicaciones, etcétera. La información que se encuentra presente en dichos modelos hace posible que herramientas externas a las propias aplicaciones razonen sobre ellas desde dos puntos de vista distintos. Por un lado, es posible razonar sobre los contenidos de los modelos en tiempo de diseño de la aplicación para ayudar en la especificación de la interfaz de la misma. Por otra parte, y como es nuestro caso, es posible razonar sobre el estado de la aplicación en tiempo de ejecución, con el objetivo de modificar el comportamiento normal de la interfaz y así facilitar servicios de valor añadido a las aplicaciones. Dentro de este trabajo se describe una arquitectura, ATOM, que permite gestionar tanto los aspectos estáticos como dinámicos de la modelización de la interfaz de usuario. Y, también dentro del marco de este trabajo, se ha

realizado una implementación de esta arquitectura, ATOMS, que da soporte a la especificación y a la gestión dinámica de modelos de tareas de usuario, de modo que se puede dar servicio orientado a tareas a otros subsistemas que se basen en él.

El resto de esta tesis está estructurada como sigue. En primer lugar, se estudian los problemas asociados a los sistemas actuales de enseñanza de software, tanto de los de ámbito comercial como de los entornos experimentales más representativos en este campo. A continuación, se describirá detalladamente la arquitectura del entorno propuesto, empezando por las capas de más bajo nivel y acabando en los subsistemas de más alto nivel del entorno. Se empezará por detallar las bases y el funcionamiento de la arquitectura ATOM -y su implementación concreta ATOMS-, el subsistema encargado de gestionar en tiempo de ejecución los modelos de tareas de usuario de las aplicaciones, y en el que CACTUS se basa. En la siguiente sección se dará una descripción detallada de CACTUS: en primer lugar se explicará su arquitectura y, más tarde, se detallará su funcionamiento, tanto desde el punto de vista de los diseñadores de cursos como desde el punto de vista de los usuarios de los cursos. Tras esa sección, describiremos las pruebas realizadas con el entorno y se discutirán detalles de su implementación. Finalmente, aportaremos las conclusiones del trabajo y las posibles líneas de investigación futuras que han quedado abiertas.

Como nota previa al lector, se hace constar que cada capítulo está estructurado de manera que al principio del mismo se incluye una descripción abreviada de los conceptos que a lo largo de la sección se detallarán a un nivel más técnico. De este modo, en una primera lectura de este documento podrá optarse por leer únicamente estas partes más conceptuales para, en una lectura más a fondo, proceder a leer completamente el documento.

2 Trabajo Previo

A lo largo de esta sección vamos a describir el trabajo previo más destacado relacionado con los temas tratados en esta tesis. Esta descripción se hará separando las tres líneas más relevantes en las que este trabajo se apoya. En primer lugar, describiremos los sistemas de guía más importantes, tanto sistemas comerciales de guía (Sección 2.1.1) como prototipos de investigación (Sección 2.1.2). A continuación, describiremos brevemente las bases fundamentales de la tecnología de diseño basada en la modelización de interfaces de usuario en que nos hemos apoyado para desarrollar nuestras ideas. Finalmente, en la Sección 2.3, se discutirán aspectos influyentes de la herramienta en la que la implementación de las ideas de esta tesis se ha basado.

2.1 Sistemas de Guía

En esta sección se describen los sistemas de guía al usuario más relevantes. Hay que hacer notar que todos estos sistemas son aproximaciones a la generación o al desarrollo de guía a los usuarios independiente del dominio de la aplicación. Otros sistemas existentes, dependientes de la aplicación [Fenchel82, Hecking87, Yu88], proponen sistemas de guía estrechamente ligados a las aplicaciones a las que sirven. El problema básico de estas aproximaciones es la ausencia de un entorno genérico con el que permitir a distintas aplicaciones beneficiarse del mismo tipo de guía, ya sea inteligente o no. Por ello, estos sistemas no permiten obtener un cierto grado de reutilización de estas componentes de guía. Un caso similar de reutilización de componentes es el que nos encontramos con los entornos de diseño de interfaces de usuario basados en modelos independientes del dominio, que incorporan todo el conocimiento dependiente de las interfaces particulares en dichos modelos, aislando así a los entornos de tener que conocer aspectos no genéricos de las interfaces.

2.1.1 Sistemas de Ayuda Tradicionales

2.1.1.1 Ayudas de Línea de Mandatos

Los primeros sistemas de ayuda que aparecieron para sistemas interactivos estaban basados en *recetas*, y asociados a aplicaciones de línea de mandatos. El contenido de estas ayudas es, básicamente, el mismo que puede encontrarse en los manuales impresos de las aplicaciones, es decir, descripciones de la sintaxis y la funcionalidad de los distintos mandatos disponibles. La ventaja fundamental de esta ayuda respecto de dichos manuales es la facilidad de invocación de la misma, ya que resulta sencillo acceder a la ayuda acerca de cualquier función en cualquier momento, facilitándose el proceso de búsqueda [Henke98]. Desde el punto de vista del diseñador de la ayuda, toda la información textual ha de ser introducida a mano, con la única ayuda de herramientas que facilitan la labor de proporcionar formato a las páginas de los manuales y de crear tablas de contenido.

En cualquier caso, pese a que tanto desde el punto de vista del usuario como desde el punto de vista del desarrollador del entorno de ayuda estos sistemas de guía son de capacidades reducidas, hay que tener en cuenta que los usuarios a los que estos sistemas iban dirigidos eran usuarios avanzados en la mayoría de los casos, pues el uso de la informática no estaba tan difundido como lo está en la actualidad.

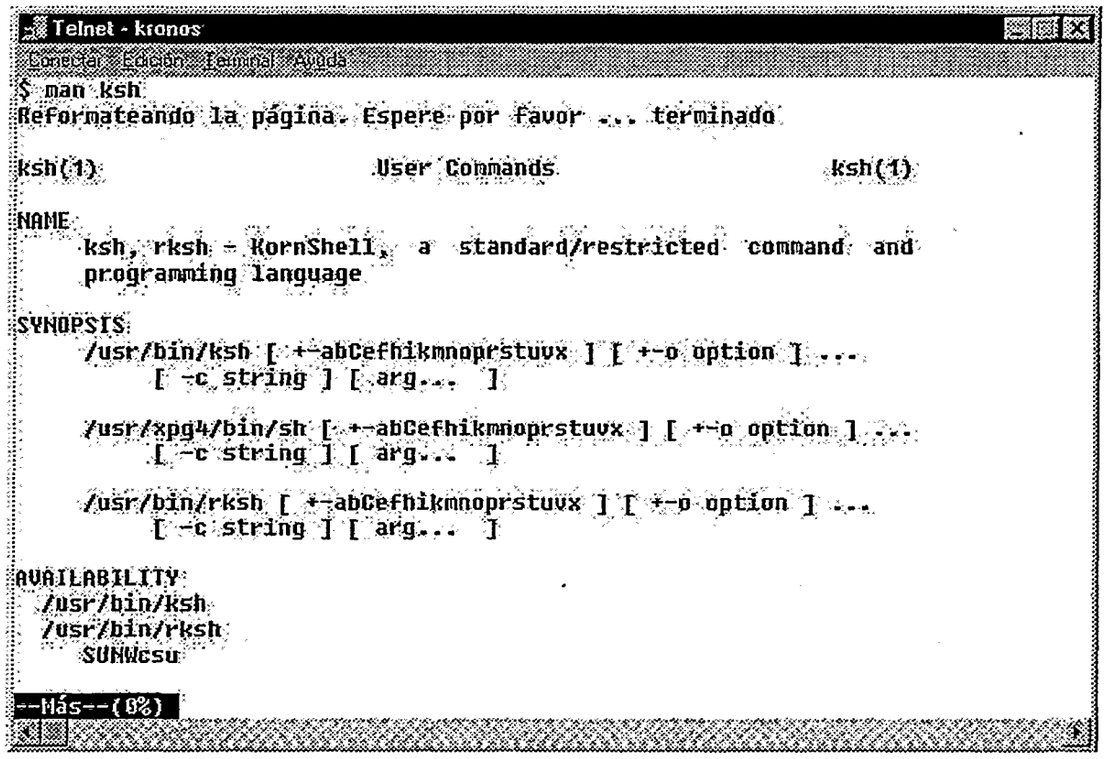

Figura 1: Ayuda de línea de mandatos.

Un ejemplo clásico de este tipo de ayuda de línea de mandatos son las páginas de ayuda del manual de UNIX (ver Figura 1). Como vemos en dicha figura, los contenidos de las ayudas de línea de mandatos enfocan la guía desde un punto de vista sintáctico, y no desde un punto de vista semántico, más cercano a las expectativas que los usuarios tienen de un sistema de esta naturaleza.

2.1.1.2 Hipertexto

Aún hoy, en casi todos los casos, la ayuda al aprendizaje que la mayor parte de las aplicaciones complejas facilitan a sus usuarios es documentación en forma de ficheros de ayuda con hiperenlaces. Estos ficheros son mostrados por aplicaciones específicas, y contienen hiperenlaces que permiten a los usuarios navegar por la información. Los usuarios pueden desplazarse entre unos documentos y otros, o entre distintas partes de un mismo documento, activando, habitualmente mediante pulsaciones del ratón, dichos enlaces. Mediante esta navegación se permite a los usuarios el acceso a temas relacionados con el tema que se esté tratando en el documento visible en cada momento: funciones similares, definiciones, explicaciones más detalladas, ejemplos, etcétera, bien sea en la misma ventana, substituyendo los contenidos actuales, o en una ventana nueva más pequeña. Actualmente, en algunos sistemas, la idea de los enlaces hipertexto ha sido generalizada a enlaces hipermedia que permiten acceder a imágenes, videos o direcciones URL.

Además de proporcionar la navegación mediante el uso de hiperenlaces, los sistemas encargados de mostrar la ayuda hipertexto proporcionan funcionalidades adicionales que ayudan a los usuarios a la hora de orientarse cuando inspeccionan los contenidos de la ayuda. Así, por ejemplo, es habitual que dichas aplicaciones faciliten una interfaz para realizar *búsquedas*, que permite a los usuarios acceder a los documentos que provean información acerca de los términos solicitados. Esta utilidad de búsqueda persigue solventar una de las mayores desventajas que provoca la presentación de información en forma de hipertexto, y que es la dificultad de encontrar a lo largo de los documentos la información relativa al tema que se está buscando. Otra de las utilidades adicionales que suelen ofrecer estos sistemas es una ayuda para la navegación, conocida como *historial*, que permite a los usuarios volver a visitar los documentos por los que anteriormente se haya pasado. De este modo, los usuarios tienen menos temor a navegar por los documentos, ya que siempre podrán ser capaces de volver a algún punto desde el que se encuentren mejor orientados y desde ahí retomar la navegación.

Como podemos ver, las facilidades citadas tienen más que ver con la búsqueda y exploración de la información que con la manera en que la ayuda se le facilita a los usuarios. Desde este último punto de vista, los sistemas de ayuda basados en hipertexto no suponen más que una adaptación de los sistemas descritos en la sección anterior a las nuevas interfaces gráficas de usuario.

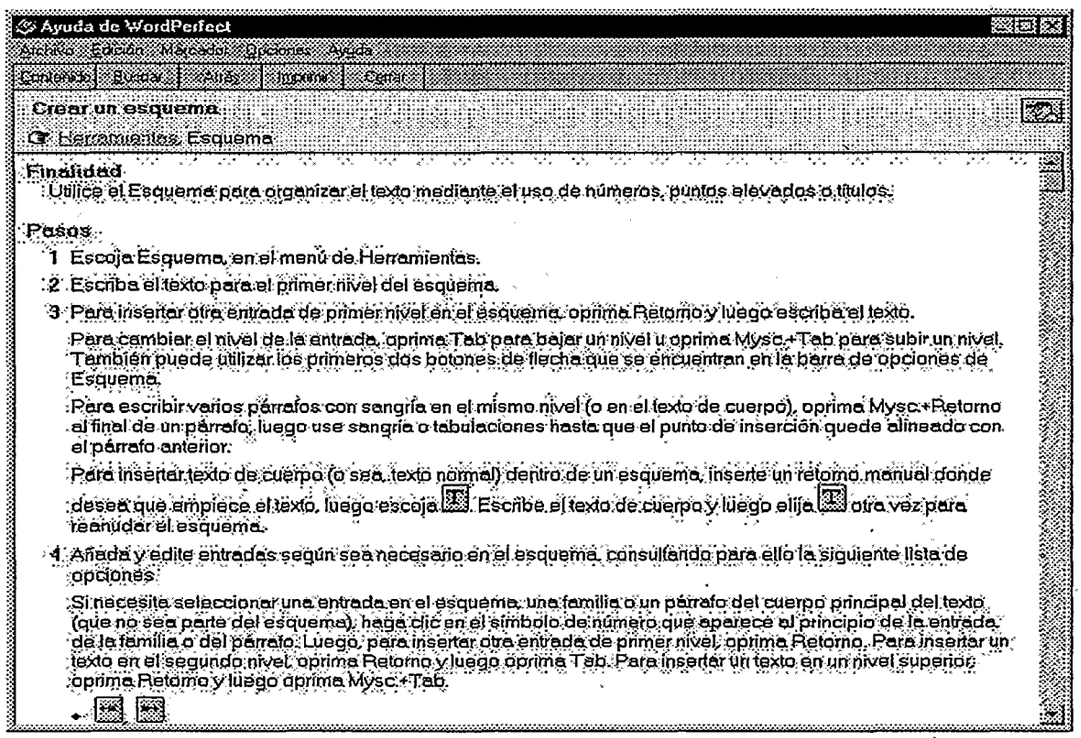

Figura 2: Ayuda basada en hipertexto.

En la Figura 2 se muestra una ventana de ayuda perteneciente a la aplicación WordPerfect 6.0 [Corel94], en la que se explica detalladamente cómo puede crearse un esquema para un nuevo documento. En este caso se aprecia que los hiperenlaces no se reducen sólo a textos, sino también a imágenes. Igualmente, en la barra de herramientas de la ventana de ayuda pueden observarse una serie de botones para el manejo del sistema de búsqueda y del sistema de navegación basado en el historial.

En sistemas de ayuda de este tipo se describe cómo es posible acceder a cada uno de los mandatos de las aplicaciones y qué funciones se supone que cada uno de ellos realizará. Las ayudas basadas en estos sistemas tienen inconvenientes relacionados con los contenidos de los ficheros [Contreras98] que limitan la utilidad de dichos sistemas de ayuda, ya que:

- Las explicaciones que se dan tienen un *nivel de abstracción muy bajo*. Los mensajes de texto, en la mayoría de los casos, se dan como una secuencia de *recetas*, interconectadas mediante hiperenlaces, que hacen referencia sólo a los objetos gráficos de la interfaz, y nunca hacen referencia a tareas de un mayor nivel conceptual. Como consecuencia de esto, las explicaciones de la componente de ayuda no están orientadas a las tareas de los usuarios, sino a las componentes gráficas de la aplicación (*widgets*), y los usuarios encuentran problemas en asociar la información que ellos están leyendo en la ayuda con las tareas conceptuales de mayor abstracción que son las que quieren llevar a cabo. Este problema de asociación de contenidos con tareas abstractas, de reestructuración de la información que reciben los usuarios, se conoce como *problema de asociación*.
- La ayuda tiene un *carácter estático* [Kearsley88]. La realización de tareas se explica completamente antes de que el usuario comience a llevar a cabo las tareas. Por ello, los mensajes que le son mostrados a los usuarios no pueden ser actualizados por el sistema de acuerdo a las opciones que el usuario escoge en cada momento concreto, ya que las explicaciones se generan previamente y no tienen en cuenta las acciones de estos usuarios. Debido a esta asincronía entre el momento en el que las explicaciones se dan y el momento en que el usuario realiza las tareas que se le explican, nos encontramos con situaciones en las que hay ciertas acciones dentro de un proceso que se explica cómo deben realizarse y que, a la hora de realizar las tareas, los usuarios no encuentran, pues han escogido caminos alternativos para realizar los procesos.
- En tercer lugar se nos presenta el problema conocido como *problema de visualización*, derivado del hecho de que las tareas se explican completamente antes de ser llevadas a cabo. Puesto que el tamaño de los dispositivos de salida habitualmente no permite mostrar simultáneamente más de una ventana con un tamaño razonable, los usuarios tienen que estar alternando entre la ventana de la aplicación y la ventana de la ayuda para realizar la tarea en paralelo a la lectura de las indicaciones. Esta incapacidad para poderse mostrar simultáneamente la ventana de la componente de ayuda y la ventana de la aplicación podría subsanarse si las explicaciones que se ofrecen a los usuarios pudieran darse en el momento adecuado, es decir, si en cada momento al usuario sólo se le facilitara la información imprescindible. En la Figura 3 puede verse cómo afecta el *problema de visualización* a una aplicación actual, en este caso el CorelDREAM 3D 8.0 [Corel97], en una pantalla capturada con la ventana de la aplicación maximizada y con una resolución de 800x600.
- Finalmente, estos sistemas de ayuda son *sistemas pasivos*. No hay un *feedback* desde el sistema de ayuda hacia los usuarios para indicarles cuándo alguno de los pasos que éste ha

realizado es incorrecto de acuerdo a las instrucciones que el sistema le ha facilitado. En este sentido, el sistema de ayuda es completamente pasivo e incapaz de darse cuenta de las acciones incorrectas (o mejorables) de los alumnos y, por tanto, de obrar en consecuencia.

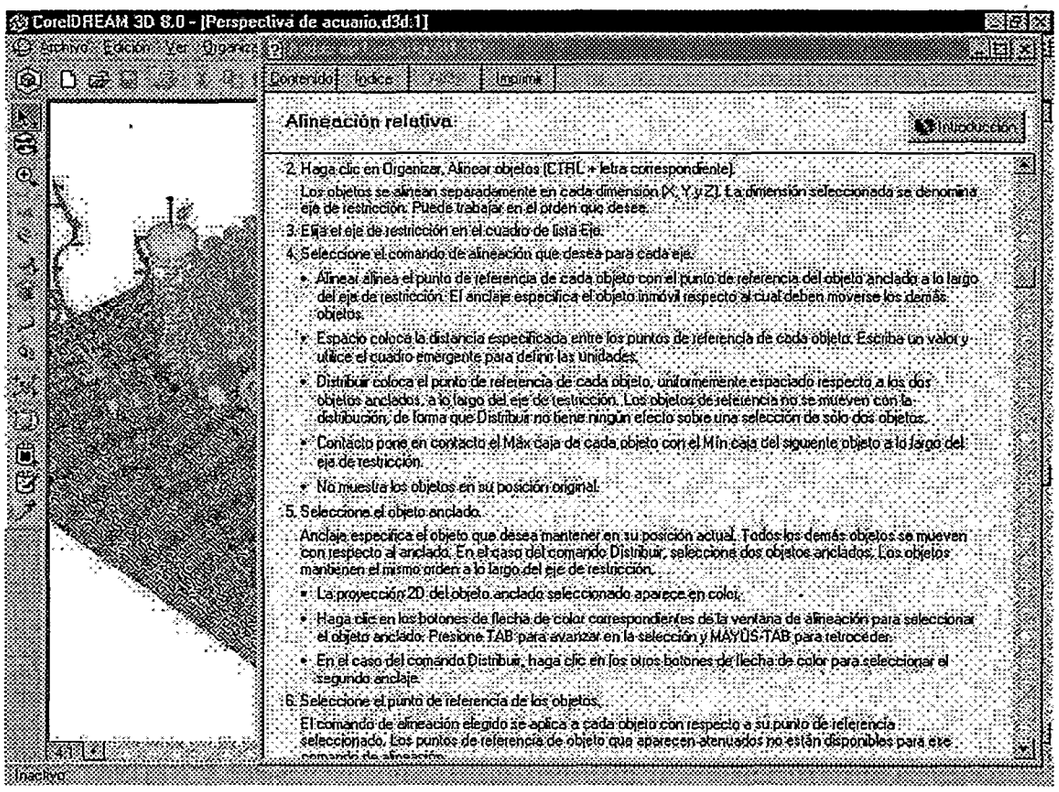

Figura 3: Problema de visualización en entornos actuales de ayuda.

Pese a que aún hoy los sistemas de ayuda mediante ficheros con hiperenlaces siguen utilizándose de manera generalizada para todo tipo de aplicaciones basadas en entornos gráficos de tipo WIMP, es evidente que poseen desventajas que los hacen poco útiles, fundamentalmente cuando se trata de ayudar a utilizar aplicaciones altamente interactivas, en las que las tareas que pueden llevarse a cabo son lo suficientemente complejas como para que se manifiesten los problemas citados.

2.1.1.3 Productos de Microsoft

Por último, dentro de los sistemas de ayuda tradicionales, vamos a incluir este apartado dedicado a la componente de guía que incorporan las últimas versiones del procesador de texto Word, el más utilizado en el entorno Windows y uno de los productos de uso más difundido dentro del mundo de los ordenadores personales. Con este conocido procesador de textos se ha editado, por ejemplo, este documento de tesis.

La veterana compañía americana de fabricación de *software* Microsoft, desde sus orígenes, ha dedicado un importante esfuerzo de investigación para lograr que sus productos sean lo más amigables posible (*user friendly*) para sus usuarios. Esto incluye no sólo su facilidad de uso (fácil accesibilidad de opciones y alta usabilidad), sino también el proporcionar componentes de guía que permitan a los usuarios noveles el fácil y rápido dominio de sus herramientas. Aquí nos vamos a centrar en su aportación al respecto de este último punto, basándonos para ello en las componentes de guía que acompañan a dos de las ultimas versiones de su procesador de texto: el Word 6.0 y el Word 97. Las mejoras incorporadas a las versiones de Office 98 y de Office 2000 son mínimas.

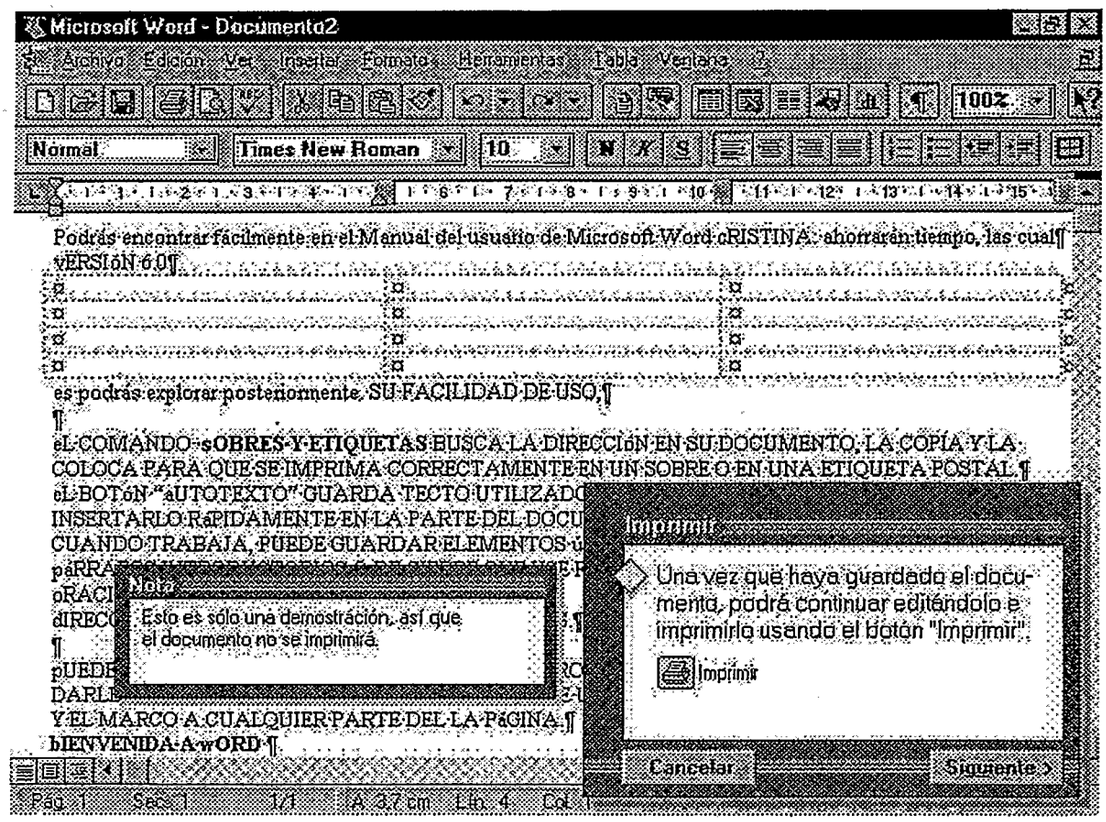

Figura 4: Tutor integrado en la versión 6.0a de MS Word.

La versión 6.0 de Word, integrada en la versión 4.0 de su paquete Microsoft Office, incluye un sistema interactivo tutor que guía a los usuarios durante el aprendizaje de la aplicación. Este sistema tutor consta de diferentes sesiones pedagógicas durante las cuales se va guiando a los alumnos en el aprendizaje progresivo del manejo de la aplicación, desde procesos simples como la creación de ficheros hasta procesos complejos como la personalización de las opciones de autoformato de textos. La sola presencia de estas sesiones pedagógicas proporciona al sistema de guía algo más que los sistemas de guía basados en hipertexto. No olvidemos que los sistemas basados en hipertexto proporcionan información a los usuarios acerca de conceptos relacionados con la aplicación o de cómo llevar a cabo ciertas tareas, pero raramente relacionan unos conceptos con otros, o las distintas maneras de realizar una misma tarea, algo que no parece un

problema para usuarios que tienen cierta familiaridad con las aplicaciones, que saben qué buscan y se sienten capaces de buscarlo en un momento dado. Sin embargo, este escenario es un poco irreal para el caso de los usuarios noveles en los que este trabajo se centra, que en un principio no saben cuáles son las tareas por las cuales deben comenzar el aprendizaje, cuáles son las más importantes, etcétera. Durante dichas sesiones pedagógicas, el tutor de Word 6.0 irá indicando a los alumnos, mediante secuencias animadas, cómo deberán realizarse las tareas más habituales.

Este tipo de tutores, pese a las mejoras evidentes que tienen sobre los sistemas basados en hipertexto, tienen importantes problemas, que se resumen a continuación:

- *Enormes costes de desarrollo*. Estos sistemas tutores no suelen trabajar sobre la *aplicación real* que intentan enseñar, sino que trabajan sobre una *versión recortada* o *modificada* que *simula* su comportamiento. De este modo, los usuarios tienen la impresión de que con lo que realmente están aprendiendo es con la aplicación real, y no con un *clon* de la misma. En el caso de Word 6.0, los alumnos no pueden acceder a la aplicación mientras el tutor está funcionando, de manera que lo que ellos ven es más próximo a un vídeo sobre el uso de la aplicación que a una sesión de trabajo con ella. Como se muestra en la Figura 4, el propio tutor avisa de que las consecuencias de las acciones que lleva a cabo son meras simulaciones de lo que realmente debería suceder. La creación de este clon de la aplicación real puede no acarrear unos costes de desarrollo tan elevados como la original pero, obviamente, su coste puede ser muy elevado.
- *Distancia entre el trabajo real y el trabajo con la herramienta tutora.* Este tipo de herramientas tutoras, como hemos mencionado, no permiten a los alumnos interaccionar con la aplicación, de manera que trabajan siempre con *ejemplos predefinidos* y nunca utilizan el contexto de trabajo de los propios alumnos. Lógicamente, la motivación de los usuarios de cara al aprendizaje sería mayor si la herramienta que les enseñara pudiera trabajar en el mismo contexto con el que ellos trabajan. Una consecuencia negativa de esto es que en ocasiones los usuarios se ven obligados a repetir el proceso de aprendizaje, debido a que tiempo que transcurre desde que ven los vídeos por primera vez hasta pueden restaurar su propio contexto y llevar a cabo la tarea sobre su propio trabajo puede ser muy largo.
- *Altos costes de mantenimiento*. Normalmente, cada cambio o mejora que se introduzca en la aplicación debería verse acompañado de la correspondiente puesta al día del sistema tutor para reflejar las modificaciones producidas, de modo que se garantice la *consistencia* entre la aplicación y lo que el sistema tutor muestra o explica a sus alumnos. A veces, este proceso de puesta al día puede ocasionar un gasto de tiempo incluso mayor que el del cambio o mejora en la aplicación para la que el tutor funciona.

Por su parte, Word97, integrado dentro del paquete Office97, abandona la filosofía de proporcionar tutores junto al programa principal. En su lugar, proporciona tanto funcionalidades avanzadas dentro de los clásicos ficheros de ayuda basados en hiperenlaces, como *agentes de confianza* (traducción de la voz inglesa *believable agents*), características que son comunes a todas las herramientas de dicho paquete.

Dentro del primer tipo de facilidades, se encuentran nuevas funcionalidades para los hiperenlaces. En la Figura 5 podemos ver un ejemplo de este tipo, en el que se explican los pasos necesarios para revisar la ortografía de un texto en un idioma distinto al seleccionado por omisión. Como puede verse, uno de los pasos de dicha tarea, marcado con el número 3, tiene como consecuencia el mostrar un cuadro de diálogo. Pues bien, el hiperenlace que se encuentra

justo después de la explicación de dicho paso permite mostrar directamente dicho cuadro de diálogo, sin tener que acceder manualmente a la opción de menú que lo muestra. Esta facilidad, pese a acortar el tiempo necesario para realizar la tarea, tiene el importante inconveniente de que los usuarios no aprenden a realizar la tarea, pues al no mostrarse qué se ha hecho para sacar el diálogo, el alumno puede no haber captado. Por tanto, estamos ante un caso parecido al de los tutores de la versión 6 de Word, ya que lo que se les muestra a los usuarios durante las explicaciones no se corresponde con el comportamiento *real* de la aplicación.

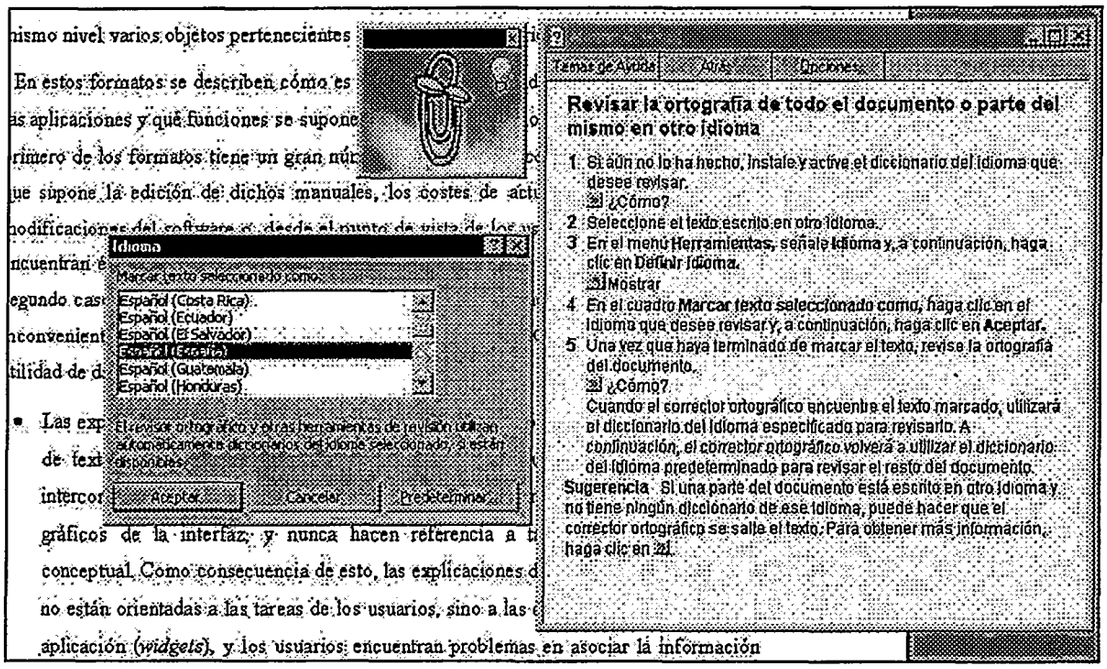

Figura 5: *Asistente* de Office97..

Por su parte, utilizando el *agente de confianza* de Word (realmente, es una componente compartida por todas las herramientas de Office), los usuarios pueden obtener ayuda directamente desde la aplicación según llevan a cabo nuevas tareas o tareas más complejas, en vez de tener que preguntar al sistema de ayuda. Esto se debe a que el agente, que basa su funcionamiento en *Redes Bayesianas*, observa el trabajo del usuario y reacciona para ofrecer consejos prácticos sobre la mejor manera de realizar las tareas, de modo que la curva de aprendizaje puede decrecer significativamente. En ocasiones el trabajo de este asistente puede ser útil, ahorrando tiempo al usuario inexperto. Por ejemplo, si en PowerPoint tratamos de seleccionar un elemento que se encuentra en el Patrón de Diapositivas de una presentación mientras estamos editando una diapositiva, el asistente se dará cuenta de que la manera de llevar a cabo esa tarea es seleccionando el elemento después de decirle a PowerPoint que nos muestre el Patrón de Diapositiva, ofreciéndose, incluso, a realizar dicha tarea por el usuario. Sin embargo, en otras ocasiones su presencia puede resultar demasiado intrusiva, incluso molesta, lo que sucede cuando no se *adivina* la verdadera intención de los usuarios, aunque solo sea porque éstos hayan realizado alguna acción de manera errónea.

Además de este comportamiento reactivo, el asistente incorpora técnicas de *Procesamiento de Lenguaje Natural* (*Natural Language Processing*) para permitir a los usuarios pedir ayuda utilizando sus propias palabras en vez de la terminología más particular de las aplicaciones. Recientemente, Microsoft ha anunciado la discontinuidad del servicio proporcionado por estos *agentes* con la próxima versión de Office, llamada Office XP [Microsoft2001a]. La versión oficial de la empresa es que las próximas versiones del producto serán tan sencillas de utilizar que nadie necesitará dichos agentes. Está por ver...

2.1.2 Trabajos de Investigación

A lo largo de esta sección vamos a detallar las principales aportaciones que se han llevado a cabo durante los últimos años desde el punto de vista de la investigación acerca de sistemas de ayuda y tutores, haciendo especial énfasis en aquellos trabajos que guardan una relación más estrecha con el trabajo expuesto en esta tesis.

2.1.2.1 Palanque et al.: Generación Automática de Ayuda

Como anteriormente hemos visto, uno de los mayores problemas que se presentan con los sistemas de guía tradicionales son sus elevados costes. Estos altos costes son consecuencia de la arraigada idea de considerar la componente de guía como algo totalmente independiente de la aplicación. Esta independencia provoca en los sistemas de ayuda dos problemas distintos: por un lado unos sistemas de ayuda muy pobres cuya relación con la aplicación a la que se refieren es mínima, como es el caso de los sistemas hipertexto y, por otro lado, sistemas de ayuda más integrados con las aplicaciones pero muy costosos de desarrollar, ya que duplican parte de la funcionalidad del sistema al que sirven. La idea de poder generar ayuda automáticamente la ayuda a partir de algún tipo de especificación formal de la aplicación va encaminada a disminuir estos altos costes de desarrollo. En esta sección se describe una primera aproximación a la generación automática de ayuda a partir de una descripción formal del sistema interactivo que, a su vez, sirve la especificación de la propia interfaz de la aplicación.

El trabajo de estos tres autores [Palanque93] describe cómo se puede generar automáticamente un sistema de ayuda contextual para aplicaciones interactivas creadas a partir de una especificación formal basada en un dialecto de *Redes de Petri de alto nivel* (*high-level Petri nets*) conocido como *Redes de Petri con Objetos*, PNOs [Sibertin-Blanc85]. Este sistema de ayuda transforma las Redes de Petri en diagramas de transición aumentados, que posteriormente son utilizados para contestar las preguntas de los usuarios referentes a los estados de estos diagramas. La utilización de modelos formales permite la especificación concisa, completa y sin ambigüedades de la parte del diálogo de la interacción hombre-computadora.

Este sistema de ayuda permite obtener respuestas para preguntas de los siguientes tres tipos, todas ellas relacionadas con el estado de disponibilidad de las distintas opciones de una interfaz:

- *¿Qué* acciones puedo realizar ahora?
- *¿Por qué* no puedo hacer esta acción?
- *¿Cómo* puedo activar esta acción?

Una red modeliza la evolución potencial del diálogo de manera que cualquier acción del usuario está asociada a una o varias transiciones de la red. Una acción puede realizarse si al menos una de sus transiciones asociadas está habilitada en la red, es decir, cada uno de los estados origen de

dicha transición tiene al menos un *token*. Si la acción no puede realizarse, entonces el *widget* asociado a la acción aparece como inhabilitado. Con esta semántica, la pregunta *¿Qué* acciones puedo realizar ahora? tiene por contestación inmediata el conjunto de transiciones habilitadas de la red.

Una *configuración* (*marking)* de la red es una asignación de *tokens* a los estados de la red. El número y posiciones de *tokens* puede variar durante la evolución de la red. La pregunta *¿Por qué* no puedo...? puede ser contestada examinando las distintas configuraciones posibles de la red: una acción estará inhabilitada sólo si ninguna de sus transiciones asociadas está habilitada en la configuración actual de la red. Así, la pregunta puede contestarse listando todos aquellos estados a los que le falta algún *token* para habilitar su transición asociada.

Para contestar las preguntas del tipo *¿Cómo* puedo activar...?, los autores construyen el *grafo de alcanzabilidad* de la red [Peterson81], cuyo cálculo es automático a partir de la Red de Petri, y que representa las diferentes configuraciones que son alcanzables por el sistema y las transiciones que hacen pasar de una configuración a las demás. Una vez construido el grafo, el algoritmo que contesta a la pregunta *¿Cómo* puedo activar...? parte del nodo que representa la configuración actual y busca un estado que sea el resultado de realizar la acción cuyo proceso de activación se desea conocer. Una vez localizado dicho estado, la secuencia de arcos seguida hasta localizarlo se corresponde con la secuencia de acciones que deben realizarse para activar la tarea y, después, realizarla. Hay que hacer notar que, para asegurar que el camino obtenido sea un camino mínimo, la búsqueda ha de ser realizada siguiendo una estrategia de búsqueda en anchura.

Este trabajo, relativo a la generación automática de la componente de ayuda de un sistema interactivo, representa sólo una parte de la búsqueda de una metodología genérica de desarrollo de aplicaciones a partir de la modelización mediante redes de Petri, con un enfoque orientado a objetos, de dichos sistemas.

El mayor inconveniente de este método de obtención de ayuda contextual radica en la complejidad de la modelización de sistemas interactivos utilizando el formalismo propuesto de Redes de Petri con Objetos. Como se muestra en algunos de los trabajos descritos en esta misma sección, otros paradigmas permiten generar las explicaciones de la ayuda basándose en representaciones más adecuadas, como modelos de presentación, modelos de secuenciamiento, o modelos de tareas. Como ventaja asociada a las representaciones basadas en métodos formales como, por ejemplo, las Redes de Petri, destacar la existencia de algoritmos que permiten la validación y simulación de las interfaces antes de ser construidas, con el ahorro de recursos que esto puede suponer.

Además, las explicaciones generadas carecen de la estructura que poseen las explicaciones obtenidas a partir de modelos jerárquicos de procesos. Nos encontramos, de nuevo, ante el *problema de asociación.* Esto es consecuencia directa de que los mensajes son obtenidos directamente a partir de Redes de Petri o del *grafo de alcanzabilidad,* que son representaciones mucho más lejanas al usuario que las descomposiciones de naturaleza jerárquica de procesos en forma de objetivos y subobjetivos.

2.1.2.2 Humanoid Hyper Help: Generación de Interfaz y Ayuda

Ya hemos visto que puede modelizarse el sistema interactivo de tal modo que el mismo modelo pueda ayudar al desarrollo de la interfaz de la aplicación y de la componente de ayuda de la misma. Ahora, se presenta la cuestión de estudiar qué sucede cuando dicha especificación integra

modelos de distintas naturalezas, lo cual puede presentar ventajas, por ejemplo, en el aspecto del diseño. Es decir, se plantea la posibilidad de generar automáticamente la ayuda para sistemas interactivos desarrollados bajo el paradigma de programación basado en modelos declarativos de la interfaz de usuario. Este paradigma, soportado por los entornos conocidos como MB-IDEs (*Model-Based Interface Development Environments*), cuentan, entre otras ventajas, con la posibilidad de dar soporte completo de una manera muy natural al desarrollo iterativo de aplicaciones [Buxton80]. De esta manera, los costes derivados del mantenimiento de la componente de ayuda se reducen al mínimo. En esta sección se describe uno de los sistemas pioneros en la generación de ayuda apoyándose en entornos de desarrollo basados en modelos declarativos.

Humanoid Hyper Help, H3 [Moriyón94], es un sistema que genera automáticamente una componente de ayuda por omisión para aplicaciones basadas en el entorno HUMANOID [Luo93, Szekely92] de desarrollo de interfaces basado en modelos declarativos [Szekely93], de manera que utiliza las mismas especificaciones que se utilizan para construir la interfaz de la aplicación. H3, además de construir automáticamente una primera versión de la componente de ayuda para la aplicación, ofrece el soporte necesario para permitir la modificación de dicha componente con el fin de adaptarla a las necesidades de diseñadores y usuarios.

H3 ofrece cuatro tipos de mensajes de ayuda:

- Mensajes que describen un elemento de la interfaz seleccionado por el usuario.
- Mensajes que contestan a preguntas del tipo *¿Qué* acciones puedo realizar?
- Mensajes que contestan a preguntas del tipo *¿Qué* se está mostrando aquí?
- Mensajes que contestan a preguntas del tipo *¿Dónde* puedo interaccionar? y *¿qué* pasará en cada caso?

Sin embargo, H3 no es capaz de producir ayuda orientada a tareas, ya que HUMANOID no incorpora modelos de tareas entre sus modelos de interfaz. Por ello, H3 no es capaz de ofrecer ayuda del tipo *¿Cómo* puedo aplicar el formato de este fichero a este párrafo de otro? o *¿Cómo* puedo añadir una cabeza de flecha al final de esta línea?.

Los mensajes de ayuda generados por H3 son concatenaciones de textos que contienen enlaces indicando dónde, en la interfaz de la aplicación, se muestra la información a la que se hace referencia en los mensajes o cómo puede accederse a ella. Asimismo, los mensajes producidos son sensibles al contexto, es decir, dependientes del estado actual de la aplicación. Así, cuando el usuario solicita ayuda acerca de un elemento inhabilitado, el sistema de ayuda explicará por qué lo está, mientras que si está habilitado explicará la función que dicho elemento cumple y a qué objetos afectará una interacción con él.

Cuando un usuario solicita ayuda apuntando con el cursor del ratón a un objeto de la interfaz y presionando la tecla de ayuda del teclado, H3 abre una ventana con los mensajes de ayuda relativos al elemento gráfico más pequeño bajo el puntero del ratón. Una paleta de botones de la ventana de ayuda permite a los usuarios solicitar ayuda acerca de otros temas, así como modificar el objeto gráfico que es el foco actual de las explicaciones. Los enlaces que hacen referencia a la interfaz de la aplicación utilizan un esquema de codificación basado en colores, de modo que cada enlace tiene asociado un color, y tanto el texto dentro del mensaje como la región de la interfaz de la aplicación a la que se refiere dicho texto son del mismo color.

En H3 los mensajes de ayuda se especifican mediante *reglas* del tipo:

When *Condiciones* **Then** *Descripción-de-Mensajes*

Las condiciones identifican el contexto en el que un mensaje en particular es apropiado, y las descripciones de los mensajes, mediante plantillas, especifican tanto los textos como los enlaces del mensaje que se generará. Cuando H3 evalúa las condiciones de una regla, las variables contenidas en ellas están asociadas a los objetos de los modelos que satisfacen dichas condiciones. H3 verifica esas condiciones accediendo a la especificación de la interfaz de la aplicación en el entorno HUMANOID. Sin dichos modelos declarativos, sería sumamente complicada la definición de reglas genéricas que funcionaran para cualquier aplicación.

La parte derecha de las reglas, es decir, las descripciones de los mensajes, consisten en piezas de texto, variables con valores asociados en las condiciones de la regla, y llamadas a funciones que recursivamente llaman a otras reglas para generar mensajes parciales embebidos. Las variables tienen dos efectos: primero, la descripción de la variable es substituida por su valor en los mensajes de texto (por ejemplo, el mandato *imprimir* es substituido por la cadena Imprimir) y, segundo, cuando los mensajes se muestran a los usuarios, H3 resalta los objetos gráficos de la interfaz de la aplicación donde se muestra el valor de la variable. Además, el sistema asigna el mismo color al texto que al objeto gráfico al que se refiere.

H3 contiene 32 reglas que cubren los mensajes por omisión generados para todos los bloques genéricos de HUMANOID (botones, menús, iconos, etcétera), los mensajes requeridos para explicar la noción de objeto seleccionado en un momento dado, y mensajes que gestionan las nociones de conjuntos de elementos que pueden ser mostradas con distintos formatos, como filas, columnas o grafos.

La calidad de los textos es crítica para el éxito de una componente de ayuda. Los seres humanos tienen mucha más facilidad para escribir buenos textos que las máquinas, de manera que H3 permite a los diseñadores modificar los mensajes generados automáticamente, para hacerlos más apropiados para cada caso particular. Los diseñadores llevan a cabo esta personalización de la componente de ayuda editando los mensajes que H3 produce en la propia ventana en que son mostrados. Este nivel de personalización de la ayuda es el nivel más simple permitido por el sistema. El siguiente nivel permite a los diseñadores añadir nuevos enlaces a los mensajes. Definir nuevos mensajes con condiciones sencillas se realiza cambiando los textos e insertando nuevas referencias, algo simple de hacer. El nivel más sofisticado de personalización de la ayuda generada es la aportación de nuevas reglas con condiciones complejas, lo cual requiere que los diseñadores aprendan el lenguaje de definición de las condiciones de las reglas.

Las facilidades de personalización de la componente de ayuda usan técnicas de *programación mediante ejemplos*, excepto para la definición de nuevas reglas con condiciones complejas, lo cual requiere añadir la especificación en el lenguaje de programación de las condiciones de las reglas. La manera de adaptar la ayuda permite a los diseñadores modificar los mensajes en la misma interfaz que posteriormente se encontrarán los usuarios finales de la aplicación, excepto en el hecho de que los mensajes pueden ser editados y que existen un panel extra de botones y menús que permiten controlar las facilidades de adaptación.

Sin embargo, a menudo el tipo de ayuda que los usuarios demandan es precisamente orientada a tareas, una ayuda que H3 no es capaz de proporcionar. En la Sección 2.1.2.5 se describe TWIW, un trabajo en el que se aumenta la expresividad de los modelos declarativos de HUMANOID,

incorporando modelos de tareas, para generar ayuda orientada a tareas. Sin embargo, como veremos, dado que la ayuda de este sistema sólo se basa en modelos de tareas, la generación de mensajes no se hace a partir de reglas como en H3.

2.1.2.3 Cartoonist: Ayuda Usando Animaciones y *Backchaining*

La técnica mayoritariamente utilizada como medio para hacer llegar la información a los usuarios, en este caso los alumnos, son los mensajes textuales, habitualmente acompañados de hipervínculos. Sin embargo, la complejidad de las aplicaciones interactivas que se desarrollan en la actualidad precisa de técnicas complementarias a las explicaciones textuales. Pese a que los mensajes en forma de texto resultan una técnica muy apropiada para proporcionar ayuda del tipo *¿cómo* puedo hacer...?, son menos apropiadas para proporcionar respuestas a preguntas del tipo *¿dónde* se encuentra...? o *¿cómo* se accede a...?. Así, las explicaciones con hipertextos dieron paso inicialmente a las *explicaciones multimedia* [Feiner90]. Sin embargo, en los entornos iniciales el grado de relación existente entre la componente de ayuda y la aplicación real era mínima, de modo que, por ejemplo, los videos mostrados durante las explicaciones no causaban efecto alguno sobre el estado que tenía la aplicación. O, por contra, la aplicabilidad de estos sistemas se reducía a ámbitos muy específicos como, por ejemplo, presentación de gráficos de negocios. Por ello, uno de los objetivos perseguidos es una mayor integración entre la componente de ayuda y la interfaz de usuario, de manera que se puedan incorporar animaciones que accedan directamente a la aplicación. En este sentido, el trabajo presentado en esta sección trata de facilitar ayuda utilizando animaciones sobre la propia aplicación.

Cartoonist [Sukaviriya90] es un sistema que genera ayuda automáticamente a partir de los modelos utilizados para construir la interfaz de una aplicación. Aparte del interés que este sistema despierta desde el punto de vista de la generación automática de ayuda, Cartoonist aporta también la capacidad de utilizar animaciones para mostrar al usuario cómo llevar a cabo las tareas. La ayuda, generada en tiempo de ejecución, es *dependiente del contexto* y contesta a preguntas del tipo *¿cómo* puedo hacer ...?. Este sistema está basado a la vez en un motor de *encadenamiento hacia atrás* (*backchaining*) y un subsistema de emulación mediante animaciones de eventos de ratón y entradas de teclado. El motor de encadenamiento hacia atrás utiliza las precondiciones y postcondiciones asociadas a las interacciones para buscar una secuencia de eventos que pueda realizar la tarea solicitada y, después, emula mediante animaciones dichos eventos. La secuencia de animaciones imita las acciones del usuario, y el controlador de diálogo de la interfaz hace responder a la aplicación ante dichas animaciones de la misma manera en que lo haría si el usuario realizara dichas acciones directamente. Este sistema fue implementado sobre el entorno de desarrollo de interfaces basado en modelos UIDE [Foley88a, Foley88b, Foley91, Foley94, Gibbs86], aprovechando las ventajas de su modelo de representación del conocimiento, que aúna información sobre la aplicación, su interfaz y las técnicas de interacción involucradas.

Cartoonist es también interesante porque muestra cómo un entorno de desarrollo basado en modelos declarativos de interfaces puede ser organizado para proporcionar control de diálogo de la interfaz de usuario en tiempo de ejecución y ayuda animada sensible al contexto a la vez, utilizando la misma especificación. Así, este sistema de ayuda utiliza las especificaciones de diseño de la interfaz de la aplicación, desde el diseño conceptual a los detalles léxicos de la interfaz deseada, y a partir de dicho conocimiento ejecuta escenarios de ayuda utilizando animaciones. Un subsistema de Cartoonist es utilizado frecuentemente para obtener valores

apropiados para las variables de los escenarios y para adaptar dichos escenarios al estado de la aplicación en el momento en que la ayuda es solicitada.

La representación y modelos utilizados incluyen el *Conocimiento de la Aplicación* (*Application Knowledge*), el *Conocimiento de la Interfaz* (*Interface Knowledge*) y las *Técnicas de Interacción* (*Interaction Techniques*). Los diseñadores especifican el *Conocimiento de la Aplicación* en Cartoonist definiendo, en primer lugar, las clases de objetos, sus atributos y las acciones que pueden realizar los usuarios para manipular las instancias* de dichas clases que tenga la aplicación. Para especificar cada una de dichas acciones, se describe qué parámetros lleva asociados, las precondiciones de la acción –las condiciones que deben cumplirse antes de poderse invocar una acción-, y las postcondiciones –las condiciones que se cumplirán toda vez que la acción haya sido llevada a cabo-. Esencialmente, las precondiciones y postcondiciones representan relaciones causa-efecto: cuando se cumpla determinada precondición y se realice una acción concreta, podrá asegurarse que las postcondiciones indicadas se cumplirán. Cartoonist utiliza la información semántica contenida en dichas condiciones para derivar las explicaciones ofrecidas en tiempo de ejecución como parte de la ayuda a los usuarios. Cada parámetro de una acción tiene también restricciones que limitan los valores que pueden ser aceptados por la acción. Las restricciones básicas consisten en la especificación de la clase a la que pertenecen los parámetros, su rango o la especificación de una limitación para sus valores, así como relaciones entre parámetros. Las restricciones de parámetros son tenidas en cuenta para la instanciación de valores de muestra que serán usados en los escenarios animados como, por ejemplo, qué objeto de una cierta clase se utilizará para una determinada acción o cuántos grados se usarán en una rotación de una pieza. Para completar la representación de una acción, el diseñador de la aplicación especifica *rutinas de aplicación* (*action routines*) asociadas a las acciones. La rutina es invocada cuando todos los parámetros de una acción tienen valores asignados, ya sea como consecuencia de una interacción del usuario o por la emulación de una animación, y la validez de dichos valores ha sido verificada por Cartoonist.

De manera similar, el *Conocimiento de la Interfaz* se compone de objetos interfaz y acciones de interfaz, utilizando el mismo esquema descrito anteriormente para el *Conocimiento de la Aplicación*. Este conocimiento permite la gestión de los objetos gráficos de la interfaz (ventanas, menús, diálogos, etcétera) de una manera independiente a la gestión del *Conocimiento de la Aplicación*.

El conocimiento acerca de las *Técnicas de Interacción* captura la información léxica sobre cómo los usuarios manipulan los objetos de la aplicación a través de sus dispositivos. Es decir, este conocimiento describe cómo a partir de interacciones básicas con los dispositivos disponibles los usuarios podrán interaccionar con la interfaz de la aplicación.

Una vez que las tres capas de *Conocimiento de la Aplicación*, *Conocimiento de la Interfaz* y *Técnicas de Interacción* han sido especificadas, éstas tienen que ser relacionadas entre sí. En la representación de una acción de la aplicación, las propias acciones y sus parámetros son unidades de información que provendrán del usuario en tiempo de ejecución para invocar las correspondientes rutinas de aplicación. Asignar unidades de información a acciones de la interfaz es relacionar esa unidad en particular con una tarea de la interfaz de usuario. Esta asignación se especifica en Cartoonist mediante *tablas de correspondencia* (*mapping tables*), en las que por un

* Anglicismo ampliamente utilizado para denotar un ejemplo o elemento de una clase o tipo.

lado se tienen las acciones y sus parámetros (*Conocimiento de la Aplicación*), y por el otro se encuentran las acciones de la interfaz. Las acciones de la interfaz deben ser ligadas, igualmente, a *Técnicas de Interacción* mediante tablas de correspondencia, de manera que se especifique mediante qué tipo de interacción un usuario podrá interaccionar con un objeto de la interfaz.

Los enlaces entre las acciones de la aplicación y las acciones de la interfaz, y las de éstas con las técnicas de interacción permiten a Cartoonist obtener una línea completa de información para realizar escenarios de ayuda animados. Partiendo de que cada capa de conocimiento en sí permite transmitir a los usuarios una parte de lo que necesitan aprender acerca de una determinada tarea, las relaciones entre dichas capas permiten ofrecer una fuente completa de información de ayuda desde el nivel léxico al conceptual.

Utilizando los modelos utilizados por el sistema, se puede inferir la secuencia completa de pasos que deben realizarse para completarse una determinada tarea con la aplicación. Por ejemplo, en un programa simple de diseño de circuitos digitales, la rotación de un objeto puede hacerse seleccionando la herramienta de rotación, después seleccionando el objeto a rotar y, finalmente, indicando los grados que se desea rotar el objeto. Para animar esta acción, Cartoonist selecciona un objeto de tipo puerta lógica (gate) de entre las distintas instancias disponibles en un momento dado, utilizando las restricciones especificadas para el parámetro que dicen que el objeto debe ser de tipo puerta lógica. Después, se escogerá un entero entre 0 y 360 para el número de grados de la rotación, de acuerdo a las restricciones establecidas. Con esto, Cartoonist ilustra el procedimiento mostrando el icono de un ratón en la pantalla, con el botón izquierdo pulsado mientras se despliega el menú donde se encuentra el mandato de rotación, soltando el botón cuando el cursor se encuentra sobre el elemento deseado, pulsando sobre el objeto elegido, y tecleando el valor escogido para el ángulo de rotación en el cuadro de diálogo que aparece a tal efecto. Las precondiciones y postcondiciones se utilizan para evaluar cuándo un contexto es adecuado para animar una acción. Por ejemplo, para rotar una puerta lógica, es necesario que exista alguna puerta. Si no existiera ninguna puerta lógica en el contexto actual de ejecución, un planificador incorporado en Cartoonist se encargaría de buscar una secuencia de acciones en el modelo de la aplicación para crear una puerta lógica. De este modo, las animaciones mostrarán cómo se crea dicha puerta en primer lugar y, tras ello, la manera de rotar dicha puerta lógica.

Utilizar animaciones para describir procesos dinámicos como la interacción hombre-computadora es, ciertamente, una característica muy atractiva. Sin embargo, como hemos visto en los párrafos anteriores, Cartoonist realiza elecciones mientras ejecuta con animaciones las acciones, escogiendo objetos y valores para los parámetros, siempre de acuerdo al contexto actual de ejecución y a las posibles restricciones que actúen sobre dichos parámetros. Estas elecciones, hechas de una manera razonable desde el punto de vista del sistema, aseguran que las animaciones tengan sentido para el sistema en la situación en que la aplicación se encuentre en el momento en que el usuario solicita ayuda al sistema. Sin embargo, estas elecciones no aseguran que los valores seleccionados sean los más apropiados desde el punto de vista de los usuarios. Esto ocasiona que, pese a que el sistema pregunta a los usuarios si éstos quieren que la situación de la aplicación se mantenga tras la ejecución de una animación o prefieren que se vuelva atrás al estado previo a la animación, lo más probable es que el usuario seleccione volver atrás y restaurar el estado original, ya que el sistema probablemente no haya realizado exactamente lo que el usuario tenía en mente. Por ejemplo, el sistema habrá seleccionado un ángulo de giro de 23 grados y no de 90, como desearía el usuario. Con respecto a ésto, nos encontramos aquí con el mismo problema al que nos referimos cuando tratábamos los sistemas de ayuda tradicionales, en

la Sección 2.1.1.2: la ayuda se ofrece en primer lugar y, a continuación, el usuario tratará de realizar su tarea a partir de la información que haya conseguido extraer de la primera fase. Como siempre, si la tarea que tiene entre manos es complicada, esta manera de proceder induce a cometer errores. Este problema se manifiesta con mayor rigor cuando la tarea para la que se solicita ayuda es suficientemente compleja, ya que en este caso el usuario debe volver a realizarla por completo tras haber deshecho lo que el sistema había realizado anteriormente al no ser exactamente lo que él deseaba.

De nuevo, nuestra opinión respecto a este punto, es que la información producida por la componente de ayuda de una aplicación interactiva debería ser entremezclada con la utilización real de la aplicación por parte del usuario, de manera que la ayuda sea ofrecida al usuario en el lugar y el momento adecuados. Así, por ejemplo, la componente de ayuda debería ser más flexible y permitir a los usuarios que seleccionaran los valores deseados para los parámetros de una manera interactiva.

Otro importante problema de este sistema es que la información que le es facilitada a los usuarios no está organizada jerárquicamente en términos de objetivos y subobjetivos, de manera que las explicaciones carecen de la estructura existente en las descripciones jerárquicas de procesos. Los modelos declarativos en los que se basa Cartoonist descomponen el conocimiento en tres niveles (*Aplicación*, *Interfaz* e *Interacción*), de los cuales sólo uno se corresponde con el nivel conceptual de la aplicación. En cambio, es muy frecuente, sobre todo cuando tratamos con aplicaciones interactivas, el encontrar tareas de alto nivel cuya modelización con un solo nivel de profundidad es tan compleja que la posterior explicación a los usuarios mediante el encadenamiento hacia atrás que usa Cartoonist sería difícilmente legible por su falta de estructura.

2.1.2.4 Pangoli y Paternó: Ayuda a Partir de Modelos de Tareas

Pese a que distintos tipos de modelos han sido utilizados para contribuir a la modelización de la interfaz de aplicaciones interactivas, el uso de los modelos de tareas merece ser tratado como caso aparte. Desde el punto de vista de la generación de ayuda, la razón para este tratamiento especial es que dichos modelos permiten incorporar el punto de vista de los usuarios acerca de la aplicación, teniendo en cuenta así no sólo el punto de vista de los diseñadores. En esta sección se describe el trabajo pionero en aportar ayuda sensible al contexto a partir de modelos de tareas.

Pangoli y Paternó [Pangoli95] describen una aproximación al diseño de una componente software para la generación automática de ayuda orientada a tareas de aplicaciones interactivas. Las ideas no requieren un gran esfuerzo de implementación, y la metodología seguida está basada en cuatro conceptos:

- Un modelo abstracto para objetos que capturen la esencia de las interacciones de los usuarios con las aplicaciones;
- Una modelización basada en tareas de la arquitectura de los sistemas interactivos;
- El uso de los operadores de concurrencia facilitados por la notación formal LOTOS [ISO88] para describir las restricciones temporales entre las distintas acciones disponibles en cada momento en entornos concurrentes.
- Un lenguaje de implementación orientado a objetos que provee soporte para una buena extensibilidad y reusabilidad.

El uso de los operadores derivados de la notación formal es importante, ya que garantiza una semántica precisa que ha sido definida matemáticamente. Esto permite la implementación de algoritmos que ofrezcan información relativa a cómo llevar a cabo las tareas. En este trabajo y en su continuación [Paternó98], la especificación de modelos de tareas se utiliza a la vez como parte del diseño de la interfaz de la aplicación y de su componente de ayuda. Es decir, la generación automática de ayuda orientada a tareas es parte de un esfuerzo más general para producir un entorno que soporte las distintas fases del desarrollo de sistemas interactivos. En esta aproximación, el diseñador primero realiza una descomposición jerárquica de las tareas, describiendo las tareas más abstractas en términos de tareas más concretas. Esta descomposición de tareas viene representada gráficamente en forma de un árbol que engloba todas las tareas de la aplicación, que puede ser creado y modificado mediante una herramienta interactiva de diseño. Los nodos más cercanos a la raíz de este árbol se corresponden con tareas muy abstractas, que dan idea de las tareas a las que el sistema deberá dar soporte, mientras que sus hojas se corresponden con *interactores* de la aplicación, es decir, con tareas que no pueden ser descompuestas más.

En este trabajo, los operadores de concurrencia temporal se utilizan para relacionar subtareas con igual nivel de abstracción. Estos operadores son: *paralelo* (*interleaving*, T1 | | | T2), donde las acciones de ambas subtareas pueden llevarse a cabo en cualquier orden; *disyunción* (*choice*, T1[]T2), donde es posible hacer una elección de entre un conjunto de subtareas y una vez que se comience a realizar una de las subtareas las demás no estarán disponibles hasta que se termine; *sincronizadas* (*sinchronization*, T1 | [a1,..an] | T2), que indica las acciones (a1,...,an) en las que dos tareas han de sincronizarse; *desactivación* (*deactivation*, T1[>T2), donde la primera tarea se desactiva una vez que la primera acción de la segunda se realiza; *habilitación* (*enabling*, T1>>T2), donde la finalización de una tarea habilita la realización de la siguiente. Además, algunas tareas pueden tener ejecución múltiple, lo que significa que cuando se terminan, sus acciones pueden comenzar de nuevo desde el principio.

Una vez que se ha especificado satisfactoriamente de manera gráfica el modelo de tareas de la aplicación, puede obtenerse automáticamente la especificación LOTOS, o bien derivarse automáticamente la correspondiente arquitectura *software*. La especificación LOTOS puede ser útil para realizar posteriores procesamientos como simulación automática de las posibles acciones, verificación de las propiedades expresadas en lógica temporal basada en acciones, etcétera.

Pese a que gran parte de la transformación desde la especificación de tareas a arquitecturas software basadas en interactores puede hacerse automáticamente, omitimos aquí la descripción detallada de cómo puede hacerse, para concentrarnos en la capacidad de derivar automáticamente ayuda orientada a tareas, tema que está directamente relacionado con el trabajo de esta tesis. Sólo mencionar que, para obtener una interfaz completa para una aplicación, el diseñador sólo tiene que definir las conexiones existentes entre los interactores y el núcleo funcional de la aplicación para especificar completamente los flujos de información.

Una ventaja de la modelización de sistemas interactivos basándose en modelos de tareas es la de que, como consecuencia, la componente de ayuda del sistema es capaz de obtener información dinámica de la ejecución de las tareas y ofrecer así ayuda orientada a tareas. La información fundamental es la asociación básica entre tareas e interactores, donde la composición de interactores puede asociarse a tareas más complejas y abstractas. De hecho, a partir de esta asociación, se puede conocer en cualquier momento qué tareas pueden realizarse y cuáles no.

Además, para aquellas tareas que pueden realizarse puede decirse qué secuencia de acciones deben ejecutarse para completarlas y, para las que no pueden ser ejecutadas en un momento dado, pueden aportarse las razones por las cuales dichas tareas están inaccesibles.

Este sistema tiene un *Gestor de Tareas* en tiempo de ejecución que recibe información dinámica acerca de los interactores activados y desactivados de entre todos los interactores que pueden reaccionar a eventos ocasionados por el usuario, la aplicación u otros interactores. Este *Gestor de Tareas* también tiene acceso, en tiempo de ejecución, al modelo de tareas de la aplicación, así como al estado de cada tarea y una descripción de su función.

En particular, una tarea básica se considera que está activa cuando su interactor asociado está habilitado. Para tareas no básicas, se considera que está activa cuando al menos una de sus subtareas lo está. Así, dada una interacción del usuario, puede decirse qué tarea básica se está realizando y cuáles son las nuevas tareas activas. De este modo, navegando por el árbol correspondiente al modelo de la aplicación se pueden encontrar qué tareas abstractas contienen a las tareas básicas que se han realizado. A partir de aquí, el sistema de ayuda puede ofrecer información de distintos tipos a varios niveles de abstracción.

Por todo lo anterior, se puede responder a los siguientes tipos de preguntas:

- *¿Por qué* no está habilitada esta tarea?
- *¿Cómo* puedo hacer esta tarea?
- *¿Cómo* puedo activar esta tarea?
- *¿Qué* tareas puedo hacer ahora?

Hay dos maneras de invocar al sistema de ayuda: o bien intentando interaccionar con un *interactor* inhabilitado o solicitando la ayuda explícitamente.

La respuesta a la pregunta *¿Qué* tareas puedo hacer ahora? Se encuentra de manera sencilla examinando el árbol de tareas de la aplicación y buscando las tareas activas. La componente de ayuda puede entonces facilitar información acerca de las tareas disponibles a los distintos niveles de abstracción hasta llegar a los más básicos, es decir, hasta llegar a las acciones que los usuarios tienen que realizar para llevar a cabo la tarea.

Cuando el usuario pregunta *¿Cómo* puedo hacer esta tarea?, la primera cosa que el sistema hace es verificar que la tarea en cuestión está activa. En este caso, se genera una breve explicación sobre cómo puede hacerse. Si la tarea no está activa, entonces el sistema comienza a buscar una manera de habilitar dicha tarea. Para ello, todo lo que necesita es descender por el árbol desde su raíz (que, por definición, siempre está activa, porque representa a la aplicación entera) hasta la tarea deseada, siempre a través de nodos activos. Según se baja hacia la tarea deseada, nos encontramos con algún nodo N desde el que no podemos seguir descendiendo, ya que la subtarea N' de N que incluye la tarea deseada está inhabilitada. Desde este punto, el sistema puede explicar por qué la tarea se encuentra inhabilitada examinando las relaciones LOTOS entre las distintas subtareas de N.

Ante preguntas del tipo *¿Cómo* puedo activar esta tarea?, el sistema tiene que tomar en consideración el árbol de tareas una vez más. Como punto interesante, cuando se trata de contestar este tipo de preguntas puede darse el caso de que no se encuentre un camino de activación, lo cual significa que en la sesión actual dicha tarea no puede ser realizada. Pangoli y

Paternó consideran que estas situaciones se deben a fallos de diseño que deberían evitarse en el diseño de las interfaces.

Partiendo de la búsqueda a través de la jerarquía que define el modelo de tareas, el sistema obtiene la lista de tareas que deben hacerse para conseguir el objetivo deseado. De este modo, se tiene la información necesaria para contestar las dos preguntas típicas sobre *cómo* habilitar una tarea y *cómo* realizarla. La primera pregunta tiene una respuesta dinámica, porque depende del estado actual de la sesión, es decir, de qué tareas se hayan llevado a cabo previamente. Por su parte, la segunda pregunta tiene una respuesta fija. Para ambas preguntas pueden existir más de una respuesta, ya que es común que haya más de una manera de realizar una tarea.

Ahora, el problema es cómo presentar toda esta información de una manera fácil de entender. El sistema realiza esto utilizando piezas de texto predefinidas, que son unidas entre sí para formar explicaciones con sentido, donde las unidades básicas consideradas son las descripciones de las tareas. El sistema utiliza la conectiva *'o'* cuando son posibles varias opciones, y la conectiva *'y'* cuando se tiene que realizar una secuencia de varias acciones. A continuación puede verse la salida dada por el sistema para preguntas sobre descomposición de tareas en una aplicación de gestión de mensajes electrónicos:

> *¿Cómo* activar la tarea: **Guardar un mensaje**?
>
> **Seleccionar un mensaje** pulsando en el título del mensaje, mostrado en el interactor 'coger'.
>
> *¿Cómo* realizar la tarea: **Guardar un mensaje**?
>
> **Teclear el nombre del fichero en el que quiere guardar el mensaje**, utilizando el interactor 'guardar'.
>
> *¿Por qué* la tarea **Guardar un mensaje** está inhabilitada?
>
> Porque primero tienes que **Seleccionar un mensaje.**
>
> *¿Qué* tareas puedo hacer ahora?
>
> Ahora puedes hacer:
>
> **Coger un mensaje que exista**
>
> o
>
> **Enviar un nuevo mensaje**

El trabajo presentado por Pangoli y Paternó tiene algunas cosas en común con parte del trabajo presentado en esta tesis. La más importante de todas es el hecho de que ambos sistemas se apoyan en modelos de tareas jerárquicos para ofrecer una componente de ayuda orientada a tareas. Este tipo de ayuda permite solventar el problema de asociación descrito en la Sección 2.1.1.2, ya que extiende la tradicional especificación funcional de los sistemas interactivos, que sólo tiene en cuenta los aspectos relacionados con las componentes internas de la aplicación, para incluir el punto de vista de los usuarios de la funcionalidad del sistema.

El conjunto de operadores de concurrencia temporal entre tareas presentados en el trabajo de Pangoli y Paternó permite expresar relaciones semánticamente más ricas que las utilizadas en el presente trabajo o que las aproximaciones basadas en precondiciones y postcondiciones de otras propuestas, como la descrita en la Sección 2.1.2.3 de esta tesis,.

Sin embargo, el subsistema DARTS presentado en esta tesis es más potente que el presentado en el trabajo de Pangoli y Paternó, ya que ofrece un entorno en el que el alumno puede practicar de una

manera *segura*, pudiéndosele permitir acceder sólo a parte de la funcionalidad de la aplicación en cada momento. De este modo, recortando en cada momento al usuario el acceso a distintas opciones de la interfaz, se evita que éste cometa errores que entorpezcan su propia actividad. La *seguridad* [Carroll87a] es un término que influye en el coste que supone el aprendizaje para los usuarios: ellos no desean aprender con un sistema en el que las nuevas acciones sean percibidas como fuentes de riesgo, es decir, que puedan causar algún daño. Así, recortar el número de opciones disponibles es una posible solución para aumentar la *seguridad* de la práctica e intentar evitar el problema conocido como la *paradoja de la producción* —centrarse en intentar utilizar las aplicaciones sin pretender aprender a utilizarlas antes, persiguiendo un aprovechamiento inmediato del uso- cuando se manifiesta en los usuarios noveles.

Otro de los puntos débiles del trabajo de Pangoli y Paternó estriba en que el sistema no tiene en cuenta las acciones que el usuario realiza, una vez que se le dan las explicaciones, para hacer un seguimiento de su actividad. Por ello, el sistema no puede ofrecer a los alumnos *feedback* acerca de su rendimiento, es decir, en la terminología empleada en la Sección 2.1.1.2, la componente de ayuda es una componente *pasiva*. Además, y dado que la ayuda que se ofrece no se actualiza dinámicamente, sino que los procesos se explican completamente cuando los usuarios solicitan la ayuda, estamos ante una componente de ayuda de carácter *estático* que, a consecuencia, puede originar el problema descrito de *visualización*.

2.1.2.5 TWIW: De Ayuda a Enseñanza *(Tutoring)*

La pasividad de la mayor parte de los entornos de ayuda existentes, que no son capaces de ofrecer indicaciones a los usuarios acerca de si han realizado correctamente las acciones que les han sido indicadas o no, se debe, fundamentalmente, a que los sistemas de ayuda no están complementados en tiempo de ejecución por un motor que se encargue de monitorizar las acciones de los usuarios y contrastarlas frente a lo que se les ha indicado que deben hacer. Su estaticidad hace que los usuarios reciban información que no tiene en cuenta el contexto exacto de las acciones que ya han realizado, de manera que en ocasiones estas explicaciones más que ayudarles pueden despistarles. Una solución sencilla para este problema es el proporcionar las explicaciones paso a paso, conforme al usuario avanza en la realización de las tareas, de manera que antes de explicar un paso se tenga en cuenta toda la información dinámica disponible sobre los pasos anteriores. En esta sección describimos un sistema encaminado a resolver estos dos problemas, y que ha servido de inspiración para parte del trabajo de esta tesis.

Teach me While I Work, TWIW [Contreras96] es una aproximación a la generación automática de tutores para las tareas de una aplicación para las que se ha definido su modelo. De este modo, este sistema, de nuevo, se aprovecha de la facilidad para razonar acerca de modelos, en este caso de tareas. Este sistema significó un primer paso para la generación de tutores para *software*, en un momento en el que los sistemas tutores existentes no se aprovechaban de las características y posibilidades derivadas del hecho de que lo que se pretendía enseñar era, en sí, un programa de ordenador.

TWIW parte de una modelización jerárquica de las tareas que forman parte de las aplicaciones. Cada tarea incorpora información sobre cómo será enseñada. TWIW construye automáticamente un sistema tutor por omisión para cada aplicación añadiendo información estandar sobre su método de enseñanza. El diseñador de la aplicación puede modificar y refinar el sistema tutor generado automáticamente modificando la información de enseñanza correspondiente a las tareas de la aplicación. Una tarea especial de TWIW, la *Tarea de Enseñanza*, se ocupa de los aspectos

genéricos relacionados con la enseñanza. TWIW incorpora, asimismo, un *Gestor de Tareas* que se responsabiliza del seguimiento de las acciones que se llevan a cabo mientras los usuarios trabajan con la aplicación.

La información que TWIW es capaz de generar en tiempo de ejecución es similar a la facilitada por el trabajo de Pangoli y Paternó, ya que ambos se basan en modelos de tareas. Es decir, pueden obtenerse explicaciones acerca de cualquier subtarea de una tarea determinada, y puede obtenerse *feedback* sobre las partes de la interfaz relacionadas con cada uno de los pasos de la tarea que está siendo explicada. En definitiva, a las explicaciones textuales habituales se le añaden las referencias a objetos gráficos de la interfaz y la posibilidad de que las distintas tareas aparezcan relacionadas jerárquicamente, formando tareas de un nivel superior de abstracción. Desde este punto de vista, la funcionalidad de TWIW no deja de ser la de un sistema avanzado de ayuda.

El aspecto fundamental de TWIW, que le permite comportarse como un sistema real de enseñanza de aplicaciones interactivas, se percibe cuando un usuario solicita que se le enseñe a realizar una tarea específica. Entonces, el usuario recibirá información precisa sobre la tarea en cuestión y sobre cómo realizarla. Tras ello, mientras el usuario trate de realizar la tarea interactuando con la aplicación original, TWIW verificará si los pasos que se están realizando son los adecuados y en el orden correcto. De acuerdo a la información de enseñanza de cada tarea particular, TWIW se comportará de una de estas dos maneras:

- Si la acción es *correcta*, entonces la ejecutará. Además, buscará cuál es el siguiente paso que debe realizarse a partir del modelo de tareas de la aplicación y realizará la acción preliminar de enseñanza asociada a dicho paso, que típicamente será la explicación de cómo deberá hacerse el siguiente paso.
- Si la acción no se corresponde con lo esperado, pueden suceder dos cosas. Si el sistema tutor se encuentra en modo *estricto* (que es el comportamiento por omisión del sistema) entonces se abrirá una ventana de aviso explicando qué era lo que supuestamente debería haber hecho el usuario, y la acción que ha intentado hacer el usuario no se llevará a cabo. Por el contrario, si el sistema tutor se encuentra en modo *libre*, se mostrará un mensaje de aviso, pero la acción se ejecutará. Además, se permite la definición de modos intermedios, dejando al diseñador la decisión acerca de dónde, cómo y cuándo un usuario puede actuar mientras se le enseña una cierta tarea.

TWIW está implementado sobre HUMANOID, el mismo entorno en el que se basó H3, pero aumentándose la expresividad del lenguaje de modelización de interfaces para incorporar modelos de tareas. Los modelos de tareas de los que se parte son modelos jerárquicos muy simples, en los que se diferencian tareas de dos tipos: *atómicas* y *compuestas*, donde las primeras forman las hojas de las jerarquías y las segundas el resto de nodos. Las tareas atómicas están directamente ligadas a las interacciones provenientes del usuario, por medio de los *comportamientos* (*behaviors*) de HUMANOID. Además, cada tarea lleva asociada información de enseñanza, que incluye dónde el usuario debe interaccionar con la aplicación para realizar una tarea, preacciones y postacciones para guiar a los alumnos y darles *feedback*, y cómo deberá comportarse el sistema si el usuario realiza una acción distinta a la esperada.

TWIW incorpora también la *Tarea de Enseñanza*. Esta es una tarea atómica que encapsula todo el comportamiento descrito anteriormente para cuando el sistema se encuentra enseñando una determinada tarea. Aparte de esto, esta tarea e similar a las demás, y es tratada de igual manera

por el *Gestor de Tareas*. Cuando el sistema entra en modo de enseñanza, la *Tarea de Enseñanza* se habilita y se inhabilitan el resto de tareas, de manera que esta tarea emula a las demás según las acciones del usuario y el estado del resto de tareas. Cuando el usuario trata de interaccionar con la aplicación, el sistema recibe un evento. TWIW verificará primero si dicho evento se corresponde con alguna tarea atómica que pueda ser activada. En caso afirmativo, el evento pasará al *Gestor de Tareas*, que ejecutará la acción asociada a dicha tarea atómica. En el modo de enseñanza, sólo la *Tarea de Enseñanza* podrá activarse, y ésta decidirá si la acción es o no apropiada, emulando o no la acción asociada, y llevando a cabo el resto de acciones asociadas con la enseñanza de la tarea en curso.

Uno de los mayores problemas de este sistema es que el modelo de tareas que utiliza es muy limitado en cuanto a la información contextual que es capaz de soportar. En particular, la ausencia de parámetros en las tareas, pese a que simplifica notablemente la implementación del sistema, limita también de manera notable su funcionalidad. Así, información contextual que de una manera natural iría asociada a determinadas tareas, es imposible de modelizar con el lenguaje de modelización de tareas de TWIW, lo cual obliga a la definición de múltiples tareas similares o a la incorporación de conocimiento extra dependiente de la aplicación. Además, las tareas de las aplicaciones actuales tienen una descomposición más compleja que la soportada por los modelos de TWIW. Así, por ejemplo, es algo muy común la existencia de tareas alternativas para obtener el mismo objetivo.

Además, el tipo de enseñanza que TWIW es capaz de ofrecer está orientado a usuarios relativamente avanzados, que tienen cierto grado de familiaridad con la aplicación que están utilizando, y tienen el conocimiento suficiente como para saber seleccionar la tarea que quieren aprender a realizar. En cambio, para un usuario novel, este escenario es irreal, y lo idóneo sería un sistema con el cual se pudieran montar cursos más complejos, que relacionaran unas tareas con otras de manera que formaran unidades pedagógicas. Estas unidades pedagógicas deberían relacionar entre sí tareas similares o complementarias, y facilitar el aprendizaje progresivo de la aplicación. Por tanto, parece deseable una herramienta que sea capaz no sólo de enseñar a realizar una tarea, sino capaz de guiar a los usuarios durante la enseñanza del manejo de las funcionalidades de una aplicación. Tanto este punto débil como el anterior son dos de los frentes a los que se atiende en el presente trabajo.

2.2 Tecnología de Diseño Basado en Modelos

Las técnicas de *desarrollo basadas en modelos* [Szekely93] dan soporte al diseño de sistemas interactivos a partir del uso de notaciones integradas de modelización, para permitir el diseño de los sistemas a partir de especificaciones con varios grados de abstracción. Estas técnicas están asistidas por herramientas conocidas como *entornos de desarrollo basados en modelos* o, del inglés, MB-IDEs (*Model Based Interface Development Environment*).

La noción de diseño basado en modelos data de unos 12 años atrás y, hoy en día, sigue en proceso de maduración y no se ha implantado todavía en ámbitos comerciales, pese a que existen diversas herramientas basadas en este paradigma desarrolladas en proyectos de investigación y se han realizado multitud de aplicaciones que las validan. Esta filosofía de diseño utiliza modelos explícitos, normalmente de naturaleza *declarativa*, que capturan la semántica de la aplicación y otros conocimientos necesarios para especificar la apariencia y el comportamiento de las interfaces de los sistemas interactivos. Bajo este paradigma, los desarrolladores de aplicaciones, en lugar de codificar las aplicaciones de manera procedural, definen modelos que representan comportamientos de la interfaz, a menudo a partir de la reutilización de modelos previos, y completan la especificación con la parte funcional dependiente de la aplicación. El objetivo es la identificación de componentes de interfaz reutilizables y la concentración de conocimiento en los modelos, lo cual significa una reducción de la cantidad de código procedural que es necesario para construir nuevas aplicaciones. El proceso general de desarrollo basado en modelos está representado en la Figura 6, y sus componentes serán descritos a lo largo de esta sección.

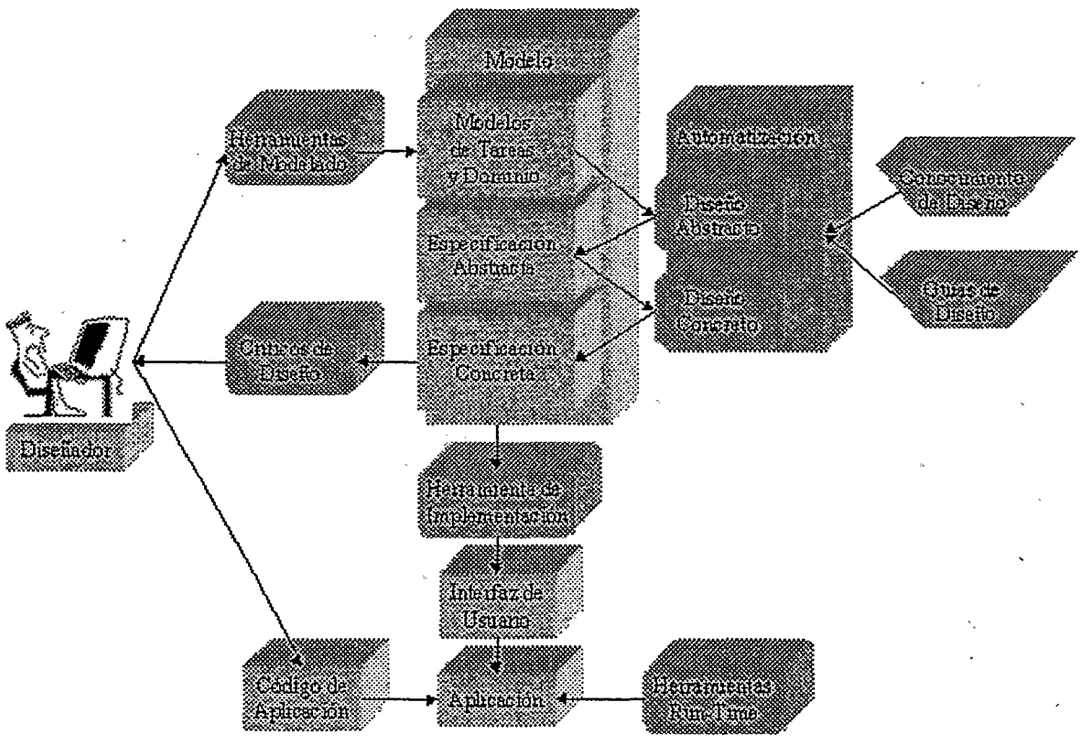

Figura 6: Proceso de desarrollo basado en modelos.

Los beneficios potenciales de esta representación explícita de los aspectos de la interfaz de usuario son importantes: generación automática de ayuda sensible al contexto, personalización de las aplicaciones, ayuda animada, diseño de las aplicaciones orientado a tareas de usuario, desarrollo independiente de la interfaz de usuario respecto del núcleo funcional de la aplicación, prototipado de interfaces, análisis de interfaces, etcétera [Puerta98]. Dado que las interfaces son cada vez más complicadas, y que los usuarios esperan cada vez más servicios de ellas, la

posibilidad de ofrecer estos servicios adicionales en tiempo de ejecución de manera fácil y a bajo coste es una de las características más atractivas de esta tecnología.

Los modelos que incorporan los MB-IDEs pueden clasificarse en tres capas, según su nivel de abstracción:

- El nivel más alto está compuesto por los *modelos de tareas* y el *modelo de dominio* de la aplicación. El primero de ellos, habitualmente consistente en una descomposición jerárquica de tareas en subtareas (pasos), representa las tareas que los usuarios pueden realizar con la aplicación; mientras que los modelos de dominio representan los datos que la aplicación trata y las operaciones aplicables a dichos datos.
- El nivel intermedio, o especificación abstracta de la interfaz de usuario, se compone de Objetos Abstractos de Interacción (Abstract Interaction Objects, AIOs), elementos de información y unidades de presentación. Los Objetos Abstractos de Interacción representan las tareas de bajo nivel de la interfaz, es decir, son abstracciones de interacciones básicas. Los elementos de información representan los datos que la aplicación deberá mostrar. Finalmente, las unidades de presentación son abstracciones de las ventanas: colecciones de AIOs y unidades de información que deben serles mostradas a los usuarios como un conjunto.
- El tercer nivel de modelización, o *especificación concreta de la interfaz de usuario*, especifica la manera en que las unidades de presentación le son presentadas a los usuarios, es decir, representa la interfaz en términos de controles gráficos, como botones, listas, líneas, círculos, etcétera, así como la *disposición espacial* (*layout*) de estos elementos en las ventanas.

Cada MB-IDE particular soportará la especificación de un cierto número de estos modelos, proporcionando soluciones por omisión, flexibles en mayor o menor medida, para el resto de modelos.

Una vez construidos los modelos, los MB-IDEs transforman la especificación concreta de la interfaz en una representación que pueda ser usada directamente por un *toolkit* o un *constructor de interfaces* (*interface builder*). Pero, en muchos casos, los MB-IDEs utilizan los modelos para generar más que la interfaz de usuario. Por ejemplo, JANUS [Balzert96], FUSE [Lonczewski96], UIDE [Foley88a, Foley88b, Foley91, Foley94, Gibbs86] y HUMANOID [Luo93, Szekely92] pueden generar partes significativas de la componente de ayuda; JANUS también genera los esquemas para la base de datos de la aplicación y parte del código necesario para gestionar dicha base de datos; y MASTERMIND [Szekely95, Szekely96a] genera el código necesario para que otras aplicaciones puedan conectarse a una aplicación y ser notificadas de cuándo se completan ciertas tareas, conocer el estado de la aplicación o invocar remotamente tareas de la aplicación, lo que permite la construcción de agentes que razonen acerca de la actividad de los usuarios.

Los MB-IDEs existentes pueden ser clasificados en dos grupos, dependiendo de cuál sea su objetivo fundamental:

- aquellos MB-IDEs cuyo objetivo primario es el automatizar el diseño de la interfaz, y
- aquellos que, por su parte, proporcionan a los desarrolladores lenguajes convenientes para expresar de manera más completa sus diseños.

El objetivo primario del primer grupo de MB-IDEs es automatizar tanto como sea posible el diseño e implementación de la interfaz de usuario. La mayor parte de estos MB-IDEs están orientados a

aplicaciones que permiten inspeccionar la información de bases de datos, modificarla, añadirla o eliminarla. Estos MB-IDEs hacen énfasis en los modelos de dominio y de tareas, no precisando que se faciliten las especificaciones abstracta o concreta de la interfaz, ya que las generan automáticamente a partir de estos modelos. Así, JANUS y las primeras versiones de MECANO [Puerta96a, Puerta96b] utilizaban sólo modelos de dominio, mientras que TRIDENT [Vanderdonckt94, Vanderdonckt95], ADEPT [Johnson92, Johnson95, Wilson93], DON [Kim93], MODEST [Hinrichs96], MUSE [Lim94] y TLIM [Paternó97, Paternó98] se centran en modelos de tareas, complementándolos con modelos de dominio.

Para que los MB-IDEs diseñen automáticamente la interfaz a partir de estos modelos, se precisan estos cinco pasos:

1. Determinar las unidades de presentación, es decir, las ventanas que serán usadas y la información que se mostrará en cada una.
2. Determinar la navegación entre las unidades de presentación, lo cual se realiza mediante un grafo de unidades de presentación que define qué unidades pueden ser invocadas desde qué otras unidades.
3. Determinar las AIOs de cada unidad de presentación, es decir, seleccionar qué componente realizará el papel de cada elemento de las unidades de presentación de una manera abstracta.
4. Relacionar los AIOs con objetos de interacción concretos, es decir, con alguno de los *widgets* disponibles en el *toolkit* sobre el que se ejecutará la interfaz.
5. Determinar el *layout* de las unidades de presentación.

Como puede observarse, los primeros tres pasos de esta secuencia tienen que ver con la generación de la especificación abstracta de la interfaz, mientras que los dos pasos finales componen la generación de la especificación concreta de la interfaz. Pese a que los MB-IDEs de esta categoría generan interfaces con muy poco esfuerzo, hay evidencias de que no es posible generar buenas interfaces, incluso para aplicaciones moderadamente simples, a partir de únicamente modelos de dominio y de tareas [Harning96, Wilson96, Szekely96b]. En particular, los pasos 1 y 3 anteriores pueden ser extremadamente complejos. El segundo paso puede derivarse a partir de la información acerca del secuenciamiento, que habitualmente viene incluida en los modelos de tareas. El paso 4 puede obtenerse a partir de los tipos de datos asociados a los atributos del modelo de dominio; y el último paso puede realizarse a partir de un estudio simultáneo de ambos modelos, mediante la aplicación de *guías de diseño* (*style guidelines*). Entre las dificultades más importantes se encuentran el que los usuarios necesitan ventanas que les muestren información obtenida desde distintos objetos; que los usuarios no quieren que se les muestre la información obtenida directamente a partir de los datos internos de la aplicación, sino quieren que dicha información les sea tratada, estructurada y resumida; y que en muchas ocasiones los formularios y tablas no permiten reflejar la misma riqueza semántica que los gráficos. Las dos primeras evidencias mantienen relación con la ausencia de una especificación para los elementos de información, mientras que la tercera significa que los AIOs son a menudo poco flexibles, de modo que difícilmente pueden determinarse de manera automática para un tipo de interfaces lo suficientemente amplio.

De esta imposibilidad para automatizar la generación de interfaces, se deriva la necesidad de colaboración entre los desarrolladores y los MB-IDEs. De este modo, en vez de automatizar, la tendencia de estas herramientas es la de colaborar en el diseño, ofreciendo sugerencias y

alternativas. Por su parte, los desarrolladores toman decisiones, aceptando sugerencias, escogiendo entre alternativas o proporcionando nuevas soluciones.

En contraposición a los sistemas que tratan de automatizar la generación de la interfaz, otro grupo de sistemas trata de facilitar a los diseñadores la labor de expresar diseños, modificarlos, modificar la plataforma a la que sus diseños van orientados, establecer los perfiles de los usuarios potenciales de sus sistemas, definir las tareas de sus programas, etcétera. Para ello, estos sistemas ofrecen lenguajes de especificación muy potentes, mediante los cuales se pueden especificar las propiedades de cualquier interfaz, y que poseen un nivel de abstracción que facilita la reusabilidad y la modificabilidad de las especificaciones. Estos lenguajes suelen permitir la especificación de modelos de los tres niveles comentados anteriormente. Normalmente, siempre tienen modelos de dominio (aunque no siempre modelos de tareas), que son usados para asociar a los AIOs los datos de la aplicación para que los usuarios los puedan modificar mediante las representaciones concretas de dichas interacciones. En este grupo se encuentran ITS [Wiecha89, Wiecha90], HUMANOID, MASTERMIND y MOBI-D [Puerta99a, Puerta99b].

Por último, hay también sistemas que incorporan un enfoque mixto. El caso más representativo es el de FUSE, que tiene un módulo de automatización, denominado FLUID [Buer96], cuya salida alimenta a otro módulo basado en especificación, de nombre BOSS [Schreiber94a, Schreiber94b].

Podemos, entonces, decir que la diferencia entre los sistemas de automatización del diseño y los basados en especificación se resume en su filosofía: mientras que los basados en especificación tienen un lenguaje de modelización *abierto*, los de automatización lo tienen *cerrado*. En los MB-IDEs de automatización los desarrolladores sólo pueden controlar el diseño utilizando unos pocos atributos que los desarrolladores de las herramientas seleccionan con dicho objetivo, limitando a los desarrolladores la posibilidad de controlar el diseño de la interfaz y, por tanto, limitando la calidad de las interfaces que pueden ser generadas. Por su parte, las facilidades para el prototipado rápido de interfaces que proporcionan los entornos de automatización no son alcanzables con los sistemas de especificación lo que, como consecuencia, conlleva un ahorro en los costes de desarrollo de las interfaces.

2.3 Sobre la Plataforma de Implementación

La creación de interfaces de usuario resulta difícil y consume muchos recursos. A menudo es un proceso largo y complejo, y llega a convertirse en todo un desafío su implementación, depurado y modificación. Un estudio mostró que una media del 48% del código de las aplicaciones estaba destinado a la interfaz de usuario, lo cual significaba alrededor de la mitad del tiempo de implementación del sistema [Myers92a].

Este alto coste de desarrollo justifica el cuidado puesto en la elección de la plataforma de implementación para las ideas de este trabajo, que han sido plasmadas en prototipos utilizando el *framework* AMULET [Myers97a, Myers97b]. Dado que todo el trabajo se apoya en la tecnología de *diseño basado en modelos de interfaces* y, en particular, en modelos de tareas, todas las ideas que durante el presente documento se expondrán podrían haber sido implementadas en cualquier MB-IDE que incorpore modelos de tareas, como los comentados en la Sección 2.2. Sin embargo, se ha optado por la implementación de un *sistema de gestión de modelos de tareas* propio, utilizándose a tal efecto AMULET. La razón para ésto es que se consideró que los MB-IDEs existentes actualmente no ofrecían el soporte necesario como para aventurarse a implementar estas ideas sobre ellos. Por otra parte AMULET ya había servido de base para la implementación

de un MB-IDE, MASTERMIND, lo que era un indicio, reafirmado por el estudio de sus principales características, de su idoneidad para la tarea. Como ya se ha dicho, todas las ideas que se expondrán en este documento son aplicables a cualquier sistema que incorpore modelos de tareas, con pequeños cambios de implementación.

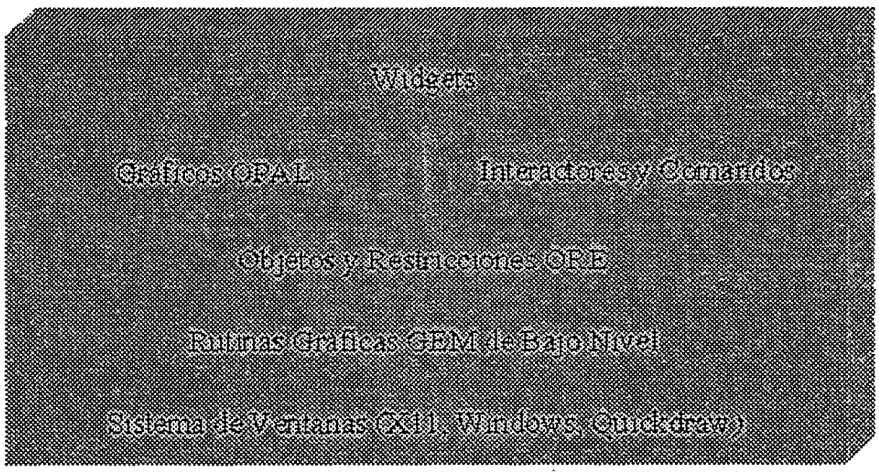

Figura 7: Capas de la arquitectura del entorno AMULET.

La presentación que se llevará a cabo a continuación del entorno de desarrollo AMULET pretende ayudar a los lectores a comprender mejor algunos de los aspectos tratados en el resto del trabajo. AMULET, *Automatic Manufacture of Usable and Learnable Editors and Toolkits*, permite la especificación en C++ de aplicaciones gráficas interactivas. Desarrollado en el departamento de *Computer Science* de la universidad de *Carnegie Mellon* (USA), aprovecha y amplía muchas de las capacidades de su antecesor, GARNET [Myers90a], basado en *Common Lisp*. El objetivo de este entorno es facilitar el diseño, prototipado, implementación y evaluación de interfaces de usuario. En la Figura 7 puede verse la representación de las capas de la arquitectura de este sistema.

Las aplicaciones basadas en este entorno son completamente portables a UNIX, Windows y Macintosh, ya que AMULET no accede directamente a los objetos gráficos de cada plataforma, sino que accede a su capa GEM (*Graphics and Events Manager*), encargada de uniformizar tanto la salida de gráficos como la entrada de eventos a las aplicaciones.

AMULET incorpora innovadores modelos de objetos, restricciones, *comandos*, salida y entrada. A continuación se describen por separado estos subsistemas:

- El *sistema de objetos*, ORE (*Object Registering and Encoding*), está basado en un paradigma de herencia *prototipo-instancia*, construido encima de C++, en el que no existe ninguna distinción entre clases e instancias, ni entre métodos y datos. Todo en AMULET viene representado por *objetos*, siendo cada uno un conjunto de atributos o *slots*. Los slots en estos objetos son similares a los atributos de otros entornos, con tres diferencias importantes. En primer lugar, los *slots* de un objeto pueden ser añadidos o destruidos en tiempo de ejecución. En segundo lugar, pueden contener métodos o datos de cualquier tipo en cualquier momento.

Y, en tercer lugar, la instanciación no sólo implica la herencia de *slots*, sino también la herencia de los valores que éstos tienen en el objeto prototipo. Posteriormente podrá asignarse un nuevo valor a los *slots* de las instancias, pasando entonces estas instancias a tener un valor *local* en dichos *slots*. Así, los *slots* de un prototipo pueden ser heredados por sus instancias, de manera que los cambios en el valor del *slot* del prototipo afectarán a todas las instancias de dicho prototipo que no lo hayan sobrescrito con un valor *local*. Este comportamiento de herencia por omisión de AMULET puede ser modificado independientemente en cada *slot*, de manera que puede hacerse que un determinado *slot* tenga siempre una copia *local* en cada objeto, evitando así que la modificación del valor de ese *slot* en un objeto afecte a todas sus instancias. Este comportamiento alternativo es el que utilizan otros sistemas más rígidos basados en el paradigma de herencia prototipo-instancia, como SELF [Chambers89], lo que acarrea una fuerte demanda de memoria en ejecución.

Para no añadir complejidad al sistema de objetos, desde los puntos de vista de implementación, de eficiencia y de desarrollo, ORE no implementa herencia múltiple, como sí que hacía el subsistema KR de su antecesor [Myers92b]. ORE ha sido, junto al modelo de entrada basado en *interactores* que se describe más adelante, básico en la implementación del trabajo presentado.

- El *sistema de restricciones* permite que cualquier valor de cualquier objeto pueda ser calculado dinámicamente por el sistema de manera transparente para los diseñadores, y soporta múltiples resolvedores. Como ejemplo especialmente interesante de coexistencia de varios resolvedores de restricciones, AMULET incorpora un resolvedor de restricciones de animaciones [Myers96a] que permite asignar animaciones a objetos preexistentes.

  Volviendo al tema de la herencia, algunos sistemas de herencia como el implementado en FormsVBT [Avrahami89] establecen que algunos *slots* hereden los valores de sus prototipos, mientras que otros lo harán de sus *dueños*. Esta confusión desaparece en AMULET debido a que siempre se hereda de los prototipos y, mediante restricciones, puede hacerse fácilmente que los *slots* adquieran los valores a partir de los *dueños*.

- Los *comandos* (*commands*) encapsulan toda la información necesaria para describir las operaciones de entrada, incluyéndose el soporte para varios métodos de *deshacer* (*undo*), *rehacer* (*redo*) o *repetir* (*repeat*) dichas operaciones. Cada vez que un usuario lleva a cabo una interacción, el sistema guarda en un objeto *comando* toda la información relevante, y llama a su método de activación para que la aplicación realice su acción asociada [Myers96b]. Mediante esta técnica, ya utilizada en el sistema Katie [Kosbie94], herramientas externas pueden razonar acerca de las acciones del usuario.

- El *sistema de gráficos* OPAL (*Object Programming Aggregate Layer*) permite la actualización automática de los gráficos. OPAL parte de las jerarquías parte-dueño de objetos ORE y del sistema de restricciones de AMULET para gestionar automáticamente la actualización de los objetos gráficos en pantalla cuando, por ejemplo, el área visible de un objeto gráfico se altera, o cuando se modifica el valor de algún atributo que afecte a la visualización de un objeto gráfico, como su posición o su color.

- Finalmente, el *modelo de entrada* basado en *interactores* es una versión refinada del que incorpora GARNET [Myers90b]. Las acciones del usuario son manejadas mediante *interactores*, objetos que permiten la reutilización y encapsulación de la información relativa a dichas acciones. Los *interactores* se asocian a objetos gráficos para que éstos puedan

responder a las interacciones del usuario. Distintos tipos de interactores predefinidos y parametrizables pueden manejar distintos tipos de comportamientos. Algunos parámetros de los *interactores* son comunes, como el evento que hace comenzar la interacción o los objetos sobre los que el interactor puede funcionar, mientras que otros son dependientes del tipo de *interactor*. Los estados que atraviesa un interactor en función de las acciones del usuario se pueden definir mediante un autómata específico asociado al mismo. A continuación se describen los tipos fundamentales de interactores del sistema, ya que se les hará referencia posteriormente, así como los *slots* particulares que definen su comportamiento:

1. *Interactor de selección* (*Am_Choice_Interactor*). Utilizado cuando se le da al usuario la posibilidad de seleccionar un elemento de entre un conjunto de elementos como, por ejemplo, cuando se selecciona un elemento de un menú o figuras dentro de una ventana gráfica. Como *slot* importantes debe resaltarse el que representa el objeto que es seleccionado.
2. *Interactor de disparo* (*Am_One_Shot_Interactor*). Representa las interacciones en las que se quiere que algo ocurra como resultado de la llegada de un evento del usuario: una pulsación de tecla, un *click* del ratón, etcétera. El atributo más representativo es el que define el elemento gráfico que se selecciona, en el caso de que el evento venga asociado a una pulsación del ratón.
3. *Interactor de mover* o *redimensionar* (*Am_Move_Grow_Interactor*). Este tipo modeliza las interacciones de cambios de posición o tamaño de objetos gráficos. Son comúnmente usados en programas tales como editores de diagramas. Los *slots* más representativos definen qué objeto se está modificando y cuáles son sus nuevos atributos para la posición o el tamaño.
4. *Interactor de edición de textos* (*Am_Text_Edit_Interactor*). Permite modificar el contenido de cualquier etiqueta de texto. Los *slots* que definen la interacción realizada son el objeto cuyo texto se ha modificado y el nuevo valor de su texto.

El sistema de interactores de AMULET es una implementación del diseño modelo-vista-controlador, MVC –*Model-View-Controller*-, introducido por Smalltalk [Krasner88]. En este diseño el modelo contiene los datos, la vista ofrece la representación visual de los datos, y el controlador manipula la vista. Sin embargo, en entornos previos, incluido el original, la vista y el controlador estaban estrechamente ligados, de modo que ambos debían modificarse cuando cambiaba el otro. De hecho, algunos sistemas como Andrew [Palay88] o Interviews [Linton89] combinaban ambos conceptos en uno único. En AMULET, los interactores permiten implementar el controlador, los gráficos de la capa OPAL facilitan la implementación de la vista y, finalmente, el modelo es implementado mediante objetos ORE o estructuras de datos convencionales. El avance fundamental radica en que los interactores son completamente independientes de los objetos gráficos a los que manipulan, de modo que pueden ser modificados por separado los unos de los otros.

El sistema ha sido ampliamente utilizado a nivel mundial, tanto por investigadores, como por estudiantes y por desarrolladores en general. Así, más de 32 proyectos de investigación a lo largo de todo el mundo se han llevado a cabo basándose en este sistema, incluyéndose varias tesis doctorales [Orosco98]; se han desarrollado 42 proyectos durante 13 cursos impartidos en 10 universidades, incluyendo las de Carnegie Mellon, Tennessee, LeTourneau (Colorado), Washington (EEUU) y Tampere (Finlandia), entre otras. Finalmente, al menos otras 14 compañías comerciales han hecho uso de este entorno para desarrollar la interfaz de sistemas tan dispares

como PRM o The Boss. El primero [AlliedSignal98], es un *Monitor de Precisión para Pistas* de aeropuertos que permite a los controladores monitorizar de manera segura el aterrizaje de los aviones en pistas paralelas bajo condiciones de reducida visibilidad, y que se encuentra operativo en aeropuertos como St. Paul en Minneapolis, JFK en New York o el nuevo aeropuerto de Hong Kong en Chek Lap Kok. El segundo de ellos [Linkworks98] es un sistema para la construcción de entornos de gestión de redes de satélites.

3 Gestión de Tareas de Usuario

Los *modelos de tareas* [Diaper89] han jugado un papel muy importante en los últimos años en el desarrollo de aplicaciones interactivas basado en conocimiento de alto nivel. Como ya hemos visto en la Sección 2.2, algunos entornos generan interfaces de usuario casi completas para aplicaciones partiendo únicamente de la descripción de las tareas de usuario. Estos sistemas basados en la modelización de tareas tratan de asistir en el diseño basándose fundamentalmente en mejorar la usabilidad de las aplicaciones y en adaptar los sistemas a las necesidades de los usuarios. Estas técnicas de modelización apuestan por el desarrollo de soluciones de diseño a partir de información sobre las tareas de los usuarios, de manera que se busca que el sistema sea compatible con las tareas a las que deben darse soporte.

Los modelos de tareas de usuario son esenciales desde el punto de vista de la implantación de una filosofía de diseño centrada en el usuario. Estos modelos describen las tareas que el usuario final realiza y qué interacciones deberán incorporarse para darles soporte. Pero estos modelos, además de ser importantes durante el ciclo de desarrollo de la aplicación, lo son también durante su ejecución, pues permiten razonar sobre ellos con el fin de proporcionar a los usuarios servicios orientados a las tareas que están realizando.

Otra de las razones principales que explican por qué los modelos de tareas son tán importantes es el hecho de que dan contexto a las acciones de los usuarios. Con la descripción de las tareas que los usuarios pueden realizar con las aplicaciones, cada paso que los usuarios lleven a cabo ya no se encontrará aislado, sino que estará relacionado con los pasos anteriores y posteriores. Este contexto viene dado por el uso conjunto de los modelos de tareas e información dinámica obtenida de las acciones de los usuarios. La importancia que esto puede tener para ciertos servicios, como las explicaciones ofrecidas a los usuarios sobre aspectos determinados de las aplicaciones, o como comportamientos adaptativos de las aplicaciones de acuerdo a lo que los usuarios han realizado o tienen que realizar, puede encontrarse en [Brézillon96].

A lo largo de este capítulo se describirá detalladamente ATOM, *Arquitectura para Gestión Orientada a Tareas* (*Arquitecture for Task Oriented Management*), una propuesta de arquitectura para *sistemas de gestión de tareas* en tiempo de ejecución. Esta arquitectura viene mostrada en la Figura 8, y será explicada en detalle durante este capítulo junto a detalles particulares del sistema ATOMS [García98b], *Sistema Avanzado para Gestión Orientada a Tareas* (*Advanced Task-Oriented Management System*), una implementación que se ha hecho de dicha arquitectura.

Los *sistemas de gestión de tareas* en tiempo de ejecución parten de la especificación de las tareas que los usuarios pueden realizar con los sistemas interactivos. Estos modelos de tareas están habitualmente organizados de manera jerárquica, y para su especificación se suele partir de un *framework* o un conjunto de *Modelos Parciales*, proporcionado por el entorno, que evite a los diseñadores partir de cero en su labor de modelización (ver Sección 3.1). Los modelos incluirán todas o algunas de las tareas que se pueden realizar con las aplicaciones, dependiendo de si se desea soporte para todas ellas o no, o de si es posible modelizar todas las tareas que se pueden llevar a cabo con las aplicaciones. El determinar qué tareas se modelizarán y cuáles se dejarán sin modelizar no es una labor sencilla y, en ocasiones, ni siquiera está en manos de los diseñadores. En efecto, en ocasiones las aplicaciones son tan genéricas que permiten realizar una cantidad de tareas limitada únicamente por la imaginación o la pericia de sus usuarios. Como ejemplo claro

de ésto, considérese el caso de una herramienta de diseño asistido. Esta aplicación tendrá, necesariamente, un número limitado de interacciones que los usuarios puedan llevar a cabo pero, sin embargo, permitirá hacer con ella tantas tareas de alto nivel como los usuarios puedan imaginar: desde dibujar una simple línea hasta diseñar el más complejo vehículo de navegación aérea. Aunque muchas otras aplicaciones no tienen unas características tan generales o abiertas como la que se acaba de citar, normalmente sólo es posible conocer en tiempo de diseño una cierta parte, mayor o menor, de las tareas que se pueden llevar a cabo con una aplicación. Es por ello que los entornos que permiten la modelización y gestión de tareas necesitan dar soporte mediante *Herramientas de Modelización* para que nuevas tareas sean añadidas a los modelos por los usuarios, según necesiten soporte de algún tipo para dichas tareas, preferiblemente de la manera más sencilla, de modo que no se precisen conocer el lenguaje de especificación de modelos que utilice el sistema. La manera más habitual de facilitar esto son los editores que, mediante técnicas visuales, permiten especificar tareas abstractas a partir de un cierto grupo de tareas que vienen predefinidas para cada aplicación. En esta tesis se propone la posibilidad de generar los modelos de tareas de manera completamente interactiva, utilizando inferencia a partir de ejemplos concretos de tareas, según se verá en la Sección 3.2.2.

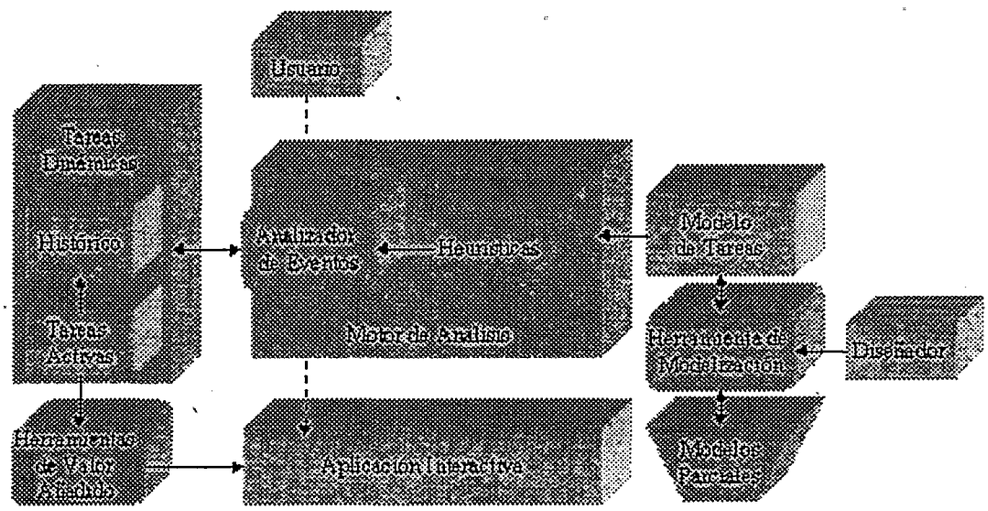

Figura 8: Arquitectura ATOM para gestión de tareas de usuario.

Una vez se tiene el modelo de tareas especificado, a partir de él y de las acciones que los usuarios realizan, los sistemas de gestión de tareas intentan determinar con exactitud las tareas de alto nivel que los usuarios están queriendo llevar a cabo. Para ello, la arquitectura propuesta incluye un módulo (*Motor de Análisis*) que utiliza técnicas de *análisis sintáctico* [Rodríguez97], similares a las que son utilizadas en el campo del *Procesamiento de Lenguaje Natural* (*Natural Language Processing*) [Maxwell94] para realizar un seguimiento de la actividad de los usuarios. Esta determinación no es una labor sencilla, ya que los modelos de tareas habitualmente contienen

gran cantidad de ambigüedades, que, en ocasiones, pueden ser resueltas mediante la aplicación de *criterios heurísticos*, como se verá en la Sección 3.3.1.

Toda vez que las tareas que se están realizando son conocidas, los sistemas de gestión de tareas pueden proporcionar soporte para que otras *Herramientas de Valor Añadido* razonen dinámicamente sobre los objetivos de los usuarios. Una ventaja fundamental de estas herramientas es su generalidad, ya que son comunes a todas las aplicaciones interactivas basadas en el mismo entorno y, por tanto, su desarrollo no implica costes adicionales a los diseñadores. Buena parte del trabajo de esta tesis se ha centrado en estas herramientas adicionales o de valor añadido, por lo que en el presente documento se describirán varias de ellas. En primer lugar, en la Sección 3.4 veremos cómo es posible acceder directamente a la interfaz de la aplicación con el objetivo de emular la realización, mediante animaciones, de las acciones que deben hacer los usuarios para llevar a cabo tareas. Un segundo caso de *Herramienta de Valor Añadido* aparecerá en la Sección 3.5, donde se describirá cómo aplicar esta filosofía para la gestión de flujos de trabajo. Posteriormente, en la Sección 4.2, mostraremos cómo es posible proporcionar guía dinámica, orientada a tareas, sobre un sistema de gestión de tareas de usuario. Después, en la Sección 0 se describirá EASE, una herramienta mediante la cual pueden describirse contextos de usuario mediante un ejemplo de cómo crear dichos contextos. Más tarde veremos cómo en los resultados de las herramientas de valor añadido descritas con anterioridad se asienta CACTUS, un sistema de gestión de cursos interactivos. Finalmente, en el Apéndice I se describirá brevemente una herramienta capaz de anticiparse a las acciones de los usuarios para evitarles tener que realizar acciones repetitivas.

3.1 Componentes de los Modelos de Tareas

Las aplicaciones *orientadas a tareas* tienen que definir sus modelos de tareas, que son representaciones jerárquicas, especificadas mediante un lenguaje declarativo, de las tareas que los usuarios pueden llevar a cabo mediante el uso de dichas aplicaciones. En el caso de ATOM, la especificación de estos modelos de tareas se descompone en dos partes, como se muestra representado en la Figura 9. Por un lado se tiene la definición de cuáles son las *tareas* disponibles en la aplicación y, por otro lado, las relaciones jerárquicas, así como otra información necesaria del modelo, viene especificada en forma de *reglas*.

Figura 9: Composición de los modelos de tareas en ATOM.

Para llevar a cabo la modelización de las tareas se utilizará un lenguaje de especificación de modelos de tareas que, habitualmente, será de naturaleza declarativa. Posteriormente, una componente de la arquitectura de ATOM deberá interpretar la especificación basada en ese lenguaje para poder ampliar la funcionalidad de las aplicaciones interactivas y así, por ejemplo, permitir la generación de cursos tutores para la enseñanza.

Cuando el diseñador del modelo de tareas especifica el modelo de tareas de una determinada aplicación, lo que está haciendo es crear instancias de los *prototipos de tareas* facilitados por la implementación particular de ATOM con que trabaje. Más tarde, cuando la aplicación se ejecute, el sistema creará instancias de dichas tareas especificadas por el diseñador, como se verá en la Sección 3.2.

A continuación se describirá, en primer lugar, en qué consisten y cómo se especifican los dos tipos de tareas existentes: *tareas atómicas* y *tareas compuestas*, ambas instancias de la *tarea genérica*. Posteriormente, se describirá cómo ambos tipos de tareas se relacionan entre sí mediante el empleo de *reglas* para permitir establecer jerarquías de tareas. Finalmente, se tratará el mecanismo de *herencia* en que se apoyarán los modelos de tareas basados en ATOM con el objetivo de fomentar la reutilización de modelos parciales de tareas y facilitar la construcción de repositorios de modelos.

3.1.1 El *Prototipo de Tarea Genérica*

ATOM facilita una tarea raíz, la *Tarea Genérica*, a partir de la cual pueden crearse las instancias que se deseen. Estas instancias se crearán siempre siguiendo el paradigma de herencia prototipo-instancia, según el cual las instancias heredan todos los atributos de su prototipo. De este modo, todos los atributos de la *tarea genérica* son heredados tanto por las *tareas atómicas* como por las *tareas compuestas*, que son instancias de ella. Además, no sólo se heredarán los atributos, sino también sus valores, de manera que los valores por omisión adoptados por los prototipos son importantes, ya que se heredarán a menos que las instancias los sobreescriban.

Atributo	Tipo	Valor por Omisión
Nombre	<CadenaTexto>	"TGenérica"
Descripción	<CadenaTexto>	Nulo
Prototipo	<Tarea>	Nulo
Parametros	<Indefinido>	Nulo

Figura 10: Atributos de la *tarea genérica.*

En la Figura 10 pueden verse representados, en tres columnas, los atributos más relevantes de la *tarea* genérica, donde la columna de la izquierda muestra el nombre, la columna central el tipo de dato esperado para el atributo y en la columna derecha se encuentra el valor que, por omisión, adopta el atributo. A continuación se detalla el significado cada uno de ellos:

Nombre. Cada tarea en los modelos de tareas ha de tener un *nombre*. Este atributo, que puede tener más de una palabra de longitud, permite a los diseñadores hacer posteriores referencias a dicha tarea para la especificación de herencia o de reglas, así como también permite que herramientas externas se refieran a estas tareas, como veremos en las Secciones 4.3.1 y 4.4.3. Además, el nombre de la tarea es importante porque deberá ser utilizado por el sistema tutor correspondiente para generar los mensajes de guía que le son mostrados a los usuarios. Así, por ejemplo, supongamos la tarea concreta de rotar una pieza en una aplicación de diseño. Un nombre apropiado para dicha tarea será *Rotar pieza.*

Descripción. Este campo es esencial para los propósitos de un sistema de enseñanza. El atributo de *descripción* contiene una descripción abstracta de la tarea, y su valor será mostrado en el momento apropiado a los usuarios durante la enseñanza de la tarea. Es importante hacer notar que este atributo sólo contendrá la descripción de la tarea asociada, es decir, información independiente de los procesos que deberán seguirse para realizarla, ya que esta información está implícita en las *reglas* del modelo y es gestionada por el sistema tutor. Las descripciones de tareas compuestas serán, por tanto, explicaciones abstractas acerca de los objetivos que se consiguen realizando dicha tarea. Por su parte, para tareas atómicas, las explicaciones podrán hacer referencia a procesos de más bajo nivel que tengan relación directa con la interfaz de usuario de la aplicación. En todos los casos, las descripciones de las tareas permiten hacer referencia a los valores que los parámetros de dichas tareas tengan asignados dinámicamente, con el fin de proporcionar información contextual a los usuarios. Siguiendo con el ejemplo anterior, una descripción adecuada sería *Rota una figura un cierto número de grado en cualquier sentido.*

Prototipo. Como se detallará más adelante, los modelos de tareas basados en la arquitectura ATOM permiten la especificación de tareas utilizando el concepto de *herencia.* Para ello, cada tarea incorpora una referencia a su *prototipo*, lo cual significa que compartirá con dicho prototipo los valores de todos aquellos atributos que no sean sobreescritos durante la especificación de dicha tarea. En el caso de la *tarea genérica*, que es la raíz de la jerarquía, este atributo tiene un valor nulo, pero para el resto de tareas este atributo tendrá una referencia a su tarea prototipo. Por ejemplo, tanto el prototipo de *tarea atómica* como el prototipo de *tarea compuesta* estarán relacionadas con la *tarea genérica* mediante este atributo.

Parámetros. Todas las tareas especificadas en ATOM pueden incluir *información contextual* acerca de su realización. Esta información contextual es de vital importancia para ofrecer a los usuarios las explicaciones sobre las tareas. Como veremos al hablar de las *tareas atómicas*, de las *tareas compuestas* y de las *reglas*, los parámetros de las tareas atómicas recogen sus valores a partir de valores relacionados con las interacciones de los usuarios, y estos valores son posteriormente transformados en parámetros de tareas compuestas, convirtiéndose así en información contextual de más alto nivel semántico.

Para la modelización de sus aplicaciones, los diseñadores no instanciarán directamente a partir de la *tarea genérica.* Ésta es la razón por la cual el tipo de dato esperado para el atributo *parámetros* de la *tarea genérica* es indefinido, ya que su tipo vendrá definido en sus instancias. De este modo, los diseñadores sólo trabajarán instanciando a partir de los prototipos de *tarea atómica*, de *tarea compuesta*, o de prototipos de tareas más específicos, que serán proporcionados por la implementación particular de ATOM con la que trabajen. Como veremos a continuación, ATOMS, la implementación que se ha realizado de la arquitectura descrita, proporciona una jerarquía completa de prototipos de tareas a partir de los cuales los diseñadores instanciarán sus tareas particulares. Igualmente, como veremos en la Sección 3.5 puede utilizarse el prototipo de *tarea genérica* para adaptarse la arquitectura ATOM con la finalidad de modelizar otros conceptos como, por ejemplo, el de *tarea remota* para la gestión de flujos de trabajo distribuidos.

3.1.1.1 Las *Tareas Atómicas*

Como se ha indicado, hay dos tipos de tareas que las aplicaciones deben definir en sus modelos de tareas: *tareas atómicas* y *tareas compuestas*. La incorporación del primer tipo de tareas a los modelos tiene por objetivo la modelización de las interacciones que los usuarios pueden llevar a cabo directamente con la interfaz de la aplicación. Por su parte, las tareas compuestas permiten la

modelización de tareas de mayor nivel de abstracción, es decir, de aquellas tareas que los usuarios tienen en mente cuando realizan un cierto conjunto de acciones más simples.

Atributo	Tipo	Valor por Omisión
Nombre	<CadenaTexto>	"Tatomica"
Prototipo	<Tarea>	Tgenérica
Patrón Interacción	<PatrónInteracción>	Nulo
Parametros	<ObtenciónParámetro>	Nulo

Figura 11: Atributos del prototipo de *tarea atómica*.

Como se muestra en la Figura 11, los atributos más importantes de las tareas atómicas son el *prototipo*, el *patrón de interacción* y las *expresiones de obtención de parámetros*, que se detallan a continuación.

Patrón de Interacción. Este atributo es el más importante de las tareas atómicas, ya que permite relacionar una *tarea atómica* dada con el tipo de interacciones del usuario asociadas a dicha tarea. Como se esquematiza en la Figura 12, cada *patrón de interacción* es una descripción abstracta de las interacciones de los usuarios, y encapsula toda la información que la componente de análisis de la actividad del usuario de la arquitectura ATOM necesita para decidir si una determinada interacción de un usuario representa una cierta tarea atómica. Los atributos de este tipo, junto a las reglas del modelo de tareas y el estado de las tareas que en cada momento el usuario esté realizando, permiten al sistema conocer con exactitud las actividades que el usuario hace y, por tanto, obrar en consecuencia.

Dado que los patrones de interacción representan el nexo necesario entre las aplicaciones y el *sistema de gestión de tareas* de usuario, el aspecto real que adoptará este atributo será dependiente de la implementación particular de la arquitectura ATOM. En el caso de ATOMS, la implementación realizada, los patrones de interacción son instancias de *patrones de objetos*. Un *patrón de objeto* es un objeto que a cada atributo asocia bien un predicado booleano que actúa sobre él, o bien otro patrón de objeto. Con esta definición recursiva se tiene que un patrón de objeto puede tener anidados otros patrones de objetos hasta niveles arbitrariamente profundos. La utilidad de los patrones de objetos radica en que representan la base para la implementación de un mecanismo de *matching* (determinación de similitudes) muy natural entre instancias de objetos y estas descripciones más abstractas de *grupos* de objetos, que pueden ser fácilmente especificados mediante lenguajes declarativos o mediante técnicas más avanzadas, como más adelante veremos. En el caso de los patrones de interacción, las instancias de objetos de las que hablamos son las interacciones de los usuarios, es decir, los eventos que llegan a las aplicaciones.

Por ejemplo, mediante un patrón de interacción podemos indicar que una tarea atómica se corresponda con la modificación de un campo de texto arbitrario de color *rojo* para hacer que su nuevo contenido contenga la palabra *"galgo"*. Para ello, bastará con definir un patrón de interacción en el que se indique que el atributo *color* del objeto situado en el atributo *objetoModificado* deba tener color *rojo* y que el atributo *texto* del este mismo objeto deba contener la palabra *"galgo"*.

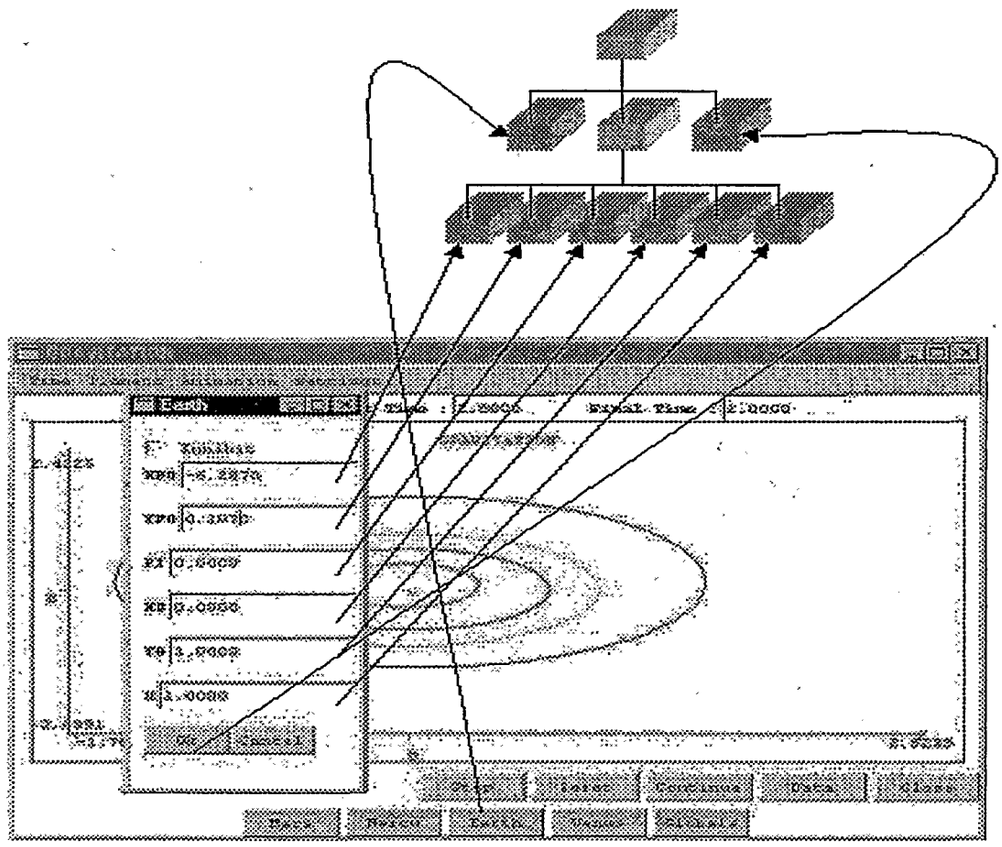

Figura 12: Correspondencia entre *tareas atómicas* e interacciones.

Como consecuencia de todo lo anterior, se puede concluir que el proceso de análisis de la actividad del usuario tendrá como uno de los papeles fundamentales el estudiar la similitud entre los eventos que llegan a la aplicación y los patrones de interacción asociados a las tareas atómicas para determinar qué tareas básicas se están realizando.

Hasta ahora hemos visto que la arquitectura ATOM debe facilitar prototipos de tareas para que se puedan modelizar tareas atómicas. Sin embargo, las implementaciones que de esta arquitectura se realicen tienen la libertad de poder proporcionar una mayor cantidad de prototipos de tareas, de modo que faciliten la labor de modelización mediante la posibilidad de reutilización de tareas más específicas y adaptadas a las necesidades de los diseñadores. A continuación, se explicará el caso particular de ATOMS, la implementación de ATOM que nos ocupa.

ATOMS proporciona no sólo el prototipo de *tarea atómica*, sino también otros prototipos de tareas atómicas más específicos. Con el fin de evitar en lo posible que los diseñadores tengan que crear sus patrones de interacción para cada tipo de tarea que necesiten definir, existe un prototipo de tarea atómica asociado a cada tipo de interactor existente en el entorno de implementación AMULET (ver Sección 2.3). Es decir, existen *prototipos de tareas atómicas* que representan selecciones, disparos de eventos, movimiento y redimensionado de objetos, y edición de textos. Cada uno de estos prototipos lleva asociado un patrón de interacción predefinido, cuyo único predicado establece que la interacción asociada a las instancias de esta tarea deberá activar en la aplicación un interactor que sea instancia de un tipo particular de interactor. Así, por ejemplo, a partir de la *Tarea Selección* se puede modelizar la tarea atómica que representa el seleccionar una figura en una aplicación de diseño, sin necesidad de conocer cómo se define el patrón de interacción para dicha tarea.

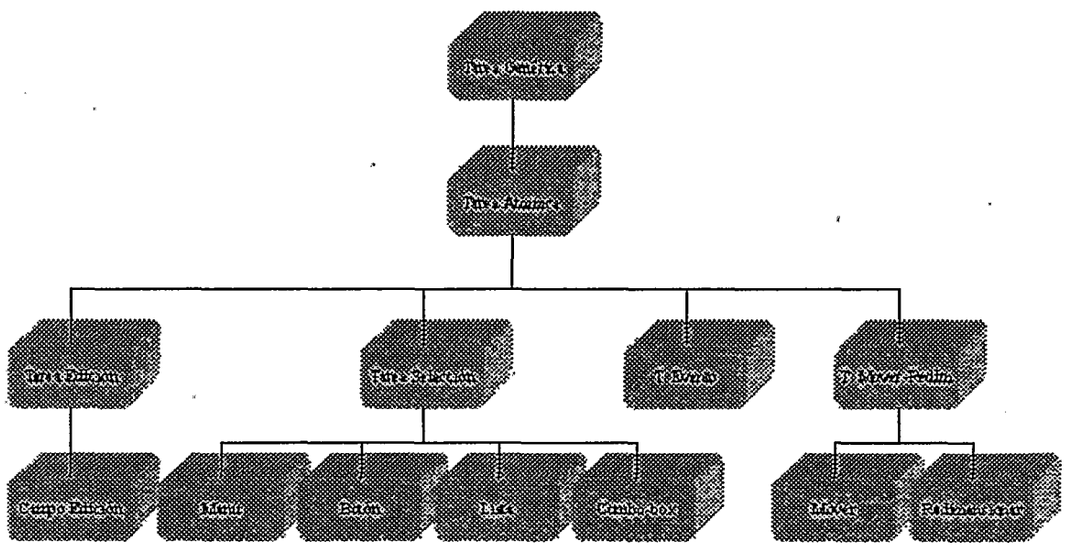

Figura 13: Jerarquía de prototipos de *tareas atómicas* de ATOMS.

Pero, además de estos prototipos de tareas básicas asociados a los tipos de interactores, existen prototipos aún más específicos, que llevan asociados patrones de interacción más complejos, y que modelizan las interacciones asociadas a los *widgets* de uso más frecuente en AMULET. Así, existen prototipos de tareas atómicas para las interacciones que se llevan a cabo con menús, botones, listas, campos de texto editables y cajas de selección. Además, ATOMS implementa un protocolo interno que permite que estos prototipos de tareas atómicas sean *parametrizables* -¡no confundir estos parámetros de los prototipos de tareas con los parámetros que proporcionan información contextual acerca de las tareas en tiempo de ejecución!- por los diseñadores cuando los instancien. Esto significa que cuando los diseñadores especifican sus tareas a partir de estos prototipos, facilitan también unos valores para algunos atributos e, internamente, ATOMS se ocupará de adaptar el patrón de interacción de la tarea para adecuarlo a dichos valores, de manera transparente para los diseñadores. De este modo, los diseñadores raramente habrán de definir sus propios patrones de interacción para las tareas atómicas, sino que simplemente partirán del

prototipo de tarea atómica que más se adapte a sus necesidades particulares y facilitarán valores para los parámetros apropiados. Típicamente, los atributos que parametrizan estos prototipos de tareas atómicas que representan a *widgets* son el nombre de la tarea, su descripción y algún otro atributo más particular como, por ejemplo, el texto identificativo que aparezca en el *widget* asociado.

La Figura 13 muestra la jerarquía completa de prototipos de tareas atómicas que ATOMS pone a disposición de los diseñadores para que modelicen las tareas atómicas de sus aplicaciones. En el nivel superior encontramos la tarea raíz de la jerarquía, de la cual se instancia el *prototipo de tarea atómica*. En el siguiente nivel nos encontramos con los *prototipos de tareas atómicas* asociadas a los distintos tipos de interactores de AMULET. Finalmente, en el nivel más bajo encontramos los *prototipos de tareas atómicas* asociados a los distintos *widgets* que provee dicho entorno de desarrollo.

Parámetros. Este atributo incluye una lista de expresiones, cada una de las cuales indicando cómo un *parámetro* de la tarea se deberá obtener a partir de valores relativos a las interacciones de los usuarios. En el caso de las tareas atómicas y de sus instancias, los valores de los parámetros provienen directa o indirectamente de valores relacionados con valores adoptados por elementos relacionados con las interacciones de los usuarios asociadas. Por tanto, al igual que sucede con los patrones de interacción, este atributo está estrechamente relacionado con la plataforma de ejecución sobre la que se asiente cada implementación particular de esta arquitectura. A continuación se describirá la solución adoptada en el caso de la implementación ATOMS.

En este sistema, los valores de los parámetros de las instancias de las tareas atómicas son obtenidos en tiempo de ejecución por el sistema a partir del objeto interactor que hace *matching* con el patrón de interacción asociado a la tarea atómica. Esto significa que los diseñadores del modelo han de tener un mínimo conocimiento de la estructura interna de la aplicación para decidir qué elementos de información le serán útiles como contexto a cada tarea y cómo obtener dichos elementos de información a partir del objeto interactor que representa dicha interacción en el entorno de desarrollo subyacente. Como ejemplo de cómo estas expresiones de obtención de información contextual para tareas atómicas se deben emplear, supongamos la tarea atómica de modificar la posición de una figura en el editor de una aplicación de diseño asistido. En este caso, el diseñador podría considerar relevante como contexto para esta tarea la figura cuya posición se está alterando, así como las nuevas coordenadas del vértice superior izquierdo de la figura. Entonces, habrá de proporcionar al sistema la manera en que dicha información relevante pueda ser recuperada a partir del objeto interactor que representa esa interacción en particular. En este caso concreto, la figura seleccionada se obtendrá, a partir del interactor, obteniendo el atributo Am_OBJECT_MODIFIED, que contiene el objeto que se está modificando. De manera similar, las coordenadas se podrán obtener a partir del objeto modificado accediendo a las propiedades Am_LEFT y Am_TOP. Como se puede ver, es evidente que el poder representar esta parte de los modelos necesita de un mínimo conocimiento del funcionamiento interno de AMULET. No obstante, y como veremos posteriormente (Sección 3.2.2), ATOMS facilita en gran medida esta tarea mediante el empleo de técnicas interactivas.

3.1.1.2 Las *Tareas Compuestas*

Es importante hacer notar que, esencialmente, todo lo que los usuarios pueden hacer con las aplicaciones son secuencias de tareas atómicas, pero es precisamente la posibilidad de definir tareas más complejas lo que confiere a los sistemas de guía la capacidad de poder ofrecer a los usuarios explicaciones con mayor riqueza semántica, estructuradas formando una jerarquía de conceptos, que descomponen cada objetivo en sus subobjetivos. En parte, es precisamente esta falta de estructura jerárquica la que hace de las ayudas basadas en *recetas* sistemas de guía poco adecuados, como vimos en la Sección 2.1.1.2. En estos sistemas, todos los pasos que deben llevarse a cabo para completar una tarea compleja se describen al mismo nivel, por ejemplo, "*debes realizar los pasos del 1 al 15*", sin ningún comentario acerca del hecho de que, por ejemplo, los pasos del 1 al 5 completan un cierto subproceso, del 6 al 9 otro, y así sucesivamente (ver representación de la Figura 14). Con el fin de modelar estos hechos, los modelos de ATOM proporciona el prototipo de *tarea compuesta*.

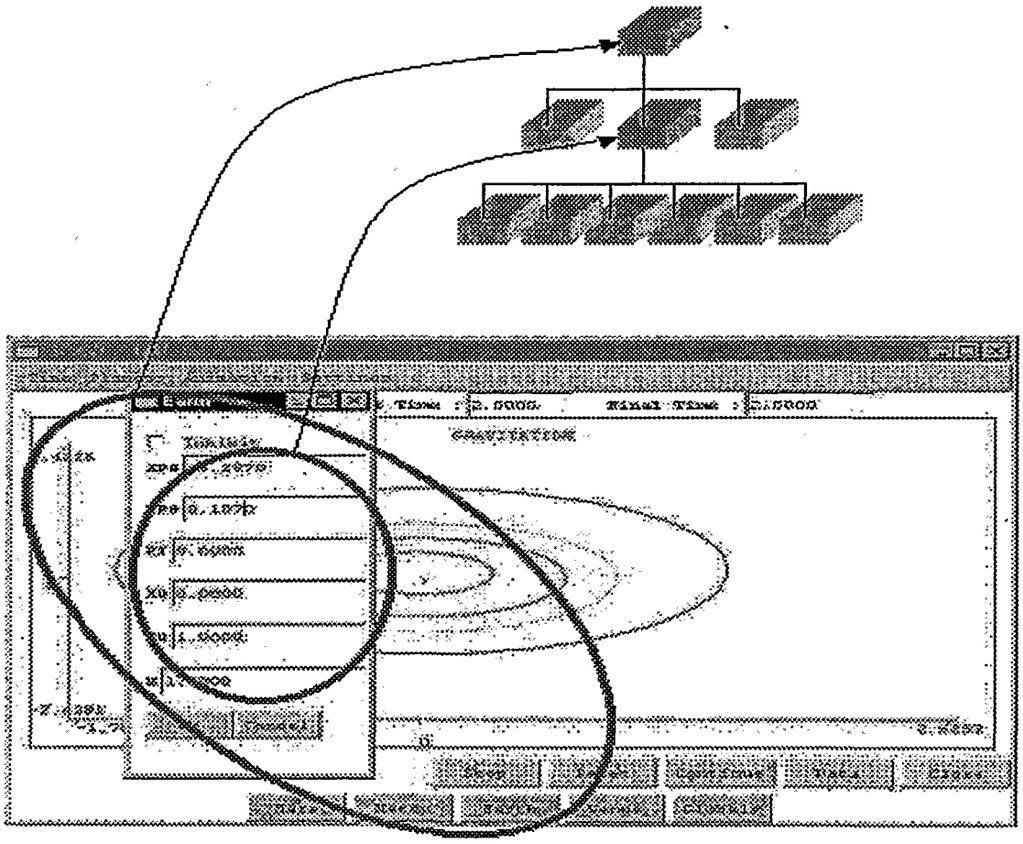

Figura 14: Correspondencia entre aplicación y tareas compuestas.

Los atributos más relevantes del prototipo de *tarea compuesta* (Figura 15) son:

Prototipo. Al ser una instancia del prototipo de tarea genérica, este atributo del *prototipo de tarea compuesta* tiene por valor la *tarea genérica*. En contra de lo que sucede con las tareas atómicas, lo habitual es que los diseñadores definan sus tareas compuestas instanciando directamente a partir de este *prototipo de tarea compuesta*, sin utilizar más niveles intermedios de

herencia, por lo que normalmente las tareas compuestas que definan los usuarios tendrán por prototipo al prototipo de *tarea compuesta*.

Atributo	Tipo	Valor por Omisión
Nombre	<CadenaTexto>	"Tcompuesta"
Prototipo	<Tarea>	TGenérica

Figura 15: Atributos del prototipo de *tarea compuesta*.

3.1.2 Establecimiento de Relaciones entre Tareas

Como ya se ha mencionado con anterioridad, los modelos de tareas de las aplicaciones, además de definir las tareas que pueden llevarse a cabo con ellas, definen las *reglas* que relacionan entre sí a dichas tareas. Cada *regla* liga una tarea compuesta, denominada *tarea padre* o *supertarea*, con un cierto conjunto de tareas, atómicas o compuestas, denominadas *subtareas* de la *tarea padre* o, simplemente, *subtareas*. Distintas *subtareas* de una misma *tarea padre* en una *regla* se dice que son *hermanas*, y así sucesivamente. De este modo, las *reglas* confieren una naturaleza jerárquica a los modelos. Gracias a esta jerarquía, puede conocerse de qué tareas se compone una tarea dada.

Además de relacionar una tarea con sus partes, las *reglas* también establecen qué tipo de relación existe entre la ejecución de las distintas subtareas de una tarea. La descripción de estas relaciones puede llegar a ser sumamente formal, como puede verse en LOTOS [ISO88], y será la implementación de la arquitectura la que determine con exactitud qué relaciones de ejecución de subtareas serán soportadas. En el caso de ATOMS, dado que su objetivo primordial es el de servir de soporte para la generación de entornos de enseñanza, se ha optado por implementar tres tipos de relaciones entre subtareas. Esta decisión se ha tomado considerando, en primer lugar, que la mayor parte de tareas pueden ser descritas usando estas tres relaciones y, en segundo lugar, que el resto no son relevantes desde el punto de vista del objetivo perseguido, ya que las ideas siguen siendo aplicables al resto de relaciones posibles. Estos tres tipos implementados son los siguientes:

- *Secuencia*. En este caso, las subtareas de una tarea dada deberán completarse en un orden preestablecido, no pudiéndose comenzar una subtarea hasta que las subtareas anteriores hayan finalizado. Este tipo de secuenciamiento suele ser el más común y se presenta, a menudo, en los procesos que abren cuadros de diálogo, en los que normalmente una subtarea modelizará el proceso de su apertura, otra los procesos que dentro del cuadro se realicen y, finalmente, otra subtarea modelizará las acciones encargadas de hacer desaparecer el cuadro de diálogo.
- *Y*. Esta relación establece que todas las subtareas de una tarea deberán completarse para que la tarea se considere terminada. El orden en que las subtareas se realizan no será determinante, ya que podrá ser cualquiera. Este tipo de secuenciamiento es muy común en los procesos que tienen lugar dentro de los cuadros de diálogo, en los que los usuarios deben

rellenar o seleccionar determinados campos, pero no se ven forzados a seguir ningún orden preestablecido.

- *Xor*. En este último caso, se considerará terminada la tarea cuando alguna de sus subtareas se realice, no siendo preciso completar el resto. La especificación de una regla con este tipo de secuenciamiento y N subtareas es equivalente a la especificación de N reglas distintas, todas ellas con la misma tarea padre y cada una con una de las subtareas, ya que durante el proceso de análisis de eventos todas las reglas tratarán de ser aplicadas y, por tanto, existirá una disyunción implícita entre ellas. Este tipo de secuenciamiento *Xor* es típico de muchas tareas que relacionan varias interacciones atómicas de efectos idénticos como, por ejemplo, la pulsación de un botón de *Guardar* y su opción equivalente de menú.

Las reglas también definen qué tareas pueden tener una *ejecución opcional*, y bajo qué condiciones serán opcionales estas ejecuciones. Por ejemplo, muchos cuadros de diálogo permiten editar propiedades para las que el sistema facilita valores por omisión, de manera que según qué condiciones se den, no tendrán por qué modificarse estas propiedades. De la misma manera, las reglas también pueden incluir información de qué tareas pueden tener *ejecución múltiple*, y bajo qué condiciones podrá seguirse realizando la tarea. Por ejemplo, cuando en un editor se desean agrupar varios objetos, la subtarea de seleccionar los objetos a agrupar puede ejecutarse tantas veces como objetos se quieran seleccionar.

Otra información presente en las reglas es la relativa a los contextos de las tareas, es decir, a los valores de sus *parámetros*. Esta información sobre el contexto tiene dos partes. Por un lado, están las restricciones que gobiernan la aplicabilidad de la regla dependiendo de los contextos de las tareas involucradas en la misma. Estas restricciones pueden forzar a que existan determinadas relaciones entre los valores que adopten los contextos de las subtareas. Por ejemplo, como parte de una tarea podemos tener una subtarea que cree un objeto y, posteriormente, otra que modifique ese objeto, y no otro.

Por otro lado, en las reglas se especifica cómo la información contextual de las subtareas de la regla se transforma en información contextual para la tarea padre de la regla. Al no estar ligadas directamente la tareas padre de una regla a la aplicación interactiva–por ser siempre tareas compuestas-, los parámetros de las tareas padre no provienen directamente de información asociada con las interacciones de los usuarios. En su lugar, los valores que adoptarán los parámetros de las tareas compuestas, una vez éstas se instancien durante la ejecución de la aplicación, provendrán de los valores que tengan los parámetros de sus subtareas en ese momento, es decir, la información fluye de abajo a arriba en la jerarquía. Y, además, la procedencia del valor de un parámetro puede ser dependiente del proceso según el cuál una tarea se haya realizado, por lo cual la información referente a cómo se realiza el paso de parámetros hacia las tareas compuestas se especifica en las reglas de los modelos. Como ejemplo de paso de parámetros, supongamos la tarea compuesta de *rotar pieza* en una aplicación de diseño gráfico. Una posible manera de descomponer la ejecución de dicha tarea en subtareas sería realizar la tarea atómica *seleccionar pieza*, que tiene por parámetro la pieza seleccionada, y, después, realizar la tarea compuesta *rotar selección*. Esta descomposición se especifica mediante una regla que tiene por *tarea padre* la tarea *rotar pieza* y por *subtareas* a *seleccionar pieza* y *rotar selección*. Además, supongamos que la tarea *rotar selección* ya está definida, así como las reglas en las que dicha tarea es la tarea padre, y que posee un parámetro *grados* que indica el número de grados que se rota la pieza. Pues bien, de manera natural, parece que los parámetros que aportan información contextual a la tarea *rotar pieza* deberán ser el objeto a rotar y los grados de la

rotación, que son obtenidos directamente a partir de los contextos de sus *subtareas* según la regla mencionada. Así, la información que determina cómo estos parámetros fluyen hacia la tarea *rotar pieza* deberá indicar cómo localizar la información contextual de origen y qué funciones se deberán aplicar a los parámetros de *origen* para convertirlos en parámetros de *destino* (en este caso, la función identidad).

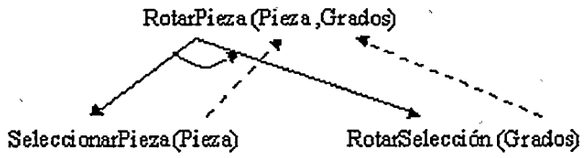

Figura 16: Descomposición jerárquica de un nivel de la tarea *RotarPieza*

Finalmente, en las reglas también se reflejan las posibles precondiciones que pueden afectar a su aplicabilidad. Por ejemplo, en el ejemplo anterior, para realizar la tarea *RotarPieza* podría especificarse como una precondición necesaria la existencia de alguna pieza en el marco de trabajo del usuario.

Destacar, asimismo, la relación existente entre las representaciones de los modelos de tareas de ATOMS y las *gramáticas de unificación* [Kay83, Shieber92] utilizadas en el contexto de tratamiento de lenguaje natural (NLP). En particular, los patrones de objetos cumplen un papel similar al de las *estructuras de rasgos* (*feature structures*) de las *gramáticas de unificación*, mientras que las reglas de los modelos de tareas y las de las *gramáticas de unificación* comparten características como la posibilidad de establecer restricciones entre la información contextual asociada a distintas tareas o a distintas *estructuras de rasgos*. Como posteriormente veremos, ATOMS también incorpora un intérprete de tareas que cumple una misión paralela a la de los procesadores de lenguaje en el campo de NLP.

3.1.3 Reutilización de Modelos Parciales

Los modelos de tareas basados en la arquitectura ATOM incorporan el concepto de herencia simple, de uso generalizado en todos los lenguajes orientados a objetos actuales. En particular, la herencia que se utiliza está basada en el paradigma prototipo-instancia, ya introducido en la Sección 2.3. Mediante la aplicación de este modelo, los objetos pueden ser vistos como *prototipos* para la creación de nuevas *instancias*.

La idea de incorporar *herencia* al proceso de modelización de tareas tiene por objetivo primario el facilitar la reutilización de componentes en los modelos. Así, el proceso de modelización de tareas se convierte en un proceso de refinamiento a partir de modelos parciales preexistentes. Este concepto de herencia vendrá plasmado en el lenguaje de programación que los sistemas basados en ATOM incorporarán para la especificación de los modelos. Si al hecho de incorporarse herencia al lenguaje de especificación de modelos se le añade un soporte en forma de herramienta interactiva, de modo que este concepto se utilice de una manera más transparente, lo que se está haciendo es fomentar la construcción y gestión de repositorios de modelos parciales, asistiendo así a las personas encargadas de modelizar las tareas de los usuarios en su cometido.

La herencia en la especificación de modelos en la arquitectura ATOM es aplicable tanto a las tareas como a las reglas de los modelos, de modo que el proceso de refinamiento de conceptos es aplicable al desarrollo de todo el modelo.

Para el primer caso, el de herencia para la especificación de una tarea, los diseñadores de modelos utilizarán como prototipo de partida el prototipo de tarea atómica, el prototipo de tarea compuesta, o cualquier otro prototipo definido previamente o que la implementación particular del sistema facilite. Una vez el diseñador haya seleccionado el prototipo de tarea que más le interese de acuerdo a la tarea que desee modelizar, facilitará a la nueva tarea un nombre, una descripción de su semántica y, en el caso de las tareas atómicas, dependiendo de en qué grado el prototipo seleccionado para su nueva tarea se ajuste a sus necesidades, deberá modificar su patrón de interacción o las expresiones que gobiernan la obtención de los valores de los parámetros de la tarea. Por ejemplo, supongamos la existencia de un prototipo de tarea *Tarea Botón* asociado a la interacción de pulsación de un botón –y que es *parametrizable* en cuanto al texto del botón-, y que se desea modelizar la pulsación de un botón con el texto *Ok* que se encuentra dentro de una caja de diálogo de título *Rellenar parámetros*. Pues bien, en este caso, además de instanciar a partir del prototipo *Tarea Botón* aportando como parámetro la etiqueta del botón, habrá que especializar la tarea definida para que haga referencia sólo a aquellos botones con etiqueta *Ok* que se encuentren dentro de una cuadro de diálogo con el título citado. En este caso, la especialización implicará una modificación del patrón de interacción de la tarea, pero en otros podrá afectar, por ejemplo, a la manera en que la tarea recoge los valores de sus parámetros. La manera en que estas modificaciones se realizarán dependerá de las soluciones que aporte la implementación de ATOM que se utilice. En particular, describiremos en la Sección 3.2 las soluciones que aporta ATOMS.

Al igual que sucede con el caso de las tareas, el concepto de herencia también es aplicable a la especificación de las reglas, con el objeto de facilitar la reutilización de modelos parciales. El uso de herencia en la especificación de reglas nos permite partir de una cierta relación entre algunas tareas para, a continuación, relajar alguna de la condiciones de dicha relación haciéndola más rígida o, simplemente, cambiarla, dependiendo del contexto en el que se quiera utilizar. La ventaja de este procedimiento frente a la redefinición completa de la regla es que el resto de condiciones de la regla permanecerá inalterado, de modo que no se tiene que especificar la regla completamente, sino sólo lo que haya cambiado de ella. Para ilustrar ésto, veamos la Figura 17. En la parte izquierda de dicha figura observamos la representación de tres reglas, cada una de las cuales difiere de las otras dos sólo en cambios relacionados tanto con la tarea padre como con la segunda subtarea de la regla. Como vemos, estas tres reglas han de especificarse íntegramente y de manera independiente las unas de las otras. Sin embargo, es obvia la existencia de un patrón de similitud entre las tres reglas, ya que todas ellas comparten la primera y tercera subtarea, además de una estructura común. Por ello, teniendo por objetivo el simplificar al máximo la especificación de las reglas, la estructura de los modelos de ATOM permite la posibilidad de que este escenario pueda ser modelado siguiendo la estructura que se muestra en la parte derecha de dicha figura. Como se ve en esta parte derecha, ATOM propone la especificación de una *regla prototipo* a partir de la cual se pueden instanciar otras reglas, de manera que sólo se indiquen las componentes que difieren de las de la *regla prototipo*, y el resto se mantienen intactas.

Casos similares al caso abstracto representado en esta figura podemos encontrarlos con relativa frecuencia. Pongamos, por ejemplo, un editor gráfico que permite efectuar un cierto grupo de tareas (T', T'', T''',...) sobre una figura que se seleccione (rotar la pieza un determinado número

de grados, modificar las propiedades del tipo de línea de su contorno, modificar las componentes RGB de su color de fondo,...). Para realizar estas acciones, normalmente se selecciona el objeto –u objetos- que será el objetivo de la acción (T1), a continuación se indicará cuál es la acción a realizar y se proporcionarán los parámetros de dicha acción (T2', T2'', T2''',...) y, finalmente, se confirmarán los valores proporcionados para la acción (T3).

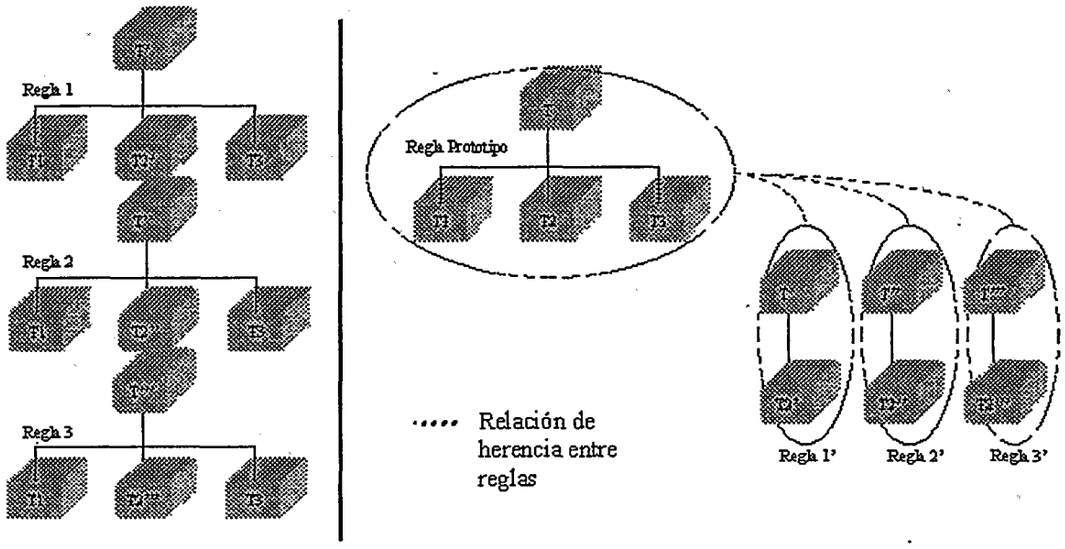

Figura 17: Aplicación de la herencia de reglas.

Según lo anterior, la arquitectura ATOM define la posibilidad de que se establezcan relaciones prototipo-instancia entre reglas. Ahora bien, ¿qué sucedería si en el modelo de tareas de una aplicación se introdujeran tanto un determinado prototipo de regla como instancia de la misma? En el ejemplo abstracto de la figura, si el usuario completara T2'', éso significaría que el usuario podría haber realizado tanto T como T'', pues tanto T2 como T2'' podrían hacer *matching* con la acción del usuario, con lo cual nos enfrentamos ante una problemática de ambigüedades, ¿cómo debería comportarse el sistema? Una posible solución es no permitir que un modelo incorpore simultáneamente una regla prototipo y una instancia suya, lo cual recortaría innecesariamente la potencia de la herencia entre reglas, ya que no permitiría especializar un prototipo para algún caso específico dejando el prototipo activo para el resto de los casos. Por lo tanto, la solución que en ATOM se establece es que puedan incluirse simultáneamente tanto reglas prototipos como reglas que sean instancias de ellas, dejándose a la fase de análisis de interacciones (ver Sección 3.2) la responsabilidad de manejar las situaciones potencialmente ambiguas que surjan. De este modo, las reglas que se encuentren más cerca de la raíz en relación jerárquica prototipo-instancia de las reglas sólo serán aplicables cuando las reglas situadas por debajo en la jerarquía no hayan podido ser aplicadas. Así, las reglas tendrán mayor prioridad cuanto más abajo estén situadas en la jerarquía. Con ello, se consigue la aplicación de reglas genéricas cuando reglas más específicas no han sido aplicables. De este modo, los diseñadores pueden crear paquetes de reglas genéricas para su uso, a partir de los cuales podrán formar sus jerarquías de reglas para adecuarlas a contextos de uso concretos. La manera concreta en que la fase de análisis de interacciones maneja

la coexistencia de prototipos de reglas e instancias suyas se describirá cuando se hable de *heurísticas*, en la Sección 3.3.1.

3.2 Especificación de Tareas en ATOMS

Una de las mayores críticas que los entornos basados en la tecnología de la modelización de aplicaciones interactivas han recibido es la de que las ventajas que mediante esta tecnología pueden obtenerse se echan a perder debido a la dificultad de uso de estos sistemas. Esta consideración se basa en la afirmación de que, en el mundo real, el esfuerzo extra que los diseñadores han de realizar para dominar el lenguaje con el cual se expresan los modelos de sus aplicaciones no compensa las ventajas que se obtienen gracias a estos sistemas [Sukaviriya94]. Por lo tanto, si se quiere conseguir que esta tecnología llegue finalmente a tener una utilización extendida, la vía es facilitar al máximo la especificación de los modelos de las aplicaciones mediante mecanismos como, por ejemplo, herramientas interactivas.

En esta dirección, esta sección presenta las ideas que dan soporte a TMT, la herramienta interactiva que tiene por objetivo facilitar la construcción de los modelos de las aplicaciones basadas en ATOMS. Como se afirmó en la Sección 2.2, los entornos de modelización de aplicaciones interactivas permiten representar los modelos mediante lenguajes de especificación, a menudo declarativos. Como paso previo a la descripción de TMT, mostraremos brevemente el lenguaje de especificación utilizado en ATOMS.

3.2.1 Lenguaje de Especificación de Modelos de ATOMS

En esta sección se presentan algunos ejemplos de tareas y reglas representadas mediante el lenguaje declarativo de especificación de modelos de ATOMS. No se pretende dar ejemplos detallados, sino tan sólo mostrar brevemente la filosofía del lenguaje. Para el lector interesado en profundizar en este lenguaje, su gramática completa puede encontrarse en el Apéndice I, mientras que en el Apéndice II puede encontrarse un ejemplo completo del uso de dicho lenguaje para la modelización de tareas.

Los ejemplos proporcionados se corresponden a la especificación del modelo de tareas de la herramienta *Schoodule*, desarrollada para la validación de los prototipos basados en las ideas de esta tesis. Este sistema utiliza una base de datos y un resolvedor de restricciones para planificar horarios de colegios. La aplicación accede a una base de datos, que refleja esencialmente cuatro entidades y una relación. La primera entidad representa a los profesores de la escuela para la que se desean planificar los horarios. La segunda de las entidades se corresponde con las diferentes asignaturas impartidas en la escuela. La información relacionada con los grupos de alumnos está reflejada en otra entidad, y la última describe las diferentes aulas disponibles en el centro escolar. Finalmente, la *relación de asignación* refleja qué profesor imparte cada asignatura para cada grupo de alumnos. A partir de esta información, *Schoodule* planifica no sólo los horarios de los distintos grupos y profesores, sino las aulas en las que cada sesión lectiva debe llevarse a cabo. Como se aprecia en la Figura 18, los usuarios interaccionan con la base de datos interna de *Schoodule* mediante cuadros de diálogo que contienen listas en las que se muestran los distintos valores que tienen las filas para una determinada columna de una tabla de dicha base de datos. Una vez seleccionados los valores deseados, confirman las selecciones con los botones, insertándose así nuevas filas y relaciones dentro de la base de datos.

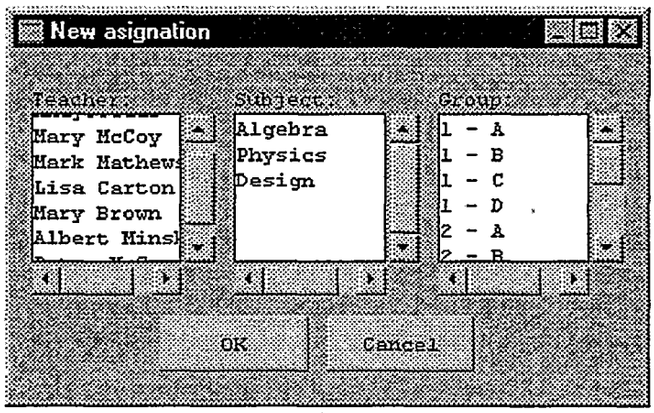

Figura 18: Adición de una nueva asignación lectiva en *Schoodule.*

A continuación se muestra la especificación asociada a la tarea atómica correspondiente a la selección de un nombre de profesor de una lista de nombres de profesores bajo el subtítulo de "*Selecciona profesor:*":

TASK *"Seleccionar el profesor de la lista"* **ISA LIST** "Profesor:" "nombre" **ALIAS** *SeleccionarProfesorDeLista*

DESCRIPTION *Selecciona el nombre del profesor de la lista 'Profesor:'*

END

Como se observa, la tarea se hace descendiente del prototipo de tarea atómica de ATOMS que modeliza las acciones sobre listas. Esta relación de herencia se establece mediante la etiqueta ISA. Los parámetros que recibe el prototipo LIST para la instanciación son *"Profesor:"* y *"nombre"*, que juegan los papeles de texto que encabeza la lista (ver Figura 18) y nombre del parámetro, dentro de la tarea *SeleccionarProfesorDeLista*, que recibirá el elemento que el usuario seleccione dentro de la lista. Los parámetros que un determinado prototipo de tarea deberá recibir durante su instanciación dependerán de cómo se haya definido dicho prototipo, de manera equivalente a los parámetros que deberá recibir una subrutina de un API tradicional. La manera mediante la cual una tarea prototipo hace uso de los valores pasados como parámetros es mediante un símbolo especial, '%', como se explicará más adelante, en la Sección 4.2.4. A continuación mostramos cómo está definida, como parte de los prototipos de tareas que ATOMS proporciona por defecto, la tarea LIST:

TASK "Seleccionar elemento de la lista %1" **ISA SELECTION ALIAS** *LIST*

DESCRIPTION Selecciona elemento de la lista de cabecera %1

// Paso de parámetros

PARAMETER %2 = *selectedItem.TEXT*

// especialización del patrón de interacción de la tarea SELECTION

PATTERN *LABEL == %1*

PATTERN *ISA == List*

END

Similarmente a como se ha definido la tarea SeleccionarProfesorDeLista, podrían definirse las tareas Seleccionar la asignatura de la lista (abreviadamente, SeleccionarAsignaturaDeLista) y Seleccionar el grupo de la lista (SeleccionarGrupoDeLista). Las tareas Confirmar y Cancelar, asociadas a los botones de etiquetas Ok y Cancel se realizan a partir del prototipo de tarea atómica que contiene el patrón de interacción que representa la pulsación de un botón. La tarea asociada al botón "Asignación", encargada de abrir el cuadro de diálogo de la Figura 18, se especifica igualmente a partir del prototipo de tarea atómica que representa la pulsación de un botón, mientras que, por su parte, la tarea asociada a la posibilidad de abrir el cuadro de diálogo mediante la selección de una opción de menú se realiza partiendo del prototipo de tarea atómica proporcionado por ATOMS para la modelización de acciones de menús (ver Apéndice II).

Por su parte, hay cuatro tareas compuestas para la modelización de la tarea *Añadir una nueva asignación de docencia* (*CrearNuevaAsignación)*. Por un lado, tenemos la tarea raíz, que modeliza el proceso global, que incluye la apertura del cuadro de diálogo, la selección de los valores deseados y la confirmación de los valores elegidos. Por otro lado, está la tarea *Abrir el cuadro de diálogo de Nueva Asignación (AbrirNuevaAsignación*), que modeliza el proceso de abrir el cuadro de diálogo mediante el cual establecer los valores de la nueva relación, y que engloba las dos posibles vías para llevar a cabo esto, es decir, mediante el menú y mediante el botón equivalente. La tercera tarea compuesta involucrada es *Establecer las componentes de la relación de docencia (SeleccionarDatos)*, que modeliza el proceso de establecimiento de valores para la nueva relación. Finalmente, la tarea *Proporcionar los valores que tendrá la relación* (*DarValores*) se ocupa de asociar la tarea *Confirmar los valores facilitados* (*Confirmar*) como condición de finalización de la tarea *SeleccionarDatos*. La jerarquía establecida por esta descomposición puede observarse representadas gráficamente en la Figura 19.

La especificación de la tarea compuesta de proporcionar valores para la nueva asignación que se desea crear puede hacerse de la siguiente manera:

TASK *"Establecer las componentes de la relación de docencia."* **ISA COMPOSED ALIAS** *SeleccionarDatos*

DESCRIPTION *Establece qué profesor impartirá una determinada asignatura a un determinado grupo*

END

Ahora, por ejemplo, la regla que permitiría relacionar las tareas SeleccionarDatos con SeleccionarProfesorDeLista, SeleccionarAsignaturaDeLista y SeleccionarGrupoDeLista es la siguiente:

RULE *ReglaSeleccionarDatos*

TASK *SeleccionarDatos*

SUBTASKS *SeleccionarProfesorDeLista* **AS** *Profesor*

SeleccionarAsignaturaDeLista **AS** *Asignatura*

SeleccionarGrupoDeLista **AS** *Grupo*

MULTIPLE *Profesor Asignatura Grupo*

OPTIONAL *Profesor Asignatura Grupo*

// La ejecución de esta tarea se puede cancelar

CANCELLING *Cancelar*

PARAMETER *nombre = SeleccionarProfesorDeLista.Params.nombre*

PARAMETER *asignatura = SeleccionarAsignaturaDeLista.Params.asignatura*

PARAMETER *grupo = SeleccionarGrupoDeLista.Params.grupo*

END

La representación esquemática completa de la tarea *CrearNuevaAsignación* puede contemplarse en la Figura 19.

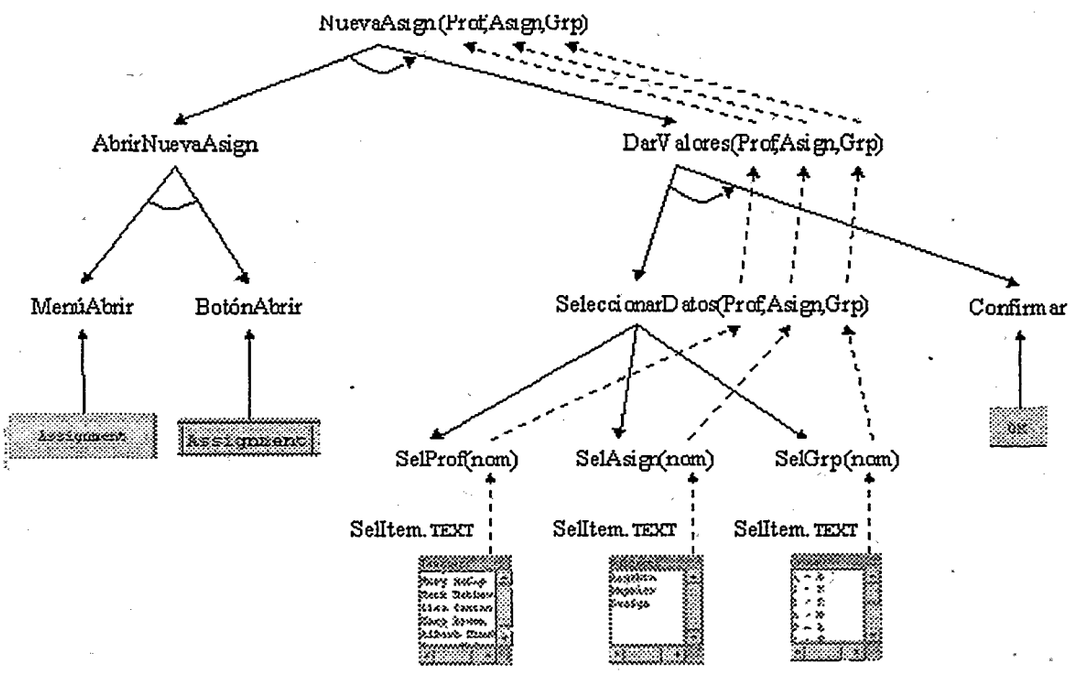

Figura 19: Descomposición jerárquica de la tarea *CrearNuevaAsignación*.

Como ejemplo de la utilidad de la especificación de modelos mediante el refinamiento de otros modelos existentes, es decir, aplicando el concepto de herencia de tareas, supongamos, en primer lugar, que tenemos modelada la tarea asociada a la pulsación de un botón con etiqueta *Ok*. La modelización de esta tarea podría ser la siguiente:

TASK *"Confirmar los valores facilitados"* **ISA BUTTON** "Ok" **ALIAS** *Confirmar*

DESCRIPTION *Confirma los valores seleccionados*

END

En el lenguaje de modelización de aplicaciones de ATOMS, la herencia de reglas se especifica de una manera natural mediante el uso de la etiqueta ISA. Sin embargo, la definición anterior puede dar lugar a incorrecciones en el caso de que se utilizara dicha tarea en reglas que fueran aplicables al pulsar botones *Ok* situados en cuadros de diálogo distintos. Por ello, se permite modificar las partes deseadas de una tarea existente para hacer que, en nuestro ejemplo, la nueva tarea represente exactamente la interacción de pulsar un botón de etiqueta *Ok* localizado dentro

modificar las partes deseadas de una tarea existente para hacer que, en nuestro ejemplo, la nueva tarea represente exactamente la interacción de pulsar un botón de etiqueta *Ok* localizado dentro de un cuadro de diálogo con etiqueta *'Nueva asignación'*. A continuación se observa cómo se puede proceder para obtener el efecto deseado, donde la tarea con nombre *ConfirmarSelecciónProfesor* es modelada de manera que, además de contener una descripción textual de la tarea distinta de la original, ha de llevarse a cabo con un botón de etiqueta *Ok* que se encuentre en un cuadro de diálogo con el título indicado, en contraposición con la tarea *Confirmar*, que es independiente del cuadro del título del cuadro de diálogo en el que dicho botón se encuentre:

TASK ISA *Confirmar* **ALIAS** *ConfirmarSelecciónProfesor*

PATTERN *OWNER.TITLE* == *"Nueva asignación"*

DESCRIPTION *Confirma los valores seleccionados para la asignación*

END

Cuando se crea una instancia de tarea a partir de un prototipo de tarea, aquellas propiedades a las que se les dé un valor sobreescribirán los valores provenientes del correspondiente prototipo. Así, en el ejemplo anterior, la descripción de la tarea *ConfirmarSelecciónProfesor* será *"Confirma los valores seleccionados para la asignación"*, y no *" Confirma los valores seleccionados"*, lo cual es muy natural, ya que la nueva tarea no tiene por qué compartir la descripción de su tarea prototipo, aunque podría darse el caso contrario. Asímismo, el lenguaje de modelización de ATOMS también permite eliminar propiedades de la modelización de la tarea que estamos especializando y, como es habitual, añadir propiedades nuevas no existentes en el prototipo.

3.2.2 Especificación Interactiva de Modelos en ATOMS: TMT

Las herramientas de modelización asisten a los diseñadores en la construcción de los modelos. Por tanto, el objetivo prioritario de estas herramientas es ocultar a los diseñadores la sintaxis de los lenguajes de modelización, proporcionándoles una interfaz conveniente para especificar las, a menudo, grandes cantidades de información que han de almacenar los modelos.

Una gran cantidad de herramientas de modelización se han construido para los diferentes entornos basados en la tecnología de modelización de interfaces. A menudo, estas herramientas están orientadas a la naturaleza específica de cada tipo de modelo particular. Estas herramientas abarcan un amplio espectro de técnicas. En el caso más simple nos encontramos desde editores para la especificación textual de modelos, como sucede con los entornos MASTERMIND e ITS. Sin duda más intuitivas resultan las herramientas interactivas proporcionadas por entornos como MECANO y MOBI-D [Eisenstein2000] para la creación y modificación de los elementos de los modelos, basadas en formularios. Otros entornos facilitan editores gráficos especiales que utilizan técnicas de programación visual, como sucede en HUMANOID, FUSE, ADEPT, TADEUS [Elwert95, Schlungbaum96], MOBI-D [Tam98] y otros entornos. Incluso, recientemente se han intentado aplicar técnicas de inducción de gramáticas jerárquicas [Nevill-Manning94] para la modelización de modelos jerárquicos de tareas de usuario, que intentan inducir estos modelos a partir de la observación de la actividad de los usuarios [Maulsby97].

Sin embargo, creemos que la aproximación más interesante para la modelización de aplicaciones se basa en el concepto de *Interfaces Demostracionales* (*Demonstrational Interfaces* [Myers93]). Bajo este concepto se engloban todos aquellos sistemas que utilizan *ejemplos* con el fin de

Estas técnicas demostracionales ya han sido utilizadas en herramientas de modelización con anterioridad. Una interesante herramienta es Grizzly Bear [Frank95]. Esta herramienta trata de ocultar a los diseñadores los entresijos de los modelos proporcionándoles una interfaz similar a la de los constructores de interfaces tradicionales o a la de los programas de dibujo: una paleta con los diferentes bloques disponibles y un área de edición donde los diseñadores pueden dibujar el aspecto de sus interfaces. Grizzly Bear extrae las entidades de los modelos a partir de los ejemplos de las interfaces que los diseñadores dibujan. Esta herramienta puede generalizar a partir de varios dibujos e inferir fragmentos del diálogo de la interfaz de usuario a partir de representaciones de los estados de la interfaz antes y después de las acciones de los usuarios.

Sin embargo, lo que en esta tesis se persigue es la generación de modelos con los que se puedan generar cursos tutores. Por tanto, se puede partir de la base de que las aplicaciones interactivas ya existen. Bajo este punto de vista, la filosofía de las interfaces demostracionales encaja perfectamente con lo que se busca, ya que los modelos pueden ser especificados mediante el uso de la propia aplicación. Esta especificación es realizada *a posteriori* respecto del diseño de la aplicación. Debido a esta independencia, se obtiene la ventaja de que esta especificación puede ser realizada por un diseñador distinto del que diseñó la aplicación.

En este sentido, un trabajo interesante y que utiliza programación mediante ejemplos para la modelización a posteriori de la aplicación es TME [Contreras98]. Este sistema permite la especificación de modelos de tareas a partir de ejemplos de interacciones realizados sobre la propia aplicación. Sin embargo, este sistema no realiza inferencia sobre distintas interacciones, de modo que no es capaz de generalizar a partir de varios ejemplos.

En una línea similar a la de TME, ATOMS proporciona la herramienta TMT (*Herramienta de Modelización de Tareas*, del inglés *Task Modeling Tool*), mostrada en la Figura 20. Esta herramienta se basa en una técnica que permite la generación de *patrones de interacción* utilizando FIND-S, un algoritmo que genera expresiones que representen a todo el conjunto de ejemplos proporcionados y que sean lo más específicas posibles para cada atributo [Mitchell82]. Esta técnica es, por tanto, conservativa, y generaliza un patrón de interacción sólo cuando existe alguna evidencia que apoye dicha generalización. Este algoritmo también ha sido utilizado para la especificación de entornos virtuales de enseñanza [Wang95, Johnson98].

Por un lado, TMT aporta el soporte necesario para capturar interacciones y generalizarlas interactivamente. Así, la herramienta permite la definición, mediante ejemplos, de las tareas atómicas. Para definir una nueva tarea atómica, el diseñador informa a la herramienta de que va a llevar a cabo una tarea atómica y la realiza. Tras este primer ejemplo, TMT es capaz de identificar el prototipo de tarea más específico que se corresponde con la interacción realizada (ver Figura 13), aunque el diseñador podrá revisar y modificar esta conclusión del sistema.

Tras este primer paso, el diseñador deberá llevar a cabo un proceso de *especialización* y otro de *generalización* de la interacción que acaba de realizar, con el fin de poder especificar con total precisión qué interacciones van asociadas a la tareas que se está definiendo. Si el diseñador no realiza ninguno de estos pasos, ATOMS utilizará como patrón de interacción de la tarea que se está especificando el patrón de interacción asociado al prototipo de tarea identificado por TMT, y como valores asociados a los atributos de dicho patrón los presentes en la interacción del diseñador. Por ejemplo, si el diseñador pulsa en un botón de etiqueta *'Cancelar'*, la tarea ya estará definida correctamente y será identificada posteriormente por el sistema sin que se haya de hacer nada más. Pero en muchas ocasiones es necesario que se especialicen más las interacciones.

estará definida correctamente y será identificada posteriormente por el sistema sin que se haya de hacer nada más. Pero en muchas ocasiones es necesario que se especialicen más las interacciones. Así por ejemplo, en el caso de que se mueva un objeto gráfico, la interacción será descrita como *'mover cualquier objeto'*, lo cual puede resultar demasiado genérico. La modificación de las interacciones incluye dos pasos: la selección interactiva del conjunto de atributos relevantes de una interacción y la generalización de los valores de dichos atributos relevantes.

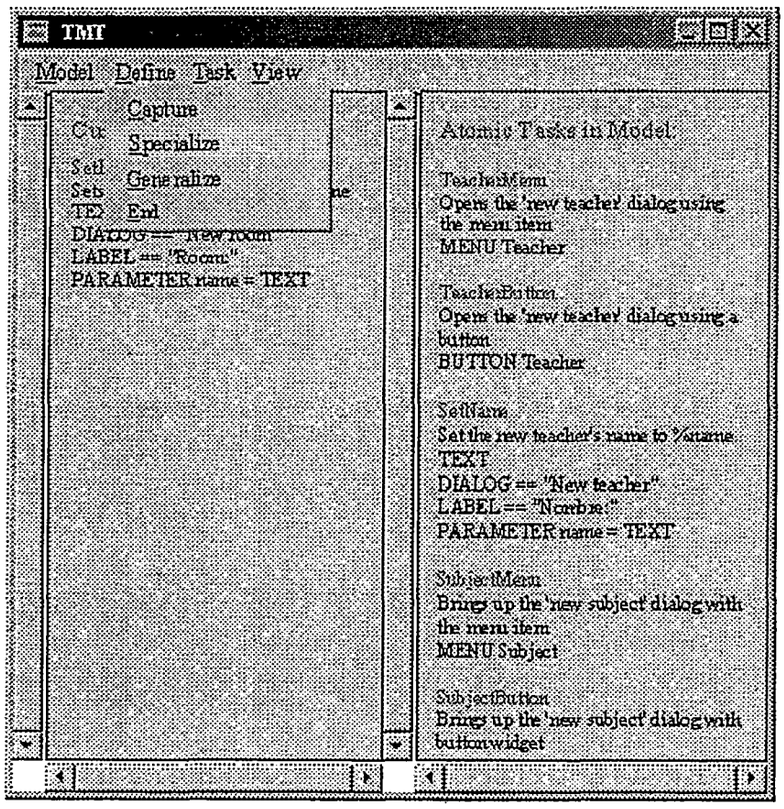

Figura 20: Interfaz de la herramienta TMT de definición interactiva de modelos.

El primer paso, *especialización*, identifica qué atributos debe tener en cuenta ATOMS, además de los correspondientes al patrón de interacción del prototipo de la tarea, para hacer el *matching* entre la tarea atómica que se está especificando y las interacciones que realiza el usuario. Por ejemplo, el diseñador puede indicar que el color del objeto que se está moviendo sí es importante y que, por contra, el tamaño del objeto no lo es. El diseñador del modelo indica ésto al sistema seleccionando interactivamente los atributos relevantes a partir de una lista que TMT le proporciona con todos los atributos asociados a la interacción que ha realizado. Esta selección implica una especialización de la interacción, ya que TMT fuerza a los atributos a adoptar los mismos valores que han tenido en la interacción que el diseñador ha realizado anteriormente. Así, siguiendo con el ejemplo anterior, si el diseñador ha proporcionado como primer ejemplo de interacción el movimiento de un objeto gráfico de color rojo e indica a TMT que el color es importante, la herramienta describirá la interacción como *'mover cualquier objeto de color rojo'*, restringiendo así las acciones asociadas a este patrón de interacción.

El segundo procedimiento *generaliza* los valores de los atributos relevantes seleccionados en el paso anterior y del patrón de interacción heredado del prototipo de tarea. Esta generalización se lleva por medio de la repetición por parte del usuario de nuevos ejemplos de la tarea atómica que está especificando. Tras cada ejemplo que se le proporcione al sistema, el sistema genera una expresión para cada atributo relevante que engloba todos los valores obtenidos para dicho atributo a lo largo de todos los ejemplos proporcionados. El conjunto de todas las expresiones así generadas formará el patrón de interacción de la tarea resultante. Por ejemplo, si en un segundo ejemplo proporcionado al sistema el usuario mueve un objeto verde, entonces el sistema generalizará la expresión resultante para el atributo *color*, de manera que la interacción pueda ser descrita en términos abstractos como *'mover cualquier objeto rojo o verde'*.

En la actualidad, TMT solo genera expresiones que incluyan la función de igualdad, ligados por el operador lógico entre expresiones || (*O*-lógico), para cada atributo del *patrón de interacción*. Por tanto, el patrón resultante es el *Y*-lógico de las expresiones resultantes para cada atributo que, a su vez, son el *O*-lógico de los valores que dichos atributos han adoptado a lo largo de los ejemplos proporcionados por el diseñador del modelo a TMT. Sin embargo, ya que ATOMS es capaz de gestionar predicados más generales, que involucren operadores aritméticos y lógicos, expresiones más complicadas pueden ser especificadas manualmente por diseñadores avanzados para proporcionar patrones abstractos de interacción más complejos.

Por otra parte, TMT también proporciona el soporte para la definición interactiva de tareas compuestas y reglas que permitan ligar entre sí tanto tareas atómicas como tareas compuestas. Como en el caso comentado para los predicados de los patrones de interacción, TMT no permite generar interactivamente todas las posibles reglas. Así, la única expresiones que puede ser especificada de manera automática para las funciones de transferencia de parámetros es la función identidad. Pese a que dicha clase de funciones es de uso muy habitual, los usuarios avanzados pueden especificar manualmente muchos otros tipos de funciones, ya que ATOMS será capaz de gestionarlas durante el análisis de la actividad de los usuarios que describiremos a continuación.

3.3 Análisis de la Actividad de los Usuarios

Los modelos de tareas, tal y como hasta ahora hemos visto, sólo contienen información estática de qué tareas puede realizar un usuario con una determinada aplicación interactiva. Sin embargo, esta información, sin acompañarse de una componente que permita contrastarla con las acciones que los usuarios están realizando en un momento dado, no permite ofrecer ningún tipo de servicio que dependa del contexto actual del trabajo de los usuarios con la aplicación. Esta componente encargada de relacionar la información contenida en los modelos de tareas de usuario con la actividad de los usuarios se denomina, dentro de la arquitectura ATOM mostrada en la Figura 8, *Motor de Análisis*. Este subsistema se encarga de analizar la actividad de los usuarios, es decir, de interpretar qué tareas está realizando el usuario con la aplicación.

Este análisis también recibe el nombre de análisis léxico-sintáctico (en inglés, *parsing*) de eventos, análisis de interacciones o análisis de tareas, por analogía con el papel que cumple la componente léxico-sintáctica (o *parser*) de los analizadores que se utilizan en el procesamiento de lenguajes [Aho86]. Estos procesadores sintácticos de lenguajes parten de la información léxica que les facilitan los analizadores morfológicos en los que se basan, para comprobar si determinadas secuencias de palabras pertenecientes a un vocabulario cumplen cierto conjunto de reglas sintácticas. Pues bien, en el caso del análisis que nos ocupa, la componente léxica viene dada por la representación de las interacciones de los usuarios, mientras que el papel de las reglas sintácticas es realizado por la información presente en los modelos de tareas de usuario. Una diferencia notable entre los analizadores léxico-sintácticos y los analizadores de tareas reside en que éstos últimos tienen que gestionar la posibilidad de que los usuarios puedan estar realizando varias tareas a la vez, de modo que intercalen acciones referentes a varias tareas distintas. Por hacer una analogía, es como si un analizador léxico-sintáctico tuviera que analizar las frases que varios emisores realizaran simultáneamente sin respetar el turno de la otra persona, de forma que sus oraciones se entrecruzaran.

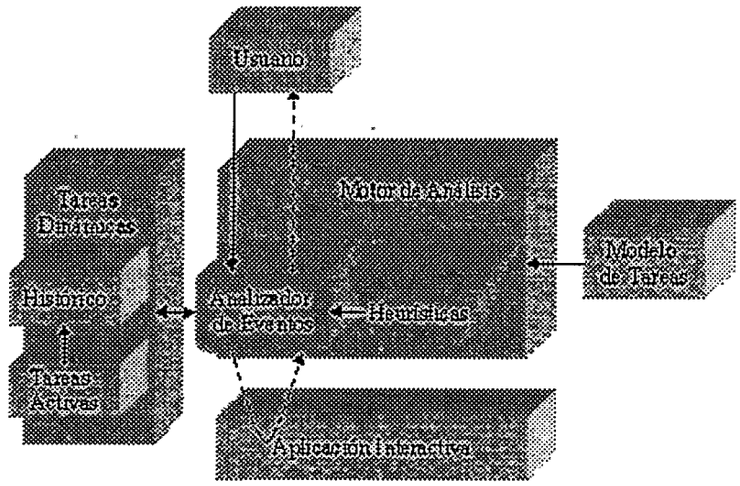

Figura 21: Flujo de información que afecta al Motor de Análisis.

Según el modelo ATOM, el corazón funcional del *Motor de Análisis*, denominado *Analizador de Eventos*, se encarga de realizar el análisis de tareas, ofreciendo así el soporte básico para que herramientas externas de valor añadido basadas en ATOM puedan razonar acerca de la aplicación y el estado en que en cada momento se encuentre. Para esta labor, el *Analizador de Eventos* tiene en cuenta tanto el modelo de tareas de la aplicación, como el estado actual de las tareas que están realizando los usuarios, representado por las *Tareas Dinámicas de la Aplicación*, y las acciones que realizan los usuarios. Además, y como veremos, en ocasiones las acciones de los usuarios resultan potencialmente ambiguas para el sistema, es decir, el *Analizador de Eventos* no es capaz de reconocer qué tareas está realizando el usuario a partir únicamente de la información anterior. En estos casos el *Analizador de Eventos* se debe encargar de la aplicación de *heurísticas* para la resolución estas ambigüedades.

Finalmente, el *Analizador de Eventos* se encargará de instanciar tareas dinámicamente conforme el usuario las realice, con el fin de mantener una representación interna, llamada *Tareas Dinámicas*, de la actividad de los usuarios. Las *Tareas Dinámicas* están organizadas de manera jerárquica, a modo de árbol. Cada uno de los árboles que representan las *Tareas Dinámicas* hace referencia a qué partes de una tarea dada han sido realizadas o se están realizando. Esta información debe ser transparente para los usuarios de la aplicación, de manera que éstos no tienen por qué tener conocimiento de su existencia ni de su evolución.

Las *Tareas Dinámicas* tienen dos partes bien diferenciadas. Por un lado se tienen las *Tareas Activas*, que se corresponden con aquellas tareas de alto nivel que han sido comenzadas pero todavía no se han terminado y, por otro lado, se encuentra el *Histórico*, representación de todas las tareas de alto nivel que se han completado durante la ejecución de la aplicación. Las representaciones de las tareas que realizan los usuarios irán pasando del área de *Tareas Activas* al *Histórico* según son completadas.

La representación esquemática detallada del procesamiento que lleva a cabo el *Motor de Análisis* de ATOM puede verse en la Figura 21.

3.3.1 El Motor de Análisis en ATOMS

En esta sección se describen detalles internos del *Motor de Análisis* de ATOMS, la implementación de la arquitectura ATOM presentada en esta tesis.

El Analizador de Eventos. Puede decirse que este módulo intenta conocer qué está haciendo el usuario, con respecto a las tareas definidas en el modelo de tareas de la aplicación, considerando el contexto facilitado por sus pasos previos. El *Analizador de Eventos* recibe y procesa la información que representa las interacciones que los usuarios realizan con las aplicaciones antes que la propia aplicación. Este análisis de las interacciones de los usuarios se realiza de manera completamente transparente desde el punto de vista de los usuarios. Una vez el analizador de eventos procesa dicha información, pueden suceder dos cosas:

- Habitualmente, a menos que alguna *Herramienta de Valor Añadido* considere oportuno lo contrario, se pasa la información relativa a las interacciones realizadas a la aplicación para que ésta la procese. En este caso, una vez la aplicación procese la interacción, la información pasará de nuevo al *Analizador de Eventos* para que, si fuera necesario, la procese de nuevo.
- En el caso de que alguna *Herramienta de Valor Añadido* lo indique, la información relativa a la interacción no será pasada a la aplicación, y la interacción será cancelada. Esto permite el filtrado de las acciones de los usuarios, dependiendo del contexto de sus interacciones.

La razón por la que el *Analizador de Eventos* procesa las interacciones antes que la aplicación es consecuencia de la segunda de las opciones citadas: para cancelar una interacción, hay que hacerlo antes de que la aplicación procese la misma. Por su parte, el que el *Analizador de Eventos* vuelva a recibir la información de la interacción, después de ser procesada por la aplicación, se debe a que, en el caso de que la interacción no sea cancelada, las tareas pueden llevar asociada información contextual que no se conozca hasta que la aplicación haya procesado la interacción. Así, por ejemplo, si una interacción permite definir los límites y el tamaño de un objeto gráfico que se desea crear, tanto la posición como el tamaño del objeto podrán saberse antes de que la aplicación procese la interacción, ya que es información relativa a la interacción, pero al objeto creado no se tendrá acceso hasta que la aplicación cree el nuevo objeto como respuesta a la

interacción del usuario. Por tanto, la manera de que se pueda asociar como información contextual de una tarea el nuevo objeto es haciendo que la información referente a la interacción se reciba, también, después de ser procesada por la aplicación.

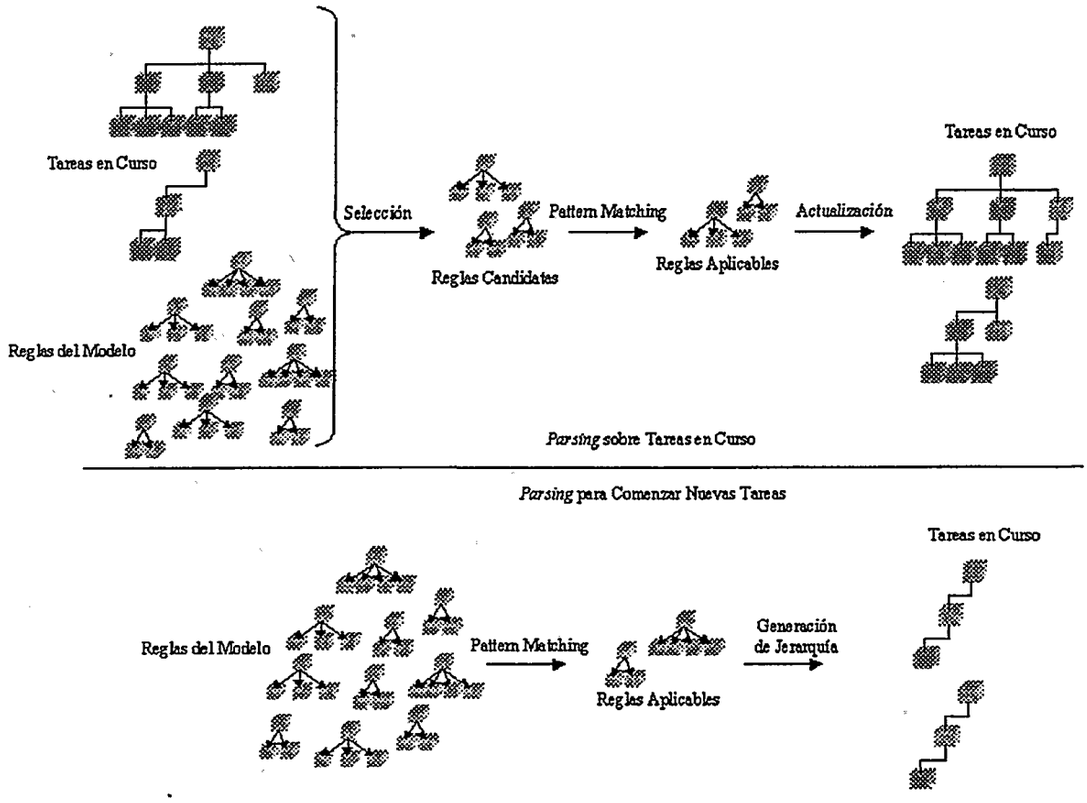

Figura 22: Proceso de análisis de interacciones del usuario.

Conocida la información acerca de la interacción del usuario, el *Analizador de Eventos* debe verificar con qué tareas atómicas concuerda (*matches*) dicha interacción. Esto es, deberá comprobar la compatibilidad entre la interacción que llega y los patrones de interacción de las tareas atómicas definidas en el modelo de tareas de la aplicación. Para ello, el *Analizador de Eventos* lleva a cabo dos procesos.

- En primer lugar, como se representa en la parte superior de la Figura 22, el *Analizador de Eventos* selecciona del modelo de tareas aquellas reglas que sean candidatas a aplicarse según el contexto actual. Esto lo realiza analizando qué tareas están actualmente en fase de ejecución, y mirando qué reglas, por tanto, están siendo aplicadas y cumplen las precondiciones que establecen. Como veremos, en un momento dado este conjunto podrá contener varias reglas. Una vez se conocen qué reglas son candidatas, se verificará si la interacción producida concuerda con el patrón de interacción asociado a cada una de las posibles tareas atómicas que puedan ejecutarse de cada regla. Con esto, se podrá determinar si la interacción actual forma parte de alguna de las tareas que están en curso en el momento de producirse. En caso afirmativo, se instancian las tareas realizadas y se interpretan las expresiones de paso de parámetros para actualizar el estado de las tareas en curso y obtener la información contextual asociada a las tareas. Como se puede ver, en este caso el *Analizador*

de Eventos conoce unas tareas objetivo y realiza un análisis de arriba abajo (*top-down*), tratando de encontrar qué tareas de niveles inferiores conducen a dichos objetivos. La determinación de estos objetivos se lleva a cabo mediante el segundo de los procesos, que se explica a continuación.

- En segundo lugar, como se ve en la parte inferior de la Figura 22, el Analizador de Eventos verifica si la interacción producida es el primer paso de alguna nueva tarea. Esto permite determinar los objetivos potenciales del usuario. Para esto, tendrá que realizar el proceso de determinación de concordancias (*pattern matching*) entre la interacción entrante y el patrón de interacción de cada una de las tareas atómicas que puedan ser el primer paso a realizar para la aplicación de una regla. Una vez determinadas qué reglas son aplicables, considerando la interacción realizada como el comienzo de una tarea, lo que se hace es aplicar sucesivamente las reglas del modelo de forma *impaciente* (*eager*), es decir, se comienzan a aplicar todas las reglas posibles para conocer qué posibles tareas de más alto nivel puede estar realizando el usuario. Después, es posible que algunas de estas posibilidades sean eliminadas mediante la aplicación de *heurísticas*. Otras, por el contrario, seguirán considerándose como tareas en curso, y serán utilizadas como tales cuando el *Analizador de Eventos* reciba la siguiente interacción, según vimos anteriormente.

  El motivo por el cual se trata de conocer lo antes posible las tareas que son el objetivo de los usuarios y que, por tanto, se busca aplicar la mayor cantidad de reglas inicialmente, es que, a efectos de proporcionar guía a los usuarios sobre cómo realizar lo que están intentando hacer, es conveniente conocer cuanto antes todas las posibilidades de qué es lo que están intentando hacer en un momento dado.

  La aplicación *impaciente* de las reglas definidas en el modelo para el caso en que un evento comience la ejecución de una tarea es un proceso costoso que puede ser acelerado mediante la aplicación de un *preproceso* en las reglas del modelo. Durante este *preproceso*, se relaciona cada regla con aquellas otras en las que el consecuente de la primera sea una tarea de igual prototipo que alguno de los antecedentes de las demás. Así, las reglas podrán ser aplicadas por el *Analizador de Eventos* sin necesidad de realizar búsquedas intensivas de reglas en el modelo.

Heurísticas. Decimos que el proceso que acabamos de describir da lugar a un estado ambiguo cuando se puede aplicar más de una regla con la llegada de una única interacción del usuario. Esto significa que el sistema no es capaz de determinar con exactitud qué tareas está realizando el usuario de entre todas las definidas en el modelo de tareas de usuario de la aplicación. La presencia de ambigüedades durante el análisis de tareas puede afectar al comportamiento del sistema en mayor o menor medida, sobre todo dependiendo de la finalidad del sistema y de por cuánto tiempo se prolongue la ambigüedad. Así, un sistema que proporcione ayuda deberá conocer qué tareas realiza el usuario cuanto antes, a fin de poder proporcionar una guía dependiente del contexto de la aplicación. En cambio, una aplicación que gestiona aspectos dinámicos de la interfaz de usuario a partir del estudio del histórico de tareas no precisa gran rapidez en la resolución de ambigüedades.

Cuando el gestor de tareas no es capaz de determinar las tareas que el usuario realiza, hay dos políticas que se pueden adoptar:

- Resolver las ambigüedades mediante la aplicación de heurísticas, de manera que se escoja uno sólo de los estados ambiguos y se descarten los demás. La bondad de esta solución depende de lo acertado de la elección de las heurísticas aplicables. La naturaleza de esta

opción puede, lógicamente, ocasionar que las acciones de los usuarios sean malinterpretadas en algunas ocasiones.

- Mantener todos los estados posibles en paralelo hasta que alguna de las acciones del usuario deshaga la ambigüedad. En este caso, el sistema puede no saber qué tareas está realizando el usuario hasta que ha realizado varios pasos, por lo que tiene su capacidad de acción limitada.

En el modelo ATOM no se especifica qué opción de las dos anteriores utilizar, y en la implementación ATOMS realizada se ha utilizado un enfoque mixto: se mantienen todos los estados posibles mediante la formación de una jerarquía de estados posibles que es podada mediante la aplicación de heurísticas adicionales, como veremos a continuación.

3.3.2 Las Tareas Dinámicas en ATOMS

La gestión de las *Tareas Dinámicas* es realizada en tiempo de ejecución, como ya hemos visto, por el *Motor de Eventos*. Éste se encarga de reflejar, tras analizar las interacciones llevadas a cabo por los usuarios, el estado actual de la aplicación. El mantenimiento correcto de qué tareas se llevan a cabo es esencial. Por un lado, permite ofrecer *feedback* acerca de las tareas que se están realizando, lo cual puede ser una ayuda tanto para el usuario final de la aplicación como para el encargado de la depuración del modelo de tareas durante la fase de diseño del modelo. Por otra parte, el mantenimiento de las *Tareas Dinámicas* sirve como base de funcionamiento de todo el sistema. Cada vez que el usuario interacciona con el sistema, el *Motor de Análisis* parte de la información de las *Tareas Activas* para ver qué tareas hay en curso, y ver si la interacción forma parte de alguna tarea por terminar o comienza alguna nueva.

Las *Tareas Dinámicas* son utilizadas tanto por el *Motor de Análisis* como entrada al análisis de interacciones, como por las *Herramientas de Valor Añadido*, que parten de la información del estado actual de las tareas para razonar sobre ellas y ofrecer a las aplicaciones servicios de alto nivel orientados a tareas. En particular, y como veremos en esta tesis, las *Tareas Activas* permiten un razonamiento acerca de la actividad más inmediata del usuario. Este es el caso que se presenta cuando, por ejemplo, se desea facilitar un servicio de guía a los usuarios. Por su parte, el *Histórico* es una representación conveniente para ofrecer servicios orientados a la automatización de secuencias repetitivas de tareas o para la adaptación de interfaces según el tipo de acciones que el usuario realice.

Extracción automática de Macros. Los usuarios de aplicaciones interactivas son todos distintos unos de otros. Las diferencias existentes entre ellos hacen de la especificación ideal de una interfaz interactiva un esfuerzo infructuoso: lo que es útil para algunos es inútil para otros. Por ello, los sistemas adaptativos han surgido como una posible vía para solucionar los problemas asociados a la adaptación dinámica de estos sistemas a las necesidades particulares de cada usuario. Sin embargo, tienen algunos problemas, fundamentalmente relativos a que rompen convenciones generales de HCI (*Human-Computer Interaction, Interacción Hombre-Máquina*) tales como la consistencia y la predectibilidad.

Los sistemas de *Extracción Automática de Macros* (*Automatic Macro Extraction*, AME), que tratan de reutilizar la entrada de los usuarios, son una forma de compromiso para la adaptabilidad de las interfaces, basándose en el concepto universalmente aceptado de adaptación del entorno de trabajo del usuario por medio del uso de macros. Estos sistemas se distinguen de los sistemas basados en *Programación por Demostración* por el hecho de que las macros son creadas de manera automática como un efecto secundario de las interacciones de los usuarios, en vez de

como algo que ha de ser especificado. En común con las técnicas de *Programación por Demostración* se tiene el hecho de que el uso de técnicas para la generalización de macros es vital para la creación de macros útiles, mediante la gestión de patrones de comportamiento de los usuarios.

Como trabajo de investigación suplementario al trabajo de esta tesis, y tomando como base la arquitectura ATOM que estamos describiendo, se ha implementado un entorno AME llamado TRAC (del inglés, *Task Repeating And Completing*, *Repetición Y Finalización de Tareas*), cuyo objetivo es la asistencia al usuario durante la realización de tareas repetitivas, y que se caracteriza fundamentalmente por su capacidad de anticipación a las acciones de los usuarios. Este trabajo puede verse descrito brevemente en el Apéndice IX.

Soporte para la Gestión de Ambigüedades. Como hemos visto, en el caso particular de la implementación ATOMS, el *Motor de Análisis* realiza un seguimiento en paralelo de todas las posibles tareas que esté realizando el usuario, de modo que no se conoce con exactitud qué tareas está realizando hasta que, bien por la aplicación de *heurísticas* o por una interacción que deshaga la ambigüedad, el sistema determine exactamente el estado. A continuación vamos a describir el algoritmo utilizado por ATOMS para realizar este seguimiento paralelo mediante el uso de *estados posibles.*

En ATOMS, cada tarea de las *Tareas Dinámicas* es una instancia, en el sentido de herencia según el paradigma prototipo/instancia, de su correspondiente prototipo especificado mediante el modelo de tareas. Esta instanciación la realiza el *Motor de Análisis* como resultado del proceso de análisis de interacciones.

Denominamos *estado posible* (EP) a cada uno de los estados globales en los que el sistema considera que pueden encontrarse las tareas de una aplicación. Cada EP contiene una cantidad arbitraria de instancias de tareas correspondientes a las diferentes actividades cuya realización está en curso. Un EP, además de contener estas instancias de tareas, tiene un atributo *booleano*, llamado *estado de análisis*, que determinará si el *Motor de Análisis* deberá utilizar dicho *estado posible* para el proceso de *parsing* de interacciones o no. Inicialmente, cuando se arranca una aplicación, se parte de un único *estado posible* que contiene el conjunto de tareas vacío, y cuyo *estado de análisis* será *verdadero*.

La Figura 23 ilustra un ejemplo de evolución de los EPS durante una sesión de trabajo interactivo con una aplicación que utiliza ATOMS. Este ejemplo será utilizado en los párrafos siguientes para ilustrar el algoritmo de gestión de ambigüedades que utiliza. En la parte izquierda de la Figura 23 aparece el estado inicial de las *Tareas Activas* tras arrancar una aplicación, donde el EP EP0 está vació, y su contorno continuo indica que su estado de análisis está activado de cara al análisis de interacciones.

Tras cada interacción, el *Motor de Análisis* mirará, para cada EP que tenga su *estado de análisis* activado, qué reglas son aplicables, y creará un nuevo EP que será descendiente del EP considerado en cada momento. Es decir, cada EP da lugar a tantos nuevos EPs como reglas sean aplicables a dicho estado tras la llegada de cada evento. Además, estos nuevos EPs se crearán conteniendo las mismas tareas y en los mismos estados que contenía el EP del que descienden, salvo los cambios asociados a la aplicación de la regla aplicada. Este proceso de aplicación de reglas se llevará a cabo recursivamente sobre los nuevos EPs cuando se comience una nueva tarea, como se explicó anteriormente al tratar el comportamiento *impaciente* del sistema.

Como vemos en la parte central de la Figura 23, la posibilidad de aplicar las reglas R1 y R2 al EP EP0 trae como consecuencia la creación de los EPs EP01 y EP02, descendientes de EP0, y cuyos contenidos son el resultado de aplicar las reglas R1 y R2, respectivamente. Además, como acabamos de comentar, el *Motor de Análisis* trata de aplicar las reglas de manera *impaciente*, de tal modo que trata de aplicar cualquier regla posible a los estados resultantes EP01 y EP02. Suponiendo que sólo sea aplicable la regla R3 al EP EP01, obtenemos el EP EP011 de la figura, descendiente del EP EP01.

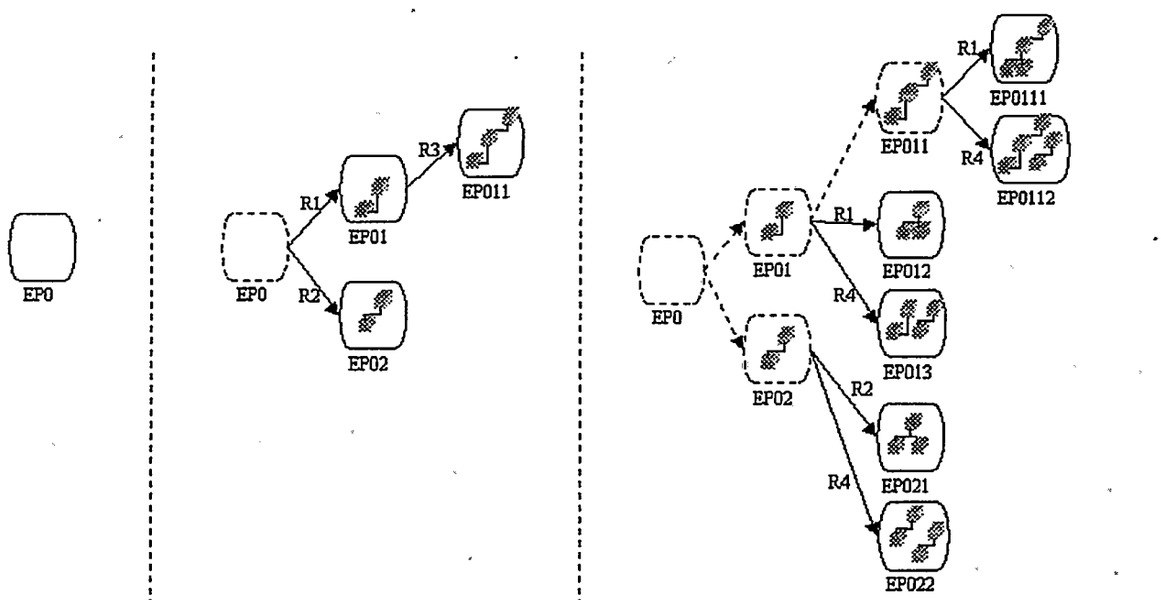

Figura 23: Ejemplo de evolución de los *estados posibles*.

El *estado de análisis* de un EP recién creado es siempre verdadero, y al EP del que desciende se le asocia un *estado de análisis falso* (a menos que se haya creado como resultado de la misma interacción del usuario, como es el caso de EP01), lo cual hará que ya no sea utilizado como parte del proceso de análisis de interacciones. De este modo, el *Motor de Análisis* va generando una jerarquía de EPs que representa los estados de las tareas que el sistema percibe como ambiguos, y que permite el tratamiento en paralelo de dichos estados ambiguos.

Para finalizar con el ejemplo de la Figura 23, veamos qué ocurre tras el procesamiento de la llegada de una nueva interacción (parte derecha). Supongamos que esta nueva interacción pueda hacer que se comience una nueva tarea mediante la aplicación de la regla R4. Esto, lógicamente, es aplicable independientemente del EP de partida que se utilice. Supongamos que, además, la misma interacción permite continuar aplicando las reglas R1 y R2 que se comenzaron a aplicar con la interacción anterior. Entonces, a partir de cada EP activo para el *parsing* se obtendrá un nuevo EP tras la aplicación de R4 y, para los EPs de partida obtenidos tras aplicar R1 se obtendrá otro EP con la llegada de la nueva interacción. Lo mismo ocurre, finalmente, para el EP proveniente de haber aplicado R2.

Resolución de Ambigüedades. Como ya hemos dicho, ATOMS gestiona las ambigüedades utilizando un enfoque mixto, basado en un seguimiento en paralelo mediante el empleo de la jerarquía de EPs que acabamos de describir, y la aplicación de *heurísticas*.

La aplicación de *heurísticas* permite podar la jerarquía de EPs de manera que se eliminan aquellas ramas asociadas a EPs que parezcan menos plausibles que algún otro EP. Las *heurísticas* que se utilizan son consecuencia de un principio básico conocido como *principio de continuidad*: *"En general, los usuarios terminan las tareas que están realizando antes de comenzar tareas nuevas."* Según este principio, la actividad de los usuarios va encaminada a completar los procesos que han comenzado, y no es habitual que inicien otros antes de terminarlos, aunque en ocasiones pueda ser así.

En términos de poda de la jerarquía de EPs, el *principio de continuidad* tiene como consecuencia la aplicación de las siguientes *heurísticas*:

- Se podan aquellos nuevos EPs que tienen más tareas que algún otro nuevo EP. Esta heurística es la consecuencia más directa del *principio de continuidad*, y da preferencia a la aplicación de reglas que no ocasionen la instanciación de tareas nuevas en un EP y, por tanto, que consideren la actividad de los usuarios como una continuación de sus anteriores acciones.
- Cuando tras el análisis de una interacción se ha creado algún nuevo EP, se podarán todos aquellos EPs existentes anteriormente a dicha interacción que no hayan tenido descendientes. Es decir, se da preferencia a aquellos EPs para los que la última interacción ha significado algún avance en las tareas que en él aparecen representadas, frente a aquellos EPs para los cuales la interacción no ha significado cambio alguno.
- Cuando tras el análisis de una interacción se ha creado algún nuevo EP en el que una o más tareas compuestas se hayan terminado, se podarán todos aquellos EPs en los que se hayan terminado menos tareas. Es decir, se da preferencia a aquellos EPs para los que la última interacción ha significado un mayor avance en la dirección de finalizar tareas de mayor nivel conceptual.

Además de estas heurísticas, ATOMS aplica otra heurística con el fin de posibilitar que un mismo modelo de tareas incluya la especificación de un prototipo de regla y de una instancia suya. Así, y con el fin lógico de que siempre se aplique aquella regla que resulte lo más específica posible, ATOMS aplica un heurístico que favorece este comportamiento, evitando así situaciones anómalas que produzcan ambigüedades innecesarias.

Estas *heurísticas* permiten podar la jerarquía de EPs cuando el *Motor de Análisis* realiza el seguimiento de tareas. Sin embargo, en ocasiones los sistemas de gestión de tareas se utilizan con fines particulares tales que permiten al sistema saber algo más de la actividad de los usuarios para reconocer con mayor exactitud qué pueden estar haciendo. Así, por ejemplo, cuando el sistema gestor de tareas está siendo utilizado para enseñar a realizar una tarea, el sistema aplica el criterio heurístico de quedarse con aquellos EPs en los que el usuario está realizando la tarea que se está enseñando, y descarta los EPs en los que no.

Cuando la jerarquía de EPs es podada, se dan casos en los que se tienen nodos del árbol con un único descendiente activo para el análisis de interacciones. La proliferación de este tipo de nodos provoca con el tiempo una ineficiencia en el sistema durante dicho proceso de análisis, ya que recordemos que el *Motor de Análisis* lo realiza utilizando todos aquellos EPs activos de la jerarquía. Para evitar esta ineficiencia, ATOMS realiza un proceso de *compactación* de la jerarquía resultante tras las podas, lo cual reduce el número de nodos inactivos que el árbol de EPs contiene.

Para ilustrar la aplicación de los criterios heurísticos anteriores, vamos a ver un ejemplo de lo que sucede continuando el mismo escenario planteado en la Figura 23. Como vimos, los EPs resultantes tras la llegada de las dos interacciones eran los que aparecían en la parte derecha de aquella figura. Ahora, aplicando el hecho de que han aparecido nuevos estados en los que no se ha creado ninguna tarea nueva, habremos de eliminar todos aquellos nuevos EPs que contienen alguna nueva tarea: EP0112, EP013 y EP022. En la parte izquierda de la Figura 24 podemos ver tachados los estados a los que la poda afecta. También vemos que el nuevo estado EP0111 es el único estado activo para el *parsing* que desciende del estado EP011, por lo cual el estado EP011 puede ser eliminado para compactar la jerarquía, al igual que le sucede a EP02. En la parte central de la Figura 24 podemos ver el resultado de la compactación una vez realizada la citada poda. Pero, como vemos en la figura, el estado EP021 contiene una tarea que se ha finalizado tras la llegada de la última interacción, mientras que no se ha terminado ninguna de las tareas en los estados EP0111 ni EP0112, por lo que podemos eliminar estos dos últimos estados, en virtud del *principio de continuidad*. Para terminar, en la parte derecha de la Figura 24 vemos el resultado de la compactación tras la segunda de las fases de poda, en la que ya sólo tenemos un único EP, con una única tarea, con lo que se han eliminado todas las ambigüedades. Además, al haber una única tarea y encontrarse finalizada, dicha tarea pasa a formar parte del histórico de tareas realizadas.

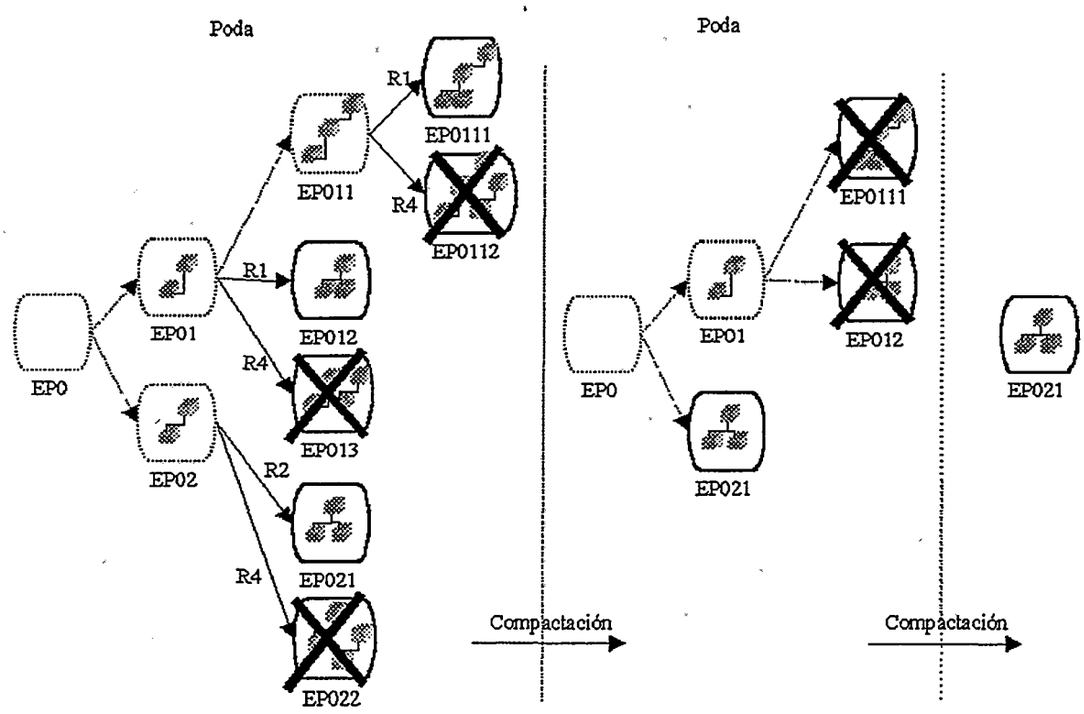

Figura 24: Poda y compactación de la jerarquía de EPs.

3.4 Emulación de Tareas

Conforme a la Figura 8, ya se han descrito todos los módulos de la arquitectura ATOM salvo uno que, bajo la denominación de *Herramientas de Valor Añadido,* engloba a todos aquellos servicios de valor añadido dependientes de la implementación, y que van enfocados a los desarrolladores de los sistemas o a sus usuarios. Para el trabajo expuesto en esta tesis se han desarrollado varias de estas herramientas dentro de la implementación de ATOM que se ha realizado, la primera de las cuales se introduce en esta sección.

Esta primera *Herramienta de Valor Añadido* responde al nombre de *Emulación.* Este bloque funcional del sistema tiene por objetivo el aportar una interfaz para que se puedan ejecutar tareas de la aplicación mediante el uso de animaciones. Este módulo es facilitado por ATOMS como un servicio que herramientas externas de valor añadido pueden utilizar como parte de la funcionalidad que provean, de manera que puedan hacer que las aplicaciones realicen tareas como si fueran los propios usuarios los que las realizaran.

Como se verá posteriormente, este módulo es la base del *Módulo de Ejecución de Escenarios,* descrito en la Sección 4.3, que a su vez es uno de los pilares de CACTUS. Durante la ejecución de las tareas, el cursor del ratón aparece por la pantalla moviéndose de la misma manera en que lo haría moverse un usuario para realizar dichas tareas, a la vez que también se indican las acciones a llevar a cabo mediante el uso del teclado. Asimismo, los usuarios reciben el mismo *feedback* y el mismo comportamiento por parte de la aplicación que percibirían en el caso de que los usuarios realizaran las tareas por ellos mismos.

Este módulo es capaz de emular las interacciones de los usuarios mediante el análisis de las tareas y reglas especificadas en el modelo de tareas de la aplicación. Así, partiendo de dicho modelo, la descripción de la tarea que se desea emular y los valores de los parámetros que se faciliten para dicha tarea, *Emulación* decide qué tareas atómicas deben ser ejecutadas, es decir, qué interacciones se deben emular y qué valores deben llevar asociados dichas interacciones.

Como ya hemos visto, en ATOM cada tarea atómica lleva asociado un *patrón de interacción* que describe de manera abstracta la interacción asociada a la tarea. En la implementación ATOMS, estos patrones de interacción vienen representados por *patrones de objetos.* Además, y como vimos en la Sección 2.3, los objetos ORE de AMULET se pueden organizar formando jerarquías de objetos. En particular, las componentes gráficas de este entorno de desarrollo y los interactores siempre se organizan de forma jerárquica, para permitir la gestión automática tanto de la apariencia de dichos objetos durante la ejecución de las aplicaciones como de la aplicación de interactores estándar a diferentes tipos de componentes gráficas. Son estas relaciones jerárquicas y los *patrones de interacción* las bases que utiliza *Emulación* para determinar que acción debe realizar y qué valores se le deben asociar a dichas acciones para realizar una tarea atómica dada. Este proceso de encontrar qué interactores deben entrar en acción y con qué valores puede considerarse como la acción inversa a la realizada por la componente de *matching* entre eventos y patrones de interacción de ATOMS.

El hecho de que ATOMS sólo incorpore una modelización de las tareas de los usuarios, y no incorpore una modelización de *Manipuladores* o de *Dispositivos,* tal y como se hace en otros entornos como HUMANOID o UIDE, hace que la lógica asociada a la descomposición de cada tipo de interacción en los distintos pasos que los usuarios deben realizar para completar dicha interacción hayan de estar codificados dentro de *Emulación.* Así, no es posible que *Emulación* realice correctamente nuevas interacciones definidas para una aplicación en particular sin antes

tener que modificar este módulo. Esto, que significa una limitación a priori, no lo es en la práctica, ya. que conviene destacar que no es habitual la definición de nuevos tipos de interactores, salvo para casos muy particulares como, por ejemplo, el reconocimiento de voz o de gestos.

Emulación proporciona un API (*Application Programming Interface*) que permite a las aplicaciones basadas en ATOMS acceder, mediante animaciones, a la realización automática de tareas de usuario de la aplicación. Este API permite indicar a *Emulación* qué tarea se desea realizar y, también, con qué información contextual, es decir, qué valores deberán tener asociados los parámetros de la tarea a ejecutar. Así, por ejemplo, se le puede pedir a *Emulación* que ejecute la tarea *CrearNuevaAsignación* (ver Figura 19) mediante el uso de animaciones, indicándosele, además, que el nombre del profesor a seleccionar sea "*Mary Brown*". Como consecuencia, *Emulación* simularía las acciones de ratón y/o teclado necesarias para emular la creación de una nueva relación de asignación cuyo parámetro *profesor* tuviera el valor "*Mary Brown*".

Es importante hacer notar que *Emulación* puede recibir peticiones de ejecución de tareas tanto atómicas como compuestas. En este último caso, la información contextual que se proporciona hace referencia a los parámetros de la tarea compuesta que a *Emulación* se le está pidiendo realizar. Pero, dado que *Emulación* sólo puede emular tareas compuestas a partir de la sucesiva emulación de una o varias tareas atómicas, estos valores de parámetros para tareas de alto nivel deben ser convertidos a valores de parámetros para tareas de niveles inferiores, hasta llegarse a obtener valores de parámetros de las tareas atómicas a ejecutar y, por tanto, a valores asociados a las interacciones del entorno subyacente de implementación. Por esto, uno de los pasos más importantes que *Emulación* ha de llevar a cabo es la transformación de los valores de los parámetros de la tarea compuesta en valores de parámetros para las subtareas que compongan dicha tarea. Este proceso, a su vez, se repetirá para estas subtareas en el caso de que sean tareas compuestas, de manera que finalmente se promocionen los valores de los parámetros hasta valores de parámetros para las hojas de la jerarquía que representa la tarea solicitada.

El tipo de información necesaria para la emulación de las interacciones es dependiente de la interacción que se quiera emular. Así, por ejemplo, en el caso del interactor de selección, la información relevante es qué elemento se desea seleccionar de entre los posibles objetos gráficos seleccionables por dicho interactor, mientras que para el interactor de edición de textos se debe proporcionar el objeto textual cuyo texto se desea editar y el valor final de dicho texto. *Emulación* puede encontrarse con que necesita emular una cierta interacción y no conoce el valor de alguno de los parámetros necesarios para dicha interacción. Esta situación se trata indicando al usuario que facilite interactivamente, en tiempo de ejecución, un valor adecuado para dichos parámetros. En los casos en los que es posible, *Emulación* también ofrece a los usuarios *feedback* gráfico acerca de las posibles opciones disponibles que tienen para escoger el valor de los parámetros. Así, en el ejemplo anterior no se facilita ningún valor ni para el nombre de la asignatura ni para el grupo para los que se quiere crear la nueva asignación, por lo que *Emulación* no conocerá qué elementos debe seleccionar dentro de cada una de las listas correspondientes. Entonces, *Emulación* hace que todos los elementos de cada una de estas dos listas parpadeen en el momento en que estas interacciones han de emularse, indicando al usuario que seleccione alguno de ellos como parte de las tareas *SeleccionarAsignaturaDeLista* y *SeleccionarGrupoDeLista*. A partir de ese momento, el sistema sólo permitirá al usuario interaccionar con el sistema para facilitarle el valor de dichos parámetros, bloqueando el resto de opciones hasta ese punto accesibles. En un sistema orientado a la comercialización al usuario.

final, habría que cuidar especialmente la forma en que el sistema realizaría las peticiones a los usuarios, de modo que se evitasen las preguntas poco inteligibles. La forma en que esto habría de realizarse sería permitiéndose que los diseñadores especificasen cuál debiese ser el contenido textual de las preguntas, así como que pudiesen seleccionar qué tareas serían *emulables* y cuáles no. En ese último caso, la utilidad de incluir dichas tareas en los modelos seguiría siendo clara, pues dichas tareas, pese a no ser *emulables*, podrían seguirse enseñando.

Para llevar a cabo la propagación de los valores de los parámetros hacia abajo en la jerarquía, *Emulación* aplica las funciones inversas de las funciones de transferencia de parámetros que se especifiquen en el modelo de la tarea cuya ejecución ha sido solicitada. Así, en el ejemplo anterior, el parámetro textual *profesor*, cuyo valor asociado es *Mary Brown*, está asociado a la tarea requerida, *CrearNuevaAsignación*, y es convertido internamente por el sistema al valor del parámetro *nombre* asociado a la subtarea *SeleccionarProfesorDeLista* de *CrearNuevaAsignación*. A su vez, el valor del parámetro *nombre* asociado a la tarea *SeleccionarProfesorDeLista* es convertido al objeto gráfico (*selectedItem*) cuyo atributo textual (*TEXT*) tiene por valor *Mary Brown*, y que irá asociado a la interacción de selección en lista correspondiente a la tarea atómica *SeleccionarProfesorDeLista*. Este último paso de conversión viene determinado a partir de la expresión de paso de parámetro contenida en la especificación de la tarea LIST, de la cual desciende *SeleccionarProfesorDeLista*. Este proceso queda esquematizado en la Figura 25, donde se muestra (utilizando nombres de tareas abreviados, por razones de espacio) cómo se realiza la propagación de parámetros hacia abajo en la jerarquía de la tarea *CrearNuevaAsignación*. De este modo, cuando dicha interacción se quiera emular se le facilitará como objeto a seleccionar aquél que tenga dicho valor para ese atributo. En este ejemplo, dado que los pasos de parámetros especificados se corresponden con la función identidad, las funciones inversas aplicadas son, igualmente, las funciones identidad. Sin embargo, éste es el caso más sencillo que se puede presentar.

En los casos en los que se encuentre con que la función inversa a aplicar no es la función identidad, o bien la función no sea biyectiva, habría que indicar al sistema cómo debería realizarse esta propagación inversa de los parámetros.

Así, por ejemplo, el primer caso se presenta frecuentemente en las funciones de paso de parámetros en las que se pretende dar formato a textos. Pongamos por caso el de una tarea de selección de una fecha, *SeleccionarFecha*. Para dicha tara tendríamos tres subtareas, a saber, selección del día, selección del mes, y selección del año. Una vez se realizaran las subtareas, los parámetros de cada subtarea se concatenarían, separándose mediante el carácter '/', para formar la fecha seleccionada por el usuario. En el caso de que se deseara emular *SeleccionarFecha* con un valor para su parámetro *fecha* de "12/Septiembre/1974", habría que llevar a cabo el *desformateado* de dicho valor, para lo que habría que permitir a los diseñadores indicar al sistema, mediante programación, cómo realizar dichas funciones inversas.

Por su parte, en el caso de enfrentarnos a funciones no biyectivas, el proceder debería variar. Esto se debe a que cuando se utilizan funciones de transferencia de parámetros no biyectivas para especificar funciones de transferencia de parámetros, no es posible convertir un valor de la función en valores únicos para las variables de dicha función. Esto significa que existen distintos valores de parámetros de tareas de un nivel inferior que dan lugar a cierto valor de un parámetro de una tarea. Así, pongamos por caso el de una tarea *CalcularImporte* de un pedido que calculara el importe a partir de el precio del objeto seleccionado por el usuario de un catálogo de productos y la cantidad de unidades de dicho producto que el usuario deseara recibir. Si ahora deseásemos

emular dicha tarea para un importe cuya cantidad pongamos fuese 100000, el sistema podría obtener dicha cantidad como el resultado de tener una unidad de un artículo de precio 100000, como dos unidades de uno que costara 50000, etcétera. Para este tipo de casos de funciones no biyectivas existirían varias soluciones, y debería permitírsele al diseñador del modelo el elegir, en cada caso, cómo debería comportarse el sistema cuando se encontrara con una función de este tipo. A continuación, se describen brevemente cada una de estas posibles soluciones:

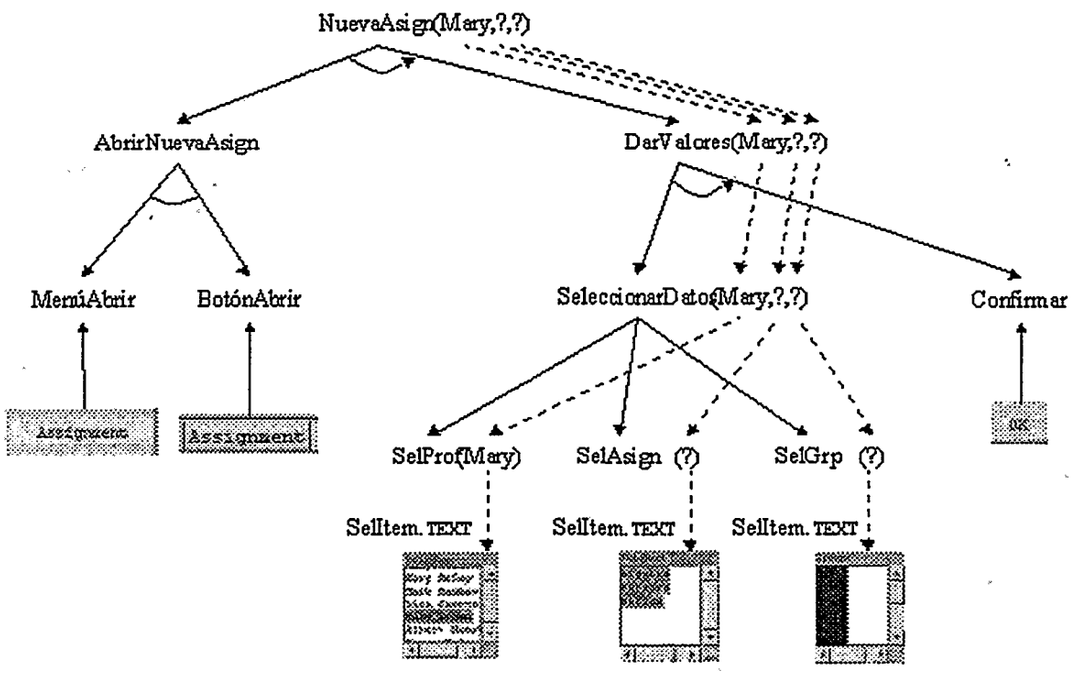

Figura 25: Propagación de parámetros hacia abajo en la jerarquía de una tarea.

- Seleccionando valores por omisión adecuados para algunas de las variables involucradas, de modo que se pueda obtener directamente el valor de la variable restante a partir del valor del parámetro resultante. Sin embargo, esta elección de valores por omisión implica el tener un mayor conocimiento acerca de la semántica de las aplicaciones. Esto es lo que se hace, por ejemplo, en Cartoonist [Sukaviriya90], donde el entorno de modelización contiene una modelización de datos con los que la aplicación opera, especificándose los diferentes rangos disponibles para cada tipo de datos y las colecciones de instancias de objetos existentes en cada momento de la ejecución. Sin embargo, estos modelos introducen una mayor complejidad en la modelización de las aplicaciones y proporcionan escasa ayuda desde el punto de vista pedagógico de cara a los usuarios noveles. Para el ejemplo descrito anteriormente (*CalcularImporte*), esta opción no sería válida porque probablemente a priori no podría determinarse un valor por defecto para, por ejemplo, el artículo a comprar, ya que la lista de éstos se determinaría dinámicamente a partir de un catálogo.
- Permitir al usuario aportar los valores de las variable involucradas hasta que el sistema, por sí mismo, sea capaz de obtener el resto de valores para las demás variables. Esta solución es una

modificación de la anterior, en el sentido que no existen valores por defecto, sino que los usuarios, en tiempo de ejecución, eligen dichos valores. En estos casos, el sistema deberá poder validar los valores seleccionados por los usuarios, de modo que solo permita aportar aquellos que permitan obtener el valor del parámetro que se desee. En el ejemplo tratado, *CalcularImporte,* este proceder haría que el sistema, mientras emulara la tarea propuesta, preguntaría al usuario qué elemento del catálogo desearía seleccionar, y a continuación propondría automáticamente el número de unidades de dicho artículo con la restricción de que el precio total fuera 100000. El sistema, en este caso, podría tener que validar que el producto seleccionado tuviera un importe que fuera divisor entero de la cantidad total.

- Ignorar aquellas funciones de transferencia de parámetros que no sean biyectivas. Al hacerse esto, *Emulación* seguirá intentando descomponer las tareas en subtareas hasta llegar a niveles atómicos y, entonces, resultará que hay información referente a parámetros de tareas o de interacciones que no se tienen y, como se mencionó anteriormente, el sistema pedirá a los usuarios que proporcionen los valores deseados de manera interactiva. En nuestro ejemplo, el sistema no podría emular la tarea *CalcularImporte* con la restriccción de aportar un valor determinado, sino que simplemente permitiría al usuario seleccionar tanto el producto como la cantidad de unidades del mismo.

Un detalle de importancia con respecto a *Emulación* que aquí se remarca es la posibilidad de que este subsistema ejecute automáticamente acciones que el usuario, o la propia lógica de la aplicación, posteriormente desee deshacer. Dado que estas acciones que se ejecutan con los sistemas interactivos a menudo no se corresponden con interacciones atómicas aisladas, sino que muy frecuentemente se trata de tareas con cierta complejidad, se ha construido sobre el *framework* AMULET (ver Sección 2.3) un mecanismo de *undo* que permite a los usuarios deshacer tareas como si se tratase de un único bloque de acciones. Podemos pensar en este mecanismo de *undo* como en el mecanismo recíproco al de la emulación de tareas, pues permite deshacer sus efectos. Este mecanismo de *undo de tareas* ofrece dos ventajas fundamentales:

- Por un lado, y de cara a los diseñadores de las aplicaciones, se permite que los sistemas para los que se definan sus modelos de tareas tengan disponible, automáticamente, además de la opción de *undo* tradicional proporcionada por el entorno de desarrollo AMULET, un *undo de tareas*. La potencia de èste mecanismo de *undo de tareas* es importante, ya que permitirá a los usuarios deshacer de un solo golpe conjuntos de acciones que para ellos tienen un significado conceptualmente único. Se trata, por tanto, de la evolución lógica desde el *undo* tradicional al *undo* de los sistemas basados en modelos de tareas.
- Por otra parte, y como posteriormente veremos, los servicios de *Emulación* son utilizados desde herramientas de más alto nivel. Para estos servicios, la existencia del citado mecanismo de *undo de tareas* ofrecerá la posibilidad de hacer que los efectos de *Emulación* sean o no persistentes con respecto de la actividad habitual de los usuarios. Es decir, una vez que se haya empleado *Emulación* con un determinado fin, podremos deshacer sus efectos automáticamente para que los usuarios puedan continuar con normalidad su trabajo.

3.5 Gestión de Flujos de Trabajo

Hasta el momento hemos descrito la arquitectura ATOM, y su implementación ATOMS, como un entorno de gestión de tareas interactivas de usuario. También hemos visto cómo pueden proporcionarse servicios adicionales mediante herramientas externas al sistema. En esta sección

vamos a describir Wf-ATOMS [García99a] [García2000c], una *herramienta de valor añadido* para la gestión de flujos de trabajo, diseñado e implementado sobre la arquitectura ATOM de gestión de tareas de usuario. Este sistema sienta las bases de aplicación de todas las ventajas que veremos a partir de la próxima sección en un entorno multiusuario distribuido. El objetivo que se pretende cubrir con la inclusión de esta sección es hacer ver cómo las ideas desarrolladas en ATOM permiten, con escasas adiciones, completar una arquitectura de gestión de flujos de trabajo que incluya aspectos novedosos.

Los flujos de trabajo (en inglés, *workflows*) representan conocimiento procedimental relacionado con procesos de negocio, administrativos, etcétera, que tienen lugar en cualquier tipo de organización. Las corporaciones a menudo intentan incrementar la calidad y disminuir los costes mediante la modelización y mejora de sus procedimientos internos de negocio. En este contexto, la utilización de flujos de trabajo permite la captura y posterior coordinación de dichos procesos.

A lo largo de los años, han aparecido muchos sistemas informáticos para el control automatizado de flujos de trabajo. Asimismo, se han propuesto muchas aproximaciones a la modelización de flujos de trabajo, dependiendo de la técnica predominante utilizada para su desarrollo [Alonso96]. Entre ellos, pueden destacarse los entornos basados en transacciones de bases de datos [Kamath95], en eventos [Weske96] y en redes de Petri [Van der Aalst94]. La aproximación de Wf-ATOMS está asada en la idea de tareas distribuídas entre diferentes usuarios.

Los sistemas de gestión de flujos de trabajo proporcionan un soporte tal que permiten modelizar la descomposición en subtareas de los procesos de trabajo y los *roles* (papeles) jugados por cada entidad de proceso. Las entidades de proceso que pueden formar parte de los flujos de trabajo incluyen recursos humanos y sistemas informáticos. Cuando una subtarea es completada, el trabajo es redirigido automáticamente a la entidad de proceso responsable de realizar la siguiente subtarea. Las tareas de un flujo de trabajo deben satisfacer ciertas restricciones que han de ser capturadas en los correspondientes modelos del flujo. En la Figura 26 podemos observar la representación esquemática de un flujo de trabajo, en particular de la introducción de un parte de gastos de viajes de un empleado. Los subprocesos realizados en cada fase del proceso se encuentran representados en óvalos, en los que en el recuadro interior se indica el perfil asociado a la entidad de proceso que los lleva a cabo, las flechas horizontales representan las relaciones temporales existentes entre los procesos, y los textos bajo los óvalos hacen referencia a las herramientas aplicativas mediante las que los usuarios llevan a cabo los procesos. El flujo de trabajo ilustrado en el ejemplo es muy sencillo, consistiendo en la simple concatenación de subprocesos atómicos.

Los beneficios más importantes que un sistema de gestión de flujos de trabajo puede proporcionar son claros, todos ellos centrados en la mejora de la gestión de los procesos. En el caso particular de Wf-ATOMS, sus ventajas básicas van orientadas a proporcionar:

- asignación automática de subprocesos a los responsables de llevarlos a cabo;
- proporcionar a laa entidades de proceso la información requerida en cada paso de una secuencia;
- gestionar automáticamente el acceso a los procesos; y
- la monitorización de los procesos involucrados en un procedimiento.

Wf-ATOMS es un entorno para la especificación y gestión de modelos de flujos de trabajo, cuyo motor de procesos esta integrado con el motor de tareas de usuario del sistema ATOMS. Como consecuencia de esta integración, Wf-ATOMS permite un mayor grado de control sobre las actividades de los flujos de trabajo del que los motores de flujos de trabajo convencionales permiten tener.

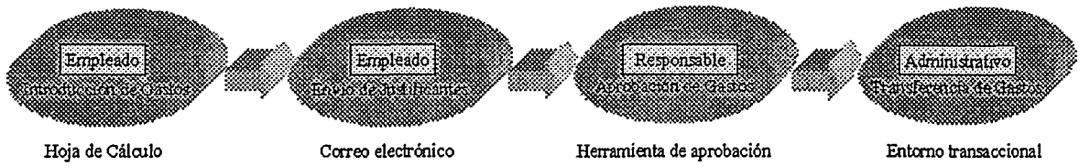

Figura 26: Ejemplo de flujo de trabajo.

Wf-ATOMS gestiona la asignación de actividades de flujos de trabajo a los usuarios. Los usuarios habitualmente se encuentran clasificados según *roles* o perfiles, que establecen las restricciones que éstos tienen con respecto a funcionalidades en los aplicativos. Así, por ejemplo, en la práctica se encuentran *roles* tales como administradores, responsables, revisores, creadores, etcétera, y a cada cual se le habilita el acceso a diferentes funcionalidades dentro del marco de un trabajo en equipo.

A continuación se va a describir cómo, utilizando las mismas técnicas de modelización introducidas para ATOM, se pueden modelizar flujos de trabajo. Posteriormente, se describirá la arquitectura diseñada sobre ATOM para dar soporte a la gestión de los flujos de trabajo capturados.

3.5.1 Modelización de procesos

Al igual que sucede en el caso de la modelización de tareas de usuario en ATOM, los modelos de Wf-ATOMS se componen de tareas y reglas. Las tareas modelizan los procesos del flujo de trabajo, mientras que las reglas, igual que sucede en ATOM, establecen las relaciones entre dichos procesos.

Como sucede en ATOM, las tareas de los flujos de trabajo forman jerarquías, en las que procesos más complejos son descompuestos en otros más simples. Así, para un caso sencillo de un flujo de trabajo referido a un proceso de creación colaborativa de un cierto documento, se definirá un modelo de flujo de trabajo en el que usuarios con un *rol* que permita la creación de contenido podrán modificar el contenido del documento hasta que un usuario con un *rol* que permita la revisión y aprobación del documento dé el visto bueno a su contenido.

Por ejemplo, en la Figura 27 podemos ver la representación esquemática del procesamiento de una hoja de gastos en una empresa. En dicha figura, los procesos están representados por una jerarquía en la que un proceso más abstracto se descompone en varios procesos más simples, cada uno de los cuales debe ser realizado con una herramienta específica, mostrada debajo, y por un perfil de usuario determinado. En primer lugar, los empleados utilizan una hoja de cálculo para reflejar los gastos que han tenido. Posteriormente, mediante un correo envían los justificantes en formato electrónico. Tras este paso, los responsables de aprobar dichos gastos

utilizan otra aplicación de gestión de pagos para realizar su aprobación y, finalmente, con un último aplicativo el personal administrativo realizará la transferencia bancaria del pago.

Wf-ATOMS, al igual que ATOMS, permite especificar los modelos de flujos de trabajo mediante un entorno basado en el paradigma de herencia prototipo/instancia. Este entorno aprovecha los mismos conceptos ya presentes en ATOMS. Así, en los modelos de flujos de trabajo, tareas o procesos de mayor grado de abstracción se descomponen en otros más simples que, de manera sucesiva, son de nuevo refinados hasta que hacen referencia a procesos ligados a tareas de aplicaciones interactivas.

En Wf-ATOMS, las tareas son de dos tipos: tareas *compuestas* y *tareas remotas*. Las primeras se corresponden con las mismas *tareas compuestas* ya tratadas en ATOM, mientras que las *tareas remotas* son específicas de Wf-ATOMS. Los procesos más abstractos son tareas compuestas, que se van descomponiendo en procesos más sencillos. Este procedimiento recurrente se repite hasta que, a determinado nivel jerárquico, se obtienen procesos atómicos que se corresponden con lo que durante este trabajo se ha denominado *tareas* de sistemas interactivos, que los usuarios deberán completar para poder proseguir el proceso de la ejecución del flujo. Las *tareas remotas* proporcionan un mecanismo mediante el cual se pueden establecer referencias entre los procesos de los flujos de trabajo y las tareas interactivas que dan soporte a dichos procesos. Normalmente, estas referencias se harán, de manera específica, a *tareas compuestas* de los sistemas interactivos, con un grado de abstracción elevado. Por lo tanto, Wf-ATOMS permite definir modelos de flujos de trabajo que funcionan sobre modelos de tareas interactivas ya definidos para los sistemas informáticos involucrados en el flujo de trabajo.

En la Figura 27 podemos ver, junto al proceso *ProcesamientoHoja* una posible modelización de dicho proceso, mientras que junto a cada una de las herramientras que participan en el flujo se encuentran representadas los modelos de las tareas involucradas. Mientras que los procesos se modelizan con Wf-ATOMS, la modelización de las tareas, como hemos visto, es parte de ATOM.

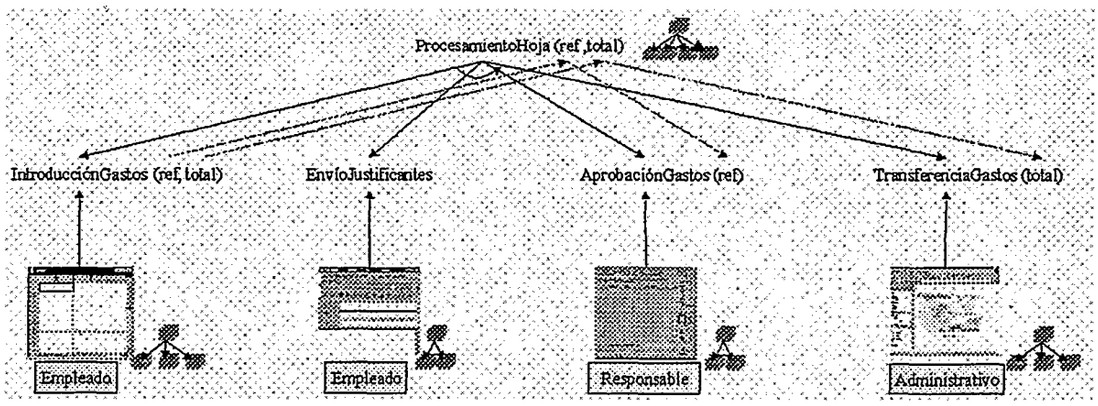

Figura 27: Modelización del procesamiento de una hoja de gastos.

Por lo tanto, para Wf-ATOMS se ha ampliado el marco de especificación de modelos de ATOMS para incluir el prototipo de *tarea remota*, que permite conectar las actividades abstractas modelizadas en los flujos de trabajo distribuídos con tareas correspondientes a aplicaciones concretas y que deben ser asignadas a usuarios con perfiles específicos. De este modo, pueden

considerarse los modelos de flujos de trabajo como modelos de tareas distribuídas y multiusuario, ya que la filosofía de gestión de ambos tipos de modelos es idéntica, variando únicamente en la gestión de perfiles de usuarios y posibilidad de acceso a varias aplicaciones interactivas ligadas entre ellas.

En la Figura 28 pueden observarse los atributos del prototipo de tarea remota.

Atributo	Tipo	Valor por Omisión
Nombre	<CadenaTexto>	"TRemota"
Prototipo	<Tarea>	TGenerica
Tarea	<Tarea>	Nulo
Aplicación	<NombreAplicación>	Nulo
Usuario	<Usuario o *Rol*>	Nulo

Figura 28: Atributos del prototipo de tarea remota.

Nombre y prototipo. Los atributos *Nombre* y *Prototipo* cumplen la misma función que cumplen en ATOMS los atributos de igual nombre, es decir, indican cuál es el nombre de la tarea y de qué tarea es instancia, en este caso de la *tarea genérica.*

Tarea y Aplicación. Por su parte, los atributos *Tarea* y *Aplicación* establecen la relación entre un determinado paso o etapa de un modelo de flujo de trabajo y el proceso interactivo que para completar dicha etapa debe llevarse a cabo, es decir, la(s) tarea(s) modelizada(s) con ATOM. En concreto, el atributo *Tarea* hace referencia a qué tarea, ya sea atómica o compuesta, debe llevarse a cabo con la *Aplicación.* Como ya hemos comentado, lo habitual será que la tarea a llevar a cabo por la persona encargada de continuar el flujo sea una tarea relativamente compleja, compuesta de más de una interacción atómica. El modelo concreto de dicha tarea se considera previamente definido conforme a las indicaciones de la Sección 3.2, dentro del modelo de tareas del aplicativo que la da soporte.

Usuario. Finalmente, el atributo *Usuario* indica qué usuario deberá llevar a cabo la tarea asignada a la etapa del proceso, o bien qué *rol* deberá tener el usuario que complete el proceso. El entorno Wf-ATOMS incorpora un sencillo módulo que permite gestionar usuarios en base a *roles,* con el fin de permitir la autenticación de los mismos.

En lo que respecta a las reglas de los modelos de procesos, éstas tienen la misma función y se especifican de igual manera en Wf-ATOMS que en el entorno ATOMS. En definitiva, establecen las relaciones entre la información contextual de los procesos, así como las relaciones de secuenciamiento en la ejecución de los mismos.

Los mecanismos de especificación de los modelos de procesos de Wf-ATOMS son los mismos que los de ATOMS. A saber, los modelos pueden ser especificados de manera declarativa o de manera interactiva.

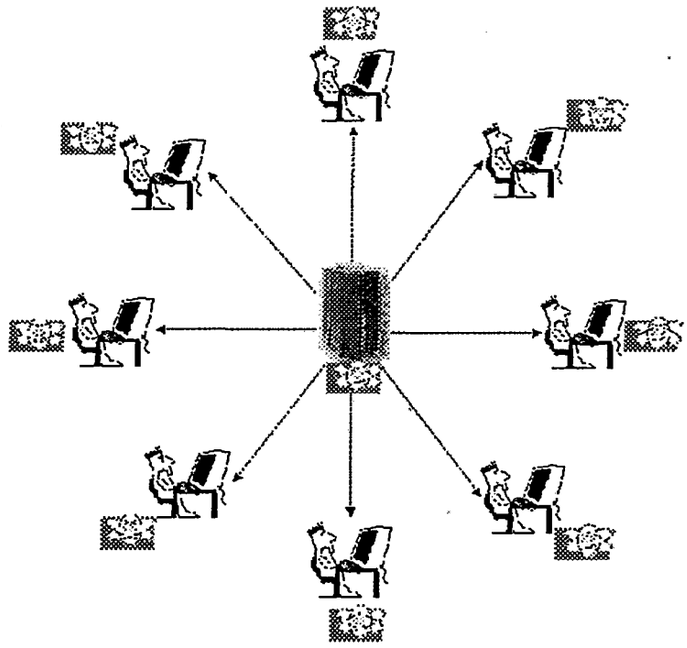

Figura 29: Arquitectura Cliente/Servidor de Wf-ATOMS.

3.5.2 Arquitectura de proceso

Adicionalmente, en Wf-ATOMS se incluye el soporte necesario para la gestión de los flujos de trabajo. La arquitectura de procesamiento de este entorno se basa en un modelo clásico cliente/servidor, como podemos apreciar en la Figura 29. Como en ella vemos, un servidor se comunica con un número indeterminado de clientes con el fin de colaborar en llevar a cabo el procesamiento de un flujo de trabajo.

En cuanto al procesamiento lógico que se lleva a cabo en cada una de las plataformas, éste lo podemos observar detallado en la Figura 30. En la parte superior de esta figura encontramos representada la lógica de servidor del sistema, mientras que en la parte inferior encontramos la lógica correspondiente a los entornos clientes. A continuación se van a describir las componentes de la arquitectura.

Cliente. Los puestos clientes disponen de una herramienta, el Gestor de Procesos de la parte inferior de la Figura 30, mediante la cual, los usuarios pueden comenzar la ejecución de los procesos que deseen si disponen del perfil adecuado. El Gestor de Procesos, uno por cada puesto de proceso cliente, tiene dos labores fundamentales:

- Por un lado, su papel es el de comunicarse con el servidor del sistema, con el fin de centralizar el seguimiento de los flujos de trabajo realizados por los diferentes usuarios. Los usuarios arrancan la ejecución de los flujos de proceso mediante el Gestor de Procesos, que informa al servidor de este hecho. Posteriormente, cada vez que un usuario realiza algún subproceso del flujo, el Gestor de Procesos informará del suceso al servidor.
- Por otro lado, cuando el servidor del sistema decide encargar un trabajo a un determinado usuario, el Gestor de Procesos situado en la máquina de dicho usuario se encarga de

comunicarse con el módulo ATOM de esa máquina, con el fin de poder conocer el estado de realización del proceso.

En resumen, el conjunto de Gestores de Procesos se encarga de la sincronización de los trabajos, permitiéndose así la interrelación entre los diferentes sistemas basados en ATOM y los flujos de procesos establecidos por los administradores. En cuanto a su diseño, la implementación del Gestor de Procesos se ha realizado como una *Herramienta de Valor Añadido* del módulo ATOM de las plataformas clientes.

Servidor. El módulo servidor de Wf-ATOMS, que podemos observar en la parte superior de la Figura 30, proporciona básicamente la posibilidad de seguir y asignar la ejecución de los procesos del flujo de procesos. La lógica de servidor es, en cuanto a la gestión de los modelos de flujos de procesos, idéntica a la de ATOM, con dos únicos matices diferenciadores:

- En primer lugar, el *input* al motor de análisis (en este caso, motor de análisis *de procesos*, en vez *de tareas*) no procede de las interacciones de los usuarios, como sucedía en ATOM. En su lugar, el *input* proviene del Gestor de Procesos del cliente, que informa de qué instancias de flujos de trabajo se desean comenzar y de la evolución los procesos que forman parte de éstos toda vez que su realización ha sido asignada a algún cliente.
- En segundo lugar, la asignación de trabajos pendientes a los distintos usuarios o *roles* ha sido implementada como una *Herramienta de Valor Añadido*, que hace uso de un módulo de gestión de perfiles de usuarios. Este módulo de asignación de procesos pendientes, *Asignación de Procesos*, se encarga de contactar con el *Gestor de Procesos* del cliente adecuado (en función de los *roles* definidos), que es quien se encargará de avisar de nuevo al *Motor de Análisis* del servidor cuando el proceso haya sido realizado.

En resumen, vemos que Wf-ATOMS se trata, básicamente, de la conexión de distintos módulos ATOM, situados en distintas plataformas clientes, con un módulo central, basado también en ATOM y con pequeñas modificaciones, que se encarga del seguimiento y distribución de los trabajos.

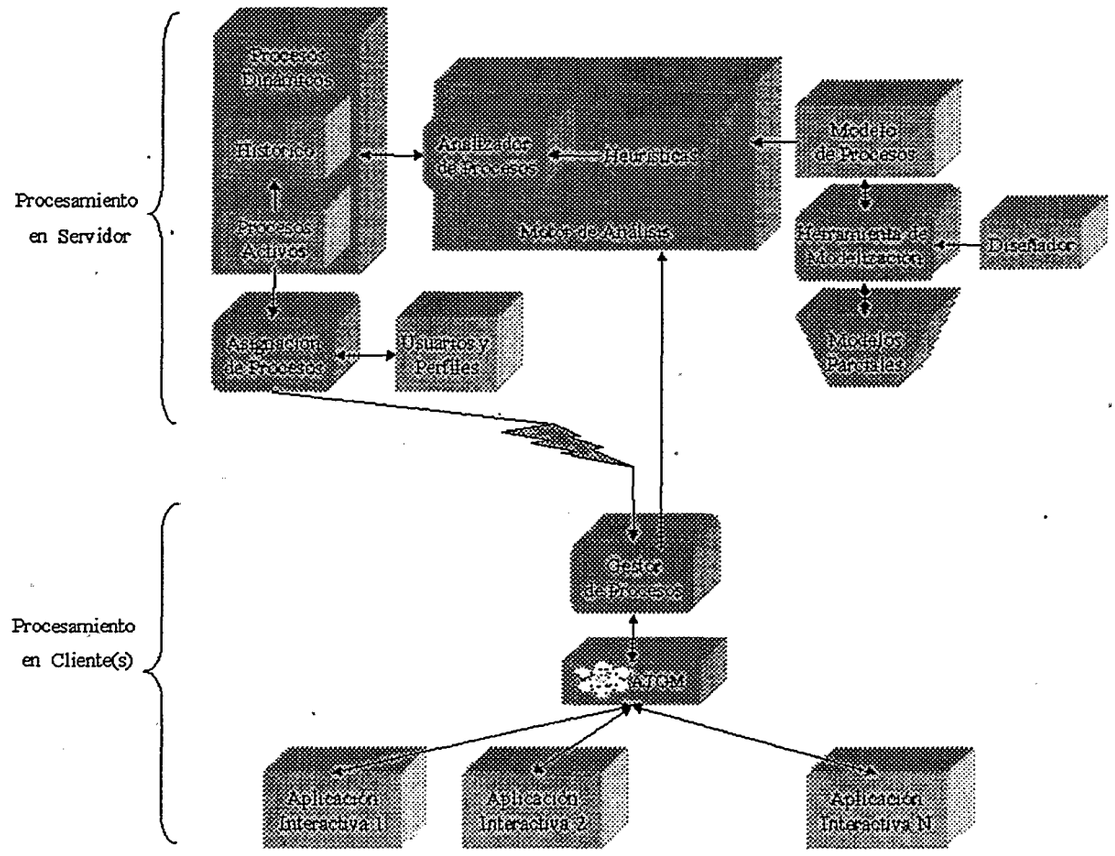

Figura 30: Arquitectura de procesamiento de Wf-ATOMS.

4 Cursos Interactivos de Enseñanza

En el capítulo anterior se ha propuesto una arquitectura y una implementación de un sistema de gestión de tareas de usuario en tiempo de ejecución, capaz de servir como base a un amplio conjunto de servicios de valor añadido orientado a facilitar la interacción entre las aplicaciones interactivas y sus usuarios. En este capítulo veremos, partiendo de la base de dicha arquitectura, cómo puede ofrecerse un entorno de enseñanza de aplicaciones orientado a los usuarios noveles.

Algunos estudios psicológicos sugieren que las explicaciones orientadas a tareas se ajustan mejor a los usuarios noveles que las explicaciones orientadas a componentes, que están más adaptadas a los usuarios expertos. Así, estudios empíricos de conjuntos de documentos, dentro del estudio TAILOR, han mostrado que los dispositivos complejos se describían normalmente utilizando un estilo descriptivo orientado a las componentes de los mismos, mientras que las enciclopedias para niños tendían a estar organizadas de una manera orientada a procesos, proporcionando información de una manera funcional [Paris89].

Uno de los mayores problemas a los que se enfrentan los usuarios noveles cuando se trata de aprender a utilizar las aplicaciones interactivas actuales es el acceso a la ayuda. Los entornos descritos en la Sección 2.1.2 ofrecen una interfaz mínima para la solicitud de ayuda, lo cual fuerza a los usuarios a estar familiarizados con las aplicaciones y crea una barrera de espacial importancia a los usuarios noveles. Esta interfaz es particularmente complicada para los entornos orientados a procesos, aunque tampoco los entornos orientados a componentes permiten un fácil acceso sin obligar a los usuarios a tener cierto grado de familiaridad con el sistema. Básicamente, la interfaz en los entornos orientados a procesos permite a los usuarios seleccionar de una lista de tareas qué es lo que quieren aprender a realizar, lo cual resulta poco práctico, fundamentalmente cuando la guía va dirigida a este grupo de usuarios sin experiencia previa. Esto es consecuencia de que los usuarios pertenecientes a este perfil precisan de un entorno más dominante que les haga propuestas de qué es lo que deben practicar para aprender a utilizar sus aplicaciones.

En esta tesis se propone un entorno en el que la guía los usuarios noveles viene facilitada utilizando una interfaz en la que los cursos proponen las actividades a realizar, y en el que la guía proporcionada está facilitada de una manera funcional, orientada a tareas. Este entorno, CACTUS [García99b, García99c, García2000a, García2000b] (acrónimo inglés de *Creating Application Courses about Tasks Using Scenarios -Creación de Cursos para Aplicaciones orientados a Tareas Utilizando Escenarios*), incorpora servicios orientados tanto a los diseñadores de los cursos como a los usuarios de los mismos.

Desde el punto de vista de los desarrolladores de cursos tutores, CACTUS permite el desarrollo interactivo de tutores para aplicaciones. De este modo, libera a los diseñadores de los cursos de gran parte del trabajo que supone el desarrollo de dichos tutores. Para ello, CACTUS integra servicios para crear, modificar, probar y depurar los cursos tutores.

Desde el punto de vista de los alumnos, CACTUS permite a los usuarios comprender los contenidos de dichos cursos, mediante la ejecución de los mismos. Durante estas ejecuciones, el entorno proporciona a los usuarios explicaciones orientadas a tareas de manera dinámica, y es capaz de hacer un seguimiento de lo que los alumnos realizan, pudiéndose actuar en consecuencia.

Los cursos tutores basados en este entorno están estructurados en un conjunto de unidades pedagógicas. Cada una de estas unidades se encarga de enseñar un cierto conjunto de tareas que se encuentran relacionadas por algún criterio particular, elegido por el diseñador del material. El criterio mediante el cual determinados procesos son asociados entre sí dentro de una unidad pedagógica debe basarse en un conocimiento previo de las funcionalidades que tiene el empleo de la aplicación en cada dominio final particular. Así, las posibilidades que ofrece a un usuario final una aplicación de hoja de cálculo no son las mismas para un administrativo que la usará para reflejar su contabilidad, que para un matemático que podría emplearla para realizar los más sofisticados cálculos. Y, como consecuencia, el material que deberá incluirse en cada unidad pedagógica deberá ser distinto, quedando a responsabilidad del desarrollador de los cursos tutores la inclusión del material más adecuado en cada caso y su estructuración.

Pese a que los cursos de CACTUS se pueden especificar utilizando un lenguaje de especificación propio, como se verá en la Sección 4.4.3, el entorno permite a los diseñadores construir y modificar los cursos de manera completamente interactiva. Para ello, CACTUS se apoya en la metáfora de representar los tutores como si fueran libros de texto, asociando las partes más importantes del curso tutor con las partes más representativas de los libros de texto. De este modo, la generación de un curso tutor se asimila a la creación de un libro interactivo, mientras que el aprendizaje se asimila al seguimiento de los contenidos del mismo.

A lo largo de esta sección, se describirá detalladamente la arquitectura en la que se apoya CACTUS para permitir la especificación y la ejecución de los cursos tutores.

4.1 Arquitectura

La arquitectura de gestión de cursos tutores en la que se basa CACTUS está apoyada en dos módulos, como se muestra en la Figura 31. El primero de ellos, DARTS, se encarga de enseñar a los usuarios a realizar determinadas tareas de las aplicaciones, mientras que el segundo, el *Módulo de Ejecución de Escenarios*, se encarga de preparar *escenarios* adecuados para llevar a cabo la enseñanza de las tareas propuestas. Ambos módulos se apoyan en la arquitectura de gestión de tareas de usuario ATOM.

DARTS es un sistema tutor que utiliza los modelos de tareas de usuario de ATOMS para generar las explicaciones proporcionadas a los usuarios. DARTS ofrece a los usuarios enseñanza dinámica, actualizando los mensajes de ayuda a las necesidades de los usuarios, según lo que vayan haciendo en cada momento. Este sistema ofrece *feedback* contextualizado, como mensajes adaptados a la labor que se está practicando y a las acciones realizadas, o la muestra de respuesta gráfica adaptada al entorno de trabajo mediante el parpadeo de objetos gráficos de la aplicación. DARTS puede actuar con varios grados de flexibilidad, filtrando las acciones incorrectas de los alumnos y forzándoles a seguir alguno de los caminos correctos, dependiendo del grado de flexibilidad establecido. En la Sección 4.2 se describirán detalladamente las características concernientes a la arquitectura de DARTS y a su implementación.

Por su parte, el *Módulo de Ejecución de Escenarios* prepara los contextos apropiados para llevar a cabo la enseñanza de las tareas de los cursos. Un *escenario* es una descripción procedural de un proceso, y estas descripciones son interpretadas por el *Módulo de Ejecución de Escenarios* con el fin de materializarlas, utilizando animaciones, en contextos apropiados para la enseñanza y práctica de tareas. Por ejemplo, supongamos que quisiéramos enseñar, en una aplicación de diseño gráfico, a pasar un diseño de 2D a 3D. En ese caso, sería razonable el preparar un *escenario*

que enseñara a los alumnos la creación del diseño de un plano de una cocina, de manera que se preparara un contexto adecuado para proceder a explicarle cómo pasar el diseño de 2D a 3D. Tanto la estructura de los *escenarios* como el *Módulo de Ejecución de Escenarios* serán descritos en la Sección 4.3.

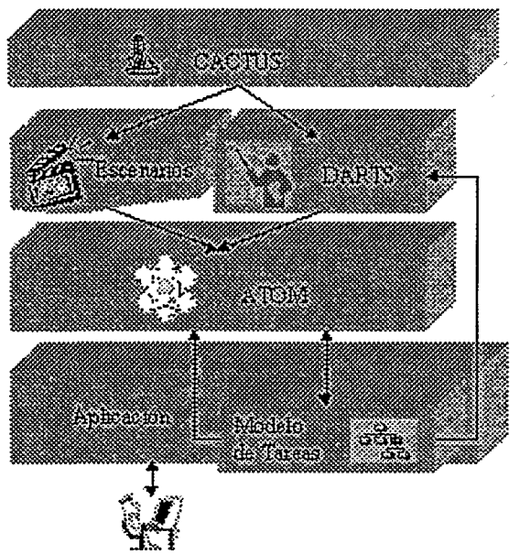

Figura 31: Arquitectura del entorno de gestión de cursos tutores.

Finalmente, en la Sección 4.4 se describirá detalladamente el funcionamiento de la interfaz CACTUS para la gestión de los cursos interactivos de enseñanza.

4.2 Enseñanza de Tareas: DARTS

DARTS (acrónimo inglés de *Dynamic and Active Real-software based Tutoring System–Sistema Tutor Dinámico y Activo basado en la propia Aplicación*), es un sistema generador de enseñanza a partir de los modelos de tareas de las aplicaciones basadas en la arquitectura ATOM. DARTS ofrece dos tipos de información a sus usuarios. Por un lado, de manera similar a como hace el trabajo de Pangoli y Paternó descrito en la Sección 2.1.2.4, explica tareas complejas en términos de tareas más simples, de modo que las explicaciones son mucho más fáciles de comprender que simples secuencias de acciones elementales como pulsaciones de ratón o de teclado, como ocurre en la mayor parte de los sistemas de guía convencionales. Por otro lado, de manera similar a como hemos visto que sucede con H3 en la Sección 2.1.2.2, DARTS ofrece información gráfica dependiente del contexto acerca de los objetos de las ventanas con los que los usuarios tienen que interaccionar para conseguir los objetivos representados por las distintas tareas, así como la manera de realizar dicha interacción. El hecho de que DARTS actúe de manera sensible al contexto es consecuencia de que la arquitectura ATOM incorpora el *Motor de Análisis* que hace un seguimiento continuo de la evolución de las tareas que los usuarios realizan, por lo que es capaz de ofrecer explicaciones adaptadas al estado de realización de las tareas de los usuarios. Por otra parte, el hecho de utilizar modelos jerárquicos de tareas permite al entorno de enseñanza DARTS ofrecer un mayor grado de abstracción que los sistemas que se basan en flujos planos de interacciones básicas.

Los objetivos obtenidos por el entorno DARTS son de gran importancia. Por un lado, el tipo de guía que los usuarios reciben resulta ser mucho más potente que aquella que se proporciona habitualmente. Y, por otro lado, las ventajas obtenidas se consiguen a precio muy bajo en comparación con los costes que tiene la obtención de la componente de guía en otros entornos, ya que los modelos de tareas pueden ser especificados de manera interactiva o mediante un sencillo lenguaje de especificación declarativo. La cantidad de pequeños detalles y diferentes aspectos que deben ser tenidos en cuenta a la hora de proporcionar la componente de guía de una aplicación es tan enorme que obtener manualmente los mismos resultados que se obtienen automáticamente mediante DARTS, sin apoyarse en ninguna herramienta externa, es una labor tremendamente complicada. Además, mediante DARTS es posible obtener la componente de guía de todas aquellas aplicaciones para las que se especifique el modelo de sus tareas, y no sólo de una aplicación. Hablamos, pues, de una herramienta general.

Como hemos visto, DARTS, como parte integrante de un entorno tutor más amplio, en nuestro caso CACTUS, recibe peticiones por parte de éste de enseñar tareas aisladas. Por tanto, será el sistema tutor quien se encargue de encauzar adecuadamente las solicitudes de enseñanza de tareas a DARTS, de modo que los cursos engloben de manera correcta tanto sesiones de enseñanza de tareas como, probablemente, otro tipo de servicios. En esta sección veremos cómo el servicio de enseñanza DARTS genera dinámicamente la información necesaria para enseñar a realizar una determinada tarea que le haya sido solicitada.

Una vez se le ha solicitado a DARTS que enseñe una tarea, hay dos tipos de información de enseñanza que el usuario puede obtener. Por un lado, el usuario puede navegar a lo largo de la jerarquía de subtareas de la tarea a enseñar, obteniendo información descriptiva sobre ellas. En adelante, denominaremos *tarea activa de enseñanza* a aquella tarea que sea el foco central de la navegación del usuario. Por otro lado, el usuario puede obtener enseñanza acerca de las acciones que debe realizar, de manera que esta información es actualizada dinámicamente conforme el usuario realiza la tarea explicada. Según esto, existe un *conjunto de tareas pendientes* que representa aquellas acciones que puede realizar el usuario en cada momento dado para llevar a cabo la tarea que se está enseñando.

Cuando la *tarea activa de enseñanza* no pertenece al *conjunto de tareas pendientes*, decimos que DARTS se encuentra en *modo descriptivo*, y que los mensajes de guía que proporciona son *mensajes de descripción de tarea* o *mensajes de acción*. En este tipo de indicaciones, los mensajes que el sistema genera ofrecen al usuario información acerca de cual es el objetivo de dicha *tarea activa de enseñanza* y de su descomposición jerárquica. En el caso contrario, decimos que el sistema se encuentra en el *modo activo*, y en este caso los mensajes que el sistema genera facilitan al usuario información sobre las acciones a realizar. Esta clasificación de modos, tipos de mensajes e información facilitada en cada modo viene reflejada en la tabla de la Figura 32.

Por omisión, cuando DARTS se encuentra ofreciendo enseñanza sobre una tarea, no facilita información acerca de las tareas de nivel superior a dicha tarea, en el caso de que existieran. Es decir, el contexto de la explicación será dicha tarea y sus correspondientes subtareas, no sus *supertareas*. Esta política tiene por objetivo el evitar la proliferación de mensajes largos, siguiendo el *principio del manual mínimo* (*minimal manual principle: "the smaller the manual, the better"* [Carroll87b]), que propone proporcionar a los usuarios noveles una cantidad de información mínima, con el fin de no distraerles de la consecución de sus objetivos. De este modo, se evita el mostrar grandes ventanas de mensajes, lo que haría de DARTS un entorno poco confortable de utilizar, como explicamos en la Sección 2.1.1.2 al hablar del *problema de*

visualización. Sin embargo, los usuarios podrán obtener bajo demanda una mayor cantidad de información contextual acerca de una subtarea, de manera que se les muestren más supertareas a partir de la *tarea activa de enseñanza*.

Condición	Modo	Mensajes	Información
Tarea activa ∈ Tareas pendientes	Activo	Activos	Siguientes Tareas Pendientes
Tarea activa ∉ Tareas pendientes	Descriptivo	Descriptivos	Descripción de Tarea Activa
		De Acción	Descomposición jerárquica de Tarea Activa

Figura 32: Modos de funcionamiento y mensajes de DARTS.

A lo largo de esta sección se va a explicar en detalle el tipo de enseñanza que proporciona DARTS, para lo que se utilizará el ejemplo de la tarea *"Añadir una nueva asignación de docencia"* (*CrearNuevaAsignación*), de la aplicación *Schoodule* ya presentado anteriormente y cuya descomposición jerárquica se mostró en la Figura 19. Para mostrar cómo el sistema es capaz de proporcionar ayuda que tenga en cuenta valores asociados a los parámetros de las tareas, haremos que la tarea mencionada se enseñe teniendo como restricción que el valor de su parámetro *nombre* sea *"Mary Brown"*. En primer lugar se describirá brevemente la interfaz que DARTS proporciona a los alumnos. Posteriormente, y mediante el ejemplo, se explicarán las facilidades proporcionadas para que los alumnos puedan explorar la manera en que las tareas están estructuradas, y para que el sistema ofrezca guía dinámica mientras los alumnos realizan las tareas. Tras esta descripción, se tratarán aspectos más específicos, como la arquitectura del entorno de enseñanza DARTS y cuestiones relacionadas con la generación de los mensajes de guía.

4.2.1 Comunicación con el Usuario

Cuando a DARTS se le solicita que enseñe a realizar una tarea como *CrearNuevaAsignación*, aparece la ventana de mensajes de DARTS, como puede verse en la Figura 33, mostrando información básica sobre la tarea*.

* Como el lector puede apreciar, tanto *Schoodule* como DARTS están en inglés, por lo que en adelante, con el fin de no entremezclar ambos idiomas, se ofrecerán transcripciones escritas de los ejemplos en vez de mediante pantallas capturadas.

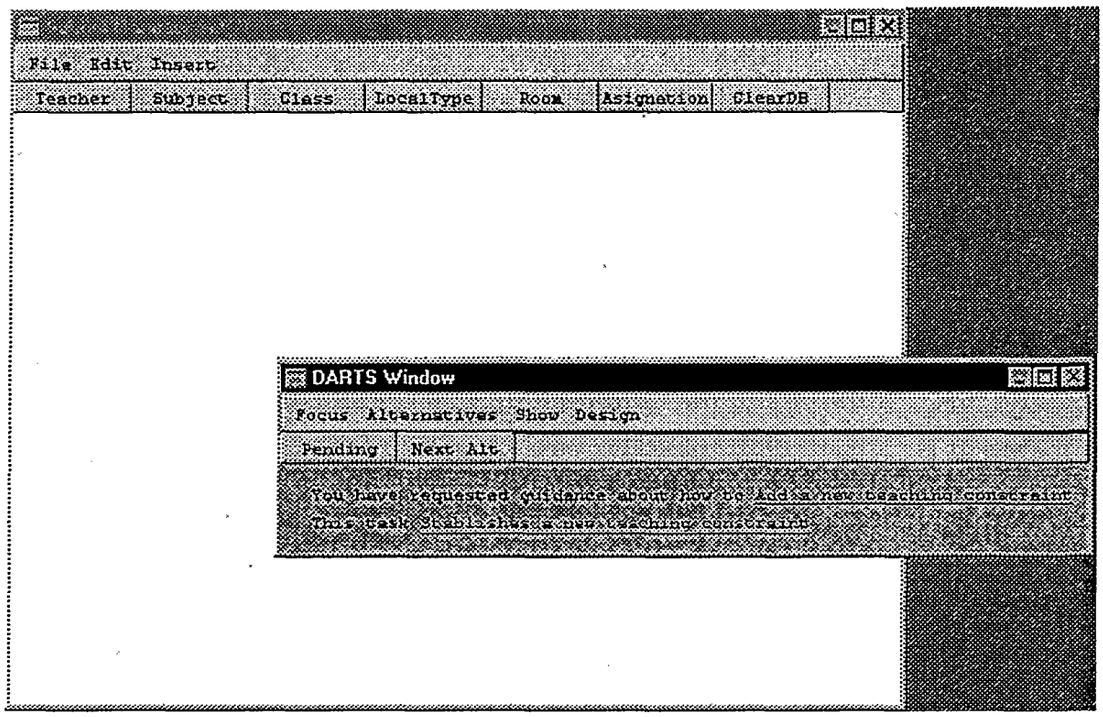

Figura 33: Guía inicial para la tarea *"Añadir una nueva asignación de docencia"*.

En este ejemplo, se muestra el siguiente *mensaje de descripción de tarea*†:

† Con el objetivo de permitir al lector de esta tesis diferenciar la procedencia de las distintas partes de las explicaciones ofrecidas, el formato de texto de los ejemplos que se darán a continuación se ajusta a las siguientes reglas:

- Textos generados por DARTS, dependientes o no del contexto: en letra **negrita**.
- Textos provenientes, directa o cuasi directamente, del contenido del modelo de tareas: en letra *cursiva*.
- Si, además, están *subrayados* y con la letra en azul, es que provienen del modelo de tareas e incorporan un hiperenlace.
- Textos elaborados por DARTS a partir de información implícita de los modelos de tareas: en letra subrayada.

Ha solicitado enseñanza sobre cómo *Añadir una nueva asignación de docencia*

Esta tarea *Establece una nueva restricción de docencia*

La ventana de mensajes de DARTS incluye únicamente mensajes simples, como el anterior, junto a dos botones. El primero de ellos, el botón *Pendientes*, hace que DARTS pase al *modo activo*, es decir, haga que su centro de atención, la *tarea activa de enseñanza*, sea alguna de las tareas del *conjunto de tareas pendientes*. El segundo botón, *Siguiente Alternativa* tiene por objetivo el simplificar la navegación entre las distintas opciones disponibles en un momento dado.

Los usuarios también pueden modificar cuál es la *tarea activa de enseñanza* mediante el uso del elemento de menú *Foco/Ir a Siguiente Tarea*, mostrar una mayor o menor cantidad de información contextual mediante las opciones *Foco/Mostrar Contexto* y *Foco/Ocultar Contexto*, y explorar las maneras alternativas de realizar alguna de las *tareas pendientes* mediante las opciones del menú *Alternativas* (*Todas*, *Siguiente Alternativa* y *Alternativa Previa*).

La opción del menú *Ver* llamada *Mostrar Jerarquía de Tarea* abre una ventana que muestra un árbol que representa la jerarquía completa de la tarea que se está explicando. Como podemos ver en la Figura 34, esta representación jerárquica muestra el árbol de la tarea para la que se va a proporcionar guía, proporcionando información sobre el estado de la ejecución de la tarea por parte del usuario. Además de esta información dinámica que muestra el estado de cada tarea (*realizada*, *incompleta*, *pendiente* o *inaccesible*), en este árbol se proporciona también información acerca de valores que debe tener asociados la información contextual de las distintas subtareas (es decir, valores de parámetros), relaciones de secuenciamiento entre las tareas hijas de cada tarea (mediante los arcos que aparecen debajo de las ramas que representan las distintas descomposiciones jerárquicas), e información relativa a la posible multiplicidad u opcionalidad de las distintas tareas (mediante lazos en las tareas para las tareas múltiples y tareas rodeadas por marcos discontinuos para las tareas opcionales). Como vemos, además, cada nodo del árbol aparece subrayado, indicándose que contiene un hiperenlace, cuya función explicaremos posteriormente.

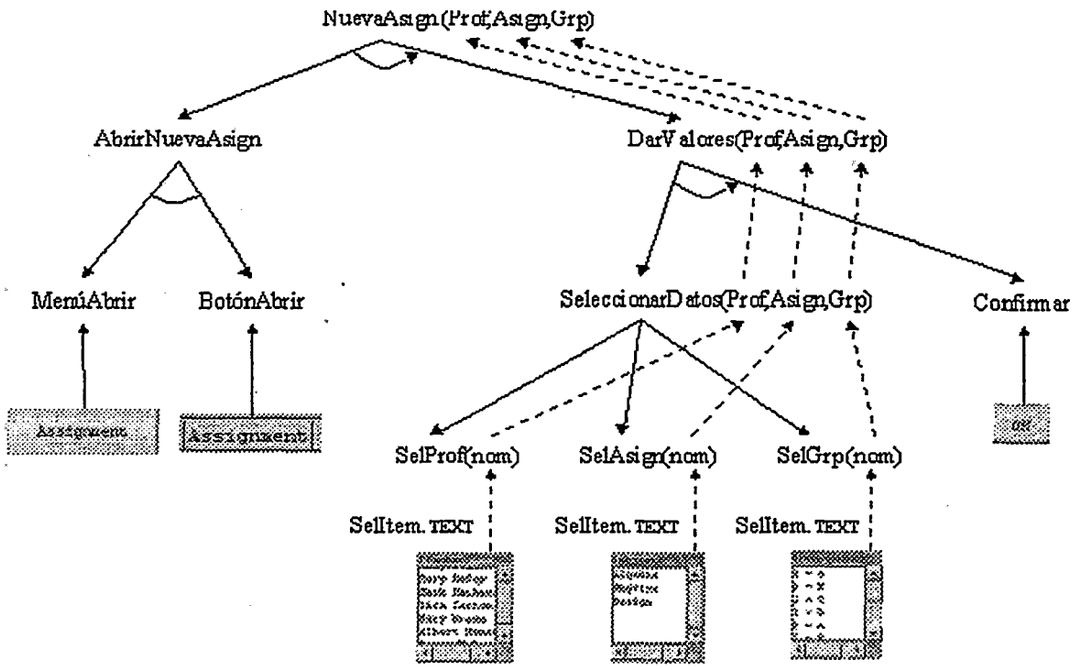

Figura 34: Representación jerárquica inicial de la tarea que se está enseñando.

Además de los botones, los usuarios también pueden interaccionar directamente tanto con la ventana de mensajes de DARTS como con la ventana que muestra la jerarquía de la tarea. En este caso, los alumnos pueden seguir los distintos hiperenlaces que aparecen durante las explicaciones proporcionadas, como se explicará en la siguiente subsección.

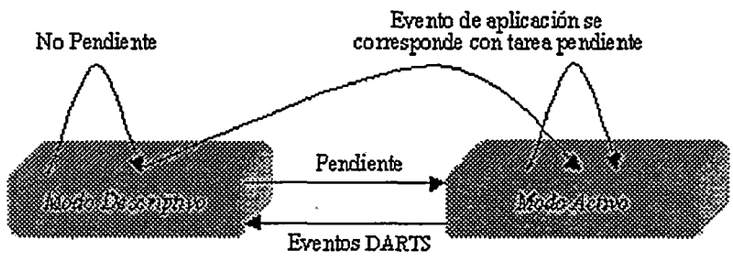

Figura 35: Cambios de *modo* de funcionamiento de DARTS.

Todos los eventos generados en la ventana de mensajes de DARTS se denominan *eventos de DARTS*, mientras que aquellos generados por el usuario interaccionando con las ventanas de la aplicación se denominan *eventos de la aplicación*. Los *eventos de la aplicación* que el sistema reconozca como correspondientes a alguna de las tareas del *conjunto de tareas pendientes*, así

como el botón *Pendiente* de la ventana de DARTS, pondrán a DARTS en *modo activo*, mientras que, en general, los eventos de DARTS situarán al sistema en *modo descriptivo*. Por su parte, los mandatos relativos al contexto (*Mostrar Contexto/Ocultar Contexto*) y a las maneras alternativas de realizar las tareas (*Todas*, *Siguiente Alternativa* y *Alternativa Previa*) no tienen ningún efecto sobre el modo de DARTS. Los cambios de *modo* de funcionamiento del sistema pueden verse representados en la Figura 35.

4.2.2 Comportamiento Estático: Navegación en *Modo Descriptivo*

Cuando el mensaje inicial aparece, la tarea activa de enseñanza es la tarea de la aplicación Añadir una nueva asignación de docencia, y el conjunto de tareas pendientes está compuesto por las tareas atómicas Seleccionar el elemento de menú Nueva Asignación y Seleccionar el botón Asignación, ambas tareas relacionadas mediante el operador de secuenciamiento Xor. Por ello, como la tarea activa de enseñanza no se encuentra dentro del conjunto de tareas pendientes, DARTS se encuentra en modo descriptivo. Veamos ahora cómo el usuario puede navegar a partir de aquí e ir obteniendo información sobre las diferentes subtareas de la tarea citada.

Como hemos visto en el ejemplo anterior, tanto los nombres de tareas como sus descripciones contienen un hiperenlace. Los usuarios pueden pinchar en estos hiperenlaces para alternar entre mensajes de *descripción de tarea* y *mensajes de acción*, o bien para modificar la *tarea activa de enseñanza*. De esta manera, al pulsar sobre el enlace de la *tarea activa de enseñanza* cuando lo que se muestra es un *mensaje de descripción de tarea*, DARTS muestra el correspondiente *mensaje de acción* (todavía un mensaje del *modo descriptivo*) para la misma *tarea activa de enseñanza*, obtenido mediante la substitución de la segunda parte del mensaje previo por la explicación de las subtareas de la tarea inicial, como se muestra a continuación:

> **Ha solicitado enseñanza sobre cómo** *Añadir una nueva asignación de docencia*
>
> **Para hacer esta tarea, usted tiene que:**
>
> *Abrir el cuadro de diálogo de Nueva Asignación*
>
> **y después**
>
> *Proporcionar los valores que tendrá la relación*

La información relativa a las distintas subtareas de una tarea dada es obtenida a partir de la información presente en las reglas del modelo de tareas de usuario. Además, dependiendo del tipo de relación de secuenciamiento temporal existente entre las subtareas en una regla, la explicación proporcionada descomponiendo el proceso en subprocesos incluye el nexo temporal adecuado (**o bien**, **y después**, **y**).

Sin embargo, si el usuario sigue un hiperenlace asociado a una tarea distinta de la *tarea actual de enseñanza*, en ese caso el sistema lo que hace es actualizar la *tarea actual de enseñanza*, dándole el valor de la tarea seleccionada. Como consecuencia de ello, lo que hace el sistema es mostrar el *mensaje de descripción de tarea* de la nueva *tarea activa de enseñanza*. Como vemos, la alternancia entre mensajes de *descripción de tarea* y *mensajes de acción*, ambos obtenidos en *modo descriptivo*, y la modificación de la *tarea activa de enseñanza*, se hace mediante el uso de los hiperenlaces. Por ejemplo, si el usuario utiliza el hiperenlace de la primera subtarea, el mensaje obtenido será:

> **Ha solicitado enseñanza sobre cómo** *Abrir el cuadro de diálogo de Nueva Asignación*

Esta tarea *Muestra el cuadro de diálogo que permite añadir una nueva asignación*

En cualquier momento los usuarios pueden pedir más información contextual utilizando la opción de menú *Foco/Mostrar Contexto*, y se le mostrará un nivel más alto de la jerarquía. El mensaje que se encuentra a continuación muestra el *mensaje de acción* contextualizado cuando la *tarea activa de enseñanza* es la misma que en el mensaje anterior (es decir, tras haber seguido el hiperenlace asociado a la *tarea activa de enseñanza* y, tras ello, seleccionar *Foco/Mostrar Contexto*):

Ha solicitado enseñanza sobre cómo *Añadir una nueva asignación de docencia*

Para ello debe *Abrir el cuadro de diálogo de Nueva Asignación*

Para hacer esta tarea, usted tiene que:

Seleccionar el elemento de menú Nueva Asignación

o bien

Pulsar el botón de Nueva Asignación

La opción *Foco/Mostrar Contexto* y la posibilidad de seguir los hiperenlaces generados por DARTS, permite la navegación completa por el árbol jerárquico de la tarea para la cual se solicitó inicialmente la guía. Estas facilidades proporcionadas por DARTS permiten al usuario obtener *mensajes de descripción de tarea* y *mensajes de acción* para cualquier subtarea, junto a un contexto que puede incluir un número variable de *supertareas*. La navegación se complementa mediante la opción *Foco/Ir a Siguiente Tarea*, que establece como *tarea activa de enseñanza* a la siguiente subtarea *hermana* de la actual *tarea activa de enseñanza* (es decir, aquella subtarea que está situada a continuación de la *tarea activa de enseñanza* según la *regla* a la que se está haciendo referencia).

4.2.3 Comportamiento Dinámico: Actualización en *Modo Activo*

Si el usuario comienza la realización de la tarea para la cual se está ofreciendo guía, o bien si establece como *tarea activa de enseñanza* alguna tarea perteneciente al conjunto de tareas *pendientes*, como podría suceder si se siguiera el hiperenlace asociado al nombre o a la explicación de la tarea *Pulsar el botón Asignación* en el mensaje anterior, entonces DARTS cambia su modo de funcionamiento, entrando en *modo* activo. Los *mensajes* activos proporcionados en este *modo* son similares a los *mensajes de acción* proporcionados en el *modo* descriptivo, excepto por el hecho de que los primeros indican las acciones que se están realizando y la siguiente o siguientes acciones que deben realizarse. En este sentido, este *modo* incorpora la componente dinámica de la que otros entornos carecen, y que permite a los alumnos recibir información dependiente del contexto de sus acciones.

Así, si tras obtenerse la información del último ejemplo mostrado, el usuario sigue el hiperenlace asociado a la tarea *Seleccionar el elemento de menú Nueva Asignación* o a su descripción, la información que se proporciona al usuario es la siguiente:

Ha solicitado enseñanza sobre cómo *Añadir una nueva asignación de docencia*

Primero, usted tiene que *Abrir el cuadro de diálogo de Nueva Asignación*

Para hacer esta tarea, usted tiene que *Seleccionar el elemento de menú Nueva Asignación*, pulsando el botón izquierdo del ratón sobre el objeto que está parpadeando en rojo

Como vemos, cuando se encuentra en *modo activo*, el sistema, además, incluye una explicación de cómo comenzar las distintas tareas que están pendientes. Dado que puede haber varias tareas pendientes, el usuario puede elegir cuál de ellas quiere ejecutar. Esta elección que realiza el usuario hace que DARTS oriente sus siguientes explicaciones hacia aquellas tareas relacionadas con la realizada por el usuario, dejando en un segundo plano las demás. Como ya se ha mencionado, los elementos del menú *Alternativas* y el botón *Siguiente Alternativa* permiten al usuario navegar a través de las distintas alternativas que se pueden llevar a cabo. Cuando existe alguna alternativa más a la que está siendo explicada, el botón *Siguiente Alternativa* aparece parpadeando, de manera que aunque el usuario sólo reciba la explicación de una de las tareas pendientes, sabe que puede obtener información de las restantes. De este modo, si a continuación el usuario pulsa este botón o selecciona la opción de menú *Alternativa/ Siguiente Alternativa*, el mensaje que DARTS mostraría sería el siguiente:

> **Ha solicitado enseñanza sobre cómo** *Añadir una nueva asignación de docencia*
>
> **Primero, usted tiene que** *Abrir el cuadro de diálogo de Nueva Asignación*
>
> **Para hacer esta tarea, usted tiene que** *Pulsar el botón de Nueva Asignación*, pulsando el botón izquierdo del ratón sobre el objeto que está parpadeando en rojo

En los casos en los que el sistema encuentra reglas que incorporan los operadores de secuenciamiento temporal *Y'* y *'Xor'*, DARTS permiten generar explicaciones recalcando este hecho y listando las distintas posibilidades disponibles. Así, si a continuación el usuario seleccionara *Alternativas/Todas,* se obtendrá una lista de todas las distintas opciones disponibles:

> **Ha solicitado enseñanza sobre cómo** *Añadir una nueva asignación de docencia*
>
> **Primero, usted tiene** *Abrir el cuadro de diálogo de Nueva Asignación*
>
> **Para hacer esta tarea, usted tiene que** *Pulsar el botón de Nueva Asignación*, pulsando el botón izquierdo del ratón sobre el objeto que está parpadeando en rojo
>
> **o bien**
>
> *Seleccionar la opción de menú Nueva Asignación*, pulsando el botón izquierdo del ratón sobre el objeto que está parpadeando en verde

Cuando se pasa al *modo activo*, DARTS muestra el contexto de la tarea formado por las supertareas de la *tarea actual de guía* que no estaban presentes en el contexto previo. En este ejemplo, una vez el usuario ha terminado la realización de la primera subtarea (*Abrir el cuadro de diálogo de Nueva Asignación*), independientemente del procedimiento utilizado para ello (menú o botón), la ventana de DARTS muestra el siguiente *mensaje activo*:

> **Está realizando la tarea** *Añadir una nueva asignación de docencia*. **Ha terminado la tarea** *Abrir el cuadro de diálogo de Nueva Asignación*
>
> **Finalmente, usted tiene que** *Proporcionar los valores que tendrá la relación*..
>
> **Primero, usted tiene que** *Establecer las componentes de la relación de docencia*
>
> **Primero, para hacer esta tarea usted tiene que** *Seleccionar el profesor de la lista*, pulsando el botón izquierdo del ratón el objeto que está parpadeando en color rojo. *Esta tarea puede hacerse más de una vez.*

Como podemos apreciar leyendo el mensaje que DARTS proporciona al usuario, vemos que el sistema ofrece *feedback* visual, sobre la aplicación, indicando con precisión dónde tiene que interaccionar el usuario. En este caso, como podemos apreciar en la Figura 36, el campo de la lista de profesores que contiene el elemento de etiqueta *"Mary Brown"* aparece parpadeando con fondo de color rojo, de modo que el usuario sabe a qué objeto gráfico se refiere exactamente la explicación textual ofrecida. Hasta este momento que se ha realizado alguna tarea, el sistema no había tenido que actualizar la información mostrada en la ventana que representa la jerarquía de la tarea que se está enseñando. Tras la realización de esta primera tarea atómica, DARTS actualiza automáticamente la información de dicha ventana, proporcionando así *feedback* sobre qué partes de la tarea permanecen activas, qué partes se han realizado, qué partes no se pueden realizar en este momento y qué partes deben realizarse a continuación. El estado que en este momento tiene esta ventana es el que aparece en la Figura 37.

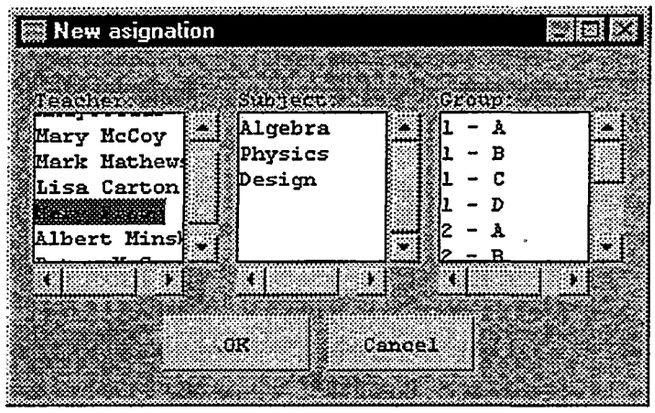

Figura 36: DARTS ofrece *feedback* visual de sobre el profesor a seleccionar.

Como podemos apreciar leyendo el mensaje que DARTS proporciona al usuario, vemos que el sistema ofrece *feedback* visual, sobre la aplicación, indicando con precisión dónde tiene que interaccionar el usuario. En este caso, como podemos apreciar en la Figura 36, el campo de la lista de profesores que contiene el elemento de etiqueta *"Mary Brown"* aparece parpadeando con fondo de color rojo, de modo que el usuario sabe a qué objeto gráfico se refiere exactamente la explicación textual ofrecida. Hasta este momento que se ha realizado alguna tarea, el sistema no había tenido que actualizar la información mostrada en la ventana que representa la jerarquía de la tarea que se está enseñando. Tras la realización de esta primera tarea atómica, DARTS actualiza automáticamente la información de dicha ventana, proporcionando así *feedback* sobre qué partes de la tarea permanecen activas, qué partes se han realizado, qué partes no se pueden realizar en este momento y qué partes deben realizarse a continuación. El estado que en este momento tiene esta ventana es el que aparece en la Figura 37.

En el mensaje ofrecido anteriormente es donde se aprecia el papel que juega la información relativa a los parámetros de las tareas en la generación de los mensajes. Para la generación de los mensajes, DARTS efectúa el procedimiento de propagación de los valores de los parámetros hacia abajo en la jerarquía de tareas que ya fue explicado en la Sección 3.4 al tratar la implementación del módulo de *Emulación* de ATOMS, y cuyo objetivo fundamental era la obtención de valores relevantes para parámetros de tareas de bajo nivel a partir de valores de parámetros para tareas de

alto nivel. Así, el valor "*Mary Brown*" correspondiente al parámetro *nombre* de la tarea *CrearNuevaAsignación*, es transformado al valor "*Mary Brown*" correspondiente al texto del objeto que se ha de seleccionar de la lista. Ésto es después utilizado por DARTS para hacer parpadear únicamente dicho elemento de la lista, y no los demás, como sucede con el resto de tareas asociadas a la selección de un elemento de una lista, como veremos a continuación.

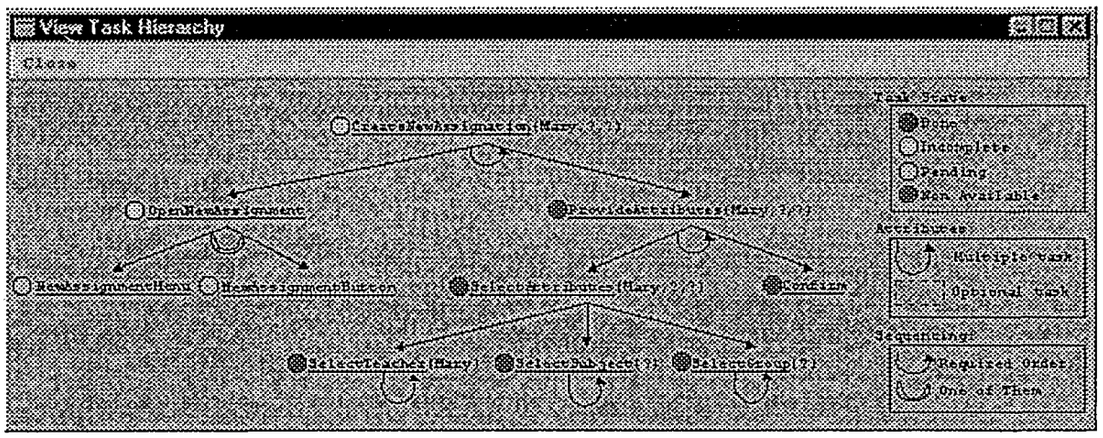

Figura 37: Panel de DARTS mostrando en verde las tareas pendientes.

Dado que nos encontramos ante un nuevo caso en el que existe la posibilidad de realizar varias *tareas pendientes*, DARTS hace parpadear el botón con etiqueta *Siguiente Alternativa*, indicando este hecho. Así, pulsando este botón o seleccionando *Alternativas/ Siguiente Alternativa*, obtenemos la descripción de cómo realizar la siguiente de las acciones disponibles:

> **Está realizando la tarea** *Añadir una nueva asignación de docencia* **Ha terminado la tarea** *Abrir el cuadro de diálogo de Nueva Asignación*
>
> **Finalmente, usted tiene que** *Proporcionar los valores que tendrá la relación.*
>
> **Primero, usted tiene que** *Establecer las componentes de la relación de docencia*
>
> **Primero, para hacer esta tarea usted tiene que** *Seleccionar la asignatura de la lista*, pulsando el botón izquierdo del ratón sobre alguno de los objetos que están parpadeando en color rojo. *Esta tarea puede hacerse más de una vez.*

Y, dado que aún queda otra *tarea pendiente* más, el usuario podrá repetir el proceso anterior, solicitando al sistema que la explique:

> **Está realizando la tarea** *Añadir una nueva asignación de docencia.* **Ha terminado la tarea** *Abrir el cuadro de diálogo de Nueva Asignación.*
>
> **Finalmente, usted tiene que** *Proporcionar los valores que tendrá la relación.*.
>
> **Primero, usted tiene que** *Establecer las componentes de la relación de docencia*
>
> **Primero, para hacer esta tarea usted tiene que** *Seleccionar el grupo de la lista*, pulsando el botón izquierdo del ratón sobre alguno de los objetos que están parpadeando en color rojo. *Esta tarea puede hacerse más de una vez.*

Adicionalmente, el usuario puede pedir que se le ofrezca, a la vez, la información referente a todas las tareas pendientes que puede realizar en un momento dado, mediante la opción de menú *Alternativas/Todas*. A continuación podemos ver los efectos de seleccionar dicha opción:

> **Está realizando la tarea** *Añadir una nueva asignación de docencia*. **Ha terminado la tarea** *Abrir el cuadro de diálogo de Nueva Asignación*
>
> **Finalmente, usted tiene que** *Proporcionar los valores que tendrá la relación*.
>
> **Primero, usted tiene que** *Establecer las componentes de la relación de docencia*
>
> **Para hacer esta tarea, usted tiene que** *Seleccionar el profesor de la lista*, pulsando el botón izquierdo del ratón sobre el objeto que está parpadeando en color rojo. *Esta tarea puede hacerse más de una vez.*
>
> **y**
>
> *Seleccionar la asignatura de la lista*, pulsando el botón izquierdo del ratón sobre alguno de los objetos que están parpadeando en color verde. *Esta tarea puede hacerse más de una vez*
>
> **y**
>
> *Seleccionar el grupo de la lista*, pulsando el botón izquierdo del ratón sobre alguno de los objetos que están parpadeando en color azul. *Esta tarea puede hacerse más de una vez.*

Como se aprecia en la Figura 38, la manera en que DARTS muestra el *feedback* visual es dependiente de la información contextual de la tarea que se está enseñando. Cuando un parámetro tiene un valor asignado, puede repercutir en una disminución de la cantidad de opciones disponibles, de modo que los elementos que el sistema hace parpadear es menor (en este caso, sólo un objeto es seleccionable).

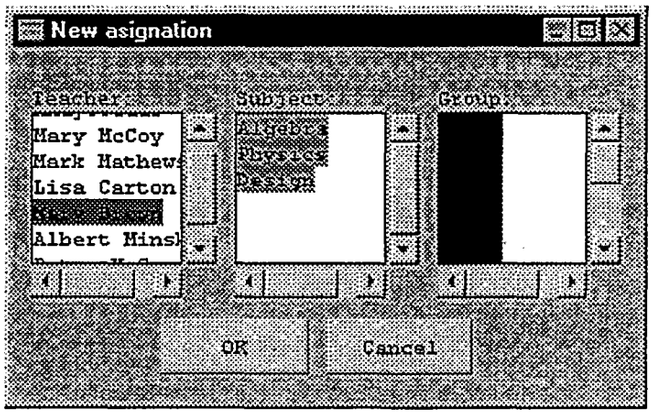

Figura 38: *Feedback* visual de DARTS sobre los datos que pueden seleccionarse.

En un caso como el actual, en el que una tarea contiene subtareas cuya ejecución viene determinada por el operador de secuenciamiento '*Y*', DARTS actualiza dinámicamente qué tareas quedan pendientes para completar la supertarea que se está llevando a cabo. Así, cuando el

usuario realiza la tarea *Seleccionar el profesor de la lista*, la información que recibirá a continuación es la siguiente:

> **Está realizando la tarea** *Establecer las componentes de la relación de docencia*
>
> **Para continuar esta tarea, usted tiene que** *Seleccionar la asignatura de la lista*, pulsando el botón izquierdo del ratón sobre alguno de los objetos que están parpadeando en color verde. *Esta tarea puede hacerse más de una vez.*

Según el usuario va realizando las acciones que le son indicadas en la ventana de DARTS, el sistema actualiza las indicaciones, mostrando constantemente las acciones sucesivas que los usuarios tienen que realizar. Como vemos en el caso anterior, puesto que las tres subtareas disponibles vienen marcadas con el atributo de *Múltiple*, hasta que el usuario haya realizado las tres subtareas, todas estarán accesibles. Sin embargo, y puesto que el usuario ya ha realizado la primera de ellas, el sistema otorgará preferencia a la realización de las dos siguientes, ya que aún no han sido realizadas.

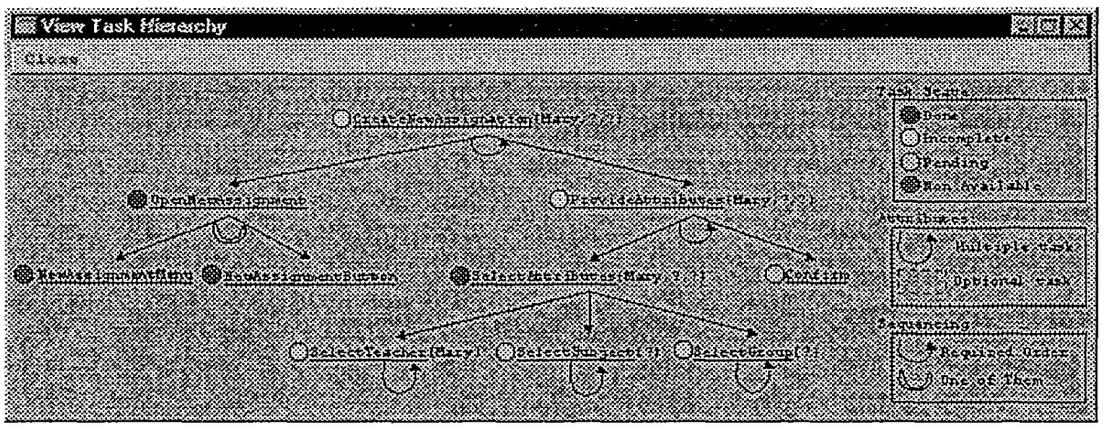

Figura 39: Panel de DARTS antes de confirmar los valores seleccionados.

Una vez que el usuario ha completado las tres subtareas anteriores al menos una vez cada una, al usuario sólo le resta llevar a cabo la subtarea *Confirmar los valores facilitados*. Este es el mensaje que el usuario recibe en este caso:

> **Está realizando la tarea** *Proporcionar los valores que tendrá la relación*. **Ha terminado la tarea** *Establecer las componentes de la relación de docencia*.
>
> **Finalmente, para hacer esta tarea usted tiene que** *Confirmar los valores facilitados*, pulsando el botón izquierdo del ratón sobre el objeto que está parpadeando en rojo.

En este momento, el estado actualizado de la ventana que muestra la jerarquía de la tarea que se está enseñando es el mostrado en la Figura 39, en donde, como puede observarse, la tarea de *Establecer las componentes de la relación de docencia* se considera terminada, aunque también se indica que sus subtareas (de posible ejecución múltiple) pueden ser ejecutadas de nuevo.

Cuando el usuario realiza esta última subtarea, DARTS informa de que la tarea ya ha sido completada exitosamente, y actualiza de nuevo la ventana que muestra la jerarquía completa (Figura 40):

Ha terminado la tarea *Añadir una nueva asignación de docencia*.

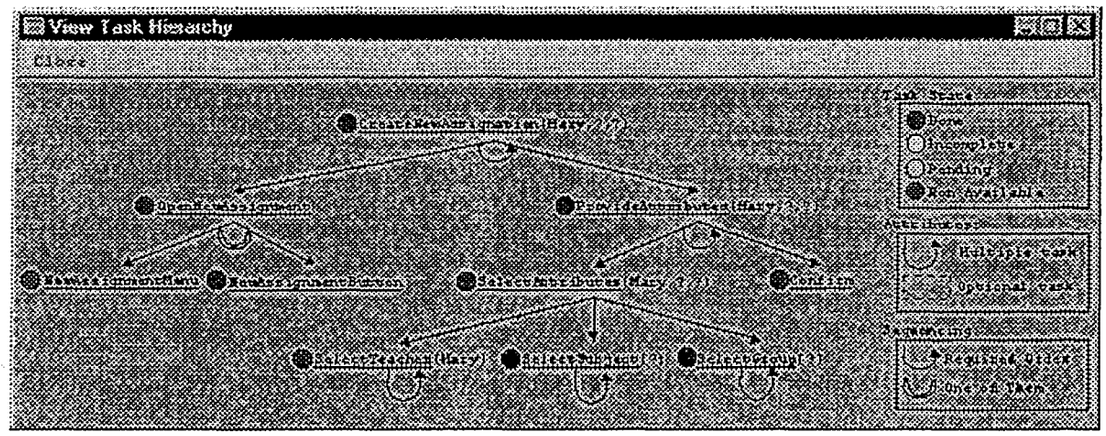

Figura 40: Panel de DARTS una vez terminada la tarea.

4.2.4 Generación de los Mensajes de Guía

El diseño de la guía que proporciona DARTS involucra dos procesos. El primer paso consiste en la especificación de los modelos de las tareas que se quieran enseñar. No es necesario modelizar la aplicación completa, tan solo es necesario modelizar aquellas tareas que deseemos utilizar con DARTS. A partir de la información contenida en estos modelos de tareas DARTS genera los mensajes de guía, por lo que la correcta especificación de las distintas partes de los modelos es vital para la obtención de mensajes efectivos de cara a la enseñanza de los procesos. En segundo lugar, los diseñadores pueden, como veremos, modificar la información de guía proporcionada por DARTS desde dentro del propio entorno de la ayuda.

La información presente en los modelos de tareas es leída y utilizada por DARTS para la generación de mensajes de guía. Estos mensajes son generados de acuerdo a una determinada gramática. En el caso de la implementación de DARTS realizada como parte del trabajo de esta tesis, esta gramática puede encontrarse completa en el Apéndice III de este documento y su lógica la describiremos en esta sección. Entre la información que se obtiene de los modelos de tareas para proporcionar los mensajes se encuentran los nombres de las tareas, sus descripciones, las relaciones de secuenciamiento entre tareas expresadas en las reglas, y la posible multiplicidad u opcionalidad en la ejecución de las tareas.

En particular, los nombres y descripciones de las tareas, como hemos visto, son cadenas de texto que son automáticamente insertadas por DARTS en las posiciones adecuadas de la gramática cuando DARTS está proporcionando guía sobre la tarea solicitada. Proporcionar nombres a las tareas es una labor que, aunque simple, resulta esencial de cara a la efectividad de un entorno de enseñanza. Los nombres de las tareas son utilizados tanto en *modo descriptivo* como en *modo activo*. De este modo, muchos de los mensajes mostrados con anterioridad que están marcados con letra *cursiva* se corresponden con nombres de tareas. Por su parte, las descripciones de tareas se utilizan en *modo descriptivo* para describir los objetivos que la realización de una tarea consigue.

Además, DARTS también utiliza la información existente en los modelos de tareas para ofrecer en sus explicaciones *feedback* sobre las relaciones temporales existentes entre las actividades que el alumno está llevando a cabo, las que tiene que hacer y las que ya ha completado. Como se pudo apreciar durante el ejemplo detallado ofrecido en la sección anterior, DARTS ofrece explicaciones en las que se incluye información que permite al alumno situar las tareas que debe realizar en el contexto de las acciones de nivel semántico superior que está realizando. Con este fin, este sistema de guía indica al alumno qué tareas ha concluido (**Ha terminado la tarea**...), qué tareas se han terminado de realizar o se tienen que realizar a continuación (**Está realizando la tarea**..., **A continuación, usted tiene que**..., **Para continuar esta tarea, usted tiene que**...), o qué papel cumplen las tareas que tiene que realizar en relación con las anteriores acciones que se han realizado (**Primero, usted tiene que**..., **Finalmente, usted tiene que**..., **Primero, para hacer esta tarea usted tiene que**..., **Finalmente, para hacer esta tarea usted tiene que**...).

Finalmente, como parte de la información estática que DARTS utiliza para la generación de la información de guía, se tiene la relativa a la posible multiplicidad u opcionalidad de la ejecución de algunas subtareas. Como vimos, estos atributos vienen definidos dentro de los modelos de tareas de ATOM como parte de las *reglas* de estos modelos. En el caso de que DARTS se encuentre con una tarea cuya ejecución esté marcada con la posibilidad de ser múltiple, DARTS, en *modo activo,* indica al usuario que puede ser ejecutada en múltiples ocasiones, siempre que la posible condición de multiplicidad asociada a dicha multiplicidad se cumpla, aunque dependerá la manera de referirse a este hecho de si la tarea ya ha sido realizada en alguna ocasión o no (*Esta tarea puede hacerse más de una vez* versus *Puede volver a realizar esta tarea*). Por su parte, cuando el atributo de opcionalidad está activado, DARTS incluye dentro del conjunto de tareas pendientes aquella tarea que puede ser realizada a continuación de la tarea opcional e indica, al explicar la tarea, que su ejecución es opcional (*Esta tarea es opcional*). Ésto ocurrirá siempre que la condición de activación de dicho atributo de opcionalidad se cumpla, ya que de lo contrario la tarea se considerará de ejecución obligada.

Pero DARTS también precisa acceder a información que no está contenida en los modelos de tareas, sino que debe ser obtenida dinámicamente a partir del contexto relativo a las actividades que realiza el alumno. Esta información permite proporcionar explicaciones que involucren el contexto de la realización de las tareas que en cada momento se presente. Como consecuencia de esta adaptación al contexto en el que las actividades se están realizando, los usuarios relacionan mejor la información que reciben con su problemática particular y, por tanto, se sienten más cómodos en el proceso de aprendizaje. Como parte de este contexto, se encuentra el hecho de que DARTS adapta sus explicaciones para adecuarse a las acciones que el alumno ha realizado con antelación. Así, en las ramas '*Xor*' el sistema adapta las explicaciones para enfocarlas directamente a la opción tomada por el usuario, obviando los posibles caminos alternativos, al no haber optado por ellos el usuario. Similarmente, el caso de encontrarse con ramas de tipo '*Y*', el sistema enfoca su atención a aquellas tareas que aún no se han realizado, sin por ello olvidar el resto. De este modo, el entorno de guía tiene en cuenta las decisiones tomadas por los usuarios a la hora de realizar unos procesos y no otros, con el fin de proporcionar explicaciones que no aporten información innecesaria y que se ciñan a las necesidades más inmediatas del usuario.

Finalmente, destacar el proceso de generación automática de explicaciones relativas al tipo de interacción que los usuarios tienen que realizar para llevar a cabo las tareas atómicas. Estas explicaciones se proporcionan a partir de los contenidos de los *patrones de interacción* provenientes de la especificación de las *tareas atómicas*. Estas explicaciones (pulsando el botón

... del ratón sobre .../en ..., tecleando ... en el campo ..., arrastrando ... unidades ...) tienen en cuenta dos tipos de información. Por un lado, hacen referencia al tipo de interacción responsable de llevar a cabo la tarea atómica en cuestión. Esta información proviene directamente del prototipo de tarea utilizado para la especificación de cada tarea atómica, ya que el paradigma de herencia prototipo/instancia hace accesible esta información y permite explicar el *cómo* de las interacciones. Para informar al usuario sobre el resto de información referente a las interacciones, se utiliza de nuevo la técnica de *propagación inversa de parámetros*, introducida en la Sección 3.4. Como vimos, mediante esta técnica se puede conseguir información relativa a las interacciones a partir de información contextual asociada a tareas de más alto nivel. Cuando esta información puede obtenerse se comunica al alumno que, dependiendo del tipo de interacción, recibe así información contextual relativa al *dónde*, *cuánto*, y *qué* de las interacciones.

La calidad de los textos es crítica para el éxito de una componente de guía. Los seres humanos tienen mucha más facilidad para escribir buenos textos que las máquinas, de manera que DARTS permite a los diseñadores modificar los mensajes generados automáticamente, para hacerlos más apropiados para cada caso particular. Con el fin de permitir la modificación de los mensajes que son generados automáticamente, DARTS incluye una opción de menú, *Diseño/ModoEdición*, que permite a los diseñadores modificar en el mismo entorno en el que aparecerán los mensajes (*in-place*) los distintos nombres y descripciones de tareas. En este modo, los diseñadores de la componente de guía pueden modificar la información de los modelos de las tareas que son utilizadas durante la guía para adecuarla al contexto del texto en que son mostradas. Esta es una característica esencial del entorno DARTS, ya que en muchas ocasiones es durante la visualización por primera vez de los mensajes de ayuda cuando los diseñadores se dan cuenta de que los nombres de las tareas o sus descripciones no son los apropiados, o no concuerdan correctamente en el contexto en que son mostrados después. Así, si, como hemos visto en los ejemplos anteriores, los nombres de las tareas y sus descripciones se han proporcionado en mayúsculas y, sin embargo, nunca son mostrados al comienzo de una frase, el diseñador puede modificar estas especificaciones utilizando directamente la interfaz de DARTS, sin necesidad de acceder directamente al modelo de tareas.

Como hemos visto, tanto las jerarquías de tareas como la información de guía asociada pueden ser reutilizadas entre distintas aplicaciones. Anteriormente vimos cómo, en el caso particular de ATOMS, se proporciona un conjunto de prototipos de tareas atómicas a partir de las cuales los diseñadores podían definirse sus propias tareas mediante la utilización del paradigma de herencia prototipo/instancia. Este paradigma puede jugar un papel importante a la hora de la generación de los mensajes de guía, permitiéndose un ahorro considerable de tiempo de diseño, ya que la información de guía puede ser reutilizada entre tareas. Para ello, se hace uso de una característica no comentada anteriormente de DARTS, que es la posibilidad de incluir el símbolo *%número* dentro de los nombres y las descripciones de las tareas. Mediante este símbolo se consigue que las descripciones hagan, automáticamente y gracias a la herencia basada en el paradigma prototipo/instancia, referencia a los valores de parámetros utilizados durante el proceso de instanciación de las distintas tareas. De este modo, en algunos casos no se necesitará ni siquiera proporcionar las descripciones de las tareas. Así, por ejemplo, la definición del prototipo de tarea asociado a la pulsación de los botones y proporcionado por ATOMS dentro de su *framework* es similar a:

TASK *"Presionar el botón %1"* **ISA SELECTION ALIAS** *BUTTON*

DESCRIPTION *Pulsa el botón con nombre %1*

... ...

// especialización del patrón de interacción de la tarea SELECTION

PATTERN *LABEL == %1*

PATTERN *ISA == Button*

END

A continuación, puede definirse la tarea atómica asociada a la tarea de pulsar un botón con la etiqueta *Ok* de la siguiente manera, evitándose el tener que aportar un nuevo nombre y una nueva descripción para dicha tarea:

TASK ISA BUTTON "Ok" **ALIAS** *Confirmar*

END

4.2.5 Filtrado de Acciones Incorrectas

Por lo visto hasta el momento, el funcionamiento de DARTS mejora al de los sistemas actuales de guía porque su comportamiento es dinámico frente al comportamiento estático de aquellos. Sin embargo, hasta el momento solo se ha tratado la integración entre el sistema de guía y la propia aplicación desde el punto de vista de que DARTS es capaz de hacer referencias a los objetos gráficos de la aplicación. Pero estas referencias, en forma de parpadeos en distintos colores, no modifican en forma alguna el comportamiento de la aplicación, que sigue manteniendo la misma funcionalidad que tenía antes de modelizar sus tareas. Es decir, no se ha descrito hasta el momento ninguna funcionalidad que haya modificado la manera en que el usuario y la aplicación interaccionan.

Como vimos al describir el *Analizador de Eventos* en la Sección 3.3.1, éste podía interaccionar con las *Herramientas de Valor Añadido* con el fin de que estos servicios pudieran decidir si las aplicaciones finales debían ser informadas de las interacciones de los usuarios o no. Así, las *Herramientas de Valor Añadido* pueden hacer que el *Analizador de Eventos* filtre las acciones de los usuarios antes de que la aplicación las conozca, como puede verse en la Figura 21. Esta capacidad de diálogo entre el *Analizador de Eventos* y las *Herramientas de Valor Añadido* proporciona un mecanismo de gran potencial mediante el cual alterar el comportamiento original de la aplicación. Este filtrado de acciones se lleva a cabo mediante la utilización de *filtros* de tareas, que son una clase especial de objetos que establecen bajo qué condiciones qué tareas deberán filtrarse.

Para ilustrar cómo pueden utilizarse los filtros de interacciones del usuario para modificar la interfaz de la aplicación y, así, controlar la actividad de los usuarios para enseñarles a realizar las tareas, vamos a ver cómo se comporta DARTS cuando las acciones de los usuarios no se ajustan a lo indicado por la guía facilitada. En estos casos, DARTS puede actuar con varios niveles de flexibilidad, según lo deseen los diseñadores. Pueden incorporarse filtros de actividad del usuario al sistema siguiendo un protocolo establecido de comunicación entre el *Analizador de Eventos* y las *Herramientas de Valor Añadido*. Inicialmente, DARTS incorpora dos modos de funcionamiento en cuanto a flexibilidad.

Por un lado, se tiene el *modo estricto*, mediante el cual el sistema no permite que el usuario realice ninguna tarea que no esté dentro del *conjunto de tareas pendientes*. En este modo de funcionamiento, cuando DARTS se da cuenta de que el usuario intenta llevar a cabo una acción errónea, bloquea la aplicación y muestra al usuario un mensaje alertándole de la situación (ver

Figura 41) para, a continuación, permitir al usuario seguir intentando realizar alguna de las tareas pendientes. En este caso, por tanto, la aplicación no responde a las intenciones del usuario que no se ajustan a la guía.

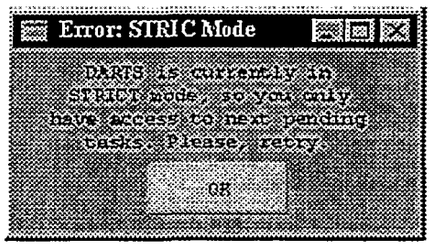

Figura 41: Mensaje de alerta ante acciones del usuario *incorrectas*.

Por otro lado tenemos el *modo libre*, mediante el cual DARTS, pese a las ventajas que el seguimiento de la actividad de los usuarios proporciona, se comporta de manera similar a como lo hacen los sistemas tradicionales de ayuda. En este modo de funcionamiento, cuando el usuario realiza alguna tarea que no se encuentre actualmente en el *conjunto de tareas pendientes*, la aplicación seguirá funcionando de igual manera en que lo haría si no se encontrara DARTS. Por lo tanto, se deja a responsabilidad de los usuarios el seguir las instrucciones del sistema o no hacerlo. En los casos en los que DARTS detecta que el usuario realiza algo que no sigue las indicaciones facilitadas, simplemente muestra durante un segundo un mensaje de aviso que no interrumpe la actividad natural del usuario y, por ello, continúa explicando las mismas tareas que antes estuviera haciendo (ver Figura 42).

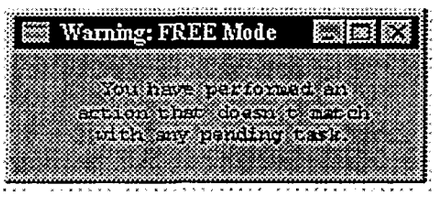

Figura 42: Mensaje de aviso mostrado en *modo libre* ante acciones *incorrectas*.

Un detalle que merece especial atención es que, como pudimos apreciar en la Figura 21, los eventos producidos por los usuarios, tras ser analizados por el *Motor de Análisis,* llegan a la aplicación. La aplicación, cuando recibe la información referente a dichas interacciones, actúa conforme a su funcionalidad y, posteriormente, es de nuevo el *Motor de Análisis* y, en concreto, el *Analizador de Eventos*, quien vuelve a recibir la información sobre la interacción del usuario. El hecho de que este módulo vuelva a procesar la información de la interacción del usuario tiene una gran importancia desde el punto de vista de la arquitectura ATOM. Como ya hemos visto, las tareas pueden hacer referencias a parámetros, cuyos valores son recogidos en tiempo de ejecución. A veces, estos parámetros pueden hacer referencia a valores que no se conocen hasta que la aplicación haya procesado la interacción. Así, por ejemplo, supongamos que estamos tratando con una aplicación de diseño gráfico en la que, como una de sus tareas más básicas, se permite dibujar un rectángulo mediante el clásico procedimiento de seleccionar la herramienta de dibujo de cuadriláteros y, a continuación, arrastrar el ratón, apretando el botón, hasta seleccionar

el contorno deseado para la figura, para posteriormente soltarlo. Pues bien, un parámetro natural para la hipotética tarea *CrearRectángulo* es, además de su posición y dimensiones, una referencia al propio objeto, por ejemplo, su nombre. Pero el rectángulo no existirá como tal hasta que la aplicación no haya procesado completamente la tarea, de modo que el *Analizador de Eventos* solo podrá conocer qué valor debe dársele al parámetro *nombreDeObjeto* tras haber pasado el evento anteriormente a la aplicación.

4.2.6 Arquitectura y Funcionamiento de DARTS

El subsistema DARTS, como una *Herramienta de Valor Añadido* que es, trabaja en conjunción con la aplicación a la que proporciona información de guía, tomando para ello como base la arquitectura de gestión de tareas ATOM. Como vimos en la arquitectura ATOM, pueden usarse *Herramientas de Valor Añadido* para complementar la interfaz de las aplicaciones. Estas herramientas ofrecen servicios genéricos, de manera automática o semiautomática, que razonan sobre las tareas que han sido especificadas en los modelos de tareas de usuario.

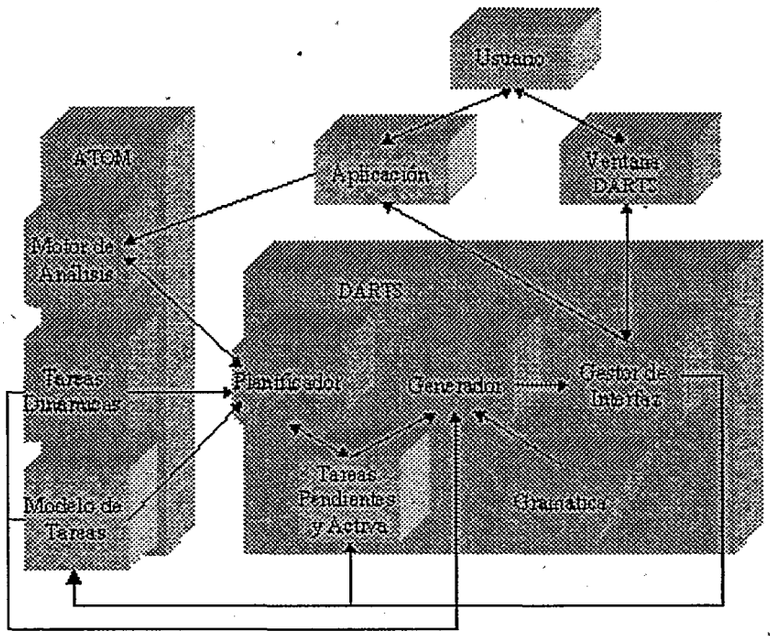

Figura 43: Arquitectura del entorno DARTS.

En particular, la implementación realizada como trabajo de esta tesis funciona sobre el entorno ATOMS de gestión de tareas de usuario. En la Figura 43 se encuentra representada la arquitectura y el funcionamiento interno de DARTS[‡]

[‡] Por motivos de claridad, la arquitectura interna de ATOM ha sido simplificada, mostrándose sólo tanto las relaciones existentes entre DARTS y el resto de subsistemas, como las relaciones entre los distintos módulos internos de DARTS.

Como puede apreciarse en esta figura, DARTS está compuesto internamente por tres módulos funcionales (un *planificador*, un *generador de mensajes* de guía y un *gestor de interfaz* de usuario), que gestionan dos áreas de datos (*tarea activa de enseñanza* y *conjunto de tareas pendientes*, y *gramática*). En esta sección se describe qué papel desempeña cada módulo y qué relaciones existen entre estos módulos y las componentes externas al sistema que se ven afectadas por ellos.

***Tarea activa de enseñanza* y *conjunto de tareas pendientes*.** Como se comentó al comienzo de la Sección 4.2, internamente DARTS lleva el control tanto de cuál es en cada momento la *tarea activa de enseñanza* como de qué tareas componen el *conjunto de tareas pendientes* para terminar de realizar la tarea que se está enseñando. Como vimos, en estos conceptos se apoya el sistema para determinar el *modo* en que está funcionando y, por tanto, el tipo de mensajes que proporciona (ver Figura 32). Estos dos elementos, la *tarea activa de enseñanza* y el *conjunto de tareas pendientes*, se mantienen actualizados merced a la labor conjunta del *Motor de Análisis* de ATOM y del *planificador* de DARTS, como veremos a continuación.

Gramática. Como ya se ha explicado, los mensajes son generados dinámicamente utilizando para ello una gramática independiente del contexto (*context-free grammar*). En la implementación realizada de DARTS, se ha empleado la gramática del Apéndice III.

Planificador. El *planificador* es, junto al *generador de mensajes*, el componente más importante del sistema DARTS. Como vimos en la Sección 3.3, el *Motor de Análisis* de ATOM tiene como misión fundamental la de analizar las tareas que el usuario realiza, con el fin de poder hacer un seguimiento de su actividad. El *planificador* de DARTS tiene tres labores fundamentales:

- Por un lado, se encarga de utilizar la información que el *motor de análisis* facilita para mantener actualizado el *conjunto de tareas pendientes* durante el proceso de enseñanza de una tarea. La manera en que el *planificador* trabaja para mantener actualizado el *conjunto de tareas pendientes* es simple. Para empezar, construye un *conjunto de tareas pendientes* inicial a partir de la descomposición jerárquica de la tarea que se desea enseñar. A partir de este conjunto de partida, tras la realización de cada tarea atómica es informado por el *motor de análisis* para indicársele qué tarea se ha realizado, momento en el cual, de nuevo a partir de la información del modelo y del *conjunto de tareas pendientes*, actualiza este conjunto con las *tareas pendientes* de ese momento.
- Por otro, este módulo también se encarga de cooperar con el *motor de análisis* durante el proceso de *parsing*, modificando así la manera por omisión de trabajar del *motor de análisis*. Como dijimos al tratar el *analizador de eventos* del *motor de análisis*, aquél se apoya en un conjunto de *heurísticas* para intentar determinar con la mayor exactitud posible qué tareas está realizando el usuario. Y, como vimos en la Sección 3.3.2, mediante estas *heurísticas* se permite a otros sistemas externos cooperar con el *analizador de* eventos en su labor. Esto es lo que hace exactamente el *planificador* para conseguir su segundo objetivo. Y, para ello, aporta la *heurística* de dar preferencia a aquellos *estados posibles* (EP) en los que la tarea que se acaba de realizar es alguna de las tareas incluidas dentro del *conjunto de tareas pendientes*. En efecto, cuando se está enseñando al usuario a realizar una tarea, es más que probable que lo que el usuario comience a hacer sea dicha tarea, y no otra. Por ello, el sistema aplicará esta *heurística* y eliminará todos aquellos EPs en los que no se encuentre la tarea objetivo, quedándose sólo con aquellos en los que sí que aparezca.

- Finalmente, el último papel del *planificador* es el de proporcionar el filtrado de acciones del usuario que sean incorrectas, como se ha explicado en la Sección anterior, para lo cual el *planificador* también se pone en contacto con el *motor de análisis*, con el fin de que éste impida a la aplicación el tener conocimiento de la interacción que el usuario ha realizado.

Generador de mensajes. Una vez el *planificador* ha determinado qué tareas componen en un momento dado el *conjunto de tareas pendientes*, entra en funcionamiento el *generador de mensajes*. El papel que este módulo juega es el de generar dinámicamente, tomando como base las tareas que se tienen que explicar y el contexto actual de las acciones del usuario, los mensajes de guía que se le van a proporcionar. Para la generación de estos mensajes, el *generador de mensajes* utiliza la *gramática* del sistema y el contexto proporcionado tanto por la información estática incluida en las *tareas* y las *reglas* de los *modelos de tareas* como la información dinámica presente en las *tareas dinámicas de la aplicación*.

Gestor de interfaz de usuario. Con el objetivo de establecer una separación entre la generación de la información de guía y la manera en que los usuarios acceden a esta información, DARTS incluye un *gestor de interfaz de usuario*, encargado de proporcionar a los usuarios el acceso bidireccional a dicha información.

Así, este gestor se encarga de gestionar tanto la salida del sistema hacia el usuario como las interacciones que éste puede realizar con la ventana de mensajes del sistema. La salida del sistema involucra dos tipos de información:

- Por un lado, DARTS aporta dos ventanas en las que el usuario recibe información y con las que puede interaccionar con distintos fines. En relación a estas interacciones (*eventos de DARTS*), hay que distinguir éstas dependiendo del perfil del usuario que las realice. Como ya hemos visto, hay dos tipos de usuarios potenciales del sistema, y cada uno de ellos tiene unas posibilidades distintas de interacción con el sistema:
  - Por un lado, se tiene a los usuarios finales del entorno, los que reciben la guía por parte del sistema. El tipo de interacciones que éstos pueden realizar incluye modificaciones en el foco de las explicaciones y en la cantidad de contexto deseado para las explicaciones (ver Sección 4.2.1). Cuando el usuario modifica alguno de estos parámetros, el entorno modifica la *tarea activa de enseñanza* y, a continuación, vuelve a generar de nuevo los mensajes de guía conforme a los nuevos criterios solicitados por el alumno.
  - Pero, además, el entorno DARTS también puede ser utilizado por los propios diseñadores de la ayuda con el fin de corregir inexactitudes o deficiencias de la información proporcionada en los *modelos de tareas*. Por esta razón, el *gestor de la interfaz de usuario* de DARTS también accede a la especificación de estos modelos, con el fin de poder modificar estas representaciones para hacer persistentes las modificaciones realizadas por el diseñador.
- Por otro lado, el sistema modifica la interfaz de la aplicación que está enseñando con el fin de ofrecer *feedback*. Este *feedback* es de dos tipos. Por un lado, se aportan dos nuevas ventanas, una destinada a ofrecer explicaciones y otra destinada a proporcionar información visual sobre el estado de realización de las distintas subtareas que forman parte de la tarea que se está enseñando. El otro tipo de *feedback* es el que hemos denominado *feedback* visual, que consiste en modificar la apariencia de los distintos objetos gráficos con los que se debe interaccionar o se ha interaccionado, con el fin de resaltar lo que el alumno debe hacer. Este

feedback se ofrece por medio de parpadeos en los objetos gráficos de la interfaz de la aplicación, como vimos en el ejemplo mostrado al comienzo de esta Sección. Con el fin de que objetos de varios tipos puedan parpadear en distintos colores en según qué contextos, el gestor de la interfaz de usuario incorpora una componente de planificación encargada de asignar distintos colores a distintos tipos de objetos gráficos. Una cuestión pendiente es determinar hasta qué punto el parpadeo de objetos gráficos en la interfaz de la aplicación puede considerarse una aportación demasiado intrusiva y desviar la atención de los usuarios, impidiéndoles centrarse en la información más conceptual formada por las explicaciones textuales.

4.2.7 Beneficios

Una vez hemos visto las capacidades de guía del subsistema DARTS, en esta subsección vamos a ver los beneficios que esta herramienta proporciona. DARTS está orientado a dos tipos de usuarios bien diferenciados. Por un lado, y más importante, se encuentran los destinatarios finales de la componente de guía, es decir, los usuarios. Por otro lado, se simplifica la generación de la guía a los diseñadores que, como hemos explicado, pueden generar la versión final de los mensajes de guía utilizando el entorno de modelización TMT y la propia interfaz de DARTS. Aunque algunos de los beneficios que DARTS proporciona lo son para ambos perfiles de usuarios, a continuación vamos a verlos por separado para cada uno de ellos.

4.2.7.1 Ventajas para el Usuario

A continuación se describen las ventajas que el entorno de enseñanza de tareas DARTS proporciona a sus usuarios finales, es decir, a los alumnos:

- El sistema proporciona a los usuarios guía dinámica mientras realizan las tareas encomendadas. Para ello, es necesario que el sistema siga continuamente la actividad de los usuarios y compare las acciones realizadas con las acciones esperadas que se han explicado con anterioridad. Por esta razón, DARTS es un sistema dinámico y activo, a diferencia de casi todos los sistemas descritos en la Sección 1 (salvo TWIW), en los que no existía relación entre las explicaciones proporcionadas y la actividad de los usuarios. Como consecuencia de esta relación entre la información facilitada y las acciones del usuario, se elimina la asincronía entre las explicaciones y las actividades descrita en la Sección 2.1.1.2.
- El entorno DARTS proporciona *feedback* a los usuarios acerca de la realización de las actividades que se están enseñando. Este *feedback* es de tres tipos. Por un lado, se tiene el *feedback* proporcionado por la representación jerárquica de las tareas que se están enseñando, que incluye la representación del estado de cada tarea, su información contextual, las relaciones de secuenciamiento, etcétera. Esta visualización dinámica, que podemos apreciar en la Figura 34, facilita a los usuarios el seguimiento de qué partes de su trabajo ya han hecho, proporcionándoseles así una manera rápida de obtener un contexto más amplio sobre lo que están realizando. Por otra parte, se ofrece lo que se denomina *feedback* visual de DARTS, que indica a los usuarios los objetos gráficos con los que deben interaccionar en cada momento y aquellos con los que ya han interaccionado. Este tipo de *feedback* (ver Figura 38) ayuda a los alumnos a comprender más fácilmente las posibilidades de interacción que tienen, enriqueciendo así las descripciones textuales que se aportan. Finalmente, y como ya hemos

visto, DARTS se encarga de proporcionar *feedback* a los usuarios acerca de cómo están realizando sus tareas, alertando cuando cometen algún error.

- Como consecuencia de la naturaleza jerárquica de los modelos de los procesos que se explican, la distancia existente entre la manera en que los usuarios estructuran los conceptos y la manera en que éstos son expuestos para la enseñanza disminuye. De este modo, el sistema pretende eliminar el *problema de asociación*, expuesto durante la Sección 2.1.1.2.
- El entorno de enseñanza DARTS puede funcionar con varios niveles de flexibilidad, desde el nivel más flexible, que permite realizar cualquier actividad en paralelo con la tarea que se está enseñando, hasta el nivel más estricto, y que no permite a los usuarios realizar ninguna otra actividad ajena a aquella para la que se está proporcionando información de guía. Esta flexibilidad incrementa la utilidad del sistema, ya que los usuarios más avanzados o con una mayor iniciativa pueden alternar el aprendizaje que están llevando a cabo con la realización paralela de otras actividades; mientras que usuarios menos iniciados o que sienten más temor a enfrentarse con la aplicación pueden hacer que DARTS funcione de manera más rígida y realice un seguimiento más estricto de sus movimientos.
- La guía se proporciona gradualmente, lo que acarrea dos consecuencias positivas.
  - En primer lugar, ayuda a los alumnos a concentrarse en partes determinadas de las tareas, que pueden tener un alto grado de complejidad e involucrar bastantes pasos a realizar. Con este objetivo, el sistema se encarga de proporcionar en cada momento la información necesaria para situar al usuario en el contexto adecuado, minimizando la intromisión con sus actividades. Proporcionar el contexto adecuado significa proporcionar información sobre las acciones que los usuarios han realizado, están realizando y deben realizar, lo que demanda una componente de seguimiento dinámico de las acciones de los usuarios.
  - Como segundo beneficio, la guía facilitada por DARTS ayuda a superar el *problema de visualización* expuesto en la Sección 2.1.1.2, ya que en cada paso la cantidad de espacio necesaria para proporcionar la información a los usuarios es reducida, de modo que la ventana de DARTS no obstruye a los usuarios en su intención de acceder a la ventana de la aplicación que están aprendiendo. De este modo, los usuarios pueden acceder simultáneamente tanto a la información proporcionada por DARTS como a la ventana de la aplicación.
- Finalmente, DARTS funciona utilizando la propia aplicación que se encarga de enseñar a utilizar. Esta característica supone una sustancial ventaja frente a la gran cantidad de entornos que sólo son capaces de enseñar a usar una aplicación utilizando para ello algo que parece dicha aplicación, pero que realmente no lo es. Así, los usuarios no sienten la frustración de estar aprendiendo a manejar algo que no es con lo que realmente se tendrán que enfrentar con posterioridad. Por tanto, los usuarios se sienten más involucrados en el proceso de aprendizaje, aprendiendo mejor los procesos que se les están explicando. Además, mediante DARTS los usuarios pueden aprender mientras realizan las tareas en sus propios entornos de trabajo. Esto significa que, por una parte, ellos pueden relacionar con mayor claridad los procesos que están llevando a cabo con la utilidad que éstos tienen en relación a sus problemas y, por otra parte, el esfuerzo que desempeñan durante el aprendizaje tendrá frutos inmediatos, ya que las tareas realizadas habrán servido para resolver el problema al que en el momento de seguir la guía se enfrentaban.

4.2.7.2 Ventajas para el Diseñador

Por su parte, DARTS también proporciona ventajas a los diseñadores de la guía. Estas ventajas se dan con respecto a la mayor parte del resto de entornos de generación de guía, tanto de los sistemas utilizados tradicionalmente, como de los desarrollados en otros trabajos de investigación. Entre estas ventajas destacamos:

- La funcionalidad aportada por la herramienta es mucho mayor que la funcionalidad que ofrecían hasta ahora muchos de los sistemas de guía, sobre todo cuando se compara con los entornos de generación de guía comerciales. Durante la Sección 2.1 se hizo una crítica de los entornos de generación de guía más comúnmente utilizados o que proporcionaban una funcionalidad más notable, señalando sus virtudes y sus carencias más importantes. Como la mayor virtud de DARTS de cara al diseñador coincidimos en señalar que es la misma que lo hace atractivo de cara a los propios usuarios: la potencia que ofrece de cara a la guía. Así, como hemos visto, DARTS permite a los diseñadores ofrecer información dependiente del contexto, seguimiento dinámico, generación de *feedback* gráfico sobre la aplicación, superación del *problema de asociación*, varios grados de flexibilidad, resolución del problema de visualización, uso de la propia aplicación e integración con la misma.
- Como consecuencia, precisamente, de la estrecha relación de la funcionalidad de guía con las funcionalidades de la aplicación, se encuentra el que los entornos de enseñanza bajo DARTS conllevan un bajo coste de desarrollo. Esta integración se debe a que se parte de un modelo de desarrollo de aplicaciones basado en modelos de tareas. Esto hace que, por un lado, el desarrollo de la funcionalidad de guía sea casi transparente de cara al diseñador, pues se realiza conforme se desarrolla la propia aplicación. La propia definición de la aplicación trae consigo la generación automática a la componente de ayuda sobre la aplicación. Y, por otro lado, al integrarse las líneas de diseño y desarrollo de ambos productos, también quedan integradas las líneas de mantenimiento. Así, cuando la aplicación se haya de poner al día, automáticamente también se pondrá al día el sistema de generación de la guía, sin que hayan de actualizarse en paralelo ambos procesos, con los costes y peligros de sincronización que ello acarrearía. Así, esta integración en el desarrollo trae consigo también la ventaja de que se garantiza la consistencia entre la componente de guía y la interfaz de la aplicación. Con todo ello, se consigue un abaratamiento en los costes de generación de la componente de ayuda durante todo el ciclo de vida del producto *software*.

  En este sentido recalcamos, de nuevo, que la implementación realizada de DARTS se apoya en modelos de tareas que no están ligados a modelos de tareas utilizados para definir la aplicación, sino que se han implementado a posteriori. Por ello, en nuestra implementación se ha perdido parte de la funcionalidad propuesta, aunque las ideas subyacentes siguen permaneciendo válidas.
- Como consecuencia directa de que el proceso de generación de la ayuda no es independiente del proceso de especificación de la interfaz de la aplicación, DARTS ofrece la ventaja de que todas las aplicaciones basadas en el mismo entorno de desarrollo comparten una misma interfaz de comunicación con el usuario final. Esto implica que los usuarios no se verán forzados a manejar interfaces de enseñanza distintas para distintas aplicaciones, sino que el manejo de todos los entornos será idéntico. Esta interfaz del sistema es, además, independiente de las aplicaciones, pudiendo por tanto evolucionar de manera paralela a las aplicaciones a las que sirve sin tener que modificar éstas.

- Finalmente, hay que resaltar como otra ventaja importante el hecho de que el mismo entorno de generación de la componente de guía ofrece la posibilidad de la modificación *in-place* de los mensajes proporcionados, como describimos en la Sección 4.2.4. Gracias a esta facilidad de edición se consigue, por un lado, dedicar el menor tiempo posible a la modificación de los mensajes que no se corresponden exactamente con el deseo del desarrollador, pues no hay un entorno de desarrollo y otro de visualización sino que ambos son el mismo. Y, por otro lado, los diseñadores comparten su mismo entorno de desarrollo con el entorno final de los usuarios, consiguiéndose por tanto una homogeneidad en el tratamiento de la información y una menor necesidad de conocimientos de desarrollo por parte de los diseñadores de la componente de guía.

4.3 Escenarios

El *Módulo de Ejecución de Escenarios* [García99b] se encarga de proporcionar una interfaz con la que preparar contextos apropiados para llevar a cabo la enseñanza de las tareas de los cursos. Estos contextos, preparados previamente a la realización de las sesiones de enseñanza por los diseñadores de los materiales didácticos, reciben el nombre de *escenarios*.

Un *escenario* representa un proceso mediante el cual la aplicación es conducida a un *estado adecuado* para poder procederse a la enseñanza de una determinada tarea. Se entiende por *estado adecuado* aquel estado de la aplicación en el que se puede realizar una cierta tarea o conjunto de tareas que, en nuestro caso, son aquellas que a continuación deseamos enseñar. Por ejemplo, la realización de muchas tareas requiere la existencia previa de determinados datos u objetos, sin los cuales dichas tareas no se pueden realizar. Los escenarios ofrecen la posibilidad de proporcoinar estos datos u objetos de forma previa a la enseñanza de las tareas, con el fin de que los alumnos dispongan de un contexto de práctica que sea el adecuado.

Los diseñadores de cursos interactivos podrán proporcionar, según su criterio, tantos posibles contextos de trabajo predefinidos como deseen. Sin embargo, que el entorno ofrezca la posibilidad de que los usuarios trabajen con escenarios predefinidos no implica que la enseñanza se realice siempre mediante la utilización de estos contextos predefinidos por el diseñador. Creemos que es precisamente uno de los atributos más interesantes de un entorno tutor la posibilidad de que las prácticas se realicen sobre el propio entorno de trabajo de los alumnos. Este enfoque ayuda a incrementar la productividad de los alumnos, pues se produce una mayor motivación durante el aprendizaje y se reduce el *problema de asociación*, como se describe en [Carroll87a].

Sin embargo, cuando el propio contexto de trabajo del alumno no es el adecuado, es conveniente proporcionar uno que cumpla con los requisitos que la enseñanza de determinadas tareas requieren. Por ejemplo, todas aquellas tareas que modifiquen atributos de objetos gráficos en un editor gráfico requieren la existencia de algún objeto de este tipo para poder llevarse a cabo. Así, cuando el contexto de trabajo del usuario no cumple este requisito, la práctica no puede realizarse. Para estos casos, el que los diseñadores proporcionen contextos de trabajo adecuados prediseñados, mediante escenarios, resulta muy útil.

Por lo tanto, el hecho de que se definan uno o varios escenarios posibles que puedan prepararse antes de la enseñanza de una tarea, no implica que sean esos contextos los que se utilicen para la práctica. En efecto, es responsabilidad del módulo que utilice los servicios del *Módulo de Ejecución de Escenarios*, el permitir o no a sus usuarios practicar en su propio contexto de

trabajo o únicamente con escenarios predefinidos. En el caso particular de CACTUS, siempre se permite al alumno decidir en última instancia si prefiere utilizar un escenario predefinido o, en cambio, desea practicar usando su contexto de trabajo. Sin embargo, esta flexibilidad de uso no está exenta de riesgo, pues se está delegando en el alumno la selección del contexto.

En otros entornos de guía, cuando a un alumno se le enseña a realizar una tarea, si dicha tarea precisa de unos requisitos previos, se habilita un mecanismo que supervisa el cumplimiento de esos prerequisitos. Normalmente, los sistemas que implementan esta supervisión, como [Sukaviriya90], se basan en el uso de precondiciones y la modelización de relaciones causa-efecto. Una vez especificada esta información, un planificador se encarga de encontrar la cadena de acciones que los usuarios han de llevar a cabo para cumplir las precondiciones de las tareas. La modelización de estas relaciones puede representar una tarea altamente tediosa y compleja.

4.3.1 Ejecución de Escenarios

Los escenarios vienen representados por la descripción del procedimiento mediante el cual se prepara un contexto de práctica. Estas descripciones procedurales son interpretadas bajo demanda por el *Módulo de Ejecución de Escenarios* que, basándose en el módulo *Emulación* de ATOMS, se encarga de hacer que el estado de la aplicación sea el que el diseñador haya previsto. En el caso de CACTUS, los servicios del *Módulo de Ejecución de Escenarios* son requeridos durante la realización de los cursos por parte de los alumnos.

La preparación de los contextos representados por los escenarios se basa fundamentalmente en la ejecución sucesiva de tareas para la consecución final del marco de trabajo deseado. Por ello, la parte principal de estos procedimientos hace referencia a la ejecución de tareas. Estas referencias a tareas juegan el mismo papel que el que juegan las llamadas del sistema en otros entornos: el *Módulo de Ejecución de Escenarios,* basándose en el módulo *Emulación* de ATOMS, ejecuta las tareas referidas utilizando animaciones.

Las animaciones gráficas e, incluso, las representaciones gráficas que proporcionan el sentido de animaciones, sobre todo cuando son utilizadas acompañadas de explicaciones textuales que indican qué es lo que las animaciones representan, mejoran la percepción humana acerca de los procesos que acaecen, como indican algunos resultados experimentales [Booher75, Palminter89]. Además, la bondad de la utilización de animaciones va más allá del impacto obtenido por la apariencia visual, teniéndose la capacidad de mostrar relaciones temporales y de tipo causa-efecto [Rieber91]. Como consecuencia de la mejora de la percepción, así como de la posibilidad de ofrecer una muestra de las relaciones temporales entre tareas y relaciones causa-efecto de las acciones, los usuarios obtienen de la creación mediante animaciones de contextos de trabajo predefinidos un refuerzo de la enseñanza de los procesos.

Otra funcionalidad importante que pueden incluir los escenarios es la opción de deshacer tareas (*undo*). En este caso, también se requiere el uso de *Emulación* para dar soporte a la ejecución de este servicio. Mediante este mecanismo se posibilita que desde los escenarios se puedan deshacer algunas de las tareas que se hayan realizado. Posteriormente veremos cómo esta capacidad es útil para, por ejemplo, proporcionar al usuario un ejemplo de cómo realizar una tarea y, posteriormente, deshacer los efectos de manera que no se hagan persistentes.

Además de la ejecución de tareas y la opción de deshacer estas tareas, es importante para un sistema que realiza animaciones el proporcionar información por otra vía a los usuarios, con el fin de que no se vean confundidos al ver cómo el sistema realiza acciones automáticamente, sin

su intervención. Con esta finalidad, también se debe poder proporcionar *feedback* textual según se preparan los escenarios, incorporándose así tanto explicaciones como referencias gráficas, mediante parpadeos de objetos gráficos, a objetos de la interfaz de usuario, al igual que se hace en DARTS.

Las funcionalidades anteriormente citadas pueden incorporarse en un escenario relacionadas entre sí mediante distintas estructuras de control de flujo, como se describirá en la siguiente sección. De este modo, pueden llegar a proporcionarse escenarios tan complejos como se desee.

El esquema de la arquitectura de ejecución de los escenarios está representado en la Figura 44. Como vemos, la descripción procedural del escenario es recogida por el *Módulo de Ejecución de Escenarios*, situado en la parte superior de dicha figura. Esta pieza es la encargada de gestionar el flujo de ejecución del escenario. Cuando es necesario, este módulo encamina algunas de las peticiones al módulo de *Emulación*. Estas peticiones se tratan de peticiones de realización de tareas y de peticiones de deshacer tareas. *Emulación*, una *Herramienta de Valor Añadido* construida sobre ATOMS, lleva a cabo las tareas realizando animaciones (ver Sección 3.4).

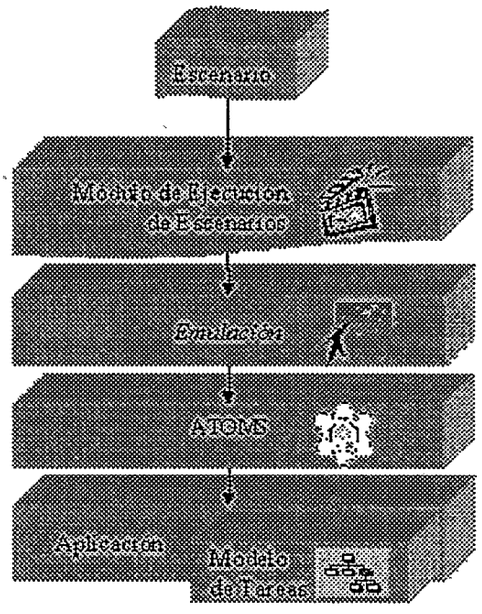

Figura 44: Esquema de flujo en la ejecución de un escenario.

4.3.2 Especificación de Escenarios

Hasta el momento hemos visto qué son los escenarios y cómo éstos son procesados de manera que proporcionen contextos de ejecución por defecto para los usuarios. Durante esta sección vamos, en primer lugar, a descender hasta el nivel básico de definición de los escenarios, para conocer qué representación final tienen. Dado que esta representación final no es sino un lenguaje formal de especificación de escenarios, los diseñadores necesitan aprender una nueva sintaxis para representar estos contextos, con los costes que esto supone. Por ello, a continuación se

explicará el soporte que desde CACTUS se le ha dado a la automatización en la generación de estas representaciones.

4.3.2.1 Lenguaje de Especificación de Escenarios

Como ya se ha mencionado, los escenarios de la arquitectura CACTUS vienen representados por medio de procedimientos expresados mediante un lenguaje de programación, diseñado con el objetivo de dar soporte a estas representaciones. Este lenguaje de especificación es interpretado en tiempo de ejecución por el *Módulo de Ejecución de Escenarios*, como hemos mostrado anteriormente. Mediante la ejecución de los escenarios, este módulo se encarga de proporcionar al usuario, temporalmente, un contexto con el que practicar el aprendizaje de las funcionalidades de su aplicación.

En esta Sección se describe someramente el lenguaje de especificación de escenarios utilizado en la arquitectura CACTUS, que finalizará con la especificación completa de un pequeño escenario. Una referencia completa de este lenguaje puede encontrarse en el Apéndice IV, al final del presente documento. Para un ejemplo más detallado que el descrito en esta sección, consulte el Apéndice V.

La estructura general de la especificación de un escenario es la que se muestra en la Figura 45.

Figura 45: Esquema general de la especificación de un escenario para CACTUS.

El lenguaje de especificación de escenarios está basado en etiquetas, lo que facilita su aprendizaje y uso, y simplifica la construcción de herramientas que automaticen su generación. Cada instrucción se compone de una etiqueta, a la que le sigue un conjunto de valores que aportan información adicional. Veremos posteriormente cómo el espíritu del lenguaje de especificación de cursos interactivos de enseñanza de CACTUS es similar al de este lenguaje de especificación de escenarios.

Los escenarios se agrupan en ficheros, formando librerías de escenarios. Al igual que las subrutinas clásicas, los escenarios pueden recibir información del servicio solicitante en forma de

parámetros, a cuyos valores pueden hacer posteriormente referencia, al igual que sucedía en el lenguaje de especificación de áreas de ATOMS.

En primer lugar, cada escenario viene definido por un nombre, que identifica a cada escenario dentro de su correspondiente librería de escenarios predefinidos y una descripción. Ambos atributos son esenciales de cara a una rápida identificación de los escenarios y los objetivos de los mismos, así como de cara a la integración del servicio de escenarios con otros servicios como, por ejemplo, la realización de cursos interactivos.

Tras estos dos atributos iniciales de los escenarios, se encuentran todas las posibles instrucciones que a un escenario se pueden incorporar, y que podemos descomponer en tres grupos.

- El primer grupo de instrucciones permite a los diseñadores de los escenarios establecer los parámetros de comportamiento del *Módulo de Ejecución de Escenarios* durante la fase de ejecución de los mismos. Dado que el *Módulo de Ejecución de Escenarios* se apoya en la funcionalidad del bloque *Emulación* de ATOMS para su labor, estos valores parametrizan su funcionamiento. Los atributos más importantes que afectan al comportamiento de *Emulación* son tres. En primer lugar, la velocidad con la que se emulan los movimientos del cursor del ratón durante la emulación de las tareas del usuario (LAPSUS[§]). En segundo lugar, la velocidad con la que las interacciones de edición de textos se producen (CHARLAPSUS), que normalmente será mayor que la anterior, con la finalidad de no ralentizar la emulación cuando estas interacciones trabajan con textos mínimamente largos. La tercera manera en la que desde un escenario se puede parametrizar el funcionamiento de *Emulación* es utilizando la velocidad con la que se refresca la visualización de la pantalla durante la ejecución (REFRESHMODE), pudiéndose hacer que ésta sea paso a paso, tarea a tarea, o únicamente se realice una vez por escenario. Finalmente, se encuentra la parametrización que indica a *Emulación* cómo se debe efectuar el tratamiento de obtención de valores de parámetros durante la propagación inversa de parámetros explicada en la Sección 3.4 (SKIP y ASK).
- En un segundo grupo se incluyen las instrucciones cuya ejecución tiene un efecto visual directo en el usuario. Este grupo de instrucciones que son advertidas de manera inmediata por el usuario son, por un lado, las instrucciones básicas que soportan la creación del contexto de trabajo del usuario y, por otro, los mensajes de *feedback* que se ofrecen. Las primeras soportan el núcleo de la funcionalidad de los escenarios, y albergan instrucciones para ejecutar y deshacer tareas de usuario (TASK y TASKUNDO, respectivamente). Estas peticiones son transladadas a *Emulación*, que es quien las maneja de forma directa, proporcionando al usuario el *feedback* adecuado. Los mensajes de *feedback* con referencias gráficas permiten a los escenarios ofrecer al usuario información adicional sobre las acciones que realizan (MSG...HIGHLIGHT).
- Finalmente, existe un último grupo de instrucciones que facilitan el control del flujo en las rutinas que definen los escenarios. Este grupo de instrucciones incluye las construcciones típicas de cualquier lenguaje de programación en lo referente al control de flujo de programas. Así, entre estas instrucciones nos encontramos con construcciones iterativas

[§] Durante esta subsección se utilizarán palabras con letras mayúsculas entre paréntesis para informar sobre el nombre de la etiqueta a la que la explicación se refiere.

(FOR...FROM...TO...STEP...ENDFOR, FOR...IN...ENDFOR, WHILE...ENDWHILE, DOWHILE...ENDDO), construcciones de ejecución condicional (IF...ELSE...ENDIF), construcción de bloques de ejecución aleatorio (ALEAT...BLOCK...ENDBLOCK...ENDALEAT), llamadas a otros escenarios a modo de *subescenarios* (CALL), y utilización de variables para almacenar valores durante la ejecución (VARIABLE y SET).

Para finalizar esta sección vamos ver una muestra de cómo especificar un escenario mediante el lenguaje expuesto anteriormente. Este ejemplo, de nombre *Ejemplo*, es utilizado internamente por CACTUS, como veremos en la Sección 4.4.2, para ejecutar una tarea que es recibida como parámetro, y se compone únicamente de cuatro instrucciones. La primera de ellas proporciona la descripción del objetivo del escenario (SCNDESC). A continuación, se encuentra una instrucción de ejecución de una tarea (TASK), en este caso de la tarea que debe ser pasada como parámetro durante la llamada al escenario. Después, la instrucción de etiqueta MSG ofrecerá al usuario un mensaje indicándole que los efectos de la tarea de ejemplo que se ha realizado inmediatamente antes van a ser deshechos. Finalmente, la última instrucción indica que el escenario deshaga los efectos de la última tarea realizada, restableciéndose así el contexto de partida en la aplicación. Veamos el código necesario para especificar este escenario:

SCENARY *Ejemplo*

// descripción del escenario

SCNDESC *mostrar un ejemplo de la tarea* **%1. DESCRIPTION**

// realizar la tarea recibida como parámetro por el escenario

TASK %1

// proporcionar al usuario un mensaje de *feedback*, indicándosele que el ejemplo va a deshacerse

MSG *El ejemplo ya se ha mostrado. A continuación, se procederá a deshacer sus efectos..*

// deshacer la última tarea realizada (en este caso, la del ejemplo)

TASKUNDO 1

ENDSCN

Para llamar a este escenario que hemos creado, simplemente se utilizará una instrucción de llamada desde el módulo que requiera el servicio, indicándose como parámetro la tarea que se desea utilizar como ejemplo. A continuación puede verse cómo hacerlo:

CALL *Ejemplo CrearNuevaAsignación*

Como indicamos al comienzo de esta sección, el lenguaje de especificación de escenarios se ha tratado de diseñar lo más simple posible con el objetivo de facilitar al máximo la creación de herramientas que ayuden en la automatización de estas especificaciones. En la siguiente subsección vamos a ver cómo, dentro del marco de este trabajo, se han proporcionado algunas ayudas para esta automatización.

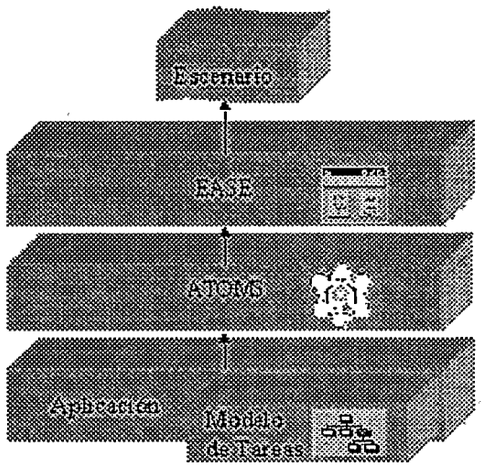

Figura 46: Esquema de funcionamiento de la herramienta EASE.

4.3.2.2 Especificación Interactiva de Escenarios: EASE

La herramienta EASE, *Aplicación Simple para la Obtención de Escenarios* (*Easy Application for Scenary Elicitation*) proporciona el soporte para la automatización de la generación de escenarios sencillos. Mediante esta sencilla herramienta es posible generar escenarios mediante el uso de la misma aplicación a la que los escenarios van referidos.

EASE es un nuevo ejemplo de *Herramienta de Valor Añadido* y que, por tanto, aprovecha las ventajas del soporte orientado a tareas de ATOMS para su funcionamiento. Como podemos ver en la Figura 46, EASE es capaz de, gracias al seguimiento de las tareas realizado por ATOMS, monitorizar la actividad del diseñador para dar soporte automatizado a la generación de escenarios. Partiendo de la actividad que los diseñadores de escenarios realizan con la aplicación para la que están realizando el escenario, EASE utiliza la información generada por ATOMS para representar las secuencias de tareas de los diseñadores mediante escenarios.

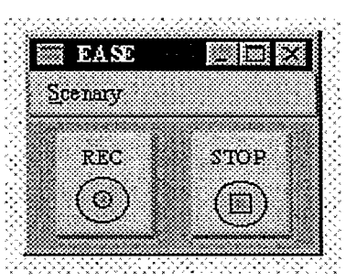

Figura 47: Interfaz de la herramienta para especificación de escenarios simples.

La interfaz de EASE, como puede verse en la Figura 47, es muy sencilla. Tan solo existen tres opciones a las que acceder, que se explican a continuación, y su funcionamiento es muy similar al

de dispositovos electrónicos como videos o DVDs y las interfaces gráficas de los entornos de grabación de macros.

Para que EASE comience a funcionar, es necesario que la aplicación para la que va a crear un escenario esté ejecutándose. En primer lugar, EASE dispone de dos botones, *Grabar y Parar*, respectivamente. La pulsación del primero de los botones hará que EASE entre en un modo de escucha, en el cual registrará todas las tareas que el usuario realice con la aplicación. A partir del momento en que el diseñador apriete dicho botón, EASE hará que todas las tareas que el usuario lleve a cabo formen parte de un nuevo escenario, hasta el momento en que se pulse el botón *Parar*. Hasta entonces, EASE habrá registrado internamente todas las tareas realizadas, y será el momento en el que el diseñador podrá acceder a la última opción, la del menú *Escenario*. Ésta permitirá que las tareas registradas sean *volcadas* en forma de escenario a una librería de escenarios seleccionada por el diseñador.

Como ejemplo de funcionamiento de EASE, supongamos que deseemos proporcionar un contexto de trabajo para enseñar a modificar los datos atribuidos a un determinado profesor con *Schoodule*. Para ello, lo que en primer lugar debemos hacer es ejecutar *Schoodule* junto a la herramienta EASE. Una vez tenemos ambas aplicaciones abiertas, seguimos el procedimiento descrito anteriormente, empezando por pulsar el botón *Grabar* de EASE. Una vez hemos hecho ésto, comenzamos a realizar las tareas de las que queramos que conste nuestro escenario. En este caso, deseamos incluir al menos un profesor en nuestra base de datos del programa *Schoodule*, a fin de que posteriormente nuestros alumnos puedan cambiar las propiedades a alguno de ellos. Por ejemplo, insertaremos un nuevo profesor de nombre "*Mary Brown*", aportando sus datos personales, como dirección y DNI. Con esta tarea, en principio, sería suficiente, pues ya se cumpliría el requisito de que existiese al menos un profesor, de cara a modificar sus datos. Por lo tanto, ahora ya se podría pulsar *Parar*, y a continuación utilizar la opción del menú *Escenario* para dar un nombre al escenario recién creado y guardarlo en una librería de escenarios existente o en una nueva. En cambio, si la tarea que deseásemos enseñar fuese la de *CrearNuevaAsignación (ver* Figura 19*)*, necesitaríamos crear un escenario mayor, pues al menos deberíamos proporcionar como contexto de enseñanza previo un profesor, un grupo de alumnos y una asignatura.

A diferencia de lo que ocurre con TMT, EASE no es capaz de realizar generalización alguna de las acciones que el diseñador realiza. Esto se debe a que EASE genera los escenarios a partir de una única ejecución de la secuencia de tareas que lo compone, de manera que no es posible hacer inferencia alguna.

Como consecuencia de esto, la mayor limitación de este procedimiento de generación de escenarios es evidente, ya que solo permite automatizar las construcciones más simples del lenguaje de especificación de escenarios, es decir, las secuencias de tareas. Por su parte, el resto de construcciones del lenguaje, como construcciones condicionales, iterativas o aleatorias, han de ser especificadas mediante codificación directa. Debido a la imposibilidad de representar de manera interactiva con EASE iteraciones o condicionales, los escenarios especificados con esta herramienta no tienen tanta generalidad como pueden llegar a tener los escenarios definidos directamente mediante programación.

4.4 Gestión de Cursos Interactivos de Enseñanza: CACTUS

Hasta el momento hemos visto los dos servicios básicos que la arquitectura de CACTUS proporciona con el fin de servir de base a una enseñanza activa, orientada a tareas, de los usuarios. Sin embargo, cuando el usuario al que se desea dar servicio es un usuario novel, precisamos una interfaz específica a través de la cual ofrecer estos servicios.

El término *usuario novel* no se refiere a menudo a usuario sin conocimientos. Es decir, son usuarios con una base de conocimientos incompleta, no necesariamente con una capacidad de razonamiento menor [Glaser84]. Son usuarios sin conocimientos desde el punto de vista de los sistemas informáticos, pero pueden ser perfectamente unos expertos en sus áreas mediante métodos más tradicionales. Así, dentro de estos usuarios noveles se engloban profesionales como los administrativos *de toda la vida* a los que, de repente, se les pide que realicen todo su trabajo con un sistema informático.

Este tipo de usuarios necesita que los sistemas les ofrezcan la guía de una manera más dominante, diciéndoles lo que tienen que hacer para llevar a cabo sus tareas [Nass95]. Creemos que no es suficiente proporcionar un buen sistema de búsqueda de información para acceder al contenido de un sistema de guía, ya que los usuarios noveles con frecuencia se sienten intimidados hacia su uso y necesitan que sea el propio sistema el que tome el control para la enseñanza.

Con el entorno que proponemos en este trabajo doctoral se afronta la problemática de la disponibilidad de material didáctico sobre el comportamiento de las aplicaciones desde dos puntos de vista distintos. En primer lugar, se persigue que los usuarios noveles puedan recibir la información de una manera que les resulte familiar, de fácil acceso y que, por tanto, les sea sencilla de utilizar. Sin estas premisas, los resultados que obtendrán con sus prácticas normalmente serán deficientes, sin llegar a sentirse cómodos con el uso de sus sistemas. En segundo lugar, se busca que los diseñadores dispongan de un entorno que les simplifique al máximo la generación del material didáctico, sin por ello tener que renunciar a la potencia y funcionalidad que puedan obtenerse de los entornos pedagógicos desarrollados. Con estos dos objetivos en mente, y sobre los servicios ya descritos tanto en la Sección 4.2 como en la Sección 4.3, lo que en esta sección presentamos es la interfaz que el entorno CACTUS proporciona tanto a los usuarios finales como a los diseñadores del material pedagógico. Como comentamos al comienzo de la Sección 4, CACTUS utiliza la metáfora de representar los cursos tutores como si fueran libros de texto, asociando las partes más importantes del curso tutor con las partes más representativas de los libros de texto. De este modo, la generación de un curso tutor se asimila a la creación de un libro interactivo, mientras que el aprendizaje se asimila al seguimiento de los contenidos del mismo. El entorno propuesto ofrece dos grupos de funcionalidades distintas, cada uno de ellos orientado a conseguir uno de los dos objetivos referidos. El primero de ambos se obtiene mediante lo que llamaremos el *modo de ejecución* de CACTUS, que permite aportar funcionalidad cara al usuario final de los cursos. Por su parte, el llamado *modo* de edición de CACTUS permite a los diseñadores disponer de un entorno con el que crear los cursos sobre sus aplicaciones de una manera interactiva y semiautomatizada.

Dividimos esta sección en cuatro subapartados. En el primero de ellos se describirá cómo se muestran los cursos, con la herramienta CACTUS, mediante la metáfora de representarlos como si de libros de texto se tratara. Así, veremos las distintas partes en que estos libros están organizados y cuáles son las funciones de cada uno de sus apartados. En la subsección siguiente se explicará cuál es la interfaz, desde el punto de vista dinámico, entre estos cursos y los alumnos; es decir, veremos cómo se ejecutan los contenidos de los libros en este entorno y como

los usuarios interaccionan con ellos. Esto es lo que hemos llamado *modo de ejecución*. En la tercero de los apartados se tratarán los cursos desde el punto de vista de cómo son creados y modificados, o *modo de edición*, es decir, nos situaremos en el punto de vista del diseñador de los contenidos didácticos y veremos cómo se utiliza el entorno. Finalmente, en la última subsección nos referiremos a las ventajas que la manera en que CACTUS representa sus cursos proporciona tanto a sus usuarios finales como a los desarrolladores.

4.4.1 Metáfora de Libro

CACTUS visualiza los cursos tutores representándolos como si dichos cursos fueran libros de texto. Mediante esta visualización, CACTUS permite a los diseñadores modificar los contenidos de los cursos tutores, y a los alumnos seguir los contenidos de los mismos. Esta manera de representar los cursos como libros de texto interactivos está orientada a ofrecer a los usuarios noveles una interfaz que les resulte familiar, de modo que se reduzcan los problemas que experimentan dichos usuarios al enfrentarse con las aplicaciones.

Esta interfaz de libros interactivos ya se ha utilizado en otros contextos de las Tecnologías de la Información. En los últimos años han aparecido algunas herramientas que permiten crear y gestionar manuales interactivos vía *World Wide Web* (WWW). Así, por ejemplo, el sistema SuperBook [Remde87] permite la generación de libros con contenidos multimedia y que permiten la navegación mediante hiperenlaces a partir de libros de texto. También sobre el contenido de esta información multimedia se han desarrollado avanzadas técnicas de búsqueda, que permiten la búsqueda de términos basada en la información semántica desprendida de éstos. Así, por ejemplo, el sistema Scatter/Gather [Hearst95] permite una estructuración de los resultados de las búsquedas en categorías de varios niveles, lo cual facilita la recuperación de documentos categorizados por áreas temáticas.

Pero para el trabajo expuesto en esta tesis, en lo que más interesados estamos es en la utilización de los libros interactivos como medio para la transmisión de material docente, campo en el que han aparecido algunos entornos en los últimos años. Algunos de ellos están orientados a facilitar la enseñanza de temas que se enmarcan en materias concretas, como las matemáticas [Antchev95] o el álgebra [Cohen99]. Otros, en cambio, proponen arquitecturas genéricas para la gestión de cursos interactivos, y adecúan los contenidos de estos cursos de acuerdo a los perfiles de sus alumnos. Entre estos entornos podemos destacar a InterBook [Brusilovsky98] y TANGOW [Carro2000] que, conforme a las actividades docentes que los alumnos completan, van permitiendo a éstos el acceso a otras partes de los cursos o van generando dinámicamente las siguientes unidades pedagógicas, ofreciéndose así la posibilidad de la enseñanza teniendo en cuenta las condiciones de contexto de sus actividades.

Incluso importantes casas comerciales prevén sacar al mercado a medio plazo libros con contenidos electrónicos que permitan a sus usuarios, los niños, el seguimiento interactivo de cursos de enseñanza o de historias infantiles [Philips96]. En este último caso, estos libros interactivos hacen su contenido más estimulante que los libros tradicionales, ya que los niños interaccionan con ellos mediante pantallas táctiles, y reciben información multimedia como textos, imágenes, videos o sonidos. Este formato permite que las historias se vuelvan más personales, adaptándose los personajes y el hilo de las historias a las preferencias de los usuarios.

Sin embargo, lo novedoso de nuestro enfoque es la aplicación del concepto de libros interactivos dentro del marco de la enseñanza de aplicaciones interactivas. Como ya se ha indicado al

comienzo de esta Sección, hasta ahora no se ha hecho mucho énfasis en la manera en que los entornos de enseñanza de aplicaciones interactivas ofrecían sus servicios, fundamentalmente de cara a los usuarios más inexpertos. CACTUS presenta sus cursos tutores como si de libros tutores se trataran.

La mayor parte de los contenidos de estos libros son generados automáticamente por CACTUS, y los diseñadores sólo tienen que preocuparse de incluir en ellos los contenidos deseados de las unidades pedagógicas. La principal aportación de estos cursos es que pueden hacer un seguimiento de la actividad de los usuarios gracias a estar funcionando sobre la arquitectura ATOM, lo que posibilita el ofrecer a los usuarios explicaciones dependientes del contexto. Al igual que sucede con los libros de texto utilizados en la enseñanza tradicional, los usuarios aprenden objetivos y procedimientos mediante la lectura de los libros, la comprensión de los ejemplos descritos, y la práctica que se obtiene al realizar los ejercicios en ellos propuestos. Sin embargo, en el caso de los libros tutores de CACTUS, la práctica se lleva a cabo bajo el seguimiento automático del tutor. Esto se complementa al ofrecerse la posibilidad de practicar utilizando escenarios predefinidos como contexto para la realización de las actividades, además de poderse usar los propios contextos de trabajo de los usuarios. Para facilitar la navegación por los contenidos de los libros, se incorporan automáticamente en éstos enlaces hipertexto. Esta incorporación automática de vínculos libera a los diseñadores de tener que facilitar explícitamente dichos enlaces que, posteriormente, pueden ser refinados, añadidos o eliminados. Además de esta navegación mediante hiperenlaces, la interfaz de los libros tutores admite la navegación por sus páginas mediante los clásicos controles *Siguiente* y *Previo*, que permiten recorrer los libros interactivos siguiendo un orden secuencial. En una implementación comercial de este sistema, lo ideal sería implementar una interfaz táctil para facilitar al máximo la navegación del alumno.

Los libros tutores que se pueden generar con CACTUS mantienen una estructura muy similar a los libros de texto tradicionales, incluyendo así muchas de las partes que dichos libros tienen. En el resto de esta sección se describirá la estructura organizativa de un libro tutor de CACTUS, mientras que en secciones posteriores se explicará el funcionamiento de dichos cursos y su proceso de creación.

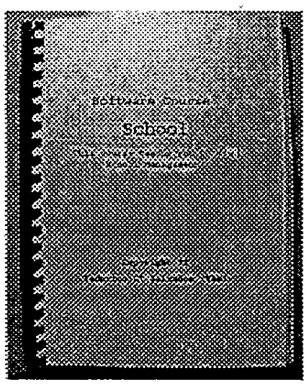

Figura 48: Portada generada para los libros tutores.

En primer lugar, CACTUS genera una portada para los libros, como se muestra en la Figura 48. Esta portada incluye dos campos principales, que son el título del curso y una descripción general

acerca de sus contenidos, a fin de aportar una idea sobre el propósito del curso. Ambos campos, título y descripción, son modificables por los diseñadores, como veremos posteriormente.

A continuación, CACTUS genera automáticamente un índice de las unidades pedagógicas del curso, siguiendo la ordenación que el diseñador ha utilizado para desarrollar dichas unidades, como se muestra en la Figura 49. Cada entrada del índice contiene el título de la unidad y una breve descripción de su contenido, siendo ambos modificables por el diseñador desde la propia interfaz del libro. Además, cada entrada del índice contiene un hiperenlace, generado automáticamente por CACTUS, a la página inicial de la unidad pedagógica asociada. Los contenidos de cada unidad pedagógica se describirán posteriormente.

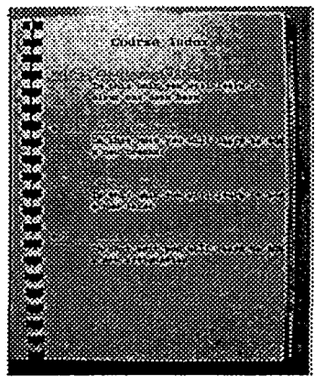

Figura 49: Índice generado para un libro tutor.

Una de las características principales de los cursos de CACTUS es que pueden ser realizados por los alumnos de distintas maneras. En este sentido, estos libros interactivos son similares a los libros tradicionales utilizados en cursos avanzados de algunas materias. Estos libros tienen habitualmente una manera por omisión de ser leídos, pero pueden ser seguidos de maneras alternativas, siempre y cuando se respeten las restricciones básicas de secuenciamiento asociadas a sus capítulos. Los cursos tutores de CACTUS incluyen la noción de secuenciamiento entre las diferentes unidades pedagógicas que los componen. CACTUS gestiona automáticamente este secuenciamiento, permitiendo a los usuarios llevar a cabo las actividades de las unidades pedagógicas dependiendo de si las actividades pertenecientes a sus unidades previas han sido realizadas anteriormente o no. La capacidad del sistema de seguir las acciones de los usuarios es lo que hace posible que CACTUS realice esta gestión de la ejecución de las unidades pedagógicas. Esta característica puede ser utilizada o no por los diseñadores, permitiéndose por tanto dar a los usuarios desde una completa libertad, hasta no darles ninguna alternativa cara al orden de seguimiento de las unidades pedagógicas. Como vamos a ver, las dos siguientes secciones de los libros tienen una estrecha relación con esta noción de secuenciamiento.

En primer lugar, el sistema genera una representación, en forma de grafo dirigido, de la estructura del curso tutor, como se muestra en la Figura 50. En este grafo, cada nodo representa una unidad pedagógica, y su color representa su estado. El estado de una unidad indica si ya ha sido ejecutada, si la unidad no ha sido aún ejecutada pero puede ser accedida en el momento actual, o si la ejecución de la unidad se encuentra bloqueada porque alguna de sus unidades predecesoras no ha sido ejecutada con éxito todavía. Los estados de las unidades son actualizados automáticamente por CACTUS según los usuarios realizan los cursos. Por su parte, en el grafo

existirá una flecha de un nodo A a otro B si la unidad A ha de ser ejecutada previamente a la unidad B. Puesto que la visualización de este grafo, implícito en la relación de secuenciamiento explicada anteriormente, muestra una visión del curso tutor más cercana a la que tienen los diseñadores de cursos que a la que tienen los alumnos, este grafo sólo es mostrado a los diseñadores, y no a los estos últimos.

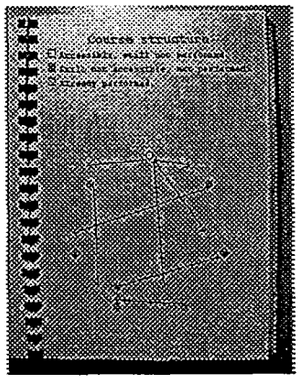

Figura 50: Grafo dirigido generado para mostrar la estructura de un libro tutor.

En segundo lugar, se genera automáticamente una hoja de propuesta con un orden de secuenciamiento para la realización del curso, como puede verse en la Figura 51. Siguiendo dicha ordenación, CACTUS garantiza que cuando se vaya a realizar la ejecución de una unidad pedagógica, ésta no se encuentre bloqueda porque una de sus predecesoras no se haya realizado previamente. Por lo tanto, el papel de esta sección es el de gestionar automáticamente y de manera transparente la noción de secuenciamiento de los cursos tutores de cara a los alumnos.

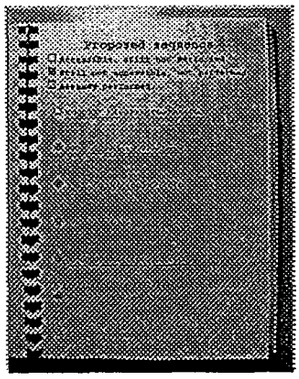

Figura 51: Ordenación propuesta para el seguimiento de un curso tutor.

Tras la hoja de propuesta se presentan tantos capítulos en el libro como unidades pedagógicas haya en el curso definido. Un ejemplo de una página de una unidad pedagógica puede verse en la Figura 52. Cada una de las unidades pedagógicas incorpora un conjunto de sesiones de enseñanza de tareas y, probablemente, cierto conjunto de ejemplos que permitirán ilustrar las explicaciones y un conjunto de escenarios que podrán ser ejecutados durante el curso. Desde el punto de vista de la presentación, se incluyen explicaciones sobre los objetivos de las sesiones de enseñanza,

sobre los ejemplos que pueden mostrarse y sobre los efectos de los escenarios que podrán ser ejecutados, así como imágenes estáticas y dinámicas, como veremos más adelante. Para los contenidos de las unidades pedagógicas, CACTUS generará explicaciones por omisión que indicarán a los alumnos las diferentes posibilidades incluidas dentro de cada unidad. La descripción de cómo se generan estas explicaciones, así como la manera en que los diseñadores pueden modificarlas, se verán durante la Sección 4.4.3.

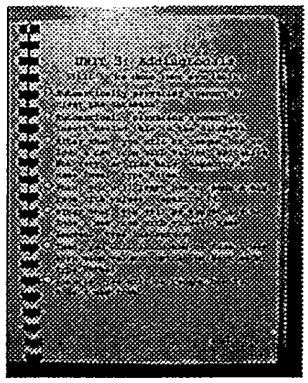

Figura 52: Aspecto de una unidad pedagógica de un libro tutor.

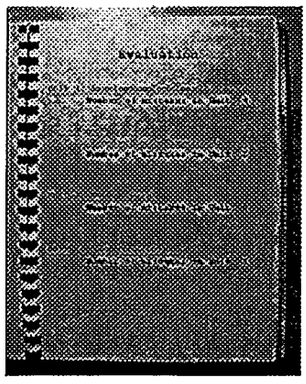

Figura 53: Sección de evaluación del rendimiento del alumno.

A continuación, CACTUS presenta una página de evaluación en la que se muestra un resumen de los resultados obtenidos durante la práctica del alumno, como puede observarse en la Figura 53. En esta página se muestra una entrada por cada unidad pedagógica del curso, acompañada por una indicación del desempeño del usuario durante dicha unidad. Esta indicación del rendimiento puede hacer referencia bien al número de errores acumulados por el usuario en cada unidad de manera global, bien al número de errores agrupados por tareas practicadas, o bien indicando con qué tarea se ha tenido un mayor grado de dificultad. Esta sección es generada y gestionada automáticamente por el sistema, sin necesidad de intervención ni del usuario ni del diseñador, quien únicamente deberá proporcionar el criterio de evaluación de cada unidad, si es que no desea utilizar el criterio por defecto del entorno (errores agrupados por tarea).

Finalmente, CACTUS genera automáticamente un glosario de tareas al final de los libros como el mostrado en la Figura 54, en el que se encuentran todas las tareas que son enseñadas durante el curso, ordenadas alfabéticamente. Cada entrada de este glosario relaciona una tarea con la lista de unidades pedagógicas en las que se puede aprender a realizar esa tarea y un hiperenlace a cada una de dichas unidades. Por tanto, el proceso de búsqueda de cómo realizar una tarea particular se reduce a una búsqueda en el glosario y el seguimiento de un hiperenlace. Este glosario de tareas es gestionado de manera transparente por CACTUS, de manera que los diseñadores no tienen que preocuparse ni por generarlo ni por actualizarlo.

En las siguientes dos subsecciones, se describirán los dos modos de ejecución de CACTUS: el *modo de ejecución* y el *modo de edición*.

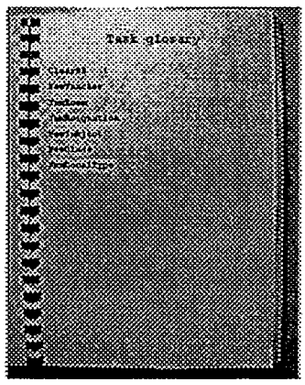

Figura 54: Glosario de tareas de un libro tutor.

4.4.2 Ejecución de los Cursos

Cuando CACTUS se encuentra en *modo de ejecución*, permite la ejecución de algunas de las partes de los libros, es decir, permite la transmisión interactiva de la información pedagógica asociada a los libros tutores a los alumnos. Este modo de funcionamiento está orientado tanto a la enseñanza como al diseño de los cursos. Así, mediante este modo se permite tanto enseñar a los alumnos cómo realizar ciertas tareas de usuario usando sus aplicaciones interactivas, como a los diseñadores probar y depurar los cursos en tiempo de desarrollo.

Algunas secciones de los libros tienen grupos de instrucciones asociadas. Podemos clasificar las instrucciones que pueden ser incluidas en los libros tutores en cuatro grupos distintos:

- El grupo primero y más importante incluye aquellas instrucciones que hacen que CACTUS se encargue de enseñar a realizar una tarea, aportar un ejemplo, ejecutar un escenario o mostrar un mensaje emergente. Es decir, este grupo engloba las instrucciones que representan actividades que tienen un reflejo inmediato sobre la actividad del usuario en cuanto a que le obligan a realizar algo o le ofrecen información de forma directa. El primer tipo, las instrucciones que enseñan a realizar una tarea, invocan directamente al módulo tutor DARTS que, siguiendo los procedimientos explicados en la Sección 4.2, se encargará de enseñar al alumno a llevar a cabo dicha tarea. Por su parte, las instrucciones de muestra de ejemplos lo que hacen es, utilizando la interfaz del *Módulo de Ejecución de Escenarios*, ejecutar un

escenario mínimo que contiene tres instrucciones: una que ejecuta utilizando animaciones la tarea que es pasada como parámetro al escenario, otra que avisa al usuario de que el ejemplo ya ha sido mostrado y de que se procederá a eliminar sus efectos, y una tercera que eliminará dichos efectos mediante una acción de *deshacer* (ver Sección 4.3.2.1). Las instrucciones de ejecución de escenarios tienen por objetivo preparar contextos adecuados que sean utilizados por las sesiones de enseñanza de tareas, y son, lógicamente, ejecutados mediante la interfaz del *Módulo de Ejecución de Escenarios*. Finalmente, los mensajes emergentes muestran *feedback* a los alumnos, utilizando para ello las clásicas ventanas *pop-up*, y que pueden mostrar información referente al contexto de las tareas e ir acompañados de parpadeos de objetos gráficos dentro de la aplicación.

- El segundo tipo incluye instrucciones necesarias para generación de los libros interactivos que representan a los cursos. Estas instrucciones no tienen un efecto directo sobre la actividad del usuario, sino sobre la interfaz gráfica de los libros interactivos con la que ellos interaccionan. Dentro de este grupo se incluyen las instrucciones para dar formato a los cursos, como descripciones, títulos, relaciones de precedencia entre sesiones pedagógicas, imágenes estáticas y dinámicas, nuevos hipervínculos incorporados por los diseñadores, etcétera.
- Un tercer grupo controla la manera en que el *Módulo de Ejecución de Escenarios* y el módulo de enseñanza DARTS y el propio CACTUS funcionan, incluyendo el grado de flexibilidad del módulo tutor (estricto o flexible), la velocidad de ejecución de los escenarios, el tipo de evaluación que se realizará en cada unidad pedagógica, el modo de depuración establecido, etcétera.
- Finalmente, el último grupo incluye estructuras de bajo nivel de control, como construcciones iterativas, bloques condicionales, bloques de ejecución aleatoria y definiciones o asignaciones de variables. Este tipo de instrucciones, en definitiva, se corresponde con instrucciones de bajo nivel de un lenguaje de programación concreto que es utilizado por CACTUS para representar internamente los contenidos de los cursos. Este tipo de instrucciones se tratará más adelante con mayor detalle, en la Sección 4.4.3 y en el Apéndice VI.

CACTUS incorpora un intérprete interno de ejecución del lenguaje de especificación asociado a los cursos. El funcionamiento de este intérprete es similar al funcionamiento de cualquier otro intérprete de algún lenguaje; es decir, se encarga de ir realizando las acciones asociadas a cada una de las instrucciones que se encuentra, siguiendo el flujo de control indicado por la especificación del curso.

En particular, las páginas correspondientes a las unidades pedagógicas suelen contener actividades de enseñanza de tareas, de muestra de ejemplos prácticos, instrucciones de ejecución de escenarios y mensajes de *feedback* a los usuarios, relacionados de manera adecuada mediante distintas estructuras de control. Mostraremos a continuación un pequeño ejemplo del contenido de una unidad pedagógica.

En un curso tutor realizado para *Schoodule* se tiene una unidad que explica cómo *añadir una nueva relación de asignación*. Lo que se desea hacer es, en primer lugar, preparar un escenario que añada profesores, asignaturas, grupos y aulas por omisión. La intención es que este escenario se ejecute como paso previo a la realización de la sesión de enseñanza relativa a la tarea de añadir la nueva relación de asignación. De esta manera, podemos asegurarnos de que el usuario no tiene ningún problema para añadir una nueva asignación durante su aprendizaje. Es decir, le estamos

facilitando un contexto de práctica adecuado. Tras ello, la sesión de enseñanza puede facilitar al alumno la posibilidad de ejecutar un ejemplo de la tarea de *añadir una nueva relación de asignación*. En caso de que el usuario desee ver el ejemplo, y con el fin de que dicho ejemplo no deje la aplicación en un estado indeseable, los efectos de la ejecución de este ejemplo deben ser deshechos. A continuación, la sesión pedagógica debe enseñar al alumno a *añadir una nueva relación de asignación*, de manera que practique siguiendo las instrucciones que se le facilitan. Finalmente, una vez que el usuario haya añadido con éxito una nueva asignación, es recomendable que el curso tutor muestre algún *feedback* al usuario sobre la tarea realizada. Con estas ideas en mente, lo que se hace para construir el curso que estamos describiendo es incluir una instrucción de ejecución de un escenario, a continuación una instrucción de ejecución de un ejemplo, seguida de una instrucción de enseñanza de una tarea y, finalmente, un mensaje informativo emergente. La ejecución de esta unidad pedagógica hace que CACTUS solicite al *Módulo de Ejecución de Escenarios* la ejecución del escenario que añade información por omisión a la base de datos de la aplicación; a continuación, pide al mismo módulo la ejecución de un ejemplo de *añadir una nueva relación de asignación* junto a su correspondiente acción de *deshacer*; después, DARTS enseñará al usuario a realizar la tarea de añadir una nueva asignación a la base de datos; y, finalmente, CACTUS muestra un mensaje emergente indicando los valores de la nueva asignación añadida. Por supuesto, el usuario puede decir al sistema que prefiere utilizar su propio contexto de trabajo, en vez de practicar con el escenario predefinido facilitado por el diseñador, así como decidir si quiere o no visualizar el ejemplo provisto por el curso tutor. Tras la ejecución del contenido de la unidad pedagógica, CACTUS actualizará automáticamente y conforme a las indicaciones proporcionadas por el diseñador la sección de evaluación del desempeño del alumno.

Cuando el usuario lleva a cabo satisfactoriamente los contenidos de una unidad pedagógica, CACTUS modifica automáticamente el estado de dicha unidad, habilitando el acceso a sus unidades sucesoras si fuera necesario. CACTUS muestra bajo el título de cada unidad su estado, que puede ser uno de los siguientes:

- *'Unidad accesible'*, establece que los contenidos de una unidad pedagógica pueden ser ejecutados cuando el usuario lo requiera;
- *'Unidad bloqueada'*, indica que el acceso a la ejecución de una unidad pedagógica está cerrado, pudiendo el alumno leer su contenido pero no ejecutarlo, ya que el contenido de alguna de sus unidades previas aún no ha sido realizado con éxito;
- *'Ya realizada'*, indica que la unidad pedagógica ya ha sido realizada con éxito por el usuario, y por tanto que sus unidades posteriores también lo han sido, o bien están accesibles; o
- *'En ejecución'*, lo que establece que el contenido de la unidad pedagógica que está siendo visualizada en ese momento está siendo ejecutado. Durante la ejecución de una unidad, CACTUS resalta en cada momento la instrucción que se está ejecutando, de manera que el usuario siempre conoce cuánto ha realizado y cuánto le queda por realizar para completar la unidad didáctica.

Finalmente, la hoja de propuesta también puede ser ejecutada, y su ejecución significa la ejecución, según el orden mostrado por la propuesta, de todas las unidades pedagógicas del curso. Ejecutando la hoja de propuesta, CACTUS hace que las relaciones de precedencia establecidas entre las distintas unidades pedagógicas de los cursos por el diseñador sean completamente transparentes a los usuarios, de modo que éstos nunca se encontrarán tratando de ejecutar una

unidad bloqueada ni se tendrán que preocupar por navegar por el libro buscando cuál debe ser la siguiente unidad pedagógica a ejecutar. Una vez ejecutadas todas las unidades, el sistema conduce al alumno a la sección de evaluación del curso, en la que puede consultar sus resultados.

El resto de secciones de los libros interactivos sólo incluyen instrucciones referentes a la disposición espacial de las componentes gráficas de las secciones de los libros, así como instrucciones internas utilizadas por el gestor de cursos para la generación de los libros interactivos asociados a los mismos.

CACTUS también incorpora la opción de guardar el estado de ejecución de un curso, para que más tarde pueda ser recuperado y continuar con su realización.

4.4.3 Especificación de Cursos

Como se comentó al comienzo de la Sección 4.4, CACTUS tiene dos modos de funcionamiento, el *modo de ejecución* y el de *edición*. Anteriormente hemos descrito el primero, y durante esta sección describiremos el segundo. El *modo de edición* de CACTUS tiene por objetivo el asistir a los diseñadores durante la fase de creación y modificación de los cursos tutores.

Desde el punto de vista de los alumnos, los cursos de CACTUS son libros interactivos que incluyen características avanzadas que permiten enseñarles a realizar tareas de usuario y hacer un seguimiento de dichas actividades. Algo similar sucede para los desarrolladores noveles de cursos, que sólo necesitan interactuar con la interfaz de CACTUS en *modo de edición* para crear los cursos de manera interactiva. Sin embargo, internamente, los cursos de CACTUS vienen definidos en ficheros de texto mediante un lenguaje de programación que es interpretado por la herramienta en el *modo de ejecución* del entorno. Sólo los diseñadores avanzados pueden necesitar acceder a esta representación interna de los cursos con el fin de sacar partido a toda la funcionalidad del sistema. La representación que CACTUS muestra de los cursos no es, por tanto, la misma que internamente tienen, sino una que el propio entorno genera para hacer los tutores más comprensibles a los usuarios y diseñadores.

En las próximas dos subsecciones vamos a ver cómo los diseñadores pueden crear los cursos, describiendo primero brevemente la manera en que se organizan internamente las especificaciones textuales de los cursos para, en la siguiente sección, ver cómo los diseñadores pueden generar éstas especificaciones por medio de técnicas no programativas utilizando el *modo de edición*.

4.4.3.1 Lenguaje de Especificación de Cursos

Como ya se ha mencionado, los cursos tutores de CACTUS están representados finalmente mediante un lenguaje de programación. Este lenguaje de especificación es interpretado en tiempo de ejecución por un intérprete, que se encargará tanto de la generación de la interfaz de libros tutores asociada a los cursos, como de llevar a cabo las acciones indicadas dentro de las sesiones pedagógicas de los cursos, y que van encaminadas a realizar la enseñanza de las tareas de la aplicación interactiva a la que el curso va destinada. En esta Sección se describe someramente el lenguaje de especificación de cursos empleado por CACTUS, lo que se hará mostrando la especificación de un pequeño curso para *Schoodule*. Una referencia completa de este lenguaje puede encontrarse en el Apéndice VI, al final del presente documento. Para un ejemplo más detallado que el descrito en esta sección, consulte el Apéndice VII.

La estructura general interna de un curso es la mostrada en la Figura 55. Como hemos visto, CACTUS se encarga de mostrar automáticamente, en forma de libros interactivos, esta representación interna, incluyéndose no sólo la generación automática de la mayoría de las partes de los libros, sino también de la disposición general de las páginas.

Figura 55: Estructura general de un curso interactivo de CACTUS.

Como podemos ver, el lenguaje de descripción de cursos de CACTUS está basado en etiquetas que describen el tipo de acción a realizar y parámetros que afectan al comportamiento de dichas acciones. En cuanto a la manera en la que dichas etiquetas se organizan a lo largo de un curso, éstos están formados por tres componentes básicos: una zona de instrucciones de inicialización del curso, otra zona de descripciones de las unidades pedagógicas, y una zona de cierre del curso.

- La zona de instrucciones de inicialización del curso incluye todas aquellas instrucciones que son ejecutadas durante el comienzo del curso, antes de que el usuario llegue a ejecutar ninguna unidad pedagógica. En esta zona también se puede asignar valores a todos aquellos parámetros que se deseen mantener constantes a lo largo de la ejecución de los cursos. Los casos más habituales de instrucciones de inicialización de los cursos atañen a la parametrización del funcionamiento de DARTS, del *Módulo de Ejecución de Escenarios* o del propio entorno CACTUS. En particular, de DARTS puede establecerse el grado de flexibilidad

empleado durante la enseñanza (etiqueta MODE**). Por su parte, del *Módulo de Ejecución de Escenarios* puede proporcionarse la velocidad con la que los escenarios se visualizan (REFRESHMODE). Y, finalmente, del propio CACTUS pueden modificarse los atributos de modo de depuración utilizado (DEBUG) o del tipo de evaluación de los conocimientos que va a realizarse al alumno (EVAL). Otra instrucción que tiene sentido incorporarse en esta zona es información genérica relacionada con la disposición gráfica de los libros, como la imagen que se situará en el fondo del libro (IMAGE) o alguna otra información adicional que en un futuro pudiera implementar el sistema, como sería el tipo de letra que se usase o los textos de encabezado y pie de página.

- En cuanto a la zona de instrucciones correspondiente a las unidades pedagógicas, los cursos de CACTUS pueden incluir tantas unidades como el diseñador desee. Cada una de ellas puede indicar la relación de precedencia entre dicha unidad y las que deben ser sus predecesoras en la ejecución del curso (PREVIOUS). Pero, además de esta relación de precedencia, los contenidos más habituales de las unidades pedagógicas son actividades de enseñanza (TUTORING), ejemplos (SAMPLE), ejecuciones de escenarios (SCENARY) y mensajes de *feedback* a los usuarios (MSG...HIGHLIGHT). Éstos últimos pueden incorporar referencias a objetos gráficos de la aplicación sobre la que ofrecen la guía. Normalmente, este tipo de instrucciones van ligadas mediante estructuras de control de flujo de ejecución, como condicionales (IF...ELSE...ENDIF), iteraciones (WHILE...ENDWHILE, DOWHILE...ENDDO, FOR...FROM...TO...STEP...ENDFOR, FOR...IN...ENDFOR), y bloques de ejecución aleatoria (ALEAT...BLOCK...ENDBLOCK...ENDALEAT). También pueden utilizarse variables (VARIABLE y SET) para, por ejemplo, establecer relaciones entre valores de parámetros de distintas tareas o valores relativos a las distintas iteraciones de un bucle. Otro tipo de instrucciones muy comunes se refiere a la inclusión de imágenes, tanto estáticas como dinámicas, así como hiperenlaces. La imágenes estáticas (IMAGE) se corresponden con mapas de bits, mientras que las dinámicas (DYNAMAGE) son instancias de objetos gráficos de la aplicación, que son dinámicamente replicadas por CACTUS durante la ejecución de los cursos y situadas en la interfaz del libro. Por su parte, los hiperenlaces (LINK) permiten a los diseñadores ofrecer nuevos caminos de navegación aparte de los que el sistema genera automáticamente para los libros, así como incluir referencias a fuentes de información externas. Además de todo lo anterior, dentro de los contenidos de las unidades didácticas pueden encontrase todas aquellas instrucciones que podemos incluir en la zona de inicio y finalización de los cursos, con el matiz de que, en el supuesto de incorporar alguna de estas instrucciones, el ámbito de permanencia de su efecto será la unidad pedagógica en la que se utilice.
- Finalmente, la parte de cierre de los cursos tiene como fin el incluir aquellas instrucciones que queramos que sean ejecutadas tras la realización, por parte del alumnos, de todas las unidades didácticas del curso. Normalmente, las instrucciones situadas en esta zona del curso proporcionarán *feedback* a los alumnos sobre sus resultados (TUTORESULT), les aportarán alguna referencia para continuar posteriormente su aprendizaje, etcétera.

** Durante esta subsección se utilizarán palabras con letras mayúsculas entre paréntesis para informar sobre el nombre de la etiqueta a la que la explicación se refiere.

4.4.3.2 Especificación Interactiva de Cursos

Como ya hemos comentado, los libros generados con el entorno que aquí describimos pueden ser modificados utilizando técnicas de manipulación directa [Shneiderman83], ya que CACTUS permite utilizar para ello técnicas de programación visual y programación por demostración [Cypher93]. El esquema general de funcionamiento de CACTUS es el mostrado en la Figura 56. En esta figura podemos apreciar que ni los diseñadores de los cursos ni sus usuarios finales necesitan acceder directamente a los ficheros de texto que sirven de soporte a los cursos, sino que simplemente interaccionan con la interfaz de CACTUS para crear y utilizar los cursos, respectivamente. Para acceder a esta funcionalidad de creación y modificación de los contenidos de los cursos, en esta sección vamos a describir el *modo de edición.*

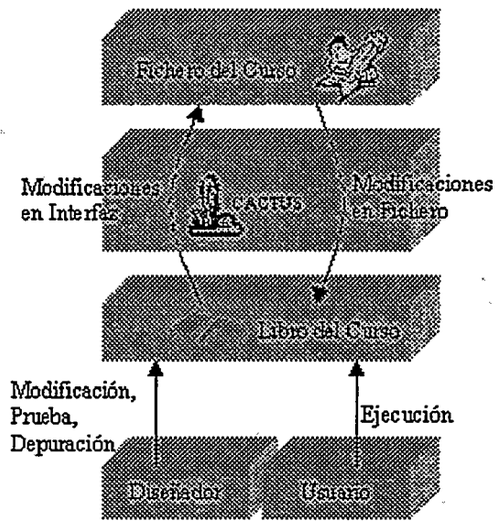

Figura 56: Interfaz para la utilización del entorno de gestión de cursos interactivos.

Las operaciones para especificar cursos pueden reducirse a la adición y eliminación de las distintas funcionalidades que CACTUS proporciona a sus cursos. Dichas funcionalidades vienen dadas, finalmente, por instrucciones de un lenguaje de definición de cursos, pero normalmente no necesitan ser conocidas más que por los desarrolladores avanzados.

Para el caso más sencillo, el de eliminar funcionalidades, el diseñador selecciona los elementos del curso que desea suprimir a partir de la representación textual del mismo, y activa la opción de *Eliminar*. Para añadir nuevas funcionalidades, el proceso es interactivo y dependiente del tipo de funcionalidad que se desee añadir, como veremos a continuación.

Se pueden insertar instrucciones que representen actividades para que practique el usuario de dos manera distintas. En la primera de ellas, el diseñador utiliza el menú *'Insertar sesión de enseñanza'*. A continuación, el diseñador seleccionará la tarea deseada de una lista que contiene todas las tareas que están especificadas en el modelo de tareas de la aplicación para la que el curso tutor se esté desarrollando. Después, el diseñador será preguntado por si desea que la enseñanza de dicha tarea se realice haciendo que alguno o algunos de los valores de los parámetros asociados a dicha tarea cumplan alguna determinada condición. Estas condiciones

pueden incluir valores constantes, expresiones calculadas en tiempo de ejecución o valores aleatorios calculados dinámicamente por el sistema. Por ejemplo, el diseñador puede seleccionar la tarea AñadirProfesor y forzar al alumno a que el nombre del profesor sea *"Mary Brown"*. La otra manera alternativa en que se pueden añadir actividades de enseñanza es utilizar el menú *'Insertar enseñanza sobre...'*, una vez la aplicación se encuentre en ejecución. Esta opción hace que CACTUS vigile y registre todas la tareas que el diseñador realice a continuación con la aplicación. Así, CACTUS insertará tantas instrucciones de enseñanza de tareas como tareas realice el diseñador hasta que éste utilice el menú *'Finalizar inserción de enseñanza'*. Por ejemplo, si el diseñador añadiera un nuevo profesor, CACTUS generará automáticamente una instrucción de enseñanza para la tarea *AñadirProfesor*. Para este modo de especificación de tareas a enseñar, CACTUS utilizará los valores que hayan adoptado los parámetros durante el ejemplo del diseñador para asociar valores a la instrucción de enseñanza. Así, si en el ejemplo provisto por el diseñador, éste inserta un profesor cuyo nombre es *"Mary Brown"*, CACTUS insertará una instrucción que enseñe a insertar un profesor con dicho nombre. Para dotar de mayor flexibilidad al entorno de especificación, el sistema ofrecerá al diseñador la posibilidad de eliminar o modificar las restricciones obtenidas sobre los valores de los parámetros, de modo que el diseñador pueda modificar la instrucción anterior para convertirla en una que enseñe a insertar un profesor, cuyo nombre sea el que el alumno desee en el momento de ejecutar el curso.

Cabe mencionar la independencia existente entre los procedimientos utilizados por los diseñadores para realizar las tareas y los que utilizaran posteriormente los alumnos a la hora de practicar. Por ejemplo, el diseñador puede utilizar opciones de menú para ejecutar determinados acciones que forman parte de tareas de más alto nivel, mientras que los usuarios podrán realizar las mismas tareas mediante la pulsación de botones. Esta independencia se consigue como consecuencia de que CACTUS está basado en modelos jerárquicos de tareas de la aplicación y no sólo en secuencias de eventos de bajo nivel. De este modo puede observar las acciones del usuario desde un punto de vista más abstracto que el basado únicamente en eventos lineales.

Para introducir una instrucción que ejecute un escenario, el diseñador puede utilizar el menú *'Insertar escenario ya definido...'*, que le permitirá seleccionar el escenario que desee que sea ejecutado desde cualquier librería de escenarios. De manera alternativa, puede escoger la opción de usar el menú *'Generador de Escenarios'*, que arrancará EASE, la herramienta de definición interactiva de escenarios descrita en la Sección 0, e insertará una referencia al escenario creado mediante dicha herramienta.

En el caso de los ejemplos, como hemos comentado anteriormente, éstos vienen definidos internamente como escenarios mínimos que contienen una instrucción de ejecución de la tarea, otra que avisa al usuario de que el ejemplo ya ha sido mostrado y de que se procederá a eliminar sus efectos, y una tercera que eliminará dichos efectos mediante una acción de *deshacer*. Para incorporar un ejemplo a un curso interactivo, los diseñadores tienen dos opciones. Por un lado, pueden utilizar el menú *'Insertar ejemplo...'* y, a continuación, seleccionar la tarea de ejemplo de una lista de tareas proporcionando los valores de los parámetros. La otra opción es acceder a *'Insertar ejemplo a definir'*, que permitirá al diseñador especificar la tarea que desea usar como ejemplo realizándola interactivamente. Dado que los ejemplos son casos especiales de escenarios, en ambos casos CACTUS incluirá en el curso una referencia al escenario indicando la tarea que se utilizará como ejemplo.

Para el resto de construcciones del lenguaje, como condicionales o iteraciones, el diseñador seleccionará la construcción que desea insertar y, dependiendo de su tipo, facilitará los

parámetros necesarios cuando CACTUS se los pregunte. Puesto que CACTUS analiza las instrucciones en tiempo de diseño del curso, gran parte de los errores son identificados durante la creación del curso, ahorrando así tiempo al diseñador. El resto de construcciones del lenguaje incluye mensajes emergentes con referencias a objetos gráficos, imágenes estáticas y dinámicas, asignaciones de variables, tipos de evaluación, modificaciones del comportamiento de DARTS y del *Módulo de Ejecución de Escenarios*, hipervínculos añadidos por el diseñador y estructuras de control de flujo.

El diseñador también puede añadir o eliminar unidades pedagógicas, y modificar interactivamente las relaciones de secuenciamiento existentes entre ellas. Estas operaciones fuerzan al sistema a que modifique ciertas secciones de los libros interactivos, tales como el índice, el grafo de precedencia, la hoja de propuesta, la evaluación o el glosario, a fin de mantener la consistencia entre las distintas partes de los libros. CACTUS genera automáticamente el aspecto de las nuevas unidades cuando son creadas, y los diseñadores pueden modificar interactivamente sus títulos y descripciones. El sistema, por omisión, realiza algunas labores tediosas, como ajustar los distintos contenidos de las páginas para que no se solapen textos con figuras, escalar correctamente imágenes, actualizar en tiempo de ejecución los gráficos dinámicos, o generar enlaces hipertexto.

Finalmente, explicaremos cómo CACTUS genera automáticamente las explicaciones presentes en las páginas de los libros tutores a partir de los contenidos de la unidades pedagógicas especificadas en los cursos. Estas explicaciones son necesarias para que el alumno no se enfrente a las descripciones internas de las tareas o a las etiquetas propias del lenguaje de especificación subyacente. CACTUS genera descripciones dependientes del contexto solamente para las instrucciones de enseñanza de tareas, ejemplos e instrucciones de ejecución de escenarios, y permite a los diseñadores modificar dichas descripciones por omisión de manera interactiva. Para actividades de enseñanza de tareas, se generan descripciones del tipo '*A continuación, usted tendrá que* ' seguidas con la descripción de la tarea, que se obtiene del modelo de tareas de la aplicación. Para las referencias a ejecuciones de escenarios, la descripción es obtenida concatenando el mensaje '*Preparando escenario para* ' con la descripción del escenario, que es extraída a partir de la especificación de dicho escenario. En el caso de los ejemplos, los alumnos podrán leer el mensaje '*Seguidamente se le mostrará un mensaje sobre cómo* ' concatenado con la explicación de la tarea. Puesto que las descripciones generadas automáticamente utilizan información proveniente de diferentes fuentes, es habitual que dichas descripciones no sean lo suficientemente atinadas como para ser mostradas directamente a los alumnos. Es decir, estas descripciones generadas automáticamente son cercanas a descripciones útiles, pero con frecuencia suelen tener fallos gramaticales o, simplemente, no son del agrado de todos los diseñadores. Por ello, CACTUS permite que los propios diseñadores modifiquen interactivamente dichas descripciones utilizando la propia interfaz del libro del curso y, posteriormente, guardar dichos cambios.

Desde el punto de vista de la depuración de los cursos, CACTUS ofrece varios niveles de apoyo. Estos grados varían desde el más simple, permitir a los diseñadores realizar una traza de la ejecución de los cursos, hasta la posibilidad de realizar una ejecución paso a paso de los cursos tutores, pudiéndose establecer puntos de ruptura, así como inspeccionar y modificar los valores de las variables de los tutores durante su ejecución.

4.4.4 Beneficios

Durante toda esta Sección 4 hemos descrito lo que supone el cuerpo del trabajo. Ya hemos descrito tanto la arquitectura propuesta, como sus componentes, su interfaz y su funcionalidad. En lo que resta de Sección vamos a completar la descripción detallada de CACTUS con un apartado en el que condensamos las principales ventajas que se consiguen con la utilización de este entorno de cara a la enseñanza del uso de aplicaciones.

Como hicimos anteriormente, esta sección la dividimos en dos para mostrar por separado las ventajas orientadas a los usuarios finales, es decir, los alumnos, y las ventajas orientadas a los desarrolladores del material de formación.

4.4.4.1 Ventajas para el Usuario

Como ocurre con todo sistema enfocado a enriquecer la interacción de los usuarios con los sistemas, las ventajas primordiales son aquellas que están orientadas a sus usuarios finales. En el caso que nos ocupa, los usuarios finales de nuestro entorno son tanto los desarrolladores de los contenidos de los cursos tutores como los alumnos que se enfrentarán a las sesiones de enseñanza. Sin embargo, creemos que la naturaleza del sistema obliga a cuidar fundamentalmente a estos últimos, pues por mucho que se facilite el desarrollo de sistemas didácticos, el objetivo prioritario es que después éstos cumplan el objetivo perseguido. En nuestro caso, no es otro que conseguir reducir las barreras que se les presentan a los usuarios noveles cuando se ponen a trabajar enfrente de una herramienta informática interactiva. A continuación describimos las principales ventajas que los cursos basados en la tecnología de CACTUS ofrecen a los alumnos:

- En primer lugar, y por estar CACTUS apoyado en el soporte de enseñanza de tareas que DARTS proporciona, durante la enseñanza individual de cada tarea se obtiene la totalidad de las ventajas de cara a los alumnos que se describieron en la Sección 4.2.7.1. A título de recordatorio rápido, estas ventajas eran el ofrecimiento de una guía dinámica conforme los alumnos llevaban a cabo las tareas, la aportación de un *feedback* visual sobre el avance de la realización de sus tareas, sobre los objetos gráficos de la aplicación con los que los alumnos tienen que interactuar y sobre el rendimiento de sus acciones, la atenuación del *problema de asociación* por el uso de modelos jerárquicos, el uso de varios posibles niveles de flexibilidad durante la práctica, la inclusión de información contextual en las explicaciones, la resolución del *problema de visualización* y la utilización de la propia aplicación, y no una réplica o maqueta de la misma, durante las sesiones prácticas.
- La utilización de ejemplos como parte de los cursos didácticos proporciona a los alumnos un refuerzo de la enseñanza de los procesos. Pese a que el uso de estas animaciones es valioso, la utilización aislada de animaciones como soporte de guía no proporciona un buen sistema de apoyo. Se deben, por tanto, proporcionar también explicaciones mediante otros medios además de las animaciones para ayudar a los usuarios a generalizar los conceptos que se tratan de mostrar.
- CACTUS proporciona un mecanismo mediante el cual puede estructurarse la enseñanza de familias de tareas, relacionadas entre sí por determinados criterios. La agrupación de varias sesiones pedagógicas de tareas en unidades pedagógicas, y la agrupación de éstas formando libros interactivos de enseñanza, permite estructurar los contenidos de los cursos para la enseñanza a los usuarios noveles. La utilización de la metáfora de representar los cursos

interactivos como si de libros se tratara permite a los alumnos disfrutar de una interfaz familiar e intuitiva durante el aprendizaje.

- La interfaz de los libros interactivos es común a todos los cursos tutores generados con CACTUS. Esta homogeneidad los convierte en herramientas sencillas de utilizar para los alumnos, que solo deben aprender a utilizar una vez el sistema de enseñanza. Por su parte, la idea de utilizar como metáfora de representación de los cursos los libros interactivos permite a los alumnos comenzar a utilizar inmediatamente el sistema, ya que desde tempranas edades todas las personas están habituadas al manejo de libros impresos.
- Los libros interactivos proporcionan los contenidos de los cursos a los alumnos. Por lo tanto, en este modelo de guía no son los alumnos los que de forma explícita deben buscar ayuda sobre la realización de determinadas tareas, sino que el propio diseñador de los cursos es quien se encarga de planificar qué procesos van a enseñarse en los cursos y, posiblemente, cuáles no. Se trata, por lo tanto, de un entorno dominante que, como vimos anteriormente, se adapta especialmente bien a las necesidades de los usuarios noveles. Estos usuarios tienen especiales dificultades con el acceso a los sistemas de guía, de manera que lo que buscan es un entorno que les introduzca en el manejo de la aplicación de manera proactiva, proporcionándoles conocimientos generales del uso de la herramienta sin que ellos tengan que enfrentarse específicamente a un problema de utilización [Nass95]. Además, la orientación a tareas de las explicaciones también va especialmente enfocada a este perfil de usuario, que se encuentra más cómodo cuando recibe explicaciones de esta naturaleza [Paris89].
- La herramienta, en su *modo de ejecución*, gestiona automáticamente, y de manera transparente al usuario, las restricciones de secuenciamiento que el diseñador de los cursos ha establecido. Gracias al seguimiento de la actividad de los cursos que realiza el sistema, los usuarios no deben preocuparse de seguir los cursos de manera que se respeten las relaciones de orden de ejecución marcadas, sino que es el propio entorno quien guía a los alumnos durante el aprendizaje a través de las distintas unidades didácticas. De este modo, con esta gestión automática del material didáctico que se proporciona, los alumnos pueden aprovechar la riqueza de posibilidades que esta noción de secuenciamiento proporciona sin tener que preocuparse de gestionar por si mismos el orden en que siguen los cursos.
- Finalmente, el entorno lleva a cabo una evaluación del desempeño de los alumnos durante las sesiones prácticas. Esta evaluación, aunque simple, proporciona a los alumnos un nivel de *feedback* adicional mediante el cual pueden saber si sus conocimientos son los adecuados para comenzar a realizar su trabajo.

4.4.4.2 Ventajas para el Diseñador

Durante la Sección 2.1 vimos que los mayores problemas que los diseñadores de las componentes de guía de los sistemas se encontraban se debían al hecho de que en los entornos tradicionales existía poca o ninguna integración entre los procesos del desarrollo de la aplicación y el desarrollo de la componente de guía. Fruto de ésto se obtienen altos costes para el desarrollo de estas componentes.

Vimos también que estos altos costes se daban durante todo el ciclo de vida de las aplicaciones. Es decir, no sólo era muy costoso el ofrecer una primera versión que enseñara en la utilización de la herramienta, sino que el mantenimiento y todas las subsiguientes versiones del *software* tenían

que realizarse sobre la componente de guía de manera independiente. Esta actualización no solo suponía unos altos costes de desarrollo, sino que, además, implicaba unos riesgos de consistencia entre la funcionalidad del *software* y los contenidos didácticos que hacía de este proceso una labor cara y complicada.

La solución descrita en este trabajo para la generación del material didáctico parte del desarrollo de las aplicaciones interactivas a partir de la modelización de sus interfaces. En particular, se utilizan modelos de tareas para representar las tareas que los usuarios pueden llevar a cabo con las aplicaciones. En estos modelos nos basamos para proporcionar la dependencia entre la componente de guía que se genera y el desarrollo de las aplicaciones.

- Por un lado, la especificación de los modelos de tareas es importante durante el desarrollo de las aplicaciones, como se describió en la Sección 2.2, particularmente por los servicios de valor añadido que son capaces de aportar a las aplicaciones. En nuestro caso, los modelos permiten a DARTS generar automáticamente una componente de guía avanzada basada en las tareas que definen.
- La especificación de aplicaciones basada en modelos proporciona también un marco genérico para la generación de cursos. Es decir, sin importar el objetivo de la aplicación que se desarrolle o su funcionalidad es posible la entrega de cursos interactivos que enseñan a utilizarla, todos ellos con una interfaz homogénea. Esto no se da en otros sistemas menos generales, como los descritos en la Sección 4.4, que están orientados a sistemas muy específicos, necesitando adaptaciones para aplicarse a otros sistemas.
- Así mismo, CACTUS reduce los costes de mantenimiento de los cursos, puesto que la mayor parte de estos costes residen en la modificación de los modelos de tareas de las aplicaciones. Con frecuencia, estas modificaciones pueden ser realizadas en ATOMS mediante técnicas visuales y de programación por demostración (Sección 3.2.2). Además, de manera sencilla CACTUS puede gestionar qué tareas del modelo de tareas de una aplicación se enseñarán en el curso que está siendo creado y cuáles no, de modo que puede avisar al diseñador de cuándo una unidad debería ser modificada porque trata de enseñar una tarea que no está modelizada, y de igual modo puede alertar de qué tareas deberían añadirse a los cursos.
- Al igual que las herramientas TMT y EASE de especificación de modelos y escenarios, CACTUS soporta técnicas de especificación de los contenidos de los cursos basadas en ejemplos. Estas técnicas permiten a los diseñadores de los cursos el enfrentarse a la generación de un curso sin necesidad de tener elevados conocimientos de programación, simplemente utilizando técnicas interactivas. Esta capacidad, fundamental, diferencia este sistema de otros en los que son necesarios grandes conocimientos informáticos, lo cual redunda en una barrera a la creación del material didáctico.
- Aunque los cursos vayan orientados fundamentalmente a usuarios noveles, CACTUS permite que los cursos incorporen contenidos de muy diversa complejidad. Para facilitar el que dentro de un curso existan contenidos de dificultad notablemente diferenciada, se ha introducido la noción de secuenciamiento entre las distintas unidades pedagógicas de los cursos. Esto permite que los cursos generados no hayan de tener necesariamente una ejecución lineal, incorporándose restricciones de ejecución para las distintas unidades, en función del estado de ejecución del resto. Al igual que sucede con la especificación de la práctica totalidad de los contenidos de los cursos, estas relaciones de precedencia en la realización de las distintas

actividades se especifican de manera completamente interactiva y, por lo tanto, no suponen un peso durante el desarrollo. Además, CACTUS hace transparente al usuario final la correcta gestión de estas restricciones, de manera que ellos no necesitan preocuparse de conocer cómo están estructurados los cursos, sino que simplemente siguen las instrucciones del entorno, que será quien se ocupe de gestionarlas.

- La generación automática que realiza el sistema de las explicaciones sobre el contenido didáctico de los cursos, unida a la flexibilidad que el entorno proporciona para la modificación de las mismas, confiere a CACTUS un gran potencial. Así, por un lado se reducen al máximo los costes de desarrollo de los contenidos, pues las descripciones realizadas por omisión liberan a los diseñadores de buena parte de la carga de trabajo. Por otro lado, se permite a los desarrolladores la total personalización de las explicaciones, pudiéndose así proporcionar contenidos con niveles de calidad máximos.
- Finalmente, el entorno ofrece facilidades de depuración que asisten a los diseñadores durante el desarrollo de los cursos. Pese al esfuerzo que se ha realizado, orientado a facilitar en la mayor medida posible el conocimiento de programación requerido por los diseñadores durante la generación de los cursos, éstos en ocasiones contendrán errores lógicos cuya detección puede llegar a ser complicada. Así, hemos completado la gama de ayudas a los diseñadores con la incorporación de mejoras que permiten desde hacer una traza de los cursos, hasta inspeccionar y permitir alterar los valores de los distintas variables y objetos del sistema. Proporcionar estos servicios ha sido posible merced a las capacidades aportadas por el propio entorno de desarrollo AMULET en lo referente a las capacidades de auto-descripción de los objetos ORE en tiempo de ejecución. Con todo ello, creemos que se ha conseguido facilitar de manera muy notable la generación de los cursos utilizando nuestro entorno.

5 Pruebas Realizadas

La herramienta CACTUS ha sido probada en la generación de cursos de aprendizaje para varias aplicaciones de prueba que han sido desarrolladas. Como vamos a ver, algunas de estas aplicaciones han sido construidas partiendo de cero, mientras que otras han sido generadas automáticamente por un compilador de modelos de simulación digital continua. Lo que sí es común a todas estas aplicaciones es que todas ellas utilizan como base el *framework* AMULET que hemos descrito.

En la mayor parte de los casos, una vez implementadas las aplicaciones de prueba, se han especificado sus modelos de tareas. Para ello, se ha utilizado TMT como herramienta de modelización, teniendo que recurrir al acceso directo al lenguaje de modelización de tareas en muy pocas ocasiones, normalmente sólo para especificar las relaciones de paso de parámetros entre tareas. En los casos en los que las aplicaciones son generadas automáticamente, también lo son sus modelos de tareas, por lo que en este caso no se ha precisado el uso de TMT.

Las aplicaciones de pruebas que se han implementado son las que se describen a continuación:

- En primer lugar, se han implementado dos interfaces interactivas para simulación digital continua de sistemas. Ambas interfaces se han obtenido a partir de la especificación de un modelo de simulación digital continua en el lenguaje OOCSMP [Alfonseca97], tras ser compilados los sistemas correspondientes con el compilador de este lenguaje. En particular, estas dos simulaciones son del sistema solar interno y las ecuaciones de Volterra [Alfonseca98]. En la Figura 57 podemos observar ambas aplicaciones de prueba ejecutándose, mientras que el Apéndice VIII se encuentra descrita de modo más detallado una de estas interfaces.

  En ambas interfaces, el usuario puede cambiar los parámetros de los objetos que intervienen en la simulación, los valores de las variables globales de la simulación, los valores de los parámetros de simulación, y añadir y quitar objetos en la simulación -en estos casos, planetas o nuevos animales a la pirámide alimenticia.

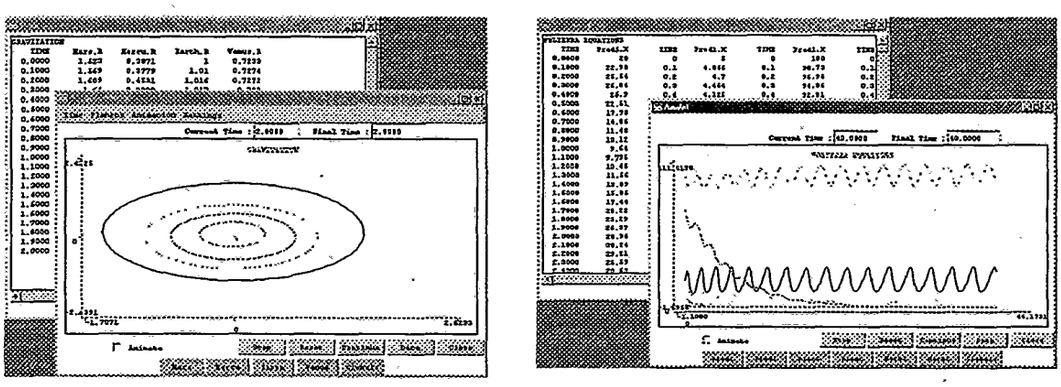

Figura 57: Interfaces de simulación digital continua utilizadas como pruebas.

- También se ha implementado un prototipo de agenda electrónica mediante el cual sus usuarios pueden tener acceso a un calendario en el que registrar sus citas. La interfaz de

usuario de esta aplicación la tenemos en la Figura 58. A diferencia de los casos anteriores, mientras que en aquellos el propio compilador de OOCSMP nos asistió generándonos automáticamente los modelos de tareas de las aplicaciones, en el caso de esta aplicación de prueba se ha definido el modelo de tareas mediante TMT a posteriori.

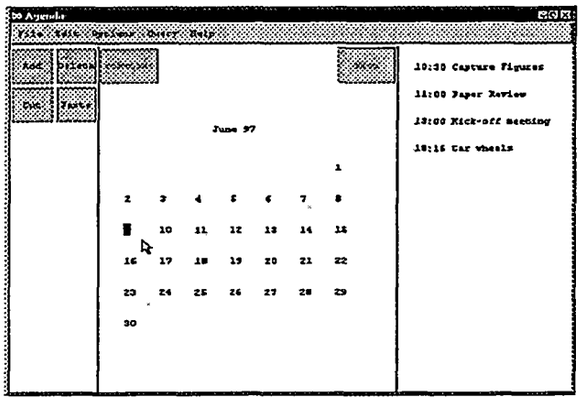

Figura 58: Agenda electrónica implementada para probar la generación de cursos.

- Una tercera aplicación de pruebas que se ha construido ha sido OOPI-TasKAD, una aplicación de diseño gráfico orientada a objetos. Podemos observar la interfaz de esta herramienta en la Figura 59.

  La mayor particularidad de esta aplicación es su funcionamiento que está basado en el paradigma de herencia prototipo/instancia. Gracias a ello, OOPI-TasKAD permite realizar diseños relativamente complejos. Cada nuevo objeto que se crea en un diseño de OOPI-TasKAD es un prototipo potencial para la creación de nuevas instancias suyas. Una vez se han creado instancias de un prototipo de objeto, se pueden aplicar operaciones tanto a las instancias de manera individual, como a todas las instancias, de manera genérica, si modificamos los atributos del objeto del que heredan. En definitiva, el modelo de herencia tiene las mismas propiedades que ORE, el sistema de gestión de objetos de AMULET.

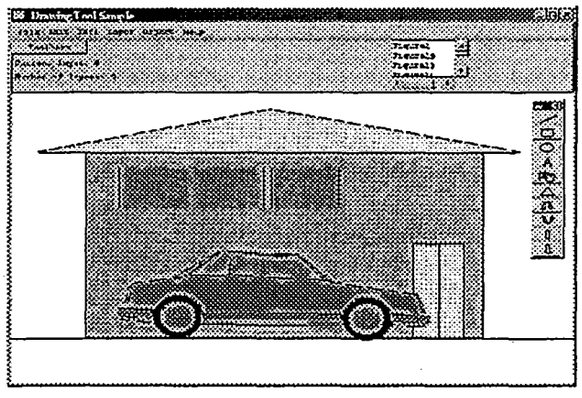

Figura 59: OOPI-TasKAD, aplicación de prueba para realizar diseños.

- Finalmente, CACTUS también se ha probado utilizando *Schoodule*, una aplicación de planificación de horarios en escuelas que hace uso de una base de datos y un resolvedor de restricciones para generar horarios escolares, y de la que se ha tratado ampliamente a lo largo de este trabajo.
- Además, también se ha generado un curso de enseñanza con CACTUS para TMT, la herramienta interactiva de generación de modelos de ATOMS. En general, las tareas de todas aquellas aplicaciones o herramientas que se implementen o estén ya implementadas utilizando AMULET son susceptibles de ser modelizadas. Y, una vez realizada la modelización, la generación de cursos con CACTUS es rápida y sencilla.

En lo que se refiere a evaluaciones experimentales del desempeño de los cursos, hasta el momento no se han realizado pruebas con alumnos reales. Las pruebas realizadas se han hecho utilizando como alumnos personas de nuestro entorno, que no responden al prototipo de sujetos a los que el sistema va dirigido, ya que están familiarizados con el uso de sistemas de la información. Pese a todo, las pruebas preliminares indican que la información de guía orientada a tareas que proporciona DARTS es más fácil de asimilar que la información proporcionada por los entornos comerciales actuales. Estos resultados se deben fundamentalmente a dos hechos. En primer lugar, que las indicaciones son proporcionadas de manera integrada con el sistema y, en segundo, que los mensajes de guía son generados de manera dinámica y facilitados al usuario en el momento en que son requeridos. No obstante, uno de los puntos sobre los que aún cabe hacer trabajo es en el de estudiar experimentalmente el grado de aprendizaje que se consigue con el empleo de tutores sobre aplicaciones, como CACTUS.

6 Conclusiones

En esta sección se resumen las principales cuestiones estudiadas durante el trabajo y qué soluciones se han aportado. No se incluyen las ventajas de todas las aportaciones, sino aquellas consideradas fundamentales de cara a solventar los objetivos fijados.

Como se comentó al comienzo del trabajo, las interfaces de usuario de las aplicaciones interactivas actuales son tremendamente complejas, fruto de la riqueza que proporcionan los nuevos entornos gráficos. Y, sin embargo, los entornos de guía para dichas aplicaciones no han evolucionado tanto como las propias aplicaciones, de manera que existe un gran desfase entre ambos tipos de sistemas. La consecuencia principal de esto es que muy pocos usuarios son capaces de exprimir la potencia de las herramientas. Este problema se agrava cuando son usuarios noveles quienes tratan de aprender a usar estas interfaces de usuario complejas. Además, hay que resaltar la importancia de la enseñanza de las interfaces, en tanto en cuanto éstas pueden emular el funcionamiento de artefactos arbitrariamente complejos. De este modo, un entorno de enseñanza de interfaces de usuario puede ayudar a la comprensión de dispositivos físicos tan complejos como puedan imaginarse.

En esta tesis se ha mostrado una técnica de construcción de cursos interactivos para estas interfaces que proporcionen a los usuarios noveles una manera cómoda de comenzar a utilizar las aplicaciones complejas. Se ha diseñado e implementado una arquitectura modular basada en la modelización de las tareas que los usuarios pueden realizar con las aplicaciones. Sobre esta modelización de tareas se ha construido ATOM, un entorno de gestión de tareas que fundamentalmente aporta la posibilidad de realizar un seguimiento de las acciones de los usuarios. Con esta base, el entorno de guía DARTS es capaz de guiar al usuario en la realización de tareas. Y, finalmente, con esta base el entorno CACTUS proporciona el soporte para construir cursos estructurados que faciliten el papel de los usuarios noveles.

En conjunto, con este entorno se han atacado los mayores problemas detectados tanto en los entornos de ayuda tradicionales basados en formas más o menos avanzadas de hipertexto, como en tutores comerciales y otros entornos más innovadores. Para ello, las aportaciones realizadas se enmarcan en tres líneas: uso de la modelización para la generación de la guía, seguimiento de la actividad del alumno y representación de los cursos mediante la metáfora de libros. A continuación se indican las principales aportaciones que corresponden a cada una de estas líneas.

- Uso de la **modelización** para la generación de la guía. La información que se proporciona a los usuarios se desprende directa o indirectamente de la misma información utilizada en el diseño de la aplicacón. Para ello nos hemos basado en el paradigma de desarrollo basado en modelos declarativos de interfaces. Las principales ventajas que proporciona la utilización de estos modelos son:
  - Una **reducción de los costes** de desarrollo y mantenimiento de la guía. La utilización de los mismos modelos utilizados para diseñar la aplicación para la enseñanza de la misma hace que no sea preciso un desarrollo independiente de la componente de guía. Además, el hecho de modificar la interfaz de la aplicación automáticamente modifica la componente de guía para reflejar los camios realizados. Esto es así porque la componente de guía construye los contenidos de manera dinámica, y no es preciso mantener unos contenidos estáticos como los utilizados de manera tradicional.

- Eliminación del **problema de asociación**. La utilización de modelos de alto nivel, en nuestro caso modelos de tareas, permite propocionar información de alto nivel semántico. Además, estudios previos [Paris89] sugieren la utilización de la guía orientada a tareas como la manera idónea de comunicar información a los usuarios noveles, en contra de otras tipologías de usuarios, que demandan información más orientada a procesos.

- **Seguimiento** de la actividad del alumno. El seguimiento de la actividad de los alumnos es vital para que éstos se involucren en el proceso de enseñanza. De lo contrario, éstos pueden percibir la enseñanza como una obligación y una actividad monótona cuya realización no aporta recompensa alguna. En este trabajo la arquitectura propuesta tiene por cimiento un entorno de seguimiento de las tareas que realizan los usuarios, y en base a esta base se soportan varias capas encargadas de generar la guía de tareas aisladas y de cursos. En nuestro caso, las principales ventajas que el seguimiento de la actividad de los alumnos proporciona son:

  - La información de guía se proporciona siempre de manera **dinámica**. De este modo se evita la explicación sobre la realización de tareas de manera aislada e independiente a la realización de éstas. Este dinamismo también facilita que la información que se comunique sea más adecuada, menos genérica, pues puede ajustarse mejor al momento en el que el alumno la recibe.

  - Puede proporcionarse **información contextual**. Como consecuencia del seguimiento en profundidad que se realiza de la actividad de los usuarios, la información que se le proporciona al alumno puede contener una mayor riqueza contextual, gracias a lo que en el marco de este trabajo se ha denominado *parámetros de las tareas*. Este contexto permite que las explicaciones hagan referencia a información específica con la que el usuario está tratando en el momento de proporcionarse la guía, en lugar de limitarse a dar explicaciones genéricas.

  - El seguimiento que el sistema es capaz de hacer de la actividad de los alumnos permite ofrecer un ***feedback*** a aquéllos sobre cómo están llevando a cabo los procesos, ya sea de manera positiva o negativa. De este modo se mejora la interfaz entre el entorno de enseñanza y el alumno, completándose el ciclo de la formación.

  - Dado que la información puede proporcionarse de manera dinámica, es posible particionar ésta en base a bloques lo suficientemente pequeños como para evitar el problema de ***visualización*** que se plantea con los sistemas de ayuda tradicionales. Esto no sería posible en un entorno estático, en el que no existiese un seguimiento que hiciese posible la descomposición dinámica de grandes procesos en subprocesos.

  - La ***seguridad*** [Carroll87a] influye en el coste que supone el aprendizaje para los usuarios: ellos no desean aprender con un sistema en el que las nuevas acciones sean percibidas como fuentes de riesgo, es decir, que puedan causar algún daño. Así, mediante el seguimiento de la actividad de los usuarios es posible recortar dinámicamente el número de opciones disponibles para aumentar la *seguridad* de la práctica, fundamentalmente de cara a la enseñanza orientada a los usuarios más noveles.

- Representación de los cursos mediante la metáfora de **libros**. La idea de utilizar como metáfora de representación de los cursos de enseñanza los libros interactivos permite a los alumnos comenzar a utilizar inmediatamente el sistema, ya que desde temprana edad todas las

personas están habituadas al manejo de libros impresos. Las principales ventajas que esta representación proporciona son:

- **Acceso** a la enseñanza. Una vez conseguida una manera eficiente de realizar la enseñanza de una determinada tarea, se plantea el problema de cómo los usuarios, específicamente aquellos de perfil más novel, acceden a dicha enseñanza. Como solución a esta problemática se ha propuesto la utilización de un entorno de perfil más dominante que permita la creación y gestión dinámica de cursos didáticos. Los cursos didácticos se comportan de manera proactiva, guiando a los alumnos por las posibilidades funcionales de la herramienta que se está enseñando. Para ello, los diseñadores de cursos incorporan a éstos la información que desean que los alumnos aprendan, de modo que así les sea más sencillo a éstos enfrentarse al sistema a aprender.
- Se proporciona un mecanismo mediante el cual puede estructurarse la enseñanza de familias de tareas, relacionadas entre sí por determinados criterios. La agrupación de varias sesiones pedagógicas de tareas en unidades pedagógicas, y la agrupación de éstas formando libros interactivos de enseñanza, permite **estructurar** los contenidos de los cursos para la enseñanza a los usuarios noveles. Además de los contenidos pedagógicos, la metafóra de representación mediante libros permite incorporar otros elementos, como ejemplos, índices, evaluaciones o relaciones de secuenciamiento.
- La utilización de la metáfora de representar los cursos interactivos como si de libros se tratara permite a los alumnos disfrutar de una **interfaz intuitiva** durante el aprendizaje. La interfaz de los libros interactivos es común a todos los cursos tutores generados con CACTUS. Esta homogeneidad los convierte en herramientas sencillas de utilizar para los alumnos, que solo deben aprender a utilizar una vez el sistema de enseñanza.

7 Dependencia de la Plataforma de Implementación

Una de las cuestiones más importantes que cabrían plantearse acerca de un entorno de enseñanza como CACTUS es el grado de aplicación que éste podría tener en la realidad. Por realidad nos referimos a una situación actual donde el uso de aplicaciones interactivas basadas en entornos gráficos es enorme. Y, como consecuiencia de este desarrollo masivo de aplicaciones gráficas interactivas, a lo largo de los años se han ido estableciendo algunos estándares *de facto* que sirven de base a dichos desarrollos. Sin embargo, y como ya hemos visto, la plataforma en la que CACTUS se basa, AMULET, no se trata de ninguno de dichos estándares *de facto*, lo que puede comprometer la aplicabilidad de los resultados que se obtengan a las aplicaciones construidas bajo los estándares utilizados en la actualidad. Durante el resto de esta sección se pretende discutir el grado de dependencia de CACTUS respecto del entorno de implementación utilizado.

En la actualidad, el desarrollo de aplicaciones basadas en entornos gráficos interactivos se suele realizar basándose en tecnologías de componentes, cuya lógica está distribuída en arquitecturas multicapa. Para llevar a cabo la gestiones típicas de dichas componentes, como son los eventos, los mensajes o la encapsulación, éstos se apoyan en servicios de objetos distribuídos. Desde hace más de diez años, el mercado de estos servicios de objetos distribuídos ha ido evolucionando, encontrándose en la actualidad consolidado en torno a dos opciones: DCOM (Distributed Component Object Model, de Microsoft), y Java/RMI (Remote Method Invocation, de Sun Microsystems).

El primero de ellos es la extensión a un modelo distribuído de COM (Component Object Model), e implementa una capa de llamadas a procedimientos remotos, basada en el estándar RPC (Remote Procedure Call) del DCE (Distributed Computing Environment), para dar soporte a los objetos remotos. Este estandar *de facto*, de carácter binario, responde más a una implementación de objetos distribuídos para entornos con sistemas operativos Microsoft que a una especificación estándar. En la Figura 60 podemos ver representado un esquema muy simple del mecanismo de funcionamiento de este modelo. Sobre este sistema de componentes distribuidas, Microsoft ha construido un modelo de arquitectura de tres capas para aplicaciones *web*, conocido como DNA (Distributed InterNet Architecture), que incorpora como elementos fundamentales el servidor web IIS (Internet Information Service), el servidor de transacciones MTS (Microsoft Transaction Server), el servicio de mensajería MSMQ (MicroSoft Message Queuing) y otros servicios de *clustering*, balanceo de carga, replicación, tolerancia a fallos, etcétera. Durante los próximos años, Microsoft pretende evolucionar esta arquitectura hacia la plataforma .NET [Microsoft2001b], orientada a la gestión del ciclo de vida software como si se tratase de un servicio, y no como un producto.

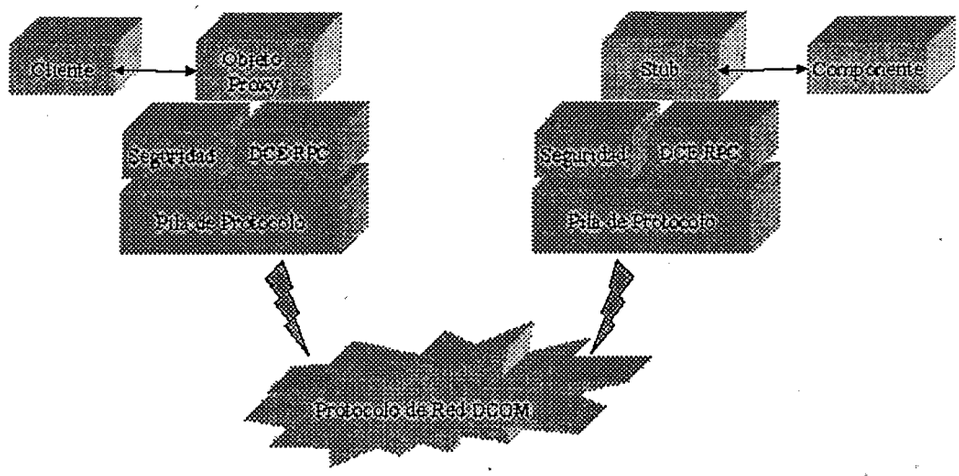

Figura 60: Arquitectura del modelo DCOM.

Por contra, Java/RMI es el núcleo de una arquitectura nacida a partir del lenguaje de programación orientado a objetos Java. Este lenguaje surgió de la demanda de una plataforma de desarrollo orientada fundamentalmente a las necesidades de Internet, y la comunicación entre las componentes Java se realiza por medio del protocolo RMI. Alrededor de Java/RMI ha ido surgiendo todo un conjunto de funcionalidades y servicios, algunos de los cuales inspirados en la plataforma CORBA (Common Object Request Broker Architecture). Este conjunto de servicios ha dado lugar a la plataforma J2EE (Java2 Enterprise Edition), cuyo esquema podemos apreciar en la Figura 61. El estándar de esta plataforma establece unas especificaciones técnicas y funcionales para los productos, existiendo distintas implementaciones dependiendo de los fabriantes (BEA, IBM, iPlanet, etc), que ofrecen así distintos parámetros de escalabilidad, rendimiento, tolerancia a fallos, etcétera. Conforme a la especificación J2EE, la codificación de la interfaz de usuario se encuentra bien en módulos JSP escritos en el lenguaje de programación JavaScript, o bien en *applets*, escritos mediante el lenguaje Java. Los módulos JSP se ejecutan en el servidor, generando dinámicamente páginas html que son mostradas por los navegadores de la manera convencional. Por su parte, los *applets* son programas precompilados, descargados dinámicamente por los navegadores e interpretados mediante lo que se conoce como la JVM (Java Virtual Machine) en las máquinas clientes, por lo que éstas precisan tener cierta capacidad de proceso. Por lo tanto, los elementos de mayor riqueza interactiva de cara al usuario se consiguen programando *applets* en Java.

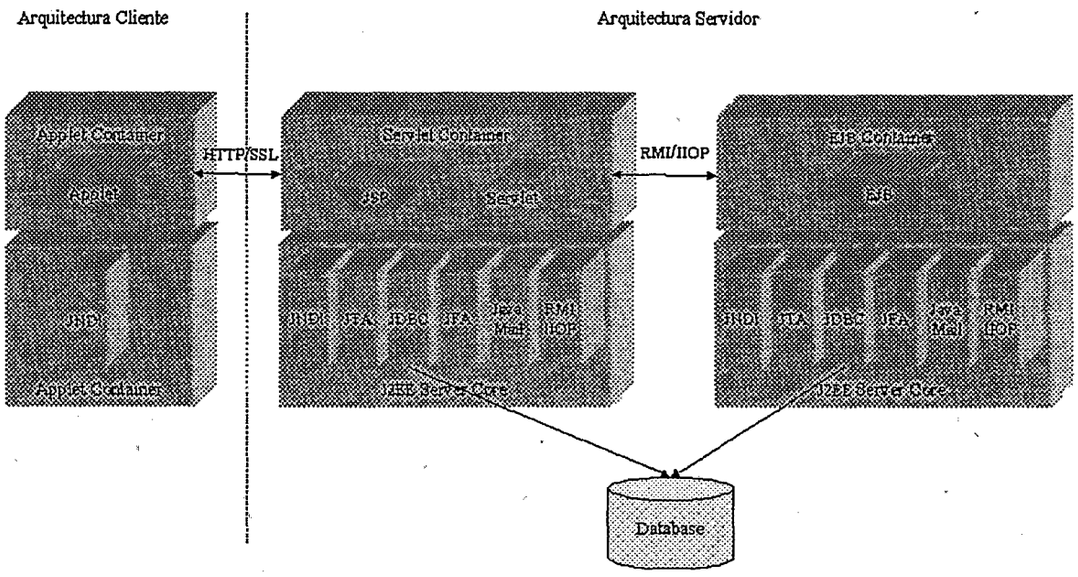

Figura 61: Arquitectura de la plataforma J2EE.

De cara a mostrar la compatibilidad de la plataforma CACTUS con los entornos más habituales que hemos descrito, .se ha realizado como parte del trabajo de esta tesis una implementación parcial del entorno basada en la plataforma Java, en su versión 1.2 del JDK (Java Development Kit). La elección de Java frente a DCOM se ha realizado en base a la idea de no perder la portabilidad que AMULET nos había ofrecido. En concreto, se ha implementado en la plataforma Java el sistema de gestión de tareas de usuario ATOM, que proporciona la base para el desarrollo del resto del entorno CACTUS, conforme a la Figura 31.

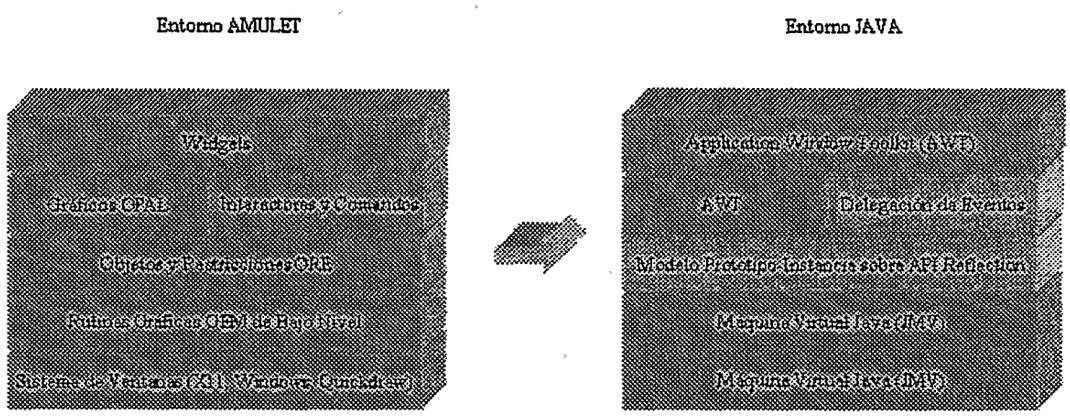

Figura 62: Modificaciones realizadas para la implementación de ATOM en Java.

Para realizar esta implemenación, se han desarrollado en Java un conjunto de clases que proporcionan las funcionalidades utilizadas por ATOMS que estaban presentes en AMULET y por contra ausentes para Java. En la Figura 62 se muestran esquematizados los cambios más relevantes introducidos en nuestras modificaciones. A la izquierda se muestra la arquitectura de

AMULET, mientras que a la derecha se muestran las componentes de Java que realizan funciones eqiuvalentes. En este último lado, las piezas con un color más claro se corresponden con aquellas que se han añadido o modificado. A continuación se describen estas equivalencias y las modificaciones fundamentales realizadas sobre la plataforma Java para la implementaciòn de ATOM sobre ella:

- El sistema de ventanas de AMULET proporciona un mecanismo para gestionar objetos gráficos como ventanas, menús, botones, etc. En el caso de Java, de esta gestión de objetos gráficos se ocupa la máquina virtual Java que, ejecutándose dentro o fuera del navegador de Internet, visualiza los objetos gráficos. Por lo tanto, esta capa no ha precisado modificaciones.
- Las rutinas gráficas de bajo nivel, GEM, de AMULET, proporcionan un mecanismo mediante el cual el código de las aplicaciones AMULET es portable. En el caso de Java, la máquina virtual proporciona el soporte para que no solo el código Java sea portable, sino que dicho código precompilado también lo sea.
- En cuanto al sistema de objetos y restricciones ORE de AMULET, la implementación descrita a lo largo de esta tesis únicamente precisaba de su modelo de objetos basado en el paradigma de herencia prototipo-instancia, y no del modelo de restricciones. Por ello, lo que se ha hecho para implementar ATOM sobre Java ha sido crear una capa sobre este lenguaje que implemente un sistema de objetos equivalente. Así, se han implementado las clases necesarias para conseguir las funcionalidades principales de un entorno de este tipo: *slots*, objetos, prototipos, iteradores, etc. Para esta implementación se ha utilizado el API estándar de Java *Reflection*, cuya funcionalidad permite analizar los atributos, interfaces y clases heredadas de una clase dada. Si se deseara implementar ATOM sobre la arquitectura DNA de Microsoft, deberían emplearse las tecnologías COM *Type Information* y *Automation*, cuya evolución en .NET se denomina .NET Reflection.
- Por su parte, la capa OPAL de AMULET proporciona funcionalidades en dos líneas. Por un lado, la posibilidad de crear jerarquías de objetos gráficos de tipo parte-dueño y, por otro, proporciona un mecanismo mediante el cual la visualización de los objetos gráficos se lleva a cabo automáticamente en base a restricciones que ligan sus propiedades con valores de otros objetos. La primera de las dos facilidades se proporciona de manera equivalente por la librería AWT de Java. Por su parte, la segunda facilita la labor del programador, pero no proporciona una mayor funcionalidad a las aplicaciones, por lo que no se ha implementado ninguna mejora a la plataforma Java para incorporar su equivalente.
- La capa de Interactores y *Comandos* de AMULET se utiliza para obtener información de alto nivel acerca de las interacciones de los usuarios con los objetos gráficos. Para implementar está capa, lo que se ha hecho ha sido modificar la funcionalidad de las componentes gráficas de Java, aportándose implementaciones estándares para los objetos *Listener* de Java apropiados (actionListener, itemListener, textListener), lo que permite el seguimiento de las interacciones más usuales de los usuarios, obteniéndose la información contextual de dichas interacciones. Esencialmente, esto es lo mismo que internamente hace AMULET: recoger y agregar los eventos del sistema operativo, proporcionando información de mayor contenido semántico.
- Los widgets de AMULET proporcionan un conjunto de componentes gráficas de alto nivel para construir de manera sencilla interfaces gráficas de usuario. Con diferencias que no son

determinantes de cara a la implementación Java de ATOM, el JDK de Java también proporciona una librería de dichos objetos gráficos, denominada AWT (Application Window Toolkit).

Una vez realizadas estas pequeñas modificaciones, la implementación de ATOM sobre Java encierra la misma complejidad que su implementación sobre AMULET. Dentro del trabajo de esta tesis, con el fin de estudiar la viabilidad del funcionamiento de ATOM en una plataforma estandar, tan solo se ha implementado ATOM. Sin embargo, y dado que toda la plataforma CACTUS se basa en la arquitectura ATOM, podemos afirmar que la portabilidad de CACTUS a la plataforma Java está garantizada y, como consecuencia, la aplicabilidad de las ventajas y funcionalidades de CACTUS al mundo Internet.

8 Trabajo Futuro

Esta tesis deja abiertas algunas lineas sobre las que es posible continuar investigando, como modelización de la interfaz de usuario, gestión de tareas distribuídas y gestión de cursos interactivos. A su vez, cada una de estas lineas de trabajo ofrece margen suficiente para varias vías de avance. En esta sección se describen estas vías de avance, distribuídas en torno a las distintas líneas abiertas.

- **Modelizacion de las tareas de usuario**. En esta linea, así como en la linea más generalista de modelización de aplicaciones, existe aún mucho margen de mejora. Como dato paradógico, ITS, el MB-IDE de uso más extendido, tiene la herramienta de modelización menos potente de cuantas hemos descrito en la Sección 3.2.2. En esta linea, en este trabajo se ha presentado la herramienta TMT como soporte a la modelización interactiva de tareas. Sobre dicha herramienta cabe realizar algunas mejoras, entre las que destaca:

  - *Integración de técnicas más potentes de aprendizaje automático* (*machine learning*), para facilitar la generación de patrones de interacción tan ricos como se pueda, de manera sencilla. Recordemos que, en la actualidad, los patrones de interacción que TMT genera no incorporan relaciones entre distintos atributos, y las únicas funciones de transferencia de parámetros que genera se basan en la función identidad.

    La gran barrera que se presenta para lograr este objetivo es que una característica deseable es que el número de ejemplos que se deban presentar a la herramienta interactiva que permita la generación de los modelos deberá ser *muy pequeña*. Habitualmente, *muy pequeña* significa que proporcionando no más de tres o cuatro ejemplos, el sistema debería ser capaz de generar una representación válida que expresase las cualidades que la interacción ha de cumplir para ser asociada a una cierta tarea atómica, o que estableciese relaciones complejas entre valores de parámetros.

    Esta observación dificulta la implementación de técnicas más sofisticadas como, por ejemplo, árboles de decisión. Parece intuitivo que al final se debe llegar a un compromiso entre potencia de la herramienta de modelización y complejidad, es decir, número de ejemplos a proporcionar, del proceso de modelización.

- **Entornos de enseñanza**. Dentro de esta categoría se agrupan las posibles vías de avance que permitirían aportar mejoras a la generación de herramientas de creación y gestión, tanto de cursos tutores como CACTUS, como de entornos de guía como DARTS. En particular, se identifican tres posibles e importantes vías:

  - *Evaluación empírica de los resultados del sistema de enseñanza*. Esta labor debería ser el trabajo a realizar a más corto plazo, dado que a raíz de ella se pueden detectar deficiencias en el sistema e incoporar mejoras. La evaluación del sistema de enseñanza CACTUS iría encaminada a medir tanto la usabilidad del sistema como la utilidad, desde el punto de vista didáctico, de los cursos con él desarrollados. Existe bastante trabajo orientado a determinar criterios que permitan medir la calidad de los sistemas de enseñanza, que pueden encontrarse en [Aedo2000].

    En nuestro caso, un aspecto fundamental a medir sería cuánto de acertada es la utilización de los libros interactivos como metáfora de interacción con los cursos. En este sentido,

cabría contrastar si la metáfora utilizada incorpora adecuadamente aspectos deseables -interactividad, integración con las aplicaciones que se enseñan, navegación mediante hiperenlaces, multimedia, etcétera-, sin por ello llegar a restingir o confundir a los alumnos, ni ser antinatural [Gentner96, Ficarra97]. Otros aspectos a medir con especial atención serían aquellos relativos a la usabilidad del entorno DARTS, tales como aspectos estéticos, de consistencia, predectibilidad de su interfaz, intuitividad, etcétera, a fin de poderse comparar con la interfaz de otros entornos, como los basados en *agentes de confianza.*

- *Integración del entorno interactivo de generación y seguimiento de cursos con la tecnología de agentes de confianza (believable agents)* [de Rosis99, Rickel98, André98, Lester99]. Estos agentes utilizan como interfaz la imagen animada de personajes que interaccionan con los usuarios para enseñarles a llevar a cabo tareas. Estos personajes se expresan de manera verbal y no verbal (movimientos, gestos, muecas), pudiendo así establecer una relación más estrecha con los alumnos, especialmente los usuarios menos expertos o que sientan una menor confianza con las aplicaciones informáticas. Algunos estudios [Koda96, Takeuchi95] muestran que los usuarios muestran más motivación cuando interaccionan con agentes representados por personajes animados que cuando lo hacen con las ventanas gráficas cotidianas. El tipo de guía y la personalidad de estos agentes pueden variar dependiendo del tipo de usuario al que se pretenda guiar. Por ejemplo, en el trabajo de [de Rosis99] se presenta un agente de confianza que diferencia el tipo de guía que proporciona (orientada a objetos u orientada a procesos), la manera de facilitarla (mínima o redundante) y su personalidad (sumisa o dominante), dependiendo del tipo de usuario con el que se interaccione (experto o novel). Tomando como base de este trabajo el sistema tutor DARTS y la modelización de las actividades de los usuarios propuesta en esta tesis, la idea sería modificar la interfaz de DARTS para integrar una guía basada en agentes de confianza dentro del sistema gestor de cursos CACTUS.
- *Integración con las tendencias actuales de acceso a los sistemas de la información.* Con esto lo que se pretendería sería una reingeniería de los sistemas presentados (ATOMS, DARTS, CACTUS...), con el fin de que su funcionalidad se pudiera aplicar a sistemas accesibles mediante navegadores estandar, como *applets* Java, controles Active X, *servlets*, o páginas dinámicas. Como ya hemos visto, en el marco de este trabajo de tesis se ha realizado una implementación parcial del sistema ATOMS para Java, con el fin de verificar la aplicabilidad de los conceptos de gestión de modelos de tareas introducidos, en el marco del tipo de aplicaciones web que en la actualidad se desarrollan.

  Una vez conseguida la funcionalidad de enseñanza accesible mediante un navegador, se permitiría la integración con entornos de generación de cursos multimedia genéricos como TANGOW [Carro2000]. Esta integración iría encaminada a mejorar el seguimiento que estos sistemas realizan de las actividades prácticas de los alumnos, pudiéndoseles permitir que éstas fueran simulaciones interactivas, juegos didácticos, demostraciones multimedia, etcétera.

- **Flujos de trabajo**. En este trabajo se ha mostrado que, sobre la base de un sistema de gestión de tareas de usuario puede funcionar un entorno de seguimiento de flujos de trabajo potente y flexible. Sin embargo, no hemos profundizado en las ventajas que un sistema de gestión de flujos de trabajo como Wf-ATOMS puede aportar de cara a la instrucción de alumnos que

participan de manera cooperativa en procesos que involucran múltiples aplicativos. En esta linea, destacamos como fundamentales vías de avance, a poder desarrollar a corto plazo sobre la base de Wf-ATOMS, las siguientes:

- *Adaptación del entorno de gestión de cursos a la plataforma Wf-ATOMS.* La actual arquitectura de CACTUS (ver Figura 31), que incluye los módulos DARTS y el *Módulo de Ejecución de Escenarios*, descansa actualmente sobre la plataforma ATOM y, por lo tanto, está orientada a la enseñanza en el marco de aplicaciones aisladas.

  Dado que la arquitectura de Wf-ATOMS (ver Figura 30) se apoya en ATOMS para el seguimiento de la ejecución de las tareas en los puestos clientes, las modificaciones a incorporar a DARTS incluirán básicamente el ampliar la lógica de generación de mensajes de guía contextuales, de manera que las instrucciones generadas proporcionen información acerca del papel que cumplen las tareas que los usuarios realizan dentro del flujo completo que determina el proceso. Es decir, si en la actualidad DARTS proporciona el contexto a partir de las acciones del usuario y los modelos de tareas, en un entorno basado en flujos de trabajo también se debería utilizar el modelo de flujo de procesos.

  En cuanto a la ejecución de escenarios, habría que diseñar un mecanismo que permitiese la ejecución automática o semiautomática de flujos de trabajo, en vez de tareas interactivas. En resumen, se trataría de que, mientras que el actual módulo de *Emulación* de ATOM se encarga de ejecutar tareas, un hipotético módulo *Wf-Emulación* se encargaría de controlar desde el servidor que se ejecutasen los distintos pasos que conformasen los procesos a realizarse en un determinado flujo de trabajo.

  Finalmente, el trabajo debería considerar la ampliación a un entorno colaborativo de las herramientas de gestión de cursos de CACTUS, tanto desde el punto de vista de la creación colaborativa de cursos como desde el punto de vista de la aportación de *feedback* durante la ejecución de los cursos, a los distintos alumnos involucrados en el aprendizaje de la realización de un flujo de trabajo. Un aspecto importante a tener en cuenta para esta labor sería el hecho de que mientras que las *tareas* realizadas con los entornos interactivos suelen ser síncronas, los *procesos* involucrados dentro de los flujos de trabajo son, con frecuencia, realizados de manera asíncrona. Así, es muy normal el caso de que estos flujos tarden en completarse horas, días e, incluso, semanas. Por lo tanto, un hipotético *Wf-CACTUS* debería contemplar esta nueva casuística de uso.

- *Evolución del entorno hacia una plataforma Web.* En efecto, las actividades hoy en día son llevas a cabo, cada vez con mayor frecuencia, desde localizaciones remotas, vía internet. Partiendo de esta situación, un hipotético entorno Wf-CACTUS como el descrito en el punto anterior, basado en una arquitectura cliente/servidor, no cumpliría las expectativas de usuario que puedan participar en flujos de trabajo utilizando herramientas accesibles mediante navegadores.

Referencias

- **[Aedo2000]**

  Aedo, I. y Díaz, P.: Evaluation criteria for hypermedia educational systems. Informática y Educación para una Sociedad Interconectada. Actas del Segundo Simposio Internacional de Informática Educativa (SIIE'2000). AIDE. Noviembre 2000.

- **[Aho86]**

  Aho, A.V., Sethi, R. y Ullman, J.D.: Compilers: Principles, Techniques and Tools. Addison-Wesley Publishing Company, Reading, MA, 1986.

- **[Alfonseca97]**

  Alfonseca, M., Pulido, E., Orosco, R. y de Lara, J.: OOCSMP : an Object-Oriented Simulation Language, ESS'97. Passau. 1997.

- **[Alfonseca98]**

  Alfonseca, M., García, F., de Lara, J. y Moriyón, R.: Generación Automática de Entornos de Simulación con Interfaces Inteligentes, Revista de Enseñanza y Tecnología (ADIE), no. 10, Diciembre 1998.

- **[Alonso96]**

  Alonso, G., Agrawal, A., El Abbadi, A., Kamath, M., Günthör, R. y Mohan, C.: Advanced Transaction Models in Workflow Contexts. En Proceedings of the 12th International Conference on Data Engineering, Nueva Orleans, Luisiana, USA, Febrero 1996.

- **[André98]**

  André, E., Rist, T. y Mueller, J.: Integrating Reactive and Scripted Behavious in a Life-like Presentation Agent. En K. P. Sycara y M. Woolridge (Eds.). Proceedings of the Second International Conference on Autonomous Agents. ACM Press, 1998.

- **[Antchev95]**

  Antchev K., Luhtalahti, M., Multisilta, J., Pohjolainen, S. y Suomela, K.: A WWW Learning Environment for Mathematics. En 4th International World Wide Web Conference: The Web Revolution. Boston, Massachusetts, USA. Diciembre 1995.

- **[Avrahami89]**

Avrahamin, G., Brooks, K.P., y Brown, M.H.: A Two-View Approach to Constructing User Interfaces. En Computer Graphics. Boston, MA: 23. pp. 137-146. Proceedings SIGGRAPH'89. 1989.

- **[AlliedSignal98]**

  AlliedSignal-Electronic and Avionics Systems: PRM, Baltimore, Maryland. http://www.alliedsignal.com, 1998.

- **[Balzert96]**

  Balzert, H., Hofmann, F., Kruschinski, V. y Niemann, C.: The JANUS Application Development Environment- Generating More than the User Interface. Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, pp. 183-206. Belgica, Junio 1996.

- **[Bauer96]**

  Bauer, B.: Generating User Interfaces from Formal Specifications of the Application. Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, pp. 141-158. Belgica, Junio 1996.

- **[Booher75]**

  Booher, H.R.: Relative Comprehensibility of Pictorial Information and Printed Words in Proceduralized Instructions. En Human Factors, 17(3), 1975.

- **[Brézillon96]**

  Brézillon, P.: Context in Human-Machine Problem Solving: A Survey. Rapport de Recherche 96/29, LAFORIA, Octubre 1996.

- **[Brusilovsky98]**

  Brusilovsky, P., Eklund, J. y Schwarz, E.: Web-Based Education for All: A Tool for Developing Adaptive Courseware. Computer Networks and ISDN Systems. Vol. 30. Nos. 1-7. pp. 291-300. Proceedings of the 7th International WWW Conference. 1998.

- **[Buxton90]**

  Buxton, W. y Sniderman, R.: Iteration in the Design of the Human-Computer Interface. In Proceedings of the 13th Anual Meeting of the Human Factors Association of Canada, pp 72-80, 1980.

- **[Carro2000]**

  Carro, R.M., Pulido, E. y Rodríguez, P.: How Adaptivity Affects the Development of TANGOW Web-based Courses. International Conference on Adaptive

Hypermedia and Adaptive Web-based Systems, AH2000, pp.-280-283. Trento, Italia, Agosto 2000.

- **[Carroll87a]**

Carroll, J.M. y Rosson, M.B.: The Paradox of the Active User. En J.M. Carroll (Ed.), Interfacing Thought: Cognitive Aspects of Human-Computer Interaction, Cambridge, MA: MIT Press, 1987.

- **[Carroll87b]**

Carroll, J.M., Smith-Kerker, p.l., Ford, J.R. y Mazur-Rimetz, S.A.: The Minimal Manual, Human-Computer Interaction, 3, 123-153, 1987.

- **[Chambers89]**

Chambers, C., Ungar, D. Y Lee, E.: An Efficient Implementation of SELF, a Dinamically-Typed Object-Oriented Language Based on Prototypes. Sigplan Notices, 1989, vol. 24, no. 10, pp. 49-70. ACM Conference on Object-Oriented Programming, System Languages and Applications, OOPSLA'89, 1989.

- **[Cohen99]**

Cohen, A.M. y Meertens, L.: The ACELA Project: Aims and Plans. Accesible desde http://www.cwi.nl/~steven/acela. 1999.

- **[Contreras96]**

Contreras, J. y Saiz, F.: A Framework for the Automatic Generation of Software Tutoring, Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, Belgica, Junio 1996.

- **[Contreras98]**

Contreras, J.: A Framework for the Automatic Generation of Software Tutoring, PhD. Thesis, Université René-Descartes, Paris, 1998.

- **[Corel94]**

Corel Corporation Limited: Wordperfect. http://www.corel.com/Office2000. Europa House, Harcourt St., Dublín 2, Irlanda. 1994.

- **[Corel97]**.

Corel Corporation Limited: CorelDREAM 3D. http://www.corel.com/products/graphicsandpublishing/draw8/dream3d.htm. Europa House, Harcourt St., Dublín 2, Irlanda. 1997.

- **[Cypher91]**

  Cypher, A.: EAGER: Programming Repetitive Tasks by Demonstration. En Proceeding de CHI'91, ACM Press, New York, 1991.

- **[Cypher93]**

  Cypher, A.: Watch What I do: Programming by Demonstration, MIT press (ed. A. Cypher), Cambridge, Ma., USA, 1993.

- **[de Rosis99]**

  de Rosis, F., de Carolis, B.N., y Pizzutilo, S.: Software Documentation with Animated Agents. 5th ERCIM Workshop on User Interfaces for All. Dagstuhl (Alemania). 1999.

- **[Diaper89]**

  Diaper, D.: Task Analysis for Human-Computer Interaction. Ellis Horwood, Books in Information Technology, 1989.

- **[Eisenstein2000]**

  Eisenstein, J. y Puerta, A.: Adaptation in Automated User-Interface Design. En Proceedings IUI'2000: 2000 International Conference on Intelligent User Interfaces. ACM Press. New Orleans, Louisiana, 2000.

- **[Elwert95]**

  Elwer, T., y Schlungbaum, E.: Modelling and Generation of Graphical User Interfaces in the TADEUS Approach, in DSV-IS95, pp. 193-208.

- **[Feiner90]**

  Feiner, S. y McKeown, K.: Coordinating Text and Graphics in Explanation Generation. Proceedings AAAI-90. Boston, MA, 1990.

- **[Fenchel82]**

  Fenchel, R.S. y Estrin, G.: Self-Describing System Using Integral Help. IEEE Transactions on System, Man and Cybernetics, Marzo-Abril, 1992.

- **[Ficarra97]**

  Ficarra, F.V.C.: Evaluation of Multimedia Components. Proceedings of International Conference on Multimedia Computing and Systems, págs. 557-564. Ottawa, 1997.

- **[Foley88a]**

Foley, J.D., Gibbs, C., Kim, W.C. y Kovacevic, S.: A Knowledge-Based User Interface Management System, Proceedings CHI'88, 1998.

- **[Foley88b]**

Foley, J.D., Kim, W.C., Kovacevic, S. y Murray, K.: The User Interface Design Environment, Proceedings of Architecture for Intelligent Interfaces: Elements and Prototypes, Monterey, California, 1998.

- **[Foley91]**

Foley, J.D, Kim, W.C., Kovacevic, S. y Murray, K.: UIDE – An Intelligent User Interface Design Environment. En Intelligent User Interfacès, J.W. Sullivan y S.W. Tyler (eds.), Addison Wesley, ACM Press, pp. 339-384. 1991.

- **[Foley94]**

Foley, J.D: History, Results and Bibliography of the User Interface Environment (UIDE), an Early Model-Based System for User Interface Design and Implementation, en DSV-IS'94, pp.3-14, 1994.

- **[Frank95]**

Frank, M.: Grizzly Bear: A Demonstrational Learning Tool for a User Interface Specification Language. En Proceedings de UIST'95, pp.75-76.

- **[García98a]**

García, F., Contreras, J., Rodríguez, P. y Moriyón, R.: Help Generation for Task Based Applications with HATS. En Proceedings EHCI'98, Creta (Grecia), Septiembre 1998.

- **[García98b]**

García, F., Rodríguez, P., Contreras, J., y Moriyón, R.: Gestión de Tareas de Usuario en ATOMS. IV Jornadas de Tecnología de Objetos, JJOO'98, Bilbao, Octubre 1998.

- **[García99a]**

García, F. y Moriyón, R.: A Framework for Distributed Task Management. Third Argentine Symposium on Object Orientation, ASOO'99. Buenos Aires, Argentina, Septiembre 1999.

- **[García99b]**

García, F.: Towards the Generation of Tutorial Courses for Applications. 5th ERCIM Workshop on User Interfaces for All. Dagstuhl (Alemania). 1999.

- **[García99c]**

García, F. y Moriyón, R.: Formación Basada en Tareas para Usuarios de Aplicaciones. Actas del Congreso Nacional de Informática Educativa. CONIED'99. Puertollano, Ciudad Real. Noviembre, 1999.

- **[García2000a]**

García, F.: CACTUS: Automated Tutorial Course Generation for Software Applications. En Proceedings IUI'2000: 2000 International Conference on Intelligent User Interfaces. ACM Press. New Orleans, Louisiana, 2000.

- **[García2000b]**

García, F. y Moriyón, R.: CACTUS: Automated Tutorial Course Generation for Software Applications. Proceeding of the First Technical Workshop of the Computer Engineering Department, pp. 26-29, ETS Informática, UAM, Madrid. Marzo 2000. URL: http://www.ii.uam.es/eng/research/workshop/garcia.pdf

- **[García2000c]**

García, F. y Moriyón, R.: Distributed Task Management by Means of Workflow Atoms. SADIO Electronic Journal of Informatics and Operations Research (http://www.dc.uba.ar/sadio/ejs/vol.3.1/Garcia.pdf). Octubre, 2000.

- **[Gentner96]**

Gentner, D. y Nielsen, J.: The Anti-Mac Interface.Communications of the ACM, vol. 39, núm. 8, págs. 70-82. 1996.

- **[Gibbs86]**

Gibbs, C., Kim, W.C. y Foley, J.D.: Case Studies in the Use of IDL: Interface Definition Language. Report GWU-IIST-86-30, Department of EE&CS, The George Washington University, Washington, DC 20052, 1986.

- **[Glaser84]**

Glaser, R.: Education and Thinking: the Role of Knowledge. American Psychologist, 39, 93-104. 1984.

- **[Harning96]**

Harning, M.: An Approach to Structures Display Design- Coping with Complexity. Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, pp. 121-138. Belgica, Junio 1996.

- **[Hearst95]**

Hearst, M.A., Karger, D.R. y Pedersen, J.O.: Scatter/Gather as a Tool for the Navigation of Retrieval Results. proceedings of the 1995 AAAI Fall Symposium on Knowledge Navigation.1995.

- **[Hecking87]**

Hecking, M.: How to Use Plan Recognition to Improve the Abilities of the Intelligent Help System SINIX Consultant. Proceedings INTERACT'87, 2nd IFIP Conference on Human-Computer Interaction. 1987.

- **[Henke98]**

Henke, H.: Are Electrons Better than Papyrus (Or Can Adobe Acrobat Reader Files Replace Hardcopy?). ACM 16th International Conference on Systems Documentation, pp.29-37. 1998.

- **[Hinrichs96]**

Hinrichs, T., Bareiss, R., Birnbaum, L. y Collins, G.: An Interface Design Tool based on Explicit Task Models. En CHI'96, pp. 269-270. 1996.

- **[ISO88]**

ISO: Information Processing Systems – Open Systems Interconnection: LOTOS – A Formal Description Technique Based on Temporal Ordering of Observational Behaviour. ISO/IS 8807, ISO Central Secretariat.

- **[Johnson92]**

Johnson, P., Markopoulos, P., Wilson, S. Y Pycock, J.: Task Based Design: Mapping between User Task Models and User Interface Designs. Proceedings of 2nd Workshop on Mental Models in HCI. University of Cambridge, pp. 176-186, 1992.

- **[Johnson95]**

Johnson, P., Johnson, H. Y Wilson, S.: Rapid Prototyping of User Interfaces driven by Task Models. En Scenario-Based Design: Envisioning Work and Technology in System Development, J. Carroll (ed.), John Wiley & Sons, pp. 209-246. Londres, 1995.

- **[Johnson98]**

Johnson, L., Rickel, J., Stiles, R. y Munro, A.: Integrating Pedagogical Agents into Virtual Environments. En MIT Press Journal Presence 7(6) Diciembre 1998.

- **[Kamath95]**

Kamath, M., Alonso, G., Günthör, G. y Mohan, C.: Providing High Availability in Very Large Workflow Management Systems. Proceedings of the Fifth International Conference on

Extending Database Technology (EDBT'96), Avignon, Francia, Marzo 1996. También disponible como IBM Research Report RJ9967, IBM Almaden Research Center, Julio 1995.

- **[Kay83]**

  Kay, M.: Unification Grammar. Technical report, Xerox Palo Alto Research Center. 1983.

- **[Kearsley88]**

  Kearsley, G.: Online Help Systems, Design and Implementation. Ablex Publishing Corp., 1988.

- **[Kim93]**

  Kim, W.C. y Foley, J.D.: Providing High-Level Control and Expert Assistance in the User Interface Presentation Design. En InterCHI93, pp. 430-437. 1993.

- **[Koda96]**

  Koda, T. y Maes, P.: Agent with Faces: The Effect of Personification. Proceedings of theFifth IEEE International Workshop on Robot and Human Communication (RO-MAN'96), pp. 189-194. 1996.

- **[Kosbie94]**

  Kosbie, D.S. y Myers, B.A.: Extending Programming by Demonstration with Hierarchical Event Histories. En Human-Computer Interaction: 4th International Conference EWHCI'94, Lecture Notes in Computer Science, vol. 876. Belín, Springer-Verlag, pp. 128-139. 1994.

- **[Lester99]**

  Lester, J.C., Voerman, J.L., Towns, S.G. y Callaway, C.B.: Deictic Believability: Coordinated Gesture, Locomotion and Speech in Lifelike Pedagogical Agents. Applied Artificial Intelligence, 13 (4-5), 1995.

- **[Lim94]**

  Lim, K.Y. y Long, J.: The MUSE Method for Usability Engineering. Cambridge University Press. Cambridge, 1994.

- **[Linkworks98]**

  Linkworks: The Boss. http://www.linkworks.co.nz, Linkworks Ltd, Wellington, NZ, 1998.

- **[Lonczewski96]**

Lonczewski, F. y Schreiber, S.: The FUSE-System: an Integrated User Interface Design Environment. Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, Belgica, Junio 1996.

- **[Luo93]**

Luo, P., Szekely, P., Neches, R.: Management of Interface Design in HUMANOID. En InterCHI93, pp. 107-114. http://www.isi.edu/isd/CHI93-manager.ps. 1993.

- **[Masui94]**

Masui, T. y Nakayama, K.: Repeat and Predict - Two Keys for Efficient Text Editing. En Proceedings del CHI'94, Human Factors in Computing Systems, Boston, Massachusetts, USA, ACM, 1995.

- **[Maulsby97]**

Maulsby, D.: Inductive Task Modeling for User Interface Customization. En Proceedings IUI'97: 1997 International Conference on Intelligent User Interfaces. ACM Press. Orlando, Florida, 1997.

- **[Maxwell94]**

Maxwell, J.T. y Kaplan, R.M.: The Interface between Phrasal and Functional Constraints, Computational Linguistics, no.4, 1994.

- **[Microsoft91a]**

Microsoft Corporation: Microsoft Visual Basic. http://msdn.microsoft.com/vbasic. One Microsoft Way, Redmond, WA 98052, USA. 1991.

- **[Microsoft91b]**

Microsoft Corporation: Microsoft Visual C++. http://msdn.microsoft.com/visualc. One Microsoft Way, Redmond, WA 98052, USA. 1991.

- **[Microsoft2001a]**

Microsoft PressPass: Farewell Clippy: What's Happening to the Infamous Office Assistant in Office XP. http://www.microsoft.com/PressPass/features/2001/apr01/04-11clippy.asp. Washinton, Nota de prensa, 11 de Abril 2001.

- **[Microsoft2001b]**

Microsoft Corporation: Plataforma de Servicios .NET. http://www.microsoft.com/net. One Microsoft Way, Redmond, WA 98052, USA. 2001.

- **[Mitchell82]**

Mitchell, T.M.: Generalization às Search. Artificial Intelligence 18, pp. 203-266, 1982.

- **[Moriyón94]**

  Moriyón, R., Szekely, P. y Neches, R.: Automatic Generation of Help from Interface Design Models, en Proceedings of CHI'94, ACM Press, 1994.

- **[Myers90a]**

  Myers, B.A. et al.: Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces. IEEE Computer, Vol. 23, no. 11, pp. 71-85. 1990.

- **[Myers90b]**

  Myers, B.A.: A New Model for Handling Input. ACM Transactions on Information Systems, vol. 8, no. 3, pp 289-320. 1990.

- **[Myers92a]**

  Myers, B.A. y Rosson, M.B.: Survey on User Interface Programming. Proceedings of CHI'92, pp. 195-202. ACM Press, New York, 1992.

- **[Myers92b]**

  Myers, B.A., Giuse, D. y Vander Zanden, B.: Declarative Programming in a Prototype-Instance System: Object-Oriented Programming without Writting Methods. Sigplan Notices, vol. 27, no. 10, pp. 184-200. ACM Conference on Object-Oriented Programming, System Languages and Applications, OOPSLA'92. 1992.

- **[Myers93]**

  Myers, B.A.: Demonstrational Interfaces: a Step Beyond Direct Manipulation. In A. Cypher, editor, Watch What I Do: Programming by Demonstration, pages 485-512. MIT Press, Cambridge, MA, 1993.

- **[Myers96a]**

  Myers, B.A., Miller, R.C., McDaniel, R., y Ferrency, A.: Easily Adding Animations to Interfaces Using Constraints. ACM Symposium on User Interface Software and Technology, UIST'96. Seattle, WA. pp. 119-128. 6-8 Noviembre 1996.

- **[Myers96b]**

  Myers, B.A. y Kosbie, D.: Reusable Hierarchical Command Objects. En Proceedings CHI'96: Human Factors in Computing Systems. Vancouver, BC, Canada, pp. 260-267. 1996.

- **[Myers97a]**

Myers et al.: The AMULET V3.0 Reference Manual. Carnegie Mellon University Computer Science Department CMU-CS-95-166-R2. System available from http://www.cs.cmu.edu/~amulet. Marzo, 1997.

- **[Myers97b]**

Myers, B.A., McDaniel, R.G., Miller, R.C, Ferrency, A.S., Faulring, A., Kyle, B.D., Mickish, A., Klimovitski, A. y Doane, P.: The Amulet Environment: New Models for Effective User Interface Software Development. IEEE Transactions on Software Engineering, Vol. 23, no. 6, pp. 347-365. Junio, 1997.

- **[Nass95]**

Nass, C., Moon, Y., Fogg, B.J., Reeves, B. Y Dryer, D.C.: Can Computer Personalities be Human Personalities? International Journal of Human Computer Studies, 43, 1995.

- **[Nevill-Manning94]**

Nevill-Manning, C.G., Witten, I.H. y Maulsby, D.: Compression by Induction of Hierarchical grammars. En Proceedings of Data Compression '94, pp. 244-253. 1994.

- **[Orosco98]**

Orosco, R.: Un modelo arquitectónico para sistemas de visualización y exploración de información. Tesis doctoral. Dtor. R. Moriyón, E.T.S. Ingeniería Informática, UAM, Octubre 1998.

- **[Palanque93]**

Palanque, A., Bastide, R. y Dourte, L.: Contextual Help for Free with Formal Dialog Design. En 5th International Conference on Human-Computer Interaction, Orlando, Florida, USA. 8-13 Agosto, 1993.

- **[Palminter89]**

Palminter, S., Elkerton, J. y Baggett, P.: Animated Demonstrations versus Written Instructions for Learning Procedural Tasks. Technical Report C4E-ONR-2, Center of Ergonomics, Dept. of Industrial and Operations Engeneering, University of Michigan, Enero 1989.

- **[Pangoli95]**

Pangoli, S. y Paternó, F.: Automatic Generation of Task-oriented Help, en Proceedings UIST'95, Pittsburgh, ACM Press, 1995.

- **[Paris89]**

Paris, C.L.: The Use of Explicit User-Models in a Generation System for Tailoring Answers to the User's Level of Expertise. En Kobsa, A. Y Wahlster (Eds.): User Models in Dialog Systems. Springer Velag, 1989.

- **[Paternó97]**

Paternó, F.D.: Understanding Task Model ans User Interface Architecture Relationships. CNUCE Internal Report, Diciembre 1997.

- **[Paternó98]**

Paternó, F.D., Breedvelt-Schouten, I.M, Koning, N.M..: Deriving Presentations from Task Models. En Proceedings EHCI'98, Creta (Grecia), Septiembre 1998.

- **[Peterson81]**

Peterson, J.L.: Petri Net Theory and the Modelling of Systems. Prentice-Hall, Englewood Cliffs, N.J., 1981.

- **[Philips96]**

Royal Philips Electronics. Vision of the Future: Interactive Books. http://www-us.design.philips.com/vof/vofsite5/vof5lev3/book3/book3.htm. 1996.

- **[Puerta96a]**

Puerta, A.R.: The MECANO Project: Enabling User-Task Automation during Interface Development.. En Proceedings of AAAI'96 Spring Symposium on Acquisition, Learning & Demonstration: Automating Tasks for Users, AAAI Press, pp. 117-121. Standford, Marzo 1996.

- **[Puerta96b]**

Puerta, A.R.: The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, pp. 19-35. Belgica, Junio 1996.

- **[Puerta97]**

Puerta, A.R.: A Model-Based Interface Development Environment. IEEE Software, 14(4), p. 41-47, Julio/Agosto 1997.

- **[Puerta98]**

Puerta, A.R.: Supporting User-Centered Design of Adaptive User Interfaces Via Interface Models. First Annual Workshop On Real-Time Intelligent User Interfaces For Decision Support And Information Visualization, San Francisco, Enero 1998.

- **[Puerta99a]**

  Puerta, A.R., Cheng, E., Ou, T., y Min, J.: MOBILE: User-Centered Interface Building. CHI99: ACM Conference on Human Factors in Computing Systems. Pittsburgh, Mayo 1999.

- **[Puerta99b]**

  Puerta, A.R. y Eisenstein, J.: Towards a General Computational Framework for Model-Based Interface Development Systems. IUI99: International Conference on Intelligent User Interfaces, Los Angeles, Enero 1999.

- **[Remde87]**

  Remde, J.R., Gomez, L.M. y Landauer, T.K.: SuperBook: An Automatic Tool for Information Exploration - Hypertext?. ACM Hypertext'87 Proceedings p.175-188. 1987.

- **[Rickel98]**

  Rickel, J. y Johnson, W. L.: Task-Oriented Dialogs with Animated Agents in Virtual Reality. En S. Prevost y E. Churchill (Eds.): Workshop on Embodied Conversational Characters, 1998.

- **[Rieber91]**

  Rieber, L.: Animation, Incidental earning and Continuing Motivation. Journal of Educational Psycology, 83 (3), 318-328. 1991.

- **[Rodríguez97]**

  Rodríguez, P., García, F., Contreras, J. y Moriyón, R.: Parsing Techniques for User-Task Recognition, Proceedings of the Fifth International Workshop on Functional Modelling of Complex Technical Systems, Paris-Troyes, Francia, Julio 1997.

- **[Schlungbaum96]**

  Schlungbaum, E. y Elwert, T.: *Automatic User Interface Generation from Declarative Models.* Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, Belgica, Junio 1996.

- **[Schreiber94a]**

  Schreiber, S.: The BOSS System: Coupling Visual Programming with Model Based Interface Design. En DSV-IS'94, pp. 161-179. 1994.

- **[Schreiber94b]**

  Schreiber, S.: Specification and Generation of User Interfaces with the BOSS-System. En EWHCI'94, pp. 107-120. 1994.

- **[Shieber92]**

  Shieber, S.: Constraint-based Grammar Formalism. Parsing and Type Inference for Natural and Computer Languages. MIT Press. 1992.

- **[Shneiderman83]**

  Shneiderman, B.: Direct Manipulation: a Step Beyond Programming Languages. IEEE Computer, Vol. 16, No. 8, Agosto 1983, pp. 578-68. 1983.

- **[Sibertin-Blanc85]**

  Sibertin-Blanc, C.: High-level Petri Nets with Data Structure. En 6th European Workshop on Petri Nets and Applications. Espoo, Finlandia, Junio 1985.

- **[Sukaviriya90]**

  Sukaviriya, P. y Foley, J.D: Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help, UIST'90, pp. 152-166, 1990.

- **[Sukaviriya94]**

  Sukaviriya, P., Kovacevic, S. (organizadores) y Foley, J.D., Myers, B.A., Olsen, D.R. y Schneider-Hufschmidt, M. (invitados): Model-Based User Interfaces: What are They and Why Should we Care?. UIST'94 panel, pp. 133-135, 1994.

- **[Szekely92]**

  Szekely, P., Luo, P. y Neches, R.: Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design, en Proceedings SIGCHI'92, pp. 507-514, ACM Press, 1992.

- **[Szekely93]**

  Szekely, P., Luo, P. y Neches, R.: Beyond Interface Builders: Model-Based Interface Tools. Proceedings de INTERCHI'93, pp. 383-390. 1993.

- **[Szekely95]**

  Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J. y Salcher, E.: Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach. En EHCI95, pp. 120-150. 1995.

- **[Szekely96a]**

  Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J. y Salcher, E.: Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach. In

Engineering for Human-Computer Interaction, L. Bass and C. Unger (eds), pp. 120-150. Chapman & Hall, 1996.

- **[Szekely96b]**

  Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics. Belgica, Junio 1996.

- **[Takeuchi95]**

  Takeuchi, A. y Naito, T.: Situated Facial Displays: Towards Social Interaction. En Katz, I., Mack, R., Marks, L., Rosson, M. y Nielsen, J. (Eds.), Human Factors in Computing Systems: CHI'95 Conference Proceedings, pp. 450-455. New York: ACM Press. 1995.

- **[Tam98]**

  Tam, R. C. M, Maulsby, D. y Puerta, A.: U-TEL: A Tool for Eliciting User Task Models from Domain Experts. En Proceedings IUI'98: 1998 International Conference on Intelligent User Interfaces. San Francisco, CA: ACM Press. 1998.

- **[Vanderdonckt94]**

  Vanderdonckt, J.: Automatic Generation of a User Interface for Highly Interactive Business-Oriented Applications. En CHI'94, pp. 41 y 123-124. ACM Press. 1994.

- **[Vanderdonckt95]**

  Vanderdonckt, J.: Knowledge-Based Systems for Automated User-Interface Generation: the TRIDENT Experience. Technical Report RP-95-010, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur. 1995.

- **[Van der Aalst94]**

  Van der Aalst, W.M.P., Van Hee, K.M. y Houben, G.J. Modeling Workflow Management Systems With High-Level Petri Nets. De los editores G. De Michelis, C. Ellis, y G. Memmi, Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms, pp 31--50, 1994.

- **[Wang95]**

  Wang, X.: Learning by Observation and Practice: an Incremental Approach for Planning Operator Adquision. Proceedings of the 12th International Conference on Machine Learning. 1995.

- **[Weske96]**

Weske, M.: Event-Based Modeling and Analysis of Distributed Workflow Executions. Fachbericht Angewandte Mathematik und Informatik 16/96-I, Universidad de Munich, 1996.

- **[Wiecha89]**

  Wiecha, C., Bennett, W. Et al.: Generating Highly Interactive User Interfaces. En CHI'89, pp. 277-282. 1989.

- **[Wiecha90]**

  Wiecha, C., Bennett, W., Boies, S., Gould, J. y Green, S.: ITS: A Tool for Rapidly Developing Interactive Applications. ACM Transactions on Information Systems, Vol. 8, No. 3, Julio 1990, pp. 204-236. 1990.

- **[Wilson93]**

  Wilson, S., Johnson, P., Kelly, C., Cunningham, J. y Markopoulos, P.: Beyond hacking: a Model Based Approach to User Interface Design. In Proceedings of HCI'93, pp. 217-231. 1993.

- **[Wilson96]**

  Wilson, S., y Johnson, P.: Bridging the Gap: From Work Tasks to User Interface Designs. Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, pp. 77-94. Belgica, Junio 1996.

- **[Yu88]**

  Yu, C-C., y Robertson, S.P.: Plan-Based Representations of Pascal and Fortran Code. Human Factors in Computiong Systems Proceedings of CHI'88, 1988.

Apéndice I : Gramática de Especificación de Modelos de ATOMS

En este Apéndice se describe la gramática del lenguaje de especificación de modelos de tareas para aplicaciones interactivas implementado en ATOMS. Para cada sistema interactivo que se desee modelizar se modelizarán un conjunto de tareas y reglas que las relacionan, siguiendo el esquema de la gramática de a continuación:

// los modelos se componen de tareas y reglas

modeloTareasATOMS ::= (tarea | regla)*

// definición de las tareas

tarea ::= **TASK** [*nuevaTarea*] [**ISA** nombreTarea expresiónValor*] **ALIAS** *alias*

[DESCRIPTION *descripciónTarea*]

descripciónInteracción*

(**PARAMETER** expresiónDePasoDeParámetro)*

(**POSTPARAMETER** expresiónDePasoDeParámetro)*

END

// prototipos de tareas facilitados por el *framework* ATOMS

nombreTarea ::= **ATOMIC** | **COMPOSED** | **TEXTEDIT** | **EDITFIELD** | **SELECTION** | **MENU** | **BUTTON** | **LIST** | **COMBO** | **ONESHOT** | **MOVEGROW** | **MOVE** | **RESIZE** | *nuevaTarea*

// descripción del *patrón de interacción*

descripciónInteracción ::= **PATTERN** condiciónCamino |

PATTERN DROP camino

// descripción de las reglas

regla ::= **RULE** *nombreRegla* [**ISA** *nombreReglaPrototipo*]

TASK *nombreTarea*

SUBTASKS (*tipoSubtarea* **AS** *nombreSubtarea*)+

SEQUENCING (**XOR** | **AND** | **SEQ**)

[**MULTIPLE** (*nombreSubtarea* condición)+]

[**OPTIONAL** (*nombreSubtarea* condición)+]

[**CANCELING** *nombreSubtarea*+]

(PRECONDITION IN *nombreSubtarea* condición)*
(PARAMETER expresiónDePasoDeParámetro)*
(POSTPARAMETER expresiónDePasoDeParámetro)*
(DEPENDENCE IN *nombreSubtarea* condiciónCamino)*
END

// gramática de expresiones del lenguaje
condiciónCamino ::= camino operadorLógico expresiónValor

camino ::= identificador[.camino]

identificador ::= *nombreVariable* |
nombreAtributoAMULET |
nombreTarea |
num |
%num |
PROTOTYPE |
ISA |
OWNER |
TUTORESULT |
ALEAT *num* | **ALEAT** *nombreLista* | **ALEAT** *nombreTarea+*

expresiónValor ::= identificador operadorAritmético expresiónValor |
condición |
identificador

condición ::= expresiónValor operadorLógico expresiónValor |
expresiónValor

operadorLógico ::= || | && | == | != | > | < | >= | <= | **ISA**

operadorAritmético ::=
+ | - | * | /

expresiónDePasoDeParámetro ::= nombreParámetro = expresiónValor |

nombreParámetro **ADDPOINT** (expresiónValor, expresiónValor)

nombreParámetro ::= *nombreDeParámetro* |
%num

Apéndice II: Modelización de la Tarea *Añadir una nueva asignación de docencia*

Como ejemplo de la utilización de la gramática de especificación de modelos descrita en el Apéndice anterior, es esta sección se muestra un ejemplo detallado de cómo se ha modelizado una tarea interactiva. En concreto, esta modelización se corresponde con la jerarquía de tareas mostrada en la Figura 19.

Tareas atómicas.

Abrir cuadro de diálogo:

// Menú Nueva Asignación

TASK *"Seleccionar el elemento de menú Nueva Asignación"* **ISA MENU** "Asignación" **ALIAS** *AbrirAsignaciónMenú*

DESCRIPTION *Muestra el cuadro de diálogo que permite añadir una nueva asignación*

END

// Botón Nueva Asignación

TASK *"Pulsar el botón de Nueva Asignación"* **ISA BUTTON** "Ok" **ALIAS** *AbrirAsignaciónBotón*

DESCRIPTION *Muestra el cuadro de diálogo que permite añadir una nueva asignación*

END

Seleccionar de listas:

// Lista de profesores

TASK *"Seleccionar el profesor de la lista"* **ISA LIST** "Profesor:" "nombre" **ALIAS** SeleccionarProfesorDeLista

DESCRIPTION *Selecciona el nombre del profesor de la lista 'Profesor:'*

END

// Lista de asignaturas

TASK *"Seleccionar la asignatura de la lista"* **ISA LIST** "Asignatura:" "asignatura" **ALIAS** SeleccionarAsignaturaDeLista

DESCRIPTION *Selecciona el nombre de la asignatura de la lista 'Asignatura:'*

END

// Lista de grupos de alumnos

TASK *"Seleccionar el grupo de la lista"* **ISA LIST** "Grupo:" "grupo" **ALIAS** *SeleccionarGrupoDeLista*

DESCRIPTION *Selecciona el grupo de la lista 'Grupo:'.*

END

Aceptar/Cancelar valores:

// Botón de Ok

TASK *"Confirmar los valores facilitados"* **ISA BUTTON** "Ok" **ALIAS** *Confirmar*

DESCRIPTION *Confirma los valores seleccionados*

END

// Botón de Cancelar

TASK *"Cancelar la tarea en curso"* **ISA BUTTON** *"Cancel"* **ALIAS** *Cancelar*

DESCRIPTION *Cancela la tarea que se está realizando*

END

Tareas compuestas.

// Tarea que modeliza el proceso completo

TASK *"Añadir una nueva asignación de docencia"* **ISA COMPOSED ALIAS** *CrearNuevaAsignación*

DESCRIPTION *Establece una nueva restricción de docencia*

END

// Tarea que hace transparente el proceso según el cuál se abre el cuadro de diálogo

TASK*"Abrir el cuadro de diálogo de Nueva Asignación"* **ISA COMPOSED ALIAS** *AbrirNuevaAsignación*

DESCRIPTION *Muestra el cuadro de diálogo que permite añadir una nueva asignación*

END

// Representación de todo el proceso realizado por medio del cuadro de diálogo

TASK*"Proporcionar los valores que tendrá la relación"* **ISA COMPOSED ALIAS** *DarValores*

DESCRIPTION *Selecciona los valores para la nueva relación de asignación*

END

// Representación de la selección de parámetros (antes de presionar el botón de
// confirmación 'Ok')

TASK *"Establecer las componentes de la relación de docencia"* **ISA COMPOSED ALIAS** *SeleccionarDatos*

DESCRIPTION *Establece qué profesor impartirá una determinada asignatura a un determinado grupo*

END

Reglas.

// Abrir el cuadro de diálogo

RULE *ReglaAbrirNuevaAsignación*

TASK *AbrirNuevaAsignación*

SUBTASKS *AbrirAsignaciónMenú*

AbrirAsignaciónBotón

SEQUENCING XOR

END

// Operaciones sobre las listas para seleccionar los datos deseados

RULE *ReglaSeleccionarDatos*

TASK *SeleccionarDatos*

SUBTASKS *SeleccionarProfesorDeLista* **AS** *Profesor*

SeleccionarAsignaturaDeLista **AS** *Asignatura*

SeleccionarGrupoDeLista **AS** *Grupo*

MULTIPLE *Profesor* **TRUE** *Asignatura* **TRUE** *Grupo* **TRUE**

SEQUENCING AND

CANCELLING *Cancelar*

PARAMETER *nombre* = *SeleccionarProfesorDeLista.Params.nombre*

PARAMETER *asignatura* = *SeleccionarAsignaturaDeLista.Params.asignatura*

PARAMETER *grupo* = *SeleccionarGrupoDeLista.Params.grupo*

END

// Operaciones sobre el cuadro de diálogo que permite añadir la nueva

// relación de asignación

RULE *ReglaDarValores*

TASK *DarValores*

SUBTASKS *SeleccionarDatos* **AS** *Datos*

Confirmar

SEQUENCING SEQ
PARAMETER *nombre = Datos.Params.nombre*
PARAMETER *asignatura = Datos.Params.asignatura*
PARAMETER *grupo = Datos.Params.grupo*
END

// Regla que permite unir todo el proceso
// relación de asignación
RULE *ReglaCrearNuevaAsignación*
TASK *CrearNuevaAsignación*
SUBTASKS *AbrirNuevaAsignación*
DarValores **AS** *Valores*
SEQUENCING SEQ
PARAMETER *nombre = Valores.Params.nombre*
PARAMETER *asignatura = Valores.Params.asignatura*
PARAMETER *grupo = Valores.Params.grupo*
END

Apéndice III: Generación de Mensajes de Guía de DARTS

En este Apéndice se muestra la gramática que utiliza internamente DARTS con el objetivo de generar mensajes de guía adecuados para la enseñanza de tareas. Los modelos de tareas se suponen generados conforme al lenguaje de especificación de modelos de tareas que se encuentra en el Apéndice I. Los textos que se encuentran en **negrita** se corresponden con textos que son mostrados tal cual a los usuarios (símbolos terminales de la gramática), ya sean no dependientes del contexto. Los textos que se generan con letra *cursiva* hacen referencia a información contenida en la modelización de las aplicaciones. Si el formato es *subrayado y en azul*, la información viene acompañada de un hiperenlace. Finalmente, los mensajes elaborados a partir de información implícita en los modelos se encuentran con letra subrayada.

// distintos tipos de mensajes facilitados por DARTS a los usuarios

msgEnseñanza ::= msgDescTarea |

msgAcción |

msgModoActivo

Mensajes del modo descriptivo.

Mensajes de descripción de tarea:

// información únicamente descriptiva del objetivo de la tarea,

// sin referencias a su proceso de realización

msgDescTarea ::= contextoMensaje

Esta tarea *descripciónTarea*

[Puede realizar esta tarea sobre cualquiera de los objetos que están parpadeando en color *refColor*]

// parte dependiente del nivel de profundidad para el contexto que se desee visualizar

contextoMensaje ::= **Ha solicitado enseñanza sobre cómo** *nombreTarea*

[Para ello debe *nombreSubTarea*]*

Mensajes de acción:

// información descriptiva que proporciona información acerca de la

// descomposición jerárquica de una tarea

msgAcción ::= contextoMensaje

Para hacer esta tarea, usted tiene que:

ExplicaciónSubtareas |

Ha terminado la tarea *nombreTarea*

explicaciónSubtareas ::= *nombreSubtarea* |

nombreSubtarea

(o bien | y después | y)

explicaciónSubtareas

Mensajes del *modo activo*:

// información detallada, enfocada a la explicación de las *tareas pendientes* de realizar

msgModoActivo ::= encabezamientoActivo

// relación temporal de la tarea con su *supertarea*

(A continuación, usted tiene que *nombreTareaCompuesta* |

Primero, usted tiene que *nombreTareaCompuesta* |

Finalmente, usted tiene que *nombreTareaCompuesta*)*

explicaciónTareasPendientes

// contexto que indica las tareas realizadas, en cuál se encuentra el usuario,

// o sobre cuál ha pedido información

encabezamientoActivo ::= **Ha solicitado enseñanza sobre cómo** *nombreTarea* |

[Está realizando la tarea *nombreTarea* **[sobre el objeto que está parpadeando | los objetos que están parpadeando]] [Ha terminado la tarea** *nombreTarea* **[sobre (el objeto que está parpadeando | los objetos que están parpadeando) en color** refColor]].

explicaciónTareasPendientes ::=

// cuando la subtarea explicada no es ni la primera

// ni la última de las subtareas de una regla

Para continuar esta tarea, usted tiene que explicaciónTareaPendiente |

// cuando la subtarea explicada es la primera parte de la tarea superior

Primero, para hacer esta tarea usted tiene que explicaciónTareaPendiente |

// cuando la subtarea explicada finaliza la tarea superior

Finalmente, para hacer esta tarea usted tiene que explicaciónTareaPendiente |

// cuando sólo hay que hacer una de las subtareas,

// o cuando hay varias alternativas con 'Y'

Para hacer esta tarea, usted tiene que explicaciónTareaPendiente

explicaciónTareaPendiente ::= *nombreTareaAtómica* cómoTareaAtómica. **[Esta tarea puede hacerse más de una vez.| Puede volver a realizar esta tarea] [Esta tarea es opcional.]**

[(y | o) explicaciónTareaPendiente]

// información obtenida a partir de la especificación de la interacción,

// mediante el *patrón de interacción,* incluida en el modelo de tareas

cómoTareaAtómica ::= tipoAcción lugarAcción valoresAcción

tipoAcción ::= pulsando el botón (izquierdo | derecho) del ratón |

tecleando |

arrastrando |

lugarAcción ::= [sobre] el objeto que está parpadeando en color refColor |

sobre alguno de los objetos que están parpadeando en color refColor |

en el campo que está parpadeando en color refColor

valoresAcción ::= (la cadena de texto *cadenaDeTexto* | la cadena de texto deseada) |

x unidades en horizontal y *y* unidades en vertical

refColor ::= **rojo | verde | azul | amarillo | morado**

Apéndice IV: Gramática de Especificación de Escenarios

En este Apéndice se muestra la gramática del lenguaje que permite la especificación interactiva de escenarios. Cada fichero de escenarios puede contener uno o más escenarios, viniendo cada uno de ellos definido según el símbolo 'escenario' de la siguiente gramática:

// definición de un escenario

escenario ::= **SCENARY** *nombreEscenario*

[**SCNDESC** *descripciónEscenario*]

instrucciónEscenario*

ENDSCN

// tipos de instrucciones que pueden incorporarse a un escenario

// condicional

instrucciónEscenario ::= **IF** condición instrucciónEscenario* [**ELSE** instrucciónEscenario*] **ENDIF** |

// iteraciones

FOR *nombreVariable* **FROM** expresiónValor **TO** expresiónValor **STEP** expresiónValor instrucciónEscenario* **ENDFOR** |

FOR *nombreVariable* **IN** *nombreLista* instrucciónEscenario* **ENDFOR** |

WHILE condición instrucciónEscenario* **ENDWHILE** |

DOWHILE condición instrucciónEscenario* **ENDDO** |

// llamadas a otros escenarios

CALL [*nombreFichero.*]*nombreEscenario* expresiónValorParámetro* |

// emulación de tarea

TASK *nombreTarea* expresiónValorParámetro* |

// *undo* de tarea

TASKUNDO *num* |

// temporización

LAPSUS expresiónValor |

CHARLAPSUS expresiónValor |

// tratamiento de parámetros *unbound*

SKIP |

ASK |

// utilización de variables

VARIABLE nombreVariable [expresiónValor] |
SET nombreVariable expresiónValor |
// mensajes de *feedback*
MSG *texto* [**HIGHLIGHT** expresiónValor+] |
// establecimiento de refresco en visualización
REFRESHMODE (STEP_BY_STEP | TASK_BY_TASK | WHOLE_SCN)

Apéndice V: Ejemplo de Especificación de un Escenario

En este Apéndice se muestra un ejemplo completo de la especificación de un escenario, según la gramática mostrada en el Apéndice anterior. La ejecución de este escenario de ejemplo ocasiona que la aplicación OOPI-TasKAD refleje el estado mostrado en la Figura 59.

SCENARY *RealizarDiseñoCoche*

SCNDESC Realiza un diseño para trabajar con él

// ajustar la velocidad de ejecución de las animaciones

LAPSUS *75*

// realizar las distintas partes del diseño, cada zona en distintas capas

LINE *Figure2 16 337 640 2 2*

ADDLAYER *2*

SETLAYER *2*

RECTANGLE *Figure19 99 81 523 257*

ADDLAYER *3*

SETLAYER *3*

RECTANGLE *Figure23 142 103 90 57*

RECTANGLE *Figure24 242 103 90 57*

RECTANGLE *Figure26 342 103 90 57*

ADDLAYER *4*

SETLAYER *4*

RECTANGLE *Figure27 529 205 67 133*

ADDLAYER *5*

SETLAYER *5*

ELLIPSE *Figure0 189 280 62 58*

ELLIPSE *Figure1 437 279 66 61*

ADDLAYER *6*

SETLAYER *6*

POLYGON *Figure7 3 435 315 435 305 252 302 253 312*

POLYGON *Figure8 3 185 310 187 300 142 297 140 303 142 307*

POLYGON *Figure11 4 156 269 154 269 149 293 151 293*

POLYGON *Figure5 1 185 317 145 312 145 308 185 311*

POLYGON *Figure3 1 253 313 252 319 435 321 435 317*

POLYGON *Figure9 1 252 302 246 290 235 282 221 279 206 282 196 289 189 299 152 297 159 263 225 255 275 219 377 224 423 262 533 278 543 308 500 304 489 291 477 283 460 281 445 289 441 295 436 304*

POLYGON Figure12 4 274 221 233 252 286 253 289 222

POLYGON Figure13 4 273 219 213 255 224 255

POLYGON *Figure14 4 293 222 376 226 414 259 293 252*

POLYGON *Figure15 4 377 223 422 261 435 262*

ADDLAYER *7*

SETLAYER *5*

LINE *Figure16 291 252 3 49*

LINE *Figure18 297 260 11 1*

LINE *Figure17 421 262 1 42*

ADDLAYER 8

SETLAYER *8*

POLYGON *Figure6 3 502 315 548 317 551 308 503 306*

POLYGON *Figure4 1 502 317 501 321 542 322 545 318*

POLYGON *Figure10 1 536 284 538 284 545 304 543 304*

ADDLAYER *9*

SETLAYER *9*

LINE *Figure28 560 206 1 131*

POLYGON *Figure29 8 99 81 29 83 340 22 671 82*

ENDSCN

Apéndice VI: Gramática de Especificación de Cursos de CACTUS

En este Apéndice se muestra la gramática utilizada internamente por CACTUS para el lenguaje de especificación de cursos interactivos de enseñanza. Los diseñadores raramente necesitarán conocer las estructuras de este lenguaje, según vimos en la Sección 4.4.3.2, ya que CACTUS permite la especificación interactiva de sus contenidos. El símbolo 'cursoCACTUS' se corresponde con un curso interactivo:

```
// curso e enseñanza interactiva
cursoCACTUS := CRS nombreCurso
        CRSDESC descripciónDeCurso
        instruccionGlobal*
        unidadCurso*
        ENDCRS

// contenido de una unidad pedagógica del curso
unidadCurso := UNIT nombreUnidad
        UNITDESC descripciónDeUnidad
        // relación de secuenciamiento entre unidades
        [PREVIOUS nombreUnidad+]
        (instrucciónGlobal | instrucciónUnidad)*
        ENDUNIT

// instrucciones que pueden ser incorporadas a la parte global de los cursos, fuera de las
        unidades
        // fondo por defecto
instrucciónGlobal ::= IMAGE ficheroDeImagen posX posY |
        // modo de funcionamiento de DARTS
        MODE (STRICT | FLEXIBLE | AUTO) |
        // Modo de debug utilizado
        DEBUG (DISABLED | TRACEALL | TRACEUSER | FULL) |
        // evaluación del comportamiento de los alumnos
        [EVAL (GLOBAL | TASK | WORST)]
        // establecimiento de refresco en visualización
        REFRESHMODE (STEP_BY_STEP | TASK_BY_TASK | WHOLE_SCN)
```

// carga/ejecución de escenarios

LOADSCN *pathFicheroEscenario* |

SCENARY [*pathFicheroEscenario.*]*nombreEscenario* (*nombreParámetro* expresiónValor)* |

// utilización de variables

VARIABLE *nombreVariable* [expresiónValor] |

SET *nombreVariable* expresiónValor |

// mensajes de *feedback*

MSG *texto* [**HIGHLIGHT** expresiónValor+] |

// utilización de imágenes

IMAGE *ficheroDeImagen posX posY* [*width heihgt*]|

// instrucciones que pueden ser utilizadas dentro de las unidades pedagógicas

// carga/ejecución de escenarios

instrucciónUnidad ::= **LOADSCN** *ficheroDeEscenarios* |

SCENARY [*ficheroDeEscenarios.*]*nombreEscenario* (*nombreParámetro* expresiónValor)* |

// enseñanza de una tarea

TUTORING *nombreTarea* (*nombreParámetro* expresiónValor)* |

// ejecución de un ejemplo

SAMPLE *nombreTarea* (*nombreParámetro* expresiónValor)* |

// *feedback* al usuario

MSG *texto* [**HIGHLIGHT** expresiónValor+] |

// utilización de imágenes dinámicas y estáticas

IMAGE *ficheroDeImagen posX posY* [*width heihgt*]|

DYNAMAGE expresiónValor+ *posX posY tamX tamY* |

// Soporte interno para modificaciones *in-place*

_DESC *descripciónDeTareaEjemploOEscenario* |

// soporte para hiperenlaces

LINK *texto* (((+ | -) *num*) | **COVER** | **INDEX** | **GRAPH** | **PROPOSAL** | **UNIT** *nombreUnidad* | **GLOSARY**) |

// modo de libertad para la enseñanza de DARTS

MODE (**STRICT** | **FLEXIBLE** | **AUTO**) |

// modo de refresco de la visualización de escenarios

REFRESHMODE (**STEP_BY_STEP** | **TASK_BY_TASK** | **WHOLE_SCN**) |

// soporte a la depuración

DEBUG (DISABLED | TRACEALL | TRACEUSER | FULL) |

// utilización de variables

VARIABLE *nombreVariable* [expresiónValor] |

SET *nombreVariable* expresiónValor |

// bloques condicionales, iterativos y de ejecución aleatoria

WHILE condición (instrucciónGlobal | instrucciónUnidad)* **ENDWHILE** |

DOWHILE condición (instrucciónGlobal | instrucciónUnidad)* **ENDDO** |

IF condición (instrucciónGlobal | instrucciónUnidad)* [**ELSE** (instrucciónGlobal | instrucciónUnidad)*] **ENDIF** |

FOR *nombreVariable* **IN** *nombreLista* (instrucciónGlobal | instrucciónUnidad)* **ENDFOR** |

FOR *nombreVariable* **FROM** expresiónValor **TO** expresiónValor **STEP** expresiónValor (instrucciónGlobal | instrucciónUnidad)* **ENDFOR** |

ALEAT *num* (**BLOCK** (instrucciónGlobal | instrucciónUnidad)* **ENDBLOCK**)* **ENDALEAT**

Apéndice VII: Ejemplo de Especificación de Cursos de CACTUS

En este Apéndice se muestra la especificación completa de un curso interactivo de CACTUS. Esta especificación sigue la gramática del lenguaje mostrado en el Apéndice anterior, aunque ha sido completamente definido de manera interactiva utilizando las facilidades proporcionadas por el sistema y explicadas en la Sección 4.4.3.2.

// Curso sobre el manejo esencial Schoodule

COURSE *Schoodule*

// indicar el modo de depuración

DEBUG DISABLED

// texto descriptivo del curso en cuestión

CRSDESC *Este curso enseña a utilizar la aplicación Schoodule*

// modo de evaluación del curso, común a todas las unidades

EVAL TASK

MSG *"Atención, comienza el curso!"*

// insertar la imagen del fondo

IMAGE bookpage.gif 0 0 **MAXWIDTH MAXHEIGHT**

// Unidad pedagógica para añadir un nuevo profesor

UNIT *AñadirProfesores*

// texto descriptivo de la unidad pedagógica

UNITDESC *En esta unidad, aprenderás a añadir un nuevo profesor*

// cargar un escenario que abra una nueva base de datos

SCENARY *bd_crs.NuevaBaseDeDatos*

SAMPLE *NuevoProfesor* nombre *"Juan Cano"* DNI *"11111111"* dirección *"Jorge Manrique, 14"*

TUTORING *NuevoProfesor* nombre *"Juan Cano"* DNI *"11111111"* dirección *"Jorge Manrique, 14"*

TUTORING *NuevoProfesor* nombre *"Julián Fernández"*

TUTORING *NuevoProfesor* DNI *"22222222"*

TUTORING *NuevoProfesor* dirección *"Corralejos, 20"*

TUTORING *NuevoProfesor*

// cerrar la base de datos descartando los cambios realizados

SCENARY *bd_crs.CerrarSinGuardar*

ENDUNIT

// Unidad pedagógica para añadir un nuevo aula

UNIT *AñadirAulas*

// Lecciones previas que se deberán enseñar con anterioridad

PREVIOUS *AñadirTiposDeAula*

// texto descriptivo de la unidad pedagógica

UNITDESC *En esta unidad aprenderás a añadir un nuevo aula*

// cargar un escenario que abra una nueva base de datos

SCENARY *bd_crs.NuevaBaseDeDatos*

// escenario para incializar varios tipos de locales

SCENARY bd_crs.*AñadirTiposDeLocalesPorDefecto*

SAMPLE *NuevoAula* ubicación *"Edificio nuevo"* capacidad *"80"* nombre *"Vivaldi"* tipo *"Laboratorio"*

TUTORING *NuevoAula* ubicación *"Edificio nuevo"* capacidad *"80"* nombre *"Vivaldi"* tipo *"Laboratorio"*

TUTORING *NuevoAula* capacidad *"54"*

TUTORING *NuevoAula* nombre *"Mozart"* tipo *"Laboratorio"*

TUTORING *NuevoAula* ubicación *"Edificio antiguo"* tipo *"Audiovisuales"*

TUTORING *NuevoAula*

// cerrar la base de datos descartando los cambios realizados

SCENARY *bd_crs.CerrarSinGuardar*

ENDUNIT

// Unidad pedagógica para añadir una nueva asignación de docencia

UNIT *AñadirAsignaciones*

// Lecciones previas que se deberán enseñar con anterioridad

PREVIOUS *AñadirProfesores AñadirAsignaturas AñadirClases*

// texto descriptivo de la unidad pedagógica

UNITDESC *En esta unidad vas a aprender a añadir asignaciones de docencia*

// cargar un escenario que abra una nueva base de datos

SCENARY *bd_crs.NuevaBaseDeDatos*

// escenario para incializar varios tipos de locales

SCENARY bd_crs.*AñadirInformaciónParaAsignación*

SAMPLE *CrearNuevaAsignación* profesor *"Juan Cano"* asignatura *"Matemáticas"* grupo *"3-B"*

TUTORING *CrearNuevaAsignación* profesor *"Juan Cano"* asignatura *"Matemáticas"* grupo *"3-B"*

TUTORING *CrearNuevaAsignación* profesor *"Fernando Echave"* asignatura *"Música"* grupo *"4-D"*

TUTORING *CrearNuevaAsignación*

// cerrar la base de datos descartando los cambios realizados

SCENARY *bd_crs.CerrarSinGuardar*

ENDUNIT

// Unidad pedagógica para añadir una nueva asignatura

UNIT *AñadirAsignaturas*

// texto descriptivo de la unidad pedagógica

UNITDESC En esta unidad vas a aprender a añadir asignaturas

// cargar un escenario que abra una nueva base de datos

SCENARY *bd_crs.NuevaBaseDeDatos*

SAMPLE *NuevaAsignatura* nombre *"Matemáticas"* horas *"5"*

TUTORING *NuevaAsignatura* nombre *"Matemáticas"* horas *"5"*

TUTORING *NuevaAsignatura* nombre *"Ciencias"*

TUTORING *NuevaAsignatura* horas *"4"*

TUTORING *NuevaAsignatura*

// cerrar la base de datos descartando los cambios realizados

SCENARY *bd_crs.CerrarSinGuardar*

ENDUNIT

// Unidad pedagógica para añadir una nueva asignatura

UNIT *AñadirClases*

// texto descriptivo de la unidad pedagógica

UNITDESC *En esta unidad vas a aprender a añadir clases*

// cargar un escenario que abra una nueva base de datos

SCENARY *bd_crs.NuevaBaseDeDatos*

SAMPLE *NuevaClase* grupo *"3-B"* alumnos *"30"*

TUTORING *NuevaClase* grupo *"3-B"* alumnos *"30"*

TUTORING *NuevaClase* grupo "*4-A*"

TUTORING *NuevaClase* alumnos *"55"*

TUTORING *NuevaClase*

// cerrar la base de datos descartando los cambios realizados

SCENARY *bd_crs.CerrarSinGuardar*

ENDUNIT

// Unidad pedagógica para añadir un nuevo tipo de local

UNIT *AñadirTiposDeAula*

// texto descriptivo de la unidad pedagógica

UNITDESC *En esta unidad vas a aprender a àñadir un nuevo tipo de aula*

// cargar un escenario que abra una nueva base de datos

SCENARY *bd_crs.NuevaBaseDeDatos*

SAMPLE *AñadirNuevoTipo* nombre *"Laboratorio"* utilidad *"Laboratorio de química"*

TUTORING *AñadirNuevoTipo* nombre *"Laboratorio"* utilidad *"Laboratorio de química"*

TUTORING *AñadirNuevoTipo* nombre "Gimnasio"

TUTORING *AñadirNuevoTipo*

// cerrar la base de datos descartando los cambios realizados

SCENARY *bd_crs.CerrarSinGuardar*

ENDUNIT

// Unidad pedagógica para repaso general

UNIT *RepasoGeneral*

// Lecciones previas que se deberán enseñar con anterioridad

PREVIOUS AñadirAsignaciones AñadirAulas

// texto descriptivo de la unidad pedagógica

UNITDESC En esta unidad ud. podrá practicar todo lo aprendido hasta ahora

// cargar un escenario que abra una nueva base de datos

SCENARY *bd_crs.NuevaBaseDeDatos*

TUTORING *NuevoProfesor* nombre *"Juan Cano"* DNI *"11111111"* dirección *"Jorge Manrique, 14"*

TUTORING *AñadirNuevoTipo* nombre *"Laboratorio"* utilidad *"Laboratorio de química"*

TUTORING *NuevoAula* ubicación *"Edificio nuevo"* capacidad *"80"* nombre *"Vivaldi"* tipo *"Laboratorio de química"*

TUTORING *NuevaAsignatura* nombre *"Geología"* horas *"5"*

TUTORING *NuevaClase* grupo *"3-B"* alumnos *"30"*

TUTORING *CrearNuevaAsignación* profesor *"Juan Cano"* asignatura *"Geología"* grupo *"3-B"*

// cerrar la base de datos descartando los cambios realizados

SCENARY *bd_crs.CerrarSinGuardar*

ENDUNIT

// instrucciones que se ejecutan después de haberse realizado todas las unidades

MSG "Enhorabuena, curso finalizado!"

ENDCOURSE

Apéndice VIII: Interface para la Simulación del Sistema Solar Interno.

En este apándice se muestra un ejemplo de cómo, a partir de una especificación en el lenguaje OOCSMP de un modelo de simulación digital continua, un compilador genera una interfaz interactiva y el modelo de tareas asociado a la misma. De este modo pueden explotarse todas las ventajas explicadas en esa tesis a las interfaces generadas con OOCSMP.

De manera resumida, el mecanismo de generación de estas interfaces de simulación se encuentra plasmado en la Figura 63. Como puede verse, un compilador recibe como entrada un programa OOCSMP que representa el sistema a simular y una serie de librería predefinidas, a partir de los cuales genera un código C++ capaz de simular el sistema; la interfaz de usuario del mismo, y el modelo de tareas de dicha interfaz. A partir de dicho modelo de tareas, como hemos visto en esta tesis, actúa ATOM para dar soporte a herramientas como CACTUS.

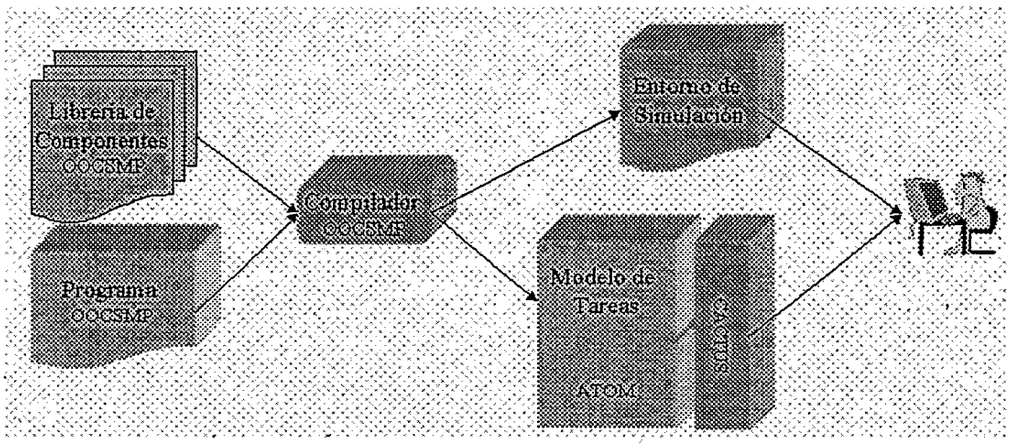

Figura 63: Funcionamiento de OOCSMP.

A continuación se muestra el código OOCSMP de un programa que representa el comportamiento del sistema solar interno (Mercurio, Venus, la Tierra y Marte), y que está compuesto de dos módulos. El primero de ellos indica cuáles son las componentes (objetos) del modelo, mientras que el segundo representa las interacciones del sistema (en este caso, las atracciones gravitatorias entre los distintos elementos).

Planet.csm:

```
* Definición de la clase Planeta
CLASS Planet {
        NAME Planet
        DATA M, X0, Y0, XP0, YP0, FI
        INITIAL
          FIR:=FI*PI/180
          CFI:=COS(FIR)
```

```
SFI:=SIN(FIR)

* Cáculos para este Planeta
DYNAMIC
 * Distancia al Sol
 R2 := X*X+Y*Y
 R :=  SQRT(R2)
 Y1 :=  Y*CFI
 Z := Y*SFI
 * Influencias mutuas
 * El Sol sobre este Planeta
 APS := G*MS/R2/R
 * Este Planeta sobre el Sol
 ASP := G*M/R2/ R
 XPP := -(ASP+APS)*X
 YPP := -(ASP+APS)*Y
 XP := INTGRL(XP0,XPP)
 YP:= INTGRL(YP0,YPP)
 X:= INTGRL(X0,XP)
 Y := INTGRL(Y0,YP)

* Interacciones mutuas entre planetas
ACTION Planet P
 * Distancia a otro planeta
 DPP2 := (P. X -X)*( P. X - X)+( P. Y - Y)*( P. Y - Y)+( P. Z - Z)*( P. Z -Z)
 DPP := SQRT(DPP2)
 * Influencias
 * El otro planeta sobre el sol
 ASP1 := G*P.M/P.R2/P.R
 * El otro planeta sobre éste
 APP1 := G*P.M/DPP2/DPP
 * Conversión de coordenadas
 Y2 := P.Y*COS(P.FIR-FIR)
 * Efecto final sobre el planeta
 XPP += APP1*(P.X-X) - ASP1*P.X
```

```
YPP += APP1*(Y2-Y) - ASP1*Y2

* Otros parámetros
PRINT R
PLOT Y,X
FINISH R=.0001
}
```

Gravitación.csm:

```
TITLE Gravitación

* Datos del Universo
DATA G:=0.00011869, PI:=3.141592653589793

* Datos del sol
DATA MS:=332999
INCLUDE "Planet.csm"

* Declaración de los planetas
Planet Mercury("Mercu",0.055271,-0.3871, 0,2.078, -9.892, 7.004)
Planet Venus ("Venus",0.81476,  0.7233, 0, 0.051,  7.39,  3.394)
Planet Earth ("Earth",1, 0, 1, -6.2899, 0.107, 0)
Planet Moon  ("Moon", 0.01235, 0, 0.9975,-6.0783, 0.107, 0)
Planet Mars  ("Mars", 0.10734, 1.5233, 0, 0.476, 5.071, 1.85)
Planet Jupiter("Jupit",317.94, 0, -5.2028, 2.754,  0.131, 1.308)
Planet InnerSys :=Mercury,Venus,Earth,Moon,Mars,Venus, Jupiter
InnerSys.STEP()
InnerSys.ACTION(InnerSys)

* Intervalos temporales y otros parámetros
TIMER delta:=.001, FINTIM:=2, PRdelta:=.1, PLdelta:=.01
* Método a utilizar para la simulación
METHOD RKSFX
```

A partir de las especificaciones anteriores, un compilador genera el código encargado de realizar la simulación, el código de la interfaz de usuario, y el modelo de tareas de dicha interfaz. A continuación se muestra el modelo de tareas que se genera en el caso de compilar el sistema del ejemplo anterior*:

```
// modelización de las tareas atómicas
TASK "Presionar botón de Stop" ISA BUTTON "Stop" ALIAS StopButton
DESCRIPTION Presiona el botón que detiene la simulación
END

TASK "Presionar botón de Reset" ISA BUTTON "Reset" ALIAS ResetButton
DESCRIPTION Presiona el botón que restablece los valores iniciales a los parámetros
END

TASK "Presionar botón de Resume" ISA BUTTON "Resume" ALIAS ResumeButton
DESCRIPTION Presiona el botón que permite continuar la simulación
END

TASK "Presionar el botón Data" ISA BUTTON "Data" ALIAS DataButton
DESCRIPTION Abre la ventana de datos
END

TASK "Presionar botón de Close" ISA BUTTON "Close" ALIAS CloseButton
PATTERN OWNER.TITLE == "Print variables"
DESCRIPTION Cierra la ventana de datos
END

TASK "Presionar el botón de propiedades para Marte" ISA BUTTON "Mars" ALIAS
        MarsButton
DESCRIPTION Comienza la edición de valores para las propiedades de Marte
END
```

* Por brevedad, solo se muestran las tareas asociadas al Planeta 'Marte', existiendo tareas análogas que corresponden al resto de planetas del sistema solar interno: Mercurio, Venus y La Tierra..

TASK "Presionar botón de confirmación de valores para Marte" **ISA BUTTON** "Ok" **ALIAS** OkMarsButton

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Confirma los valores seleccionados para Marte

END

TASK "Descartar la selección de valores para Marte" **ISA BUTTON** "Cancel" **ALIAS** CancelMarsButton

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Descarta valores seleccionados para Marte

END

TASK "Establecer el valor de la propiedad XP0 de Marte" **ISA TEXT** "XP0" "XP0" **ALIAS** SetXP0Mars

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Establece el valor de XP0 para Marte

END

TASK "Establecer el valor de la propiedad YP0 de Marte" **ISA TEXT** "YP0" "YP0" **ALIAS** SetYP0Mars

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Establece el valor de YP0 para Marte

END

TASK "Establecer el valor de la propiedad FI de Marte" **ISA TEXT** "FI" "FI" **ALIAS** SetFIMars

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Establece el valor de FI para Marte

END

TASK "Establecer el valor de la propiedad X0 de Marte" **ISA TEXT** "X0" "X0" **ALIAS** SetX0Mars

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Establece el valor de X0 para Marte

END

TASK "Establecer el valor de la propiedad Y0 de Marte" **ISA TEXT** "Y0" "Y0" **ALIAS** SetY0Mars

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Establece el valor de Y0 para Marte

END

TASK "Establecer el valor de la propiedad M de Marte" **ISA TEXT** "M" "M" **ALIAS** SetMMars

PATTERN OWNER.TITLE == "Mars"

DESCRIPTION Establece el valor de M para Marte

END

TASK "Establecer valores de propiedades para Marte" **ISA COMPOSED ALIAS** SetMarsValues

DESCRIPTION Establece valores de propiedades para Marte

END

RULE SetMarsValuesRule

TASK SetMarsValues

SUBTASKS SetXP0Mars **AS** XP0

SetYP0Mars **AS** YP0

SetFIMars **AS** FI

SetX0Mars **AS** X0

SetY0Mars **AS** Y0

SetMMars **AS** M

OkMarsButton **AS** OkPart

CANCELING CancelMarsButton

MULTIPLE XP0 YP0 FI X0 Y0 M

OPTIONAL XP0 YP0 FI X0 Y0 M

SEQUENCING AND

PARAMETER XP0 = XP0.Params.XP0

PARAMETER YP0 = YP0.Params.YP0

PARAMETER FI = FI.Params.FI

PARAMETER X0 = X0.Params.X0

PARAMETER Y0 = Y0.Params.Y0

PARAMETER M = M.Params.M

END

TASK "Modificar valores de las propiedades de Marte" **ISA COMPOSED ALIAS** ChangeMarsSettings

DESCRIPTION Modifica valores de las propiedades de Marte

END

RULE ChangeMarsSettingsRule

TASK ChangeMarsSettings

SUBTASKS MarsButton **AS** Button

SetMarsValues **AS** Values

CANCELING CancelMarsButton

SEQUENCING SEQ

PARAMETER XP0 = XP0.Params.XP0

PARAMETER YP0 = YP0.Params.YP0

PARAMETER FI = FI.Params.FI

PARAMETER X0 = X0.Params.X0

PARAMETER Y0 = Y0.Params.Y0

PARAMETER M = M.Params.M

END

TASK "Comenzar el establecimiento de valores de parámetros globales" **ISA BUTTON** "Globals" **ALIAS** GlobalsButton

DESCRIPTION Comienza el establecimiento de valores de parámetros globales

END

TASK "Confirmar valores para parámetros globales" **ISA BUTTON** "Ok" **ALIAS** OkGlobalsButton

PATTERN OWNER.TITLE == "Globals"

DESCRIPTION Confirma valores para parámetros globales

END

TASK "Descartar valores establecidos para parámetros globales" **ISA BUTTON** "Cancel" **ALIAS** CancelGlobalsButton

PATTERN OWNER.TITLE == "Globals"

DESCRIPTION Descarta valores establecidos para parámetros globales

END

```
TASK "Establecer el valor de PI" ISA TEXT "PI" "PI" ALIAS SetPI
PATTERN OWNER.TITLE == "Globals"
DESCRIPTION Establece el valor de PI
END

TASK "Establecer el valor de MS" ISA TEXT "MS" "MS" ALIAS SetMS
PATTERN OWNER.TITLE == "Globals"
DESCRIPTION Establece el valor de MS
END

TASK "Establecer el valor de G" ISA TEXT "G" "G" ALIAS SetG
PATTERN OWNER.TITLE == "Globals"
DESCRIPTION Establece el valor de G
END

TASK "Establecer el valor de Delta" ISA TEXT "delta" "delta" ALIAS Setdelta
PATTERN OWNER.TITLE == "Globals"
DESCRIPTION Establece el valor de Delta
END

COMPOSED SetGlobalValues
DESCRIPTION Sets global values
END

RULE SetGlobalValuesRule
TASK SetGlobalValues
SUBTASKS SetPI AS PI
        SetMS AS MS
        SetG AS G
        Setdelta AS delta
        OkGlobalsButton AS OkPart
CANCELING CancelGlobalsButton
MULTIPLE PI MS G delta
OPTIONAL PI MS G delta
```

SEQUENCING AND

PARAMETER PI = PI.Params.PI

PARAMETER MS = MS.Params.MS

PARAMETER G = G.Params. G

PARAMETER delta = delta.Params.delta

END

TASK "Permitir modificar los valores de los parámetros globales" **ISA COMPOSED ALIAS** ChangeGlobalSettings

DESCRIPTION Permite modificar los valores de los parámetros globales

END

RULE ChangeGlobalSettingsRule

TASK ChangeGlobalSettings

SUBTASKS GlobalsButton **AS** Button

SetGlobalValues **AS** Values

CANCELING CancelGlobalsButton

SEQUENCING SEQ

PARAMETER PI = Values.Params. PI

PARAMETER MS = Values.Params. MS

PARAMETER G = Values.Params.G

PARAMETER delta = Values.Params.delta

END

TASK "Establecer el tiempo actual para la simulación" **ISA TEXT** "Current Time:" "ctime" **ALIAS** SetCTime

PATTERN OWNER.TITLE == " Gravitación"

DESCRIPTION Establece el tiempo actual para la simulación

END

TASK "Establecer el tiempo final para la simulación" **ISA TEXT** "Final Time:" "ftime" **ALIAS** SetFTime

PATTERN OWNER.TITLE == " Gravitación "

DESCRIPTION Establece el tiempo final para la simulación

END

TASK "Habilitar/deshabilitar barra de scroll horizontal" **ISA SELECTION ALIAS** AtomicScrollXButton

PATTERN LABEL == "Scroll X"

DESCRIPTION Habilita/deshabilita barra de scroll horizontal

POSTPARAMETER estado = SELECTED

END

TASK "Habilitar/deshabilitar barra de scroll vertical" **ISA SELECTION ALIAS** AtomicScrollYButton

PATTERN LABEL == "Scroll Y"

DESCRIPTION Habilita/deshabilita barra de scroll vertical

POSTPARAMETER estado = SELECTED

END

Apéndice IX: TRAC, Gestión de Macros

Las interfaces generadas sobre ATOM ofrecen soporte para el gestor de macros TRAC (del inglés, *Task Repeating And Completing, Repetición Y Finalización de Tareas*). TRAC es un sistema que permite automatizar secuencias repetitivas de tareas, actividad muy común en aplicaciones muy ligadas al trabajo con bases de datos, como *Schoodule,* o en entornos de simulación, como los generados a partir de OOCSMP, en donde los usuarios con frecuencia formulan sus hipótesis a base de repetir experiencias del tipo '*what if...*' y contrastar los resultados con experiencias similares realizadas previamente.

Una ventaja importante es el hecho de que TRAC razona sobre tareas de alto nivel, lo que proporciona una gran independencia del procedimiento seguido por el usuario para llevar a cabo sus tareas, pues es capaz de determinar que el usuario está realizando tareas idénticas siguiendo distintos procedimientos, es decir, mediante diferentes caminos alternativos.

TRAC permite automatizar procesos basándose en inferencia a dos niveles: inferencia de tareas repetitivas e inferencia de nuevos valores de contexto para las tareas a automatizar. El primer nivel se encarga de la detección de patrones de tareas repetidos a lo largo del historial de tareas realizadas recientemente por el usuario. Una vez este primer nivel infiere cuáles deberían ser las siguientes acciones a realizar por el usuario, se lleva a cabo el segundo nivel de inferencia, esta vez para los contextos sobre los que el usuario ha llevado a cabo las tareas a partir de las cuales se infirieron las futuras.

Cuando ambos niveles de inferencia tienen éxito, en el sentido de que infieren unívocamente las siguientes acciones y el contexto de las mismas, entonces el sistema se ofrece al usuario a realizar automáticamente las acciones inferidas. Por supuesto, el usuario puede no aceptar dicho ofrecimiento o modificar interactivamente los valores del contexto determinados por el sistema, aportando los que él desee.

La referencia más cercana TRAC lo constituye *Repeat and Predict* [Masui94], un potente sistema para la automatización de acciones del usuario para el popular editor de texto EMACS. TRAC generaliza algunos de los conceptos de *Repeat and Predict* para aplicarlos a entornos no específicos y basados en interfaz de usuario gráfica, donde el contexo de las acciones cobra mayor relevancia. Y, más importante aún, TRAC actúa sobre tareas de alto nivel, mientras que *Repeat and Predict* lo hace en base a un flujo plano de eventos de bajo nivel, es decir, pulsaciones de teclas.

A diferencia de otros sistemas de automatización de procesos, TRAC no garantiza el acierto en sus predicciones, como ningún sistema basado en la inferencia, pero a cambio es más sencillo de manejar que aquellos. Esta gran sencillez se debe a que los usuarios no tienen que grabar previamente las macros y después ejecutarlas, ya que simplemente se limitan a aceptar o no las sugerencias que el sistema ofrece. En nuestro caso, el papel de la *grabación de las macros* lo juega, por un lado, el diseño del modelo de tareas de la aplicación y, por otro, la aplicación de las técnicas inductivas que a continuación se describen.

Inferencia de tareas a automatizar.

Lo primero que hace TRAC cuando se le solicita la automatización de una secuencia de tareas es comprobar en qué estado se encuentran las tareas que el usuario está realizando. La interfaz de

TRAC tan sólo tiene un botón, *Automatizar*, cuya semántica varía dependiendo del estado actual de las tareas del usuario. Así, si el usuario tiene actualmente alguna tarea en ejecución, es decir, si hay alguna tarea en las *Tareas Activas*, lo natural es que la petición que se le está realizando al sistema sea una solicitud de *finalización* de dicha(s) tarea(s). En caso contrario, lo natural es que el usuario haya realizado ya alguna secuencia de tareas y lo que quiera sea que el sistema repita la secuencia, generalizada para nuevos valores de los parámetros de las tareas de la secuencia.

Una vez TRAC determina cuál es la semántica de la petición de *automatización*, el sistema tiene que determinar qué tareas deben ejecutarse para automatizar lo que, en otro caso, el usuario debería hacer a mano.

- En el primer caso, en el que el usuario desea completar lo que actualmente se encuentra haciendo, TRAC simplemente consulta qué tareas están a medio realizar y trata de inferir, a partir de la secuencia de tareas semejantes inmediatamente anteriores del *Histórico*, los valores de los parámetros que deberán tener las tareas que se quieren completar. En ocasiones, como vimos en 3.3.1, ATOMS no puede determinar de manera unívoca cuál es la intención del usuario, qué tareas está realizando, de manera que TRAC se dirige al usuario para determinar con exactitud qué es lo que se está haciendo, es decir, cuál de los estados posibles del análisis es el correcto.
- Por su parte, si el usuario no tiene ninguna tarea a medio realizar, el proceso es más complejo, y TRAC aplica reglas de inferencia simples para determinar qué secuencia de tareas desea repetir el usuario.

  Las reglas para la determinación de qué tareas se desean repetir están inspiradas en las utilizadas en el sistema *Repeat and Predict*. En ese trabajo, existen dos reglas de inferencia, la segunda de las cuales no se aplica a menos que la primera no pueda aplicarse.

  - Regla #1. Si lo último que se ha realizado han sido dos secuencias consecutivas de operaciones, es decir, lo último ejecutado ha sido algo del tipo XX, entonces se considera dicha secuencia X de operaciones como la operación a automatizar, tomándose dicha secuencia lo mayor posible en cuanto a su longitud. Por ejemplo, si el usuario tecleara 'abccabcc', se escogería como X la secuencia 'abcc' y no 'c'. Esta primera regla obliga al sistema a repetir una secuencia que el usuario ya ha repetido anteriormente en una ocasión.
  - Regla #2. Si se encontrara un patrón XYX antes de pedir la repetición, donde X e Y son secuencias no vacías de operaciones, entonces la secuencia de acciones Y habrá de ser repetida, donde X se escoge lo más larga posible e Y lo menor posible para un tamaño de X dado. Por ejemplo, dada la secuencia 'abracadabra', se tomará 'abra' como X y 'cad' como Y, no 'a' como X y 'br' como Y. La manera de escoger los tamaños de las subcadenas es tal que se minimiza la probabilidad de error en la predicción, eligiéndose primero aquellas X que faciliten la máxima información. Esta segunda regla completa secuencias en las que una cierta parte de la secuencia ya ha sido repetida por el usuario, pero otra parte aún no.

  En el caso de TRAC, a diferencia del caso de *Repeat and Predict*, cada letra mayúscula representa una cierta sucesión de tareas compuestas, y no una cierta secuencia de eventos básicos de teclado. En el caso de TRAC, si consideramos una tarea T como la secuencia de las

tareas X e Y, la Regla #2 se convierte en el caso de la automatización para la *finalización* de tareas ya descrito, por lo que en nuestro caso solo se aplica la siguiente regla:

- Regla #1: Si existe un cierto patrón XZX, deberá repetirse la secuencia de tareas X. En esta regla se busca la aplicación para valores de la longitud de Z cuanto menores mejor, incluyendo la secuencia vacía, y longitudes de X cuanto mayores mejor, sin incluir la secuencia nula. Esta regla es la extensión natural de la primera regla de *Repeat and Predict*, donde la secuencia de tareas Z representa tareas *espúreas* del usuario, es decir, tareas aisladas que no tienen relación con las tareas representadas por la secuencia X. TRAC verifica tras cada interacción del usuario la aplicabilidad de esta regla, de modo que, cuando es aplicable, el propio sistema se ofrece para realizar la secuencia de tareas X en lugar del usuario. Esta técnica, denominada *anticipación*, fue ofrecida por primera vez por el sistema EAGER [Cypher91].

TRAC va aplicando la regla anterior y preguntando al usuario si la predicción que se ha hecho para la secuencia de tareas a ejecutar es o no correcta. En caso de que no sea lo deseado por el usuario, TRAC prosigue con la aplicación de la misma hasta dar con otra posibilidad.

Inferencia de contexto para las tareas a automatizar.

Una vez TRAC conoce qué tareas ha de ejecutar, a continuación tiene que realizarlas. Para ello, TRAC utiliza el módulo *Emulación* (Sección 3.4). A este módulo, además de qué tareas se desean ejecutar, se le deben proporcionar los valores del contexto sobre el que se van a realizar dichas tareas. Por tanto, y a diferencia del sistema *Repeat and Predict*, TRAC tiene que inferir los valores de los parámetros que no quedan determinados por las restricciones del modelo.

Para la inferencia de los valores de un determinado parámetro, los pasos que TRAC sigue son dependientes del tipo del valor del parámetro. TRAC toma como valores de partida para la inferencia los que han adoptado los parámetros durante las iteraciones previas a la que se va a automatizar.

- Para valores numéricos, TRAC verifica si a lo largo de las iteraciones, éstos han seguido la pauta de una progresión aritmética o geométrica, en cuyo caso obtiene el siguiente valor como el siguiente valor de la serie.
- Para cadenas textuales, TRAC verifica si existe algún patrón definido entre las cadenas. Esto es, verifica si las cadenas contienen alguna subcadena cuya evolución se rija por alguna progresión aritmética o geométrica.
- Cuando los parámetros representan objetos gráficos, el proceso es mucho más complejo y en muchos casos quedará sin solución, por lo que será *Emulación* quien posteriormente permita al usuario especificar interactivamente el contexto.

  Sin embargo, gracias a la jerarquía parte-dueño que forman los objetos gráficos en las aplicaciones AMULET, en casos específicos sí es posible hacer una inferencia acertada sobre los parámetros gráficos. Por ejemplo, cuando en iteraciones anteriores siempre se han utilizado objetos gráficos *hermanos* (*partes* del mismo *dueño*), lo natural es que las tareas a *automatizar* se repitan sobre alguno de ellos (si existe un patrón definido que afecte de manera cíclica a un conjunto de *hermanos*), o sobre alguno del resto de sus *hermanos*. En este último caso, para determinar cuál de los *hermanos* será el contexto, TRAC se basa en la búsqueda de relaciones gráficas entre los objetos gráficos de las iteraciones previas para

determinar cuál puede ser el siguiente: relaciones espaciales (posición, área), forma (en base a la jerarquía de herencia) y color.

Una vez determinados los valores de los parámetros de las tareas a realizar, TRAC utiliza *Emulación* para realizar las tareas mediante animaciones. Emulation, además, ofrece una interaz mediante la cual los usuarios pueden proporcionar interactivamente los valores de los parámetros para los que TRAC no ha concretado un valor.

Reunido el tribunal que suscribe en el día de la fecha, acordó calificar la presente Tesis doctoral con SOBRESALIENTE CUM LAUDE

Madrid, 25-1-02

El Presidente

El Secretario

Fdo. Miguel Jca Negras

Vocal

Vocal