TESIS DOCTORAL

Generación de cursos interactivos y análisis colaborativo del proceso de aprendizaje

Autor:

Miguel Ángel Mora Rincón

Directores:

Roberto Moriyón Salomon

Francisco Saiz López

Universidad Autónoma de Madrid

Escuela Politécnica Superior

Departamento de Ingeniería Informática

Octubre 2006

Agradecimientos

En primer lugar quiero agradecer a Mª Carmen, por todo el amor, cariño y ánimos prestados durante todo este tiempo. También por ser el mejor modelo de constancia y trabajo que uno puede seguir, y que ha permitido en gran medida que esta tesis llegue a buen puerto.

Por supuesto, gracias a mis tutores, Roberto y Paco, por todos sus consejos, su ayuda, por la paciencia sin límite que han demostrado y por crear el mejor ambiente de trabajo que podía imaginar.

Gracias también a mi familia, amigos y compañeros, por todo su apoyo y comprensión. Muchas gracias también por esa pizca de presión ejercida en los momentos que más la necesitaba.

Muchas gracias a todos, sin vosotros esto no sería posible.

Índice de contenido

Índice de Figuras

Índice de tablas

Capítulo 1. Introducción

Uno de los problemas principales a los que se enfrentan los sistemas actuales para la generación de cursos interactivos tiene lugar a la hora de dar soporte a la interacción síncrona de más de un usuario, alumno o profesor, con el material didáctico elaborado.

El soporte del trabajo en grupo es muy importante ya que la interacción entre los individuos genera actividades extra, como las explicaciones, discusiones, etc., que permiten que se dé un mayor número de mecanismos cognitivos que ayudan al alumno en la comprensión del problema y por lo tanto mejoran la calidad del aprendizaje, [Dillenbourg 1999]. Estas ventajas de la colaboración ya se conocen desde antes del uso de las aplicaciones informáticas educativas en el aula, [Vygotsky 1993], [Piaget 1932], y el estudio de los mecanismos para fomentarla son el origen de numerosos estudios desde hace varias décadas, [Doise 1984], [Light 1985], [Howe 1991].

Sin embargo existen muy pocas aplicaciones educativas que permitan directamente el trabajo colaborativo síncrono de los estudiantes, menos aún las que facilitan el trabajo a distancia de forma síncrona. Esto es debido a que a la complejidad de crear una aplicación educativa interactiva se añade la complejidad de convertirla en una aplicación colaborativa.

En el entorno informático, el concepto de trabajo colaborativo mediante ordenador, CSCW en sus siglas en inglés, se ha convertido en un marco eficaz para modelar aquellas tareas que llevan a cabo grupos de personas que trabajan con un objetivo común, [Greenberg 1999]. Esta situación se ha visto favorecida tanto por las actuales tendencias organizativas, que cada vez se centran más en el trabajo en equipo, como por la mayor disponibilidad de recursos tecnológicos en el área de las redes de comunicación.

Como parte del desarrollo del CSCW, en los últimos años se han llevado a cabo proyectos de investigación de gran relevancia dedicados al estudio de técnicas colaborativas que permitan mejorar el proceso de aprendizaje [Lipponen 1997][Kozma 1999]. Esto ha dado lugar a una nueva área de investigación llamada CSCL (del inglés “Computer Supported Collaborative Learning”). Estas aplicaciones permiten la interacción simultanea de más de un alumno, entre ellos o con el profesor. Sin embargo, los sistemas actuales

de soporte al aprendizaje colaborativo presentan serias limitaciones en cuanto al papel que puede desempeñar el tutor.

La principal limitación radica en la necesidad de atender a un gran número de estudiantes simultáneamente, y con el menor tiempo de respuesta posible. Para ello se hace necesaria la creación de un sistema que ayude al profesor a guiar a los estudiantes que necesitan más ayuda en cada momento, y poder revisar posteriormente qué han estado haciendo durante la interacción con la aplicación educativa.

1.1 Objetivos y resultados principales

El objetivo principal de esta tesis es estudiar herramientas que faciliten la creación de material didáctico que permita el trabajo colaborativo y facilite que el profesor pueda guiar y supervisar el trabajo de los estudiantes.

El objetivo general anterior se ha abordado dividiéndolo en otros tres claramente diferenciados:

- Se ha creado un modelo abstracto, llamado aprendizaje colaborativo guiado, de las características que han de poseer las aplicaciones colaborativas para permitir el análisis y revisión colaborativos del trabajo realizado, incluyendo la elaboración de propuestas para su mejora.
- Con este modelo se ha diseñado una arquitectura que permite el desarrollo de aplicaciones que soporten el trabajo colaborativo entre varias personas, permitiendo además el análisis de la historia producida durante las sesiones de colaboración. El diseño anterior se ha validado mediante la implementación de un Framework en Java, FACT, que permite a los distintos grupos de usuarios analizar o revisar el trabajo de otros grupos, tanto simultáneamente, es decir, de forma síncrona, como de forma asíncrona. Así pues, se puede conseguir que el tutor pueda revisar el trabajo realizado por los alumnos, ayudarles a completarlo, a la vez que permite a grupos de alumnos aprender colaborativamente.
- Se ha definido un tipo básico de aplicaciones para el aprendizaje colaborativo guiado, suficientemente genéricas como para poder cubrir

el aprendizaje de materias arbitrarias y con características que hacen especialmente simple su desarrollo. Se ha estudiado la factibilidad de diseñar y especificar interactivamente aplicaciones arbitrarias de este tipo. La tesis formula una propuesta de requisitos para el desarrollo de una herramienta que permita el diseño interactivo de este tipo de aplicaciones. La propuesta se ha validado mediante el desarrollo de dos herramientas de autor. Una para la creación de aplicaciones colaborativas mediante demostración, sin necesidad de programar, DFACT, y otra, basada en la anterior, especializada en la creación de material sobre Matemáticas simbólicas con un alto nivel de interacción, ConsMath. Está última herramienta permite al diseñador, mediante programación por demostración, crear cursos o aplicaciones flexibles que permitan aprender unos conceptos teóricos y practicar con ejercicios generados automáticamente, los cuales pueden ser adaptados por los alumnos. Los cursos se pueden crear de forma que evalúen los conocimientos adquiridos y ayuden al alumno a resolver ejercicios, en un espacio de problemas previamente acotado por el profesor durante el diseño del curso.

1.2 Descripción de las aportaciones

FACT es un framework para el desarrollo de aplicaciones que soporten el aprendizaje colaborativo guiado. Este framework requiere realizar unas mínimas modificaciones en las aplicaciones para poder adaptarlas. El soporte colaborativo permite a los educadores o profesores descubrir el grado de adquisición de conocimiento de los alumnos, a la vez que permite ayudarles durante el proceso de aprendizaje, trabajando síncrona o asíncronamente según sea necesario. Una de las características principales de las aplicaciones de FACT es que los usuarios pueden pasar del trabajo colaborativo síncrono al análisis de dicho trabajo sin necesidad de realizar ninguna acción especial ni cambiar de aplicación.

En comparación con otras propuestas para la construcción de aplicaciones colaborativas, como Habanero, [Jackson 1999], y Disciple, [Li 1999], FACT requiere más trabajo para adaptar la aplicación de partida, pues exige la utilización de un protocolo específico para el tratamiento de los eventos del

usuario, pero a cambio es el primer sistema con el que las aplicaciones construidas permiten la revisión constructiva de historias de trabajo y colaboración, dando con ello soporte al aprendizaje colaborativo guiado.

En cuanto a las herramientas de autor, DFACT se basa en FACT para el soporte colaborativo, y lo amplia para permitir el diseño por demostración de las aplicaciones. A su vez ConsMath integra DFACT, especializando esta herramienta en la creación de aplicaciones educativas sobre matemática simbólica. Con esta colección de herramientas se pueden crear aplicaciones que ofrezcan ayuda a los alumnos de forma automática cuando los problemas a resolver son sencillos, o de forma guiada por el profesor cuando los problemas o conceptos a aprender son más complejos.

Son muy pocas las herramientas que se pueden relacionar por sus características con DFACT y ConsMath. Concretamente, CTAT, [Koedinger 2004], y MathEdu, [Díez 2003], permiten la creación mediante demostración de tutores inteligentes que ayudan a los estudiantes a resolver interactivamente problemas de Matemáticas y, en el caso de CTAT, también de otras materias. CTAT es una herramienta desarrollada en la universidad de Carnegie Mellon, para simplificar la creación de los tutores inteligentes. Esta herramienta está basada en las experiencias con el uso y desarrollo de PAT, un tutor inteligente distribuido por la empresa Carnegie Learning, y que se utiliza actualmente en más de 1400 colegios de EE.UU.

En comparación con DFACT y ConsMath, CTAT ofrece un mecanismo de diseño de aplicaciones similar, pero limitado a especificar el comportamiento de diálogos interactivos basados en formularios, mientras que DFACT permite diseñar toda la aplicación mediante demostración. Además, en CTAT es necesario especificar los casos concretos de uso y no permite generalizar la aplicación, mientras que en ConsMath se pueden diseñar por demostración plantillas de interacción reutilizables con sólo variar las condiciones iniciales. Por ejemplo, esto se consigue generando enunciados de problemas aleatoriamente o definiendo llamadas a la resolución de un subproblema o de un ejemplo sencillo dentro de la resolución de otro problema. Además, tanto DFACT como ConsMath son las primeras herramientas de este tipo que permiten la construcción de cursos interactivo que soportan el aprendizaje colaborativo guiado.

La comparación de ConsMath con MathEdu ofrece resultados semejantes: por una parte, MathEdu tiene una capacidad de tratamiento simbólico de las fórmulas matemáticas similar al de ConsMath, pero no incorpora la posibilidad de transmitir este comportamiento a nivel de otros tipos de objetos, como por ejemplo los gráficos. Además, MathEdu no está concebido para facilitar el aprendizaje colaborativo.

1.3 Estructura de la memoria de tesis

En el resto de la memoria veremos en detalle cómo se ha conseguido el objetivo principal planteado, siguiendo la división en subobjetivos que hemos visto. Para ello comenzaremos en el capítulo 2 con una revisión del estado del arte en las áreas relacionadas con los distintos subobjetivos. Estas áreas son el soporte teórico en el que se basa el aprendizaje colaborativo, el soporte tecnológico para el trabajo en grupo y el aprendizaje colaborativo, y las técnicas de programación interactiva y por demostración orientadas a la creación de aplicaciones educativas.

En el capítulo 3 se presenta el modelo de aprendizaje colaborativo guiado. El capítulo 4 introduce FACT y en el capítulo 5 estudiaremos las herramientas para la creación de aplicaciones colaborativas por demostración, DFACT y ConsMath. Para terminar, en el capítulo de conclusiones, veremos un resumen de las aportaciones de esta tesis y las principales líneas de trabajo futuro abiertas.

Capítulo 2. Estado del arte

En este capítulo estudiaremos el estado del arte en las áreas de investigación relacionadas con esta tesis. El área principal es el aprendizaje colaborativo mediante ordenador, CSCL. Para estudiar este área es necesario comprender cada una de sus componentes principales, el aprendizaje colaborativo por un lado, y por otro el trabajo colaborativo mediante ordenador. Comenzaremos por las teorías en las que se fundamente el aprendizaje colaborativo, apartado 2.1, después estudiaremos los aspectos tecnológicos del trabajo colaborativo mediante ordenador, apartado 2.2, para concluir viendo los avances de investigación más importantes, desde el punto de vista de esta tesis, tanto en herramientas de desarrollo de aplicaciones como en las propias aplicaciones CSCL, apartado 2.3.

Por último estudiaremos el área de programación interactiva de aplicaciones y tutores inteligentes, apartado 2.4, en la que se basan las herramientas de autor DFACT y ConsMath. Estudiaremos dos aspectos fundamentales que han demostrado complementarse bien, el diseño de la interacción mediante restricciones y la programación por demostración.

2.1 Aprendizaje y aprendizaje colaborativo

En esta sección mostraremos el marco teórico sobre el que se sustenta el aprendizaje colaborativo, comenzaremos por aclarar qué se entiende por aprendizaje colaborativo, para posteriormente exponer las ideas y teorías más importantes en este área.

La definición más extendida de aprendizaje colaborativo es una situación en la cual dos o más personas aprenden, o intentan aprender algo juntos. Esta definición es muy amplia, admitiendo situaciones muy diferentes. Por ejemplo, podría incluirse desde situaciones con dos personas aprendiendo mediante la resolución conjunta de un problema durante unas horas, hasta una comunidad profesional desarrollando una cultura específica a lo largo de varias generaciones [Dillenbourg 1999].

La situación típica de aprendizaje colaborativo se da en un grupo reducido de personas, entre 2 y 5, normalmente con unos niveles similares de conocimientos, cuando éstos colaboran durante unas horas para aprender algo juntos, ya sea resolviendo un problema o ayudándose a comprender un

tema de estudio. Sin embargo, cuando nos referimos a aprendizaje colaborativo mediante ordenador, o “Computer-Supported Collaborative Learning”, CSCL con sus siglas en inglés, podemos encontrarnos con grupos mayores, por ejemplo una clase completa que sigue un curso de varios meses, encontrándonos por tanto con problemas añadidos que será necesario tener en cuenta en este tipo de situaciones.

El proceso general de aprendizaje consiste en la realización de una serie de actividades que fomentan los mecanismos cognitivos, como la inducción, deducción, adquisición de nuevo conocimiento, etc. Estos mecanismos también se dan tanto en el aprendizaje individual como en el aprendizaje colaborativo. La diferencia que encontramos es que la interacción entre los individuos genera otras actividades extra, como las explicaciones, discusiones, etc., que permiten que se dé un mayor número de mecanismos cognitivos.

Estos mecanismos extras que surgen de la interacción entre los individuos se pueden dar también individualmente, como ocurre por ejemplo durante el diálogo egocéntrico, con uno mismo. En cualquier caso, el simple hecho de realizar una actividad entre varias personas no garantiza que se aprenda mejor o más rápido. Una de las tareas más importantes en el aprendizaje colaborativo es estudiar cómo aumentar la probabilidad de crear situaciones que favorezcan en mayor medida el aprendizaje. En general tenemos 4 formas de crear estas situaciones [Dillenbourg 1999]:

- Creando las condiciones iniciales adecuadas, eligiendo cuidadosamente la composición de los grupos, la forma de comunicarse, etc. Se han realizado numerosos estudios para intentar determinar cuáles son las mejores condiciones, y la conclusión es que al depender de tantas variables no hay unos valores ideales para las condiciones iniciales, sino que dichas condiciones han de ser adaptadas a la experiencia concreta. [Dillenbourg 1995]
- Creando un escenario basado en roles que requiera la colaboración. Por ejemplo, para la realización de una práctica para la implementación de la simulación de un ecosistema, en grupos de dos alumnos, a uno de los alumnos se le podría dar acceso a los detalles sobre el ecosistema, y al otro alumno se le daría acceso a la información sobre como construir un

simulador. De esta forma, controlando el acceso a la información que tiene cada miembro del grupo, incentivamos su colaboración.

- Creando reglas de interacción, por ejemplo obligando a que cada individuo dé su opinión, o creando interfaces de usuario semi estructuradas, donde existen unos diálogos tipo que se han de usar [Baker 1996]. El inconveniente de este método es que puede limitar demasiado la interacción, siendo necesario el mantenimiento de una mínima flexibilidad.
- Realizando un seguimiento y regulando la interacción. De esta forma el profesor puede favorecer la interacción dando indicaciones o moderando el grupo, por ejemplo. También se pueden crear mecanismos de autorregulación como dar una retroalimentación sobre el grado de consenso en las decisiones, el número de aportaciones de cada miembro del grupo, etc.

Piaget ya destacó el potencial productivo que había en situaciones donde los individuos poseían puntos de vista conflictivos a la hora de resolver una tarea de forma colaborativa, [Piaget 1932], sobre todo entre niños, ya que no dio mayor importancia a la interacción de los niños con los adultos.

Basándose en esta idea se han llevado a cabo estudios donde se creaban grupos de niños para resolver tareas, en materias que les permitían discutir y tener puntos de vista conflictivos. Estos estudios reflejan que los niños que participaban en las tareas de forma colaborativa, formando parejas o grupos pequeños para resolver problemas, consiguen un progreso individual mayor, aparte de un mayor progreso en la tarea, que aquéllos que trabajaban solos. Doise también demostró que era suficiente con atribuir una visión conflictiva a otro niño, aunque no estuviera presente, para que se obtuvieran progresos, [Perret-Clermont 1980][Doise 1984].

Analizando la interacción en los grupos mediante el uso de grabaciones de vídeo Light y Glachan, [Light 1985], encontraron que aquellos grupos en los que se argumentaba más eran los que más mejoraron.

Howe, Tolmie & Anderson, [Howe 1991], formaron grupos de estudiantes para estudiar la trayectoria de los objetos cuando son lanzados desde aviones. Los grupos se formaron basándose en las respuestas a varios exámenes previos, de forma que crearon grupos de estudiantes cuyas

respuestas previas eran parecidas y grupos con respuestas dispares. Lo que encontraron fue que las parejas que avanzaron más, comparando los resultados previos con los posteriores a la colaboración, fueren aquellas parejas que diferían en sus predicciones y concepciones iniciales. En un estudio posterior, donde el criterio para formar los grupos tenía que ver con los resultados y la estrategia utilizada para resolver un problema previo, llegaron a una conclusión similar [Howe 1992].

Sin embargo otros estudios concluyen que a veces lo más efectivo es la interacción entre niños del mismo nivel [Light 1994]. De estos estudios se desprende que el concepto de conflicto no es suficiente para explicar la mejora en las actividades colaborativas.

El niño construye su idea del mundo interactuando con otros, 'negociando' los significados de las cosas. Por este motivo será necesario un nivel mínimo común que sustente la argumentación y el entendimiento entre las partes y por otro lado niveles distintos de conocimiento en algunas materias fomentarán la negociación y la participación. En general es necesario que los miembros del grupo posean unos niveles mínimos comunes en las diferentes habilidades o conocimientos requeridos para realizar una actividad colaborativa, ya que de lo contrario la actividad se convertiría en un mero reparto de tareas. Por otro lado, el que los miembros posean perfiles o concepciones iniciales diferentes, de los aspectos relativos a la actividad a realizar, fomentará la participación en la tarea colaborativa. La concurrencia de estas dos circunstancias, por un lado unos conocimientos comunes que garanticen el entendimiento entre las partes, y una cierta heterogeneidad del grupo, permitirá que los estudiantes puedan realizar tareas que no podrían realizar por si solos y que interioricen las habilidades del grupo, aprendiendo por tanto nuevas habilidades o ideas.

2.1.1 Relación entre sociedad, cultura y aprendizaje

Como acabamos de ver en el apartado anterior, los mecanismos cognitivos en el aprendizaje tienen una fuerte componente social. Este enfoque social ha sido ampliamente desarrollado por L. S. Vygotsky, [Vygotsky 1978], a comienzos del siglo XX, y por sus discípulos, principalmente A. R. Luria y A. N. Leontiev. Sin embargo no ha sido hasta finales del mismo siglo cuando

empezaron a considerarse sus trabajos, debido en gran parte a la prohibición en Rusia durante años, por razones políticas[1], de cualquier uso de los mismos.

El enfoque de Vygotsky se basa en la idea de que "la naturaleza psicológica humana representa la superposición de las relaciones sociales interiorizadas, que se han transformado en funciones para el individuo y en formas de la estructura individual" [Wertsch 1981]. De esta forma, según Vygotsky, cada función en el desarrollo cultural del niño aparece por duplicado, primero aparece en su relación con otras personas, inter-psicológico, en la sociedad, y en según lugar dentro del niño, intra-psicológico, una vez interiorizada la función. Esto ha dado origen a la llamada "psicología cultural", [Shweder 1993][Cole 1998]. Este enfoque es opuesto al de Piaget, y conlleva que la componente social tenga mucha más importancia que en las ideas constructivistas, como veremos a continuación.

La psicología cultural se centra en el estudio de las actividades humanas superiores, a diferencia de muchas otras perspectivas que parten de temas biológicos. En las teorías englobadas en la psicología cultural la palabra cultura hace referencia al medio para la actividad humana, en general, incluyendo herramientas o artefactos, ritos e instituciones que se han ido desarrollando a lo largo de la historia. Por lo tanto, estas teorías tienen muy presente el contexto socio cultural y la trama de la vida social organizada [Crook 1994].

Una de las teorías más importantes de la psicología cultural es la teoría de la actividad. Esta teoría se basa en la idea de que la forma en la que el ser humano actúa sobre la naturaleza es a través de las herramientas, por lo tanto actuamos de forma mediada por las herramientas. Un tipo muy importante de herramientas son las simbólicas, como el lenguaje o los signos en general. Gracias al lenguaje podemos controlar otros procesos psicológicos y nuestros propios procesos, como el recuerdo o el pensamiento. Estas herramientas, técnicas o psicológicas, se transmiten

[1] Esta prohibición fue debida a la asociación de sus ideas con la paidología, que es una disciplina muy parecida a la psicología de la educación, y que fue prohibida por decreto del Comité Central del Partido Comunista Ruso, además de por un conflicto con las ideas de Stalin [Wertsch 1988]. Por lo tanto no se reanudaron sus publicaciones hasta 1956, tras la muerte de Stalin y 22 años después de la muerte de Vygotsky.

culturalmente, de generación en generación, a través de actividades como la instrucción. Esta idea, que se formalizó en la teoría de la actividad, no fue formulada directamente por Vygotsky, sino que su principal responsable fue uno de sus discípulos, A.N. Leontiev, aunque ellos insisten en que las raíces de la teoría de la actividad se encuentran esencialmente en los propios escritos de Vygotsky, [Wertsch 1988] pág. 208.

Según la teoría de la actividad, "Cultural-Historical Activity Theory", o CHAT en sus siglas en ingles, la estructura del conocimiento humano esta caracterizada por un triángulo con el sujeto y el objeto en la base y el medio o herramientas en la parte superior[2], Figura 1.

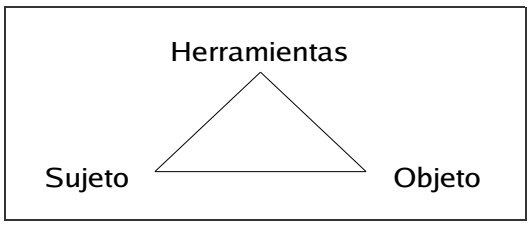

Figura 1: Modelo de Vygostky de la acción mediada

El sujeto es un agente consciente y activo, mientras que el objeto representa el objeto de su conocimiento/acción sobre el que está pensando o actuando. La línea directa entre sujeto y objeto representa las capacidades innatas o instintivas que posee el sujeto, esto representa actividades y aprendizaje no mediado.

Las acciones mediadas se ayudan de herramientas para llevarse a cabo, estas herramientas pueden ser materiales o simbólicas. Las herramientas simbólicas pueden ser el propio lenguaje, conceptos, signos o modelos. Con el lenguaje podemos trabajar con ideas que no hemos recibido directamente, sino que son producto de la experiencia de generaciones pasadas. Cuando el sujeto aprende de otros miembros de su cultura cómo se usan esas herramientas se dice que se apropia de objetos culturales. Esta apropiación se lleva a cabo participando en el uso de estos objetos culturales con otros miembros más expertos de esa cultura, y el aprendizaje resulta de las propias experiencias y prácticas del sujeto con el objeto, de forma tutelada

[2] Podemos encontrar una ampliación del modelo gráfico original, de forma que contemple los conceptos de división del trabajo y actividades en grupo en [Engeström 1987].

por un experto. Por lo tanto no es una mera transmisión de información del experto al sujeto, sino que el sujeto hace suya la herramienta, [Leontjev 1981].

Las actividades mediadas requieren una estructura más compleja de acciones, ya que el sujeto ha de tener en cuenta los tres elementos del triángulo y sus relaciones de forma global para llevarlas a cabo.

Desde el punto de vista de CHAT la apropiación de herramientas simbólicas ha de ocurrir dentro de la zona de desarrollo próximo (ZPD en sus siglas en inglés), [Vygotsky 1978]. La zona de desarrollo próximo se define como la región, en el espacio de desarrollo intelectual del sujeto, que está entre lo que ya conoce, o es capaz de realizar independientemente, y el nivel de desarrollo potencial, determinado por las actividades que puede realizar mediante la colaboración con otros sujetos más adelantados o la tutorización de otras personas. Cuando se está interactuando en la zona de desarrollo próximo se participa en actividades mediadas que no se podrían realizar independientemente, de esta forma el sujeto se está apropiando de nuevos objetos culturales y avanza en su propia capacidad de resolución independiente de problemas.

Por ejemplo, una implicación directa de la ZPD tiene que ver con las pruebas y exámenes que miden el nivel de desarrollo mental del niño. La mayoría de estas pruebas suponen que los problemas que el niño puede resolver por sí solo, indican el nivel de su desarrollo mental en ese momento. Sin embargo estas pruebas sólo pueden medir la parte del desarrollo del niño que se ha completado. Mediante la experimentación Vygotsky demuestra que por ejemplo 2 niños con edad mental de 8 años pueden resolver problemas destinados a edades muy dispares, de 10 y 12 años respectivamente, si se les da una pequeña ayuda, un planteamiento inicial, etc. Experimentalmente se demuestra que un niño con una ZPD más amplia tendrá un mejor rendimiento escolar, siendo de esta forma la ZPD una herramienta más valiosa que el resultado de los exámenes individuales para la dinámica del progreso intelectual [Vygotsky 1993].

2.2 Sistemas colaborativos

En este apartado estudiaremos los distintos aspectos de los sistemas colaborativos asistidos por ordenador, comenzando por su definición y clasificación, para terminar centrándonos en los aspectos de diseño e implementación de estos sistemas. Este estudio nos permitirá comprender mejor cómo son estos sistemas colaborativos y sus requisitos desde un punto de vista más técnico. Además nos servirá de base, junto con el apartado anterior, para estudiar los sistemas de aprendizaje colaborativo asistidos por ordenador.

El Trabajo Colaborativo Asistido por Ordenador, o **CSCW** *(Computer Supported Cooperative Work)*, en sus siglas en inglés, es la actividad coordinada y asistida por ordenador para la resolución de problemas y comunicación llevada a cabo colaborativamente por un grupo de individuos, [Greif 1988]. Llamamos **Groupware** al software multiusuario que soporta los sistemas CSCW. También se usa el término CSCW para referirnos a la disciplina que guía el diseño y desarrollo apropiado del groupware [Greenberg 1991].

Actualmente el término groupware parece estar encaminado a perder su utilidad, ya que es cada vez más alta la proporción de aplicaciones groupware. Algunas predicciones ya apuntaban que actualmente la mayoría del software sería groupware [Baecker 1993] y que los sistemas operativos incorporarían primitivas groupware como un servicio más a las aplicaciones [Greenberg 1997]. Estas predicciones se han cumplido en gran medida: por ejemplo las principales aplicaciones ofimáticas, como Microsoft Office, u OpenOffice, incorporan herramientas de revisión colaborativa, los sistemas operativos, como Microsoft Windows o la mayoría de distribuciones de Linux, incorporan herramientas para compartir aplicaciones o incluso el escritorio, así como herramientas de comunicación en grupo con soporte de indicadores de presencia, mensajería instantánea, etc.

El aumento en el número de aplicaciones groupware se ha logrado en gran medida gracias a la disponibilidad de conexiones de alta velocidad de forma relativamente económica, que ha permitido el uso de aplicaciones groupware a través de Internet de una forma eficiente por un número cada vez mayor de usuarios, y a la aparición de estándares como el protocolo de transmisión en

tiempo real, o RTP[3] en sus siglas en inglés, usado fundamentalmente para transmisión de vídeo o audio, H.323[4] y SIP[5], para gestión de sesiones, mensajes instantáneos o presencia, RDP[6], para compartir aplicaciones, etc. Estos estándares han mejorado notablemente la interoperatividad entre distintas aplicaciones groupware y entre distintos sistemas operativos o plataformas hardware.

Dada la gran cantidad de aplicaciones de trabajo en grupo, o groupware, a continuación procederemos a ver algunas clasificaciones, para posteriormente entrar a estudiar los aspectos de diseño e implementación de estas aplicaciones.

2.2.1 Clasificación de los sistemas colaborativos

Una de las clasificaciones más conocidas es la matriz espacio-tiempo, Tabla 1.

Tiempo \ Espacio	Mismo lugar	Diferente lugar
Síncrono	Interacción cara a cara	Interacción síncrona distribuida
Asíncrono	Interacción asíncrona	Interacción asíncrona distribuida

Tabla 1: Matriz Espacio-Tiempo

En los sistemas síncronos la colaboración ocurre en un espacio de tiempo muy cercano y se suele hablar de colaboración en tiempo real, mientras que en los asíncronos no es necesaria esta cercanía en el tiempo y la

[3] RTP es un protocolo estandarizado de internet, STD-64, definido originalmente como RFC-1889 y actualmente como RFC-3550: http://tools.ietf.org/html/3550

[4] H.323 es una recomendación ITU-T que define los protocolos para sesiones de comunicación de audio y vídeo a través de un red basada en paquetes. Actualmente se utiliza fundamentalmente en aplicaciones de Voz sobre IP (VoIP)

[5] SIP es un protocolo estándar IETF para definir sesiones multimedia que pueden incluir audio, vídeo, mensajes instantáneos o realidad virtual. Se utiliza sobre todo en aplicaciones VoIP

[6] Este protocolo es una extensión del ITU-T T.128 APPLICATION SHARING PROTOCOL

colaboración transcurre en momentos más distantes. En el otro eje, el espacio o lugar, tenemos sistemas en los que la colaboración ocurre en un mismo lugar o entre usuarios que se encuentran en lugares distantes.

Un ejemplo de colaboración síncrona pueden ser los sistemas de soporte para conferencias, como las "electronic meeting rooms" en las que la interacción ocurre en un mismo lugar; o los sistemas para teleconferencias, en donde la interacción ocurre entre lugares que pueden ser distantes.

Como sistemas asíncronos podemos tener los tablones de noticias físicos tradicionales, en los que la interacción ocurre en un mismo lugar, o electrónicos, en los que la interacción puede ser remota. También son destacables dentro de los sistemas asíncronos distribuidos el correo electrónico o los sistemas de gestión del flujo de trabajo, o *workflow*.

Algunos sistemas colaborativos no pueden ser clasificados en una categoría concreta de esta clasificación, ya que hay distintos grados de colaboración y muchos sistemas admiten de forma integrada la colaboración síncrona y asíncrona. Actualmente se tiende a construir sistemas que soporten colaboración del tipo "En Cualquier Momento, En Cualquier lugar".

2.2.1.1 Taxonomía de las aplicaciones

Otra clasificación que nos puede ayudar a comprender esta área es la clasificación en función del tipo de aplicación. Así tenemos por ejemplo:

1. **Sistemas de mensajes**

   Estos sistemas se caracterizan por usar el concepto de mensaje para pasar la información de unas partes a otras. Un claro ejemplo de este tipo de aplicación es el correo electrónico. Por lo general son sistemas asíncronos, aunque pueden incluir sistemas síncronos para la notificación de algunos tipos de mensajes.

2. **Editores multiusuario**

   En este grupo entran las aplicaciones de manipulación de documentos que permiten la edición por parte de múltiples usuarios, tanto de forma síncrona como asíncrona.

En el caso de editores síncronos, como SubEthaEdit[7], o Writely[8], suele existir algún tipo de coordinación o bloqueo, para por ejemplo impedir que varios usuarios modifiquen simultáneamente una misma sección de un documento, mientras que se permite el acceso de lectura a todas las secciones.

A veces este control es demasiado limitado, ya que hace más lenta la edición, y se hacen necesarios otros tipos de mecanismos como algoritmos de serialización de eventos, que permitan agrupar los cambios en bloques mayores, o mecanismos de control optimistas, que permitan la edición libre, resolviendo los conflictos cuando surgen, de forma automática o manual.

En los editores asíncronos los usuarios modifican o revisan partes de documentos y envían los cambios al resto de colaboradores. Se hace por tanto necesario algún tipo de control de versiones. Para solucionar este problema se pueden usar sistemas genéricos de control de versiones, como por ejemplo CVS[9], o Subversion[10]. Este último sistema está reemplazando progresivamente a CVS, que era el sistema dominante, ya que gestiona de forma más eficiente las versiones, tanto en ficheros de texto como binarios.

3. Sistemas de soporte de decisiones en grupo

Estos sistemas sirven de ayuda para mejorar la productividad en la toma de decisiones sobre problemas no estructurados. Esta mejora puede ser tanto para acelerar el proceso como para mejorar la "calidad" de la decisión tomada. Por ejemplo, ZingThing, [Fitzgerald 2004], ofrece un entorno de aprendizaje y toma de decisiones estructurado, donde los participantes normalmente siguen una serie de pasos para exponer sus ideas, reflexionar o votar de diferentes formas.

---

7 SubEthaEdit es un editor síncrono para MAC. Para más información se puede consultar http://www.codingmonkeys.de/subethaedit

8 Writely es un editor colaborativo para la web, comprado recientemente por Google. Para más información consultar http://www.writely.com/

9 CVS es un sistema de control de versiones utilizado fundamentalmente para el desarrollo colaborativo de software. Está disponible en http://www.nongnu.org/cvs/

10 Subversion o SVN surge como una mejora de CVS, destacando una gestión más eficiente de ficheros binarios. Está disponible en http://subversion.tigris.org/

4. Soporte de conferencias

En este apartado tenemos los sistemas síncronos que dan soporte a las conferencias, tanto locales como remotas. Así pues tenemos los siguientes tipos:

4.1. Conferencias en Tiempo Real: Este es el caso por ejemplo de las conferencias locales o remotas en las que el ordenador integra sistemas de toma de decisión, aplicaciones compartidas, pizarras electrónicas, etc.

4.2. Teleconferencias: En este caso no tiene por qué usarse un ordenador, y se trata únicamente de comunicación de audio y/o vídeo.

4.3. 'Conferencias de Escritorio' (Desktop conferencing): Los dos casos anteriores han dado lugar a este, que no es más que una integración de las teleconferencias junto con las conferencias en tiempo real. Netmeeting®[11] , gnomeMeeting, o Microsoft Messenger son ejemplos de aplicaciones que permiten este tipo de conferencias.

En general hay dos alternativas para la compartición de aplicaciones, por un lado tenemos sistemas como NetMeeting, VNC, o Remote Desktop, que permiten compartir aplicaciones mono-usuario sin modificarlas y por otro lado tenemos los sistemas específicamente diseñados para tener en cuenta la presencia de múltiples usuarios o que permiten la adaptación de la interfaz de la aplicación compartida a las preferencias o papeles de los distintos usuarios. El inconveniente de esta segunda opción es que el mayor grado de flexibilidad ofrecido impide el uso compartido, sin ser modificadas, de aplicaciones que no están diseñadas para estos sistemas. También tenemos sistemas como Disciple [Li 1999] que permiten ambos tipos de aplicaciones.

También se han creado estándares, como hemos visto en el apartado 2.2, que han fomentado el desarrollo de bibliotecas genéricas de conferencia. Esto ha provocado la aparición de un gran número de aplicaciones de este tipo, que permiten compartir aplicaciones, vídeo conferencia, mensajería instantánea, etc.

[11] NetMeeting: http://www.microsoft.com/windows/netmeeting

5. **Agentes inteligentes**

   Los agentes inteligentes se pueden usar para simular otros usuarios, aunque suelen estar limitados a un conjunto de tareas predeterminadas. Son usados, por ejemplo, en algunos juegos cuando el número de usuarios es insuficiente.

   En general realizan operaciones que podrían hacer otros usuarios, mostrando además un comportamiento similar al que mostraría un usuario normal ante el resto de colaboradores.

6. **Sistemas de coordinación**

   Estos sistemas permiten ver las acciones del usuario y las relevantes de otros usuarios en el contexto de un objetivo común. Permiten crear informes sobre el trabajo realizado y/o restante, programar alertas, planificar, fijar citas, etc.

7. **Entornos integrados de trabajo en grupo**

   Estos entornos son una combinación de las aplicaciones anteriores y permiten crear un entorno unificado para el uso de herramientas y aplicaciones de trabajo en grupo. Suelen incluir calendario y ficheros compartidos, votación, correo y mensajería instantánea, gestión de proyectos, etc. En el apartado 2.3.2 veremos algunos ejemplos de estos entornos.

2.2.2 Construcción de aplicaciones colaborativas

En los últimos años se han desarrollado un gran número de aplicaciones que explotan el trabajo colaborativo, lo que ha motivado la aparición de algunas herramientas para el desarrollo de estas aplicaciones, como ClockWorks [Graham 1996], Habanero [Jackson 1999], Collabrary [Boyle 2002] o ANTS [Lopez 2003], que estudiaremos en el apartado 2.3.1.

La construcción de herramientas genéricas para la construcción de aplicaciones colaborativas es una tarea bastante compleja, en gran medida por la diversidad de aplicaciones y de áreas de conocimiento que implica su desarrollo, como el área de Interacción persona-ordenador, Redes de

comunicaciones, Inteligencia artificial, Sistemas Distribuidos, Teoría Social, etc.

A continuación detallaremos las distintas propiedades que se han de tener en cuenta en el desarrollo de aplicaciones groupware [Graham 1999], para terminar viendo una arquitectura genérica de las aplicaciones colaborativas.

2.2.2.1 Identificación de requisitos

Estas propiedades a tener en cuenta están divididas en distintos grupos de requisitos, usando el modelo Clover [Calvary 1997], el cual tiene la siguiente estructura:

1. Requisitos de Producción: El "espacio de producción" denota los objetos compartidos que son manipulados colaborativamente, tal como documentos compartidos. Las funciones y requisitos en este espacio hacen referencia a la visión y manipulación de estos objetos.
2. Requisitos de Coordinación: Estos requisitos tienen que ver con la coordinación y los protocolos necesarios para que varias personas trabajen en grupo. Estos protocolos pueden ser puramente "sociales" o estar definidos formalmente, por ejemplo, mediante un sistema de control de flujo de trabajo.
3. Requisitos de Comunicación y Consciencia: El "espacio de comunicación" denota los sistemas que soportan la comunicación entre los colaboradores que, como vimos anteriormente, puede ser tanto asíncrona como síncrona.
4. Requisitos tecnológicos: En este grupo veremos otros requisitos que no son exclusivos de las aplicaciones de groupware, pero que son igualmente importantes, como son la adaptabilidad a los recursos, la reutilización de aplicaciones existentes, la tolerancia a fallos, etc.

2.2.2.1.1 Requisitos de Producción

En este espacio estudiaremos los siguientes requisitos relacionados con la manipulación de los objetos compartidos:

1. Acoplamiento de las Interfaces de Usuario, integración de la colaboración síncrona y asíncrona, y el mantenimiento de la consistencia

El sistema debería permitir que los usuarios puedan trabajar tanto en un espacio de trabajo privado como compartido, integrando las ventajas de la colaboración síncrona y asíncrona.

Los problemas surgen cuando hay que sincronizar el espacio de trabajo. En este caso se hace necesario el uso de versiones y de métodos para mezclar distintas versiones, así como para resolver los posibles conflictos.

2. Flexibilidad

   Los usuarios deben poder configurar su interfaz de usuario, tal como se hace en los sistemas mono-usuario, con la complejidad añadida de tener que mantener diferentes formas de ver el sistema para cada usuario. En algunos casos esto no es posible, por ejemplo en los sistemas basados en presentar una misma interfaz para todos los usuarios.

   También deberían poder decidir sobre el control de acceso a lo que están haciendo, qué partes son privadas, o qué partes son de sólo lectura, por poner algunos ejemplos.

3. Historia: edición, revisión, anotación, hacer y deshacer acciones, filtrado y gestión de versiones.

   El sistema debería proporcionar una historia sobre lo que se ha hecho y quién lo ha hecho. La historia a su vez se puede tratar como un objeto compartido más, teniendo la posibilidad de hacer anotaciones, revisarla colaborativamente o individualmente, deshacer o rehacer parte de la historia, filtrarla y crear versiones.

A continuación profundizaremos en las distintas soluciones al primer requisito del espacio de producción, ya que es el requisito principal y uno de los más complejos de conseguir, desde el punto de vista tecnológico. Veremos por separado cada uno de los tres aspectos que lo componen, acoplamiento de las Interfaces de Usuario, integración de la colaboración síncrona y asíncrona, y finalizando con las distintas técnicas para el mantenimiento automático o semiautomático de la consistencia.

Acoplamiento de las Interfaces de Usuario

Hay dos alternativas opuestas de acoplamiento entre las interfaces de los usuarios, **WYSIWIS** "*What You See Is What I See*", y **WYSINWIS**, "*What You See Is Not What I See*", y toda una graduación intermedia entre ellos.

1. WYSIWIS: En esta primera alternativa tenemos los sistemas que nos aseguran que todos los usuarios ven lo mismo, lo cual elimina gran parte de flexibilidad al sistema, ya que los usuarios no pueden personalizar la aplicación. Sin embargo podemos optar por esta solución a niveles más abstractos que la interfaz de usuario, o sólo en parte de la interfaz.

   Este modelo apareció con las primeras aplicaciones multiusuario y ha ido evolucionando para permitir mayor flexibilidad, dando origen a interfaces de usuario personalizables que permiten una mayor flexibilidad pero que dificulta la interacción entre los usuarios al tener distintas visiones del modelo de datos compartido.

   Actualmente se siguen usando interfaces idénticas para todos los usuarios, sobre todo cuando necesitamos compartir aplicaciones existentes, aunque esto nos ofrece una colaboración bastante limitada. Por ejemplo sistemas como VNC[12], o Disciple [Li 1999] permiten compartir aplicaciones existentes entre varios usuarios de una forma muy sencilla.

   En este tipo de sistemas sólo tenemos una instancia de la aplicación, que se ejecuta en una de las máquinas como una aplicación normal. La salida gráfica de la aplicación se envía al resto de usuarios capturando la interacción entre la aplicación y el servidor gráfico mientras que la entrada de teclado y ratón de los usuarios remotos se envía a la instancia de la aplicación como si dicha entrada se hubiese producido en local para la aplicación. Estos sistemas permiten compartir aplicaciones de forma transparente, pero dependen en gran parte del sistema operativo.

   Con la aparición de Java existe también la posibilidad de compartir aplicaciones Java de forma independiente a la plataforma. Sin embargo, esto aún no es una tarea sencilla, aunque se ha simplificado desde la versión Java 2.0 (JDK 1.2). El problema en Java se debe

[12] En http://www.realvnc.com el equipo que inventó VNC mantiene una versión mejorada, con fines comerciales, aunque hay multitud de implementaciones gratuitas disponibles en la Web.

fundamentalmente al tipo de información que manejan los controles gráficos de la biblioteca AWT.

En la biblioteca AWT de Java existen fundamentalmente tres problemas que dificultan la compartición de forma transparente de aplicaciones o Applets:

a) El modelo usado en la biblioteca AWT es el de un componente Java que encapsula un componente nativo del Sistema Operativo.

b) Los eventos producidos por el usuario contienen información que depende del sistema operativo.

c) Otro problema importante es cómo identificar los componentes que han producido los eventos para saber a qué componente corresponde enviar los eventos en las otras instancias de la aplicación.

   De estos tres problemas los dos primeros se han solucionado en gran medida con la aparición de SWING. SWING es una biblioteca gráfica similar a AWT, pero con la gran diferencia de estar escrita completamente en JAVA. De esta forma se puede conseguir captar los eventos de una forma más sencilla e independiente al sistema operativo, por ejemplo interponiendo un componente transparente entre la aplicación y el usuario que capture toda la información antes de llegar a la aplicación.

   Uno de los mayores problemas con este tipo de aplicaciones es la dificultad para manejar las aplicaciones entre varios usuarios, sobre todo si están comunicados mediante redes con alta latencia o bajo ancho de banda, ya que las operaciones que realizan los distintos usuarios se pueden mezclar de forma que los resultados no son los esperados por ninguno de ellos. Para evitar este y otros problemas es necesario usar otros modelos que sitúen el estado compartido a un nivel con más carga semántica.

2. WYSINWIS: En este caso tenemos los sistemas que no mantienen sincronizado todo el estado compartido de la aplicación. En estos sistemas los usuarios pueden trabajar de forma síncrona o asíncrona, por lo tanto será necesario implementar mecanismos para gestionar las distintas versiones del documento y unirlas cuando sea necesario. En este tipo de

aplicaciones los usuarios comparten sólo una mínima parte del estado de la aplicación y no es necesario que estén viendo la información de la misma forma.

Podemos tener distintos grados de acoplamiento entre las distintas interfaces de usuario, por varios motivos [Dewan 1995]:

a) Debido a la variedad de aplicaciones multiusuario: Por ejemplo, en las aplicaciones síncronas el acoplamiento se realiza en "tiempo real" mientras que en las asíncronas, como el correo electrónico, el momento en que se acoplan los datos lo decide el usuario al enviar los mensajes.

b) Variedad de fases: Durante el transcurso de una sesión colaborativa la colaboración pasa por distintas fases, algunas de las cuales pueden requerir un acoplamiento en tiempo real, por ejemplo al comienzo de una tarea, para poner de acuerdo a todos los colaboradores sobre qué realizará cada uno, mientas que en otras fases los usuarios pueden estar trabajando de forma "privada", por ejemplo en tareas que no sea necesaria la colaboración.

c) Eficiencia: En general, si queremos mantener un grado alto de acoplamiento de los datos necesitamos transmitir un mayor número de eventos de notificación y aumentará el tiempo de respuesta del programa. El sistema debería poder ajustar el grado de acoplamiento de forma que se mantenga el tiempo de respuesta por debajo del tiempo de respuesta máximo aceptado por los usuarios.

d) Variedad de "roles": En el sistema podemos tener usuarios que desempeñen papeles bien diferenciados, pudiendo tener usuarios interesados más en unos datos que en otros, por lo que cada tipo de usuario podría tener una interfaz de usuario adaptada a sus necesidades.

3. Interfaces 3D

Un tipo especial de interfaces de usuario son las interfaces 3D presentes en los Entornos Virtuales Colaborativos (Collaborative Virtual Environments, o CVEs). Estos sistemas usan la tecnología de realidad virtual para visualizar un mundo lleno de usuarios que interactúan con los objetos y el resto de usuarios.

El uso de esta tecnología esta disponible en los ordenadores actuales gracias a la aparición de estándares, como el lenguaje VRML, la mejora y abaratamiento de los dispositivos que soportan gráficos 3D y la mejora de las comunicaciones en Internet.

Los CVEs se pueden usar también como interfaz de un sistema de gestión de documentos [Benford 1997]. En los mundos virtuales se pueden representar fácilmente los documentos y su entorno, simulando lo que ocurriría en el sistema clásico de gestión de documentos, una oficina. Estos entornos permiten simular pizarras virtuales, escritorios, personas y sus gestos, con la ventaja de proporcionar una interfaz más cercana a la realidad.

En estos sistemas también son aplicables los modelos vistos en los apartados anteriores, aunque a un nivel superior de abstracción. Los CVEs que presentan los mismos datos a cada usuario se pueden ver, bajo otro punto de vista, como sistemas del tipo WYSIWIS. Parece por tanto interesante permitir distintos grados de acoplamiento, al igual que ocurría en las interfaces de usuario 2D. Por ejemplo unos usuarios podrían ver sólo los objetos en los que están interesados para facilitarles su manipulación.

4. Reacoplamiento de las interfaces de usuario

Como hemos visto anteriormente, es interesante soportar distintos grados de acoplamiento en las interfaces de usuario; sin embargo esto puede originar problemas a los distintos usuarios cuando necesitan realizar una colaboración más estrecha.

Una solución a este problema podría ser restaurar la configuración de los usuarios a una por defecto, para realizar el acoplamiento sobre la misma configuración, pero esta solución es muy costosa, en especial para los usuarios, ya que obligaría a estos a reconfigurar su interfaz cada vez que requieran colaborar con otros usuarios.

Para evitar este problema es necesario automatizar el proceso de reacoplamiento permitiendo unos cambios graduales en la interfaz de usuario. La interfaz de usuario se puede adaptar ligeramente en función de los usuarios con los que están colaborando más estrechamente, de forma

que en la configuración final influyan las preferencias del usuario y la de los usuarios que están más próximos.

Una posible implementación de este acoplamiento se puede basar en controlar la transparencia de los objetos en un entorno virtual, [Smith 1998]. Por ejemplo, los usuarios pueden hacer transparentes una serie de objetos para concentrar su trabajo en otros objetos más importantes para ellos. En este caso, a la hora de colaborar con otros usuarios necesitan de alguna forma saber que objetos ven los otros usuarios. La solución consiste en modificar la transparencia de los objetos en función del valor que tiene dicha transparencia en los usuarios más cercanos. De esta forma relativamente sencilla podemos conseguir que los usuarios próximos en el espacio virtual vean un "mundo" similar.

Integración de la colaboración síncrona y asíncrona

Como vimos en la clasificación mediante la matriz espacio-tiempo, los sistemas se pueden dividir en sistemas síncronos y asíncronos. En general, la elección de un tipo de sistema u otro dependerá en gran medida de las características del problema que queremos solucionar y de cómo queramos solucionarlo.

Por ejemplo, los sistemas de gestión de flujo de trabajo permiten coordinar a varias personas formalmente en la cooperación para realizar un trabajo, mientras que un sistema de colaboración multimedia permitiría que esas mismas personas realizasen su trabajo colaborativamente, pero de una forma informal, coordinada por los propios colaboradores.

Sin embargo, las características de un problema no siempre son uniformes en toda la extensión del mismo, y es posible que en algunos momentos sea más interesante un tipo de colaboración u otro. Un sistema que pretenda ser de utilidad para resolver un gran número de problemas deberá soportar ambos tipos de colaboración de una forma integrada.

La forma de integración dependerá de cual es la forma predominante. En el caso de la integración para la realización de proyectos, de sistemas de gestión de flujo de trabajo y de sistemas de colaboración multimedia, la forma predominante será la asíncrona, ya que por lo general los distintos usuarios

suelen trabajar en documentos privados, y únicamente necesitan trabajar síncronamente con el mismo documento en puntos concretos del desarrollo.

Una posible integración es la propuesta en WOTEL [Weber 1997]. En este sistema se integra la gestión de flujo de trabajo con la planificación de actividades que serán llevadas a cabo por varios usuarios. Varios usuarios colaboran en la realización de una actividad gracias al uso de tele-conferencias, aplicaciones compartidas y documentos compartidos.

También podemos integrar colaboración asíncrona en sistemas predominantemente síncronos. Una forma de realizar esto es extendiendo el sistema de paso de mensajes, para permitir mensajes asíncronos de algunos colaboradores. También es importante la persistencia de las sesiones de colaboración, y necesitaremos mecanismos para almacenarlas, para permitir que se incorporen nuevos colaboradores en cualquier momento y para poder mezclar o crear versiones de las sesiones de colaboración [Jackson 1999].

Una de las ventajas de almacenar las sesiones de colaboración es que nos permite realizar consultas en los datos producidos o analizar las sesiones. Por ejemplo, esto se puede utilizar como un mecanismo de documentación de prototipos [Lindstaedt 1997] o como un medio de aprendizaje.

Mantenimiento de la consistencia

En entornos colaborativos en los que varios usuarios están trabajando simultáneamente sobre los mismos datos es importante asegurar la consistencia, o al menos poder detectar inconsistencias para permitir una corrección automática o asistida por los usuarios.

Este mismo problema aparece en los sistemas de bases de datos y se soluciona mediante el uso de transacciones. Las transacciones permiten que un conjunto de operaciones se realicen de forma indivisible como si fueran una operación atómica. En el caso de aparecer conflictos se usa el bloqueo para evitar la posible inconsistencia.

En los sistemas colaborativos también podemos usar el bloqueo para evitar conflictos. Esto puede provocar problemas de eficiencia, que se pueden solucionar disminuyendo la granularidad de dicho bloqueo.

Los problemas aumentan cuando la colaboración se lleva a cabo a través de redes con una latencia alta. En estos casos el uso de bloqueos no es práctico ya que necesitamos esperar a la concesión del bloqueo antes de operar y esto se traduce en retrasos importantes cada vez que se ejecuta una acción para la que necesitamos un bloqueo.

Para solucionar el problema de la latencia es necesario permitir un control optimista en el que los usuarios realicen las operaciones localmente, detectando posteriormente los conflictos para poder encontrar la solución. En esta sección veremos algunos métodos para detectar y solucionar conflictos automáticamente.

En los entornos de colaboración distribuidos es muy común tener los datos replicados para cada usuario. De esta forma evitamos tener un punto central que disminuiría la concurrencia y aumentaría la latencia. Para mantener la consistencia, sistemas como Habanero, [Jackson 1999], o Disciple, [Li 1999], replican las operaciones sobre los datos en todas las instancias de las aplicaciones.

En principio podemos mantener la consistencia de los datos si realizamos las operaciones en el mismo orden en todas las instancias de la aplicación. Pero esto, al igual que en el caso de los bloqueos, aumentará el tiempo de reacción debido a la latencia de la red, aunque usemos algoritmos distribuidos para determinar el orden de las operaciones.

Para poder dar una solución válida en sistemas de alta latencia, que admita un alto nivel de concurrencia y que gestione la consistencia, necesitamos información semántica sobre las acciones realizadas. En general, la solución ideal será una combinación entre bloqueo de granularidad fina, gestión optimista de la concurrencia e información semántica sobre las operaciones que nos permita determinar las dependencias y la forma de resolver los conflictos. A continuación veremos en detalle estas soluciones.

1. Disminución de la granularidad en el bloqueo

   Para disminuir la granularidad de los bloqueos podemos basarnos en la descomposición de los objetos compartidos a bloquear en subobjetos. Así, creamos un árbol de bloqueos de múltiple granularidad, según el nivel del

árbol en el que se encuentra. Los bloqueos se pueden asociar a tipos de datos simples, listas o estructuras y a operaciones sobre estos tipos de datos.

Cada bloqueo tiene asociado una serie de modalidades de bloqueo: sólo lectura, escritura, escritura-lectura, modos de bloqueo parciales, que afectan únicamente a algún campo o elemento de una secuencia, o modos de bloqueo asociados a una operación, como inserción o borrado. De esta forma cuando llega una petición de bloqueo de un tipo hacia un objeto este bloqueo se propaga hacia arriba en el árbol de bloqueos, cambiando de tipo si es necesario. Para gestionar el funcionamiento de los bloqueos podemos usar una serie de tablas que nos indican cómo se propagan los bloqueos, y la compatibilidad de los distintos modos de bloqueo, [Munson 1996].

Los bloqueos se pueden usar también en sistemas con gestión de concurrencia optimista, donde se asume que el recurso no estaba bloqueado, y se conseguirá por tanto el bloqueo. En estos casos los bloqueos son de utilidad para detectar inconsistencias. Una vez detectada la inconsistencia, por el sistema de gestión de concurrencia, se han de deshacer las operaciones realizadas por el usuario que no consiguió el bloqueo, de forma interactiva o automática, de esta forma conseguimos un sistema de bloqueo optimista, con el que podemos disminuir considerablemente las latencias de las operaciones.

2. Modelización de las operaciones

Para permitir una mayor concurrencia, es necesario crear modelos que representen las dependencias de las operaciones realizadas. De esta forma se minimizan los conflictos, pudiéndose ejecutar las operaciones sobre los datos en distinto orden, consiguiendo el mismo resultado final y manteniendo la intención de los usuarios al realizar las operaciones [Ellis 1989]. Estos modelos son también de gran utilidad para implementar otras operaciones como el “deshacer” colaborativo o en grupo [Ressel 1999].

El algoritmo dOPT, [Ellis 1989], es uno de los primeros algoritmos que intenta mantener la intencionalidad de los usuarios, a la vez que permite la ejecución de las operaciones en distinto orden. El problema es que este algoritmo no cubre todas las posibles situaciones. Para solucionar esto

surgen algunas alternativas como adOPTed, [Ressel 1996], GOT, [Sun 1997], o [Suleiman 1997]. Estos algoritmos se basan en modificar las operaciones, cuando se han de replicar en cada instancia de la aplicación compartida, para adaptarlas a la nueva situación de forma que se preserve la intención del usuario que originó la operación.

Por ejemplo, supongamos la siguiente situación: tenemos un editor colaborativo y el documento actual contiene la cadena "abc". En ese momento dos usuarios distintos quieren borrar las letras "a" y "b" respectivamente. Estas letras, en el instante anterior a la ejecución de cada operación, se encuentran en las posiciones 1 y 2, por lo tanto las operaciones que se ejecutaran en el editor del primer usuario serán borrar la posición 1, borrar la posición 2, mientras que en el otro serán borrar la posición 2, borrar la posición 1. Al ejecutar estas operaciones, en el primer caso quedara la cadena "b" mientras que en el otro quedara la cadena "c". En este caso la intención era borrar "a" y borrar "b" y no se ha respetado. Para conseguir esto es necesario transformar las operaciones que se realizaran en un contexto distinto al original de forma que se respete la intención original del usuario.

Para mantener la intención de los usuarios, aunque se cambie el orden de ejecución de las instrucciones, se definen una serie de funciones de transformación. Las funciones de transformación modifican una operación para adaptarla al cambio de contexto que produce otra operación preservando la intención del usuario. Si una operación realmente no afecta a la ejecución de otra operación ambas se podrán realizar en paralelo o en cualquier orden sin que sea necesario aplicar ninguna función de transformación.

Suponiendo que todos los usuarios parten del mismo estado inicial, podemos definir el estado de una aplicación como un vector donde cada componente v[i] representa el número de operaciones que ha recibido del usuario i correspondiente. Este vector se manda junto con cada operación realizada y permite conocer en qué estado estaba la aplicación que originó la operación. Se asume que cada usuario emite las operaciones en forma secuencial.

El algoritmo permite que las operaciones realizadas por un usuario A se ejecuten en local de forma inmediata, mientras que las operaciones emitidas por un usuario B serán ejecutadas en A cuando éste haya ejecutado todas las operaciones que B había ejecutado cuando emitió la operación. El único problema que se plantea es que, cuando se ejecuta la operación de B, se pueden haber realizado más operaciones que modifican el contexto que B tenia cuando emitió la operación. Para adaptar la operación de B al nuevo contexto de A se usan las funciones de transformación, que modifican la operación emitida por B mediante transformaciones sucesivas realizadas con todas las operaciones ejecutadas por A que no fueron ejecutadas en el contexto de la operación emitida por B.

Si no es posible la transformación de las operaciones al nuevo contexto, podemos usar un método genérico de transformación basado en hacer y deshacer operaciones de forma que se ejecuten en el mismo orden en todos los usuarios, asegurando el estado final, aunque no se puede asegurar el mantenimiento de la intencionalidad [Sun 1996]. En este método se mantiene un buffer histórico de operaciones ejecutadas, y se sigue el siguiente procedimiento:

a) Cuando llega una operación que no tiene el contexto adecuado para ejecutarse, basándonos en el vector de operaciones se detecta la posición que debería ocupar en el buffer histórico.

b) Se deshacen las operaciones posteriores.

c) Se ejecuta la operación en el contexto adecuado y finalmente se ejecutan de nuevo las operaciones deshechas.

Este método de hacer / deshacer puede complementar las funciones de transformación, simplificando su creación, ya que permite separar el mantenimiento de la intencionalidad con el mantenimiento del estado final [Sun 1997].

La operación de deshacer se puede simplificar usando las funciones de transformación, [Ressel 1999]. Deshacer una operación global es relativamente sencillo, ya que se ejecuta en el estado actual y no necesita

ninguna transformación, pero en el caso de deshacer una operación local hay algunas complejidades añadidas.

En el "deshacer local" la intención del usuario es deshacer la última operación que él produjo. El problema es que después pueden llegar otras operaciones que hacen variar el contexto de la operación de deshacer. Una solución al problema puede ser transformar el contexto de la operación de deshacer mediante las funciones de transformación. De esta forma podemos deshacer una operación aunque esta no sea la última operación realizada.

3. Mantenimiento de la intencionalidad en caso de conflictos.

   En los apartados anteriores hemos visto dos alternativas para el mantenimiento de la consistencia, mediante el uso de bloqueos y sistemas de gestión optimista de la concurrencia que intentan mantener la intencionalidad de los usuarios.

   Sin embargo ninguna de las dos alternativas anteriores permite mantener la intencionalidad en caso de operaciones conflictivas. Por ejemplo supongamos un editor gráfico de objetos en el que un usuario pretende mover un objeto a una posición y otro usuario quiere mover ese mismo objeto a otra posición distinta. En este caso no hay forma de transformar las operaciones para mantener la intención de los usuarios, ya que sus intenciones son conflictivas.

   Una alternativa para este caso es la gestión de réplicas o versiones de objetos [Chen 1999]. En este sistema, cuando varios usuarios ejecutan operaciones de intencionalidad conflictiva se crean copias de los objetos en la interfaz de los usuarios. En este caso son los propios usuarios los que solucionan los conflictos, eligiendo una de las copias del objeto en cuestión como la mejor, o incluso descartando todas las copias conflictivas.

2.2.2.1.2 Requisitos de Coordinación

Para mantener la consistencia a nivel de los datos podemos usar distintos algoritmos que la gestionan de forma automática, sin embargo, es necesario disponer de mecanismos que nos permitan, de forma flexible, mantener la

consistencia a nivel de los usuarios. Estos mecanismos han de proporcionar medios para controlar qué tareas han de hacer o pueden hacer los distintos usuarios en cada momento. Estos mecanismos o protocolos pueden ser puramente "sociales" o estar definidos formalmente, por ejemplo, mediante un sistema de control de flujo de trabajo.

Por ejemplo, en los proyectos de desarrollo uno de los puntos más complejos tiene lugar a la hora de integrar las distintas partes en que se ha dividido. Este tipo de tareas revela la necesidad de coordinación existente en proyectos en los que participan un gran número de personas, y la complejidad que supone dicha coordinación [Grinter 1998].

Según la aplicación pueden ser necesarios los siguientes requisitos de coordinación:

1. Coordinación basada en 'roles': Distintos usuarios pueden desempeñar distintas tareas dependiendo del "tipo de usuario", por ejemplo en un sistema en el que participan tutores y alumnos, los tutores podrían modificar el espacio de ejercicios, mientras que los alumnos únicamente podrían resolverlos.
2. Control de recursos de uso único[13]: Cuando tenemos recursos con los cuales sólo un usuario puede actuar al tiempo, tales como canales de audio unidireccionales o documentos donde sólo una persona puede escribir a la vez, es necesario coordinar el acceso a estos recursos, por ejemplo mediante el paso de un testigo que indique quién tiene el control del recurso, o con un moderador.
3. Automatización de tareas: Puede ser necesario disponer de sistemas de alarmas automáticas o que realicen determinadas tareas simples o repetitivas que descarguen de trabajo a los colaboradores. En este aspecto puede ser de gran utilidad el uso de agentes "inteligentes".
4. Control de accesibilidad a los recursos: Los usuarios necesitan poder controlar qué partes pueden ser vistas y de que modo. Por ejemplo, un usuario podría mostrar a otros un objeto compartido indicándoles las operaciones que pueden realizar, permitiendo coordinar así qué acciones pueden realizar otros usuarios en los datos creados por cada uno.

[13] Esto también es llamado con el término inglés 'Floor control'

5. Votaciones: También puede ser de gran utilidad disponer de un sistema de votación ya que ésta puede ser una tarea muy común en muchos sistemas colaborativos.

Podemos dividir la coordinación en tres niveles: nivel de usuario, nivel de sistema y nivel de objetos. Por un lado tenemos el **nivel de usuario**, donde podemos identificar los distintos roles que desempeña cada uno y cómo están organizados. En el otro extremo, el **nivel de objetos**, tenemos los detalles del control de acceso a los distintos objetos y subobjetos. El **nivel de sistema** es el que permite la gestión eficiente de la coordinación, definiendo políticas de uso, las cuales coordinan los distintos papeles que pueden tomar los usuarios con los derechos de acceso / uso del sistema.

A continuación veremos los aspectos principales de cada uno de estos tres niveles.

1. Nivel de usuario: Identificación de roles

   La identificación de roles nos permite simplificar bastante la tarea de coordinación, ya que permite definir la coordinación en términos de tipos de usuario en lugar de en usuarios concretos. Además esto nos permite ligar papeles con usuarios en tiempo de ejecución de una forma sencilla y flexible.

2. Nivel de sistema: Políticas

   Las políticas definen de forma general cómo se comporta el sistema en cuanto a la coordinación, y de esta forma se pueden predeterminar gran parte de los aspectos de coordinación, basándose en el estado actual de los objetos del sistema y los usuarios que participan en él. Estas políticas sirven, por tanto, para establecen criterios a la hora de resolver situaciones de conflicto.

   Estas políticas modelan la colaboración limitando la interacción. Es por tanto necesario encontrar el punto de equilibrio entre definir políticas lo suficientemente restrictivas como para que el sistema sea gestionable y políticas permisivas que permitan una interacción entre los distintos colaboradores rica y fluida.

   La definición de las políticas se puede hacer por separado del diseño de la aplicación. De esta forma una aplicación genérica como por ejemplo un

editor colaborativo puede utilizarse para fines tan diversos como la enseñanza o la coedición de un artículo, simplemente cambiando la forma de coordinar los distintos colaboradores. Para este fin se han creado algunos lenguajes que nos permiten modelar los aspectos de colaboración en una aplicación, [Yang 2002], [Li 1998], [Cortés 1996].

Las políticas del sistema se pueden modelar, en la mayoría de los casos, mediante mecanismos de gestión de control de acceso a los recursos por parte de cada usuario. En este caso, el sistema de gestión de la coordinación será el que interpreta la definición de las políticas y genera los permisos de acceso a los distintos objetos en tiempo de ejecución.

3. Control de acceso

El control de acceso gestiona el acceso a los distintos objetos, recursos o funcionalidad del sistema. Este tipo de control se ha desarrollado tradicionalmente en sistemas operativos y de bases de datos, identificándose dos tipos de control de acceso, el control de acceso básico y el meta control de acceso.

Una de las diferencias con el control de acceso definido en los sistemas de bases de datos es la complejidad, debido fundamentalmente a la diversidad de objetos, usuarios, aplicaciones y operaciones que se encuentran en los sistemas colaborativos [Dewan 1998].

a) Control de acceso básico: Éste define las medidas de protección de los recursos básicos del sistema.

   En el control de acceso básico se definen asociaciones entre objetos y entidades que definen las operaciones que las entidades pueden realizar sobre los objetos. Esto se suele implementar mediante listas de control de acceso asociadas a cada objeto a proteger.

   Los objetos a proteger pueden estar agrupados formando una jerarquía mediante diversas relaciones, como "es parte de" o "es un". Estas jerarquías, al igual que ocurría con la gestión de bloqueos, también permiten crear jerarquías en el control de acceso.

   El control de acceso puede ser en cuanto al acceso a los objetos y su formato, como la escritura y lectura de un atributo, o en cuanto a derechos de acoplamiento, como el derecho de transmitir o recibir

cambios de otros usuarios. También se pueden especificar controles de acceso a información que no es propia de la aplicación, como por ejemplo datos sobre otros usuarios y lo que han hecho en el sistema, permitiéndonos en este último caso implementar el anonimato o el uso de seudónimos.

b) Meta Control de Acceso: Define cómo se protegen las medidas de control de acceso.

   El meta control de acceso define el control de acceso al control de acceso y en principio su gestión es muy similar al control de acceso básico. La mayor diferencia es la naturaleza de los objetos controlados, siendo en este caso controladas las propias estructuras del control de acceso, en lugar de los objetos propios de la aplicación.

   Gran parte de los mecanismos del meta control de acceso se pueden tomar del control de acceso básico mientras que su funcionamiento o semántica se puede abstraer del tipo de aplicación controlada, pudiéndose por tanto automatizar en gran medida su gestión.

   Algunos aspectos del meta control de acceso, como la decisión sobre a quien pertenecen en principio los permisos o los mecanismos de delegación o revocación de permisos, pueden depender del tipo de aplicación o de los intereses de los usuario en un momento dado.

   En el caso de la propiedad de los permisos es típico el uso de la figura del administrador, que es el único que puede dar permisos sobre los objetos a otros usuarios. Otra alternativa es definir permisos de administración por objeto. La figura del administrador se puede implementar definiendo los permisos sobre el objeto "sistema" que engloba el resto de objetos, y estableciendo herencia de permisos.

   Otro aspecto importante es la forma de delegar / revocar los permisos. Un usuario podría tener derecho a delegar sus derechos o parte de ellos a otros usuarios, pudiéndose propagar los permisos entre los distintos usuarios. El problema se plantea a la hora de revocar los permisos. En algunos casos puede interesarnos revocar un permiso a un usuario, mientras que en otras deseamos revocar los permisos a ese

usuario y los usuarios que lo obtuvieron de él, siendo necesario por tanto realizar un seguimiento de los permisos.

2.2.2.1.3 Requisitos de Comunicación y Consciencia

El "espacio de comunicación" denota los sistemas que soportan la comunicación entre los colaboradores que, como vimos anteriormente, puede ser tanto asíncrona como síncrona. Podemos tener los siguientes mecanismos de comunicación, como parte de la aplicación o herramienta colaborativa:

1. Intercambio síncrono y asíncrono de mensajes

   La forma fundamental de comunicación entre personas es mediante el paso de mensajes. Para esta tarea se puede usar el correo electrónico o los sistemas tipo IRC, pero también es necesario integrar sistemas de comunicación mediante vídeo, audio o pizarras compartidas.

2. Anotaciones

   Las anotaciones sirven como medio de comunicación asíncrona. La ventaja sobre otros medios asíncronos como el correo electrónico es que se pueden asociar a determinados objetos o a puntos en la historia de la comunicación, ayudando en este último caso a la revisión o análisis de la colaboración.

3. Consciencia (o *Gestalt views*, en su término Inglés)

   Para que la colaboración pueda llevarse a cabo de forma efectiva unos usuarios han de conocer qué hacen el resto de usuarios. Para ello, se han de suministrar "vistas" que sinteticen la información necesaria, que en su detalle podría ser privada pero que se puede hacer pública una vez sintetizada. Por ejemplo, en un sistema de votación, los votos individuales son privados, pero el recuento es público.

   Esta consciencia es necesaria, en la mayoría de las ocasiones, debido a la forma de interacción de los usuarios en este tipo de sistemas, en los que el "campo de visión" es muy reducido, limitado a la pantalla del ordenador, y no tenemos mucha información sobre lo que hacen otros usuarios.

Esta falta de información puede conducir a que se adopten políticas de colaboración que minimicen la interacción entre los usuarios, lo cual puede ser una mala estrategia en sistemas o entornos muy cambiantes. Para ampliar esta información puede ser útil conocer con quién se está en el entono compartido, dónde está trabajando, qué está haciendo, o qué pretende hacer.

El tipo información de consciencia que podemos tener se puede clasificar como sigue [Steinfield 1999]:

a) Información de actividad: Este tipo de consciencia incluye conocimiento sobre qué están haciendo los usuarios, la relación entre las distintas actividades, o qué ha hecho cada usuario.

b) Información de disponibilidad: Esta información nos indica la disponibilidad de los distintos usuarios, si están presentes actualmente en el sistema, cuánto tiempo llevan ausentes, o cuándo está prevista su vuelta.

c) Información de procesos: Por ejemplo en los sistemas de gestión del flujo de trabajo están muy bien definidos los procesos y quién ha de realizarlos.

d) Información de perspectiva: A la hora de colaborar es importante poder anticipar las acciones de otros usuarios. Para ello necesitamos, no sólo información pasada sobre las acciones realizadas, sino también información sobre cómo podrían ocurrir las distintas acciones. Esto incluye información sobre conocimientos y creencias de los otros colaboradores.

e) Información del entorno: Esta información hace referencia a los eventos que ocurren fuera del espacio de trabajo que pueden tener implicaciones en la actividad del grupo.

Para suministrar esta información a los usuarios podemos usar diversos mecanismos, por ejemplo, [Gutwin 1996]:

a) Punteros remotos: En las aplicaciones de tipo WYSIWIS puede ser útil usar punteros remotos para señalar partes la ventana al resto de usuarios, como complemento a los mecanismos de conferencia como canales de audio o de vídeo.

b) Barras de *scroll* multiusuario: Se usan en los editores compartidos para indicar en qué parte del documento está actuando cada usuario.

c) Ventanas del tipo WYSIWID (*What You See Is What I Do)*: Estas ventanas permiten que unos usuarios vean lo que hacen otros usuarios, en un espacio reducido.

d) Vistas en miniatura y vistas de radar: Las vistas en miniatura muestran todo el espacio compartido en una sola ventana. Las vistas de radar muestran información sobre otros usuarios, dentro de la vista en miniatura, como por ejemplo la posición del espacio compartido donde están trabajando actualmente.

2.2.2.1.4 Requisitos tecnológicos

En este último apartado veremos otros requisitos que no son exclusivos de las aplicaciones de groupware, pero que son igualmente importantes, como los siguientes:

1. Adaptabilidad de recursos: Las aplicaciones groupware deberían poder adaptarse a los distintos entornos software y hardware de los usuarios, como velocidad de interconexión con el resto de participantes o dispositivos hardware, como cámaras de vídeo o tarjetas de sonido Full-duplex.
2. Preferencias del usuario: Los usuarios deberían poder tomar decisiones en cuanto al uso de los recursos, como por ejemplo elegir la calidad del sonido o del vídeo
3. Reutilización de aplicaciones existentes: En muchas ocasiones es demasiado complejo adaptar las aplicaciones existentes, por lo que el sistema de desarrollo de groupware debería dar soporte a este tipo de aplicaciones, proporcionando mecanismos genéricos para compartirlas.
4. Información del estado de la red: También podría ser necesario, tanto para los usuarios como para la aplicación, conocer el estado de la red, si el resto de usuarios están recibiendo los datos, si la red esta colapsada, o si hay algún fallo.
5. Tolerancia a fallos: Uno de los aspectos tecnológicos más importantes en las aplicaciones groupware es la tolerancia a fallos, debido sobre todo a la

dependencia de la red de comunicación y del número de usuarios, ya que esto aumenta considerablemente la posibilidad de fallos. El sistema debe estar preparado en previsión de que se pierda la comunicación con cualquier usuario, y debería poder restablecerla con él en cualquier momento.

2.2.2.2 Arquitectura genérica de las aplicaciones colaborativas

Una forma simplificada de ver una aplicación colaborativa es como un editor compartido de documentos [Dewan 1994]. Bajo este enfoque la colaboración se centra en mantener un documento o estado compartido entre varios colaboradores.

Generalizando la idea del editor colaborativo podemos representar una aplicación colaborativa como formada por distintas capas que se comunican [Munson 1996].

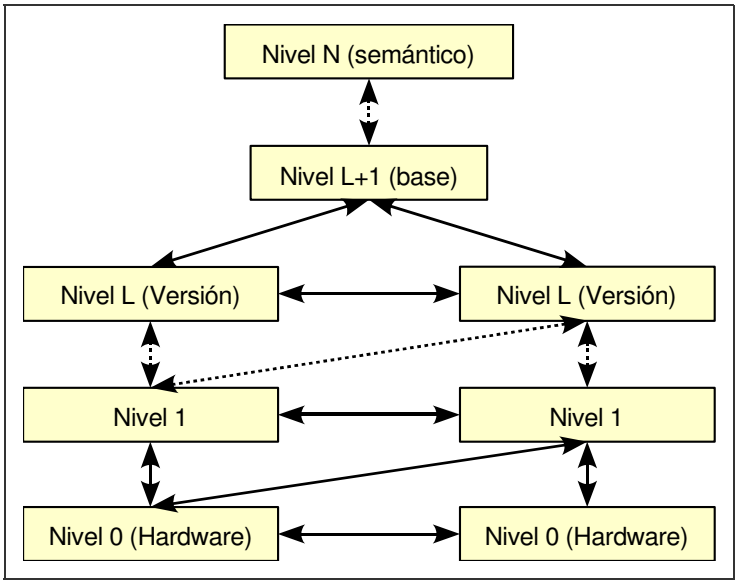

Figura 2: Arquitecturas Multiusuario

En esta arquitectura, Figura 2, la entrada y salida del usuario es procesada por una serie de niveles, que soportan el modelo del editor colaborativo. Según este modelo, cada usuario interactúa con la aplicación manipulando la representación visual de los objetos semánticos de la aplicación o documento.

Los niveles más altos son abstracciones de los niveles inferiores. El nivel inferior (el de hardware) es el más cercano al usuario, y con el que mantiene una mayor interacción. Algunos de estos niveles son compartidos y otros son versiones o replicados. Por ejemplo, en las aplicaciones WYSINWIS la interfaz de usuario puede ser diferente, Nivel 0, mientras que los niveles de abstracción más altos son compartidos.

Los distintos niveles se comunican mediante eventos, algunos de ellos serán eventos de colaboración y otros eventos de interacción, según soporten o no funciones de colaboración. Los niveles compartidos gestionan eventos producidos por todos los usuarios mientras que los otros niveles están asociados a un determinado usuario.

Este enfoque es muy simple, aunque engloba un gran número de aplicaciones colaborativas y nos permite identificar las posibles alternativas en la resolución de algunos aspectos importantes en este tipo de aplicaciones:

1. Acoplamiento de las interfaces de usuario

   Como hemos visto, hay dos alternativas opuestas de acoplamiento entre las interfaces de los usuarios, WYSIWIS y WYSINWIS. En esta arquitectura el nivel de acoplamiento se puede identificar según el nivel base compartido, obteniendo un acoplamiento mayor cuanto más cerca de la interfaz de usuario, nivel 0, esté el nivel compartido.

2. Control de la Concurrencia

   El control de la concurrencia ha de satisfacer principalmente las siguientes propiedades:

   a) Consistencia: El sistema ha de mantener el estado compartido de forma consistente, o al menos debería detectar las posibles inconsistencias para permitir a los usuarios solucionarlas.

   b) Concurrencia: El mantenimiento de la consistencia puede estar en conflicto con el mantenimiento de la concurrencia. En general, cuanta más concurrencia permita el sistema más costoso y complejo se hace el mantenimiento de la consistencia del estado compartido.

3. Control de Acceso

   El control de acceso tiene dos aspectos importantes: por un lado impide a usuarios o programas realizar algunos tipos de operaciones que no tienen permitido, por otro lado puede ser un mecanismo de protección de datos frente a errores de los usuarios.

   La granularidad del control de acceso puede ir desde el nivel de aplicación, nivel de documento, o puede ser de un grano más fino. En general asocia usuarios con posibles acciones que puede realizar sobre el estado compartido o parte de él.

4. Deshacer colaborativamente

   En los editores monousuario es común encontrar la acción de deshacer. Esta acción también puede utilizarse en editores colaborativos. Tenemos distintos tipos de deshacer: los usuarios pueden deshacer sus propias operaciones, o deshacer la última operación que les afecta. La operación de deshacer, a su vez, puede tener un efecto local o global y afectar a todos los usuarios.

Desde un punto de vista funcional podemos ver los distintos módulos del sistema divididos en 4 tipos, [Ellis 1997]:

1. Módulos de datos (*keepers*): Estos módulos son los encargados de conservar el estado compartido de la aplicación. Un ejemplo de este tipo de módulos son las Bases de datos
2. Coordinadores: Este tipo de módulos se encargan de las actividades de grupo, coordinando o sincronizando las acciones de los usuarios, un ejemplo destacado de este tipo son los sistemas de gestión de flujo de trabajo.
3. Comunicadores: Estos módulos son los encargados de gestionar las comunicaciones entre los distintos módulos y/o usuarios. Las comunicaciones pueden ser síncronas, como la vídeo conferencia, o asíncronas, como el correo electrónico.

4. Agentes: Son módulos especializados en tareas concretas que interactúan con el grupo de colaboradores. Hay muchos tipos de agentes, pero podemos resumirlos en tres tipos fundamentalmente, según su ámbito de actuación:
   a) Autómatas o agentes de la aplicación: Estos son los encargados de las reglas o computación propias de la aplicación.
   b) Autómatas o agentes de la interfaz de usuario: Se encargan de capturar y presentar la información a los usuarios.
   c) Autómatas o agentes inteligentes: Este último tipo son agentes que actúan como usuarios no humanos del sistema. Estos agentes se pueden encargar por ejemplo de realizar las tareas repetitivas, como si fuese otro usuario del sistema.

2.3 Aprendizaje colaborativo por ordenador (CSCL)

En los apartados anteriores hemos revisado el marco teórico sobre el que se sustenta el aprendizaje colaborativo, así como los aspectos técnicos y requisitos tecnológicos de los sistemas informáticos para el trabajo en grupo. La unión del aprendizaje colaborativo y los sistemas informáticos para el trabajo en grupo son la base sobre la que se sustenta el paradigma de aprendizaje colaborativo por ordenador, o CSCL en sus siglas en ingles. Este paradigma también tiene mucho que ver con el aprendizaje a distancia, ya que una gran parte de las aplicaciones CSCL permiten de forma transparente el aprendizaje a distancia.

Otros antecedentes del paradigma de CSCL son los sistemas de aprendizaje por ordenador, o Computer Assisted Instruction (CAI) en ingles. En los sistemas CAI el alumno usa el ordenador como herramienta de aprendizaje, para resolver problemas, etc. La evolución de los CAI con la incorporación de Inteligencia Artificial da lugar a los Sistemas Tutores Inteligentes, o Intelligent Tutoring Systems (ITS) en ingles. Los sistemas ITS intentan conseguir que cada alumno tenga asignado un tutor, aunque sea virtual, de forma que pueda recibir una educación más adaptada a sus conocimientos y ritmo de aprendizaje.

Actualmente hay una tendencia a integrar la capacidad de colaboración en los tutores inteligentes para intentar aprovechar las ventajas del aprendizaje colaborativo, aprendizaje a distancia, y de los tutores inteligentes. Una forma de conseguir este objetivo es mediante la utilización de agentes inteligentes, actuando el agente como un participante más de las sesiones colaborativas, ayudando al profesor al facilitarle el seguimiento de las actividades, o a los alumnos, con avisos, sugerencias, etc. De esta forma el agente inteligente se convierte en una especie de ayudante virtual, [Chen 2003].

De especial interés en el área de CSCL es el Aprendizaje Basado en Problemas, o Problem Based Learning (PBL) en inglés. Este tipo de aprendizaje comenzó como una reforma curricular introducida a finales de los 60 en la facultad de ciencias de la salud de la universidad de McMaster [Spaulding 1969]. Se basa en la formación de grupos de aprendizaje con un tutor, y se aprende trabajando con una colección de casos clínicos ideados para la enseñanza. El contacto con el paciente suele ser simulado, por motivos prácticos, y durante el proceso se va anotando los problemas e ideas que surgen, así como las áreas de conocimiento en los que los miembros del grupo sienten que no están suficientemente preparados para comprender el problema estudiado.

Koschmann propone seis principios para conseguir un aprendizaje e instrucción efectivos en dominios complejos y desestructurados [Koschmann 1996]. El paradigma PBL nos puede ayudar a que la instrucción cumpla con dichos principios. Estos principios son:

1. Principio de Multiplicidad: Cuando el conocimiento es complejo, dinámico, sensible al contexto, etc., se deben promover diferentes perspectivas, representaciones y estrategias, que permitan adquirir la riqueza de los conceptos de la forma más completa posible.
2. Principio de Actividad: El aprendizaje no es un proceso pasivo y cuando se trata un problema complejo suele ser necesaria la búsqueda activa de más información. Por lo tanto es necesario enseñar a buscar información y participar de forma activa en el proceso de aprendizaje.
3. Principio de Acomodación y Adaptación: La instrucción debe beneficiarse del conocimiento previo del alumno. También se debe poder monitorizar su aprendizaje para corregir ideas erróneas, y se debe poder adaptar de

forma flexible para aprovechar al máximo el esfuerzo del alumno por aprender.

4. Principio de Autenticidad: El contexto en el que se da el aprendizaje afecta a la habilidad del alumno en aplicar posteriormente ese conocimiento, por lo tanto se debe centrar la instrucción en problemas reales, o lo más parecidos posible a los auténticos.
5. Principio de Articulación: La articulación o estructuración de los conceptos mejora el aprendizaje, ya que fuerza la creación de un discurso, que ha de ser coherente, relacionando por lo tanto distintos conceptos, lo cual mejora la retención de la información. También nos da un mecanismo para la abstracción del conocimiento del contexto en el que fue adquirido. Esto se debe a que el alumno tiene que ver las cosas desde distintas perspectivas para poder articular las relaciones entre los conceptos, determinando así que elementos son generalizables y cuales son específicos del contexto.
6. Principio de Incompletitud[14]: Una de las características del conocimiento en dominios complejos es que no se puede ver como un sistema cerrado, con un final definible o completo. El aprendizaje por lo tanto es dinámico, y va cambiando conforme se adquiere nuevo conocimiento. Por lo tanto se ha de fomentar que el alumno aprenda por sí mismo, ya que el proceso de aprendizaje continuará indefinidamente. Se ha de fomentar también la habilidad para identificar recursos eficientemente, para mejorar su propio conocimiento, y una actitud crítica hacia su propio conocimiento, ya que con la nueva información que adquiera será necesario reconsiderar su validez.

El aprendizaje basado en problemas se puede beneficiar de la tecnología. En el área de CSCL existen varios sistemas enfocados a facilitar este tipo de aprendizaje en distintas áreas científicas, como por ejemplo STEP [Steinkuehler 2002], SMILE [Lamberty 2001] o Zebu [Tiessen 1999]. Estos sistemas permiten a los estudiantes organizar, exponer y discutir ideas formando grupos de trabajo, normalmente supervisados por un profesor o tutor.

[14] El término original utilizado en la literatura en Inglés es Termlessness, y denota la cualidad de algo que no tiene finalización.

Algunas experiencias también demuestran que se puede realizar PBL distribuido, es decir, a distancia, utilizando herramientas groupware genéricas, como Microsoft NetMeeting [Cameron 1999]. El problema que encontramos al utilizar herramientas genéricas es que el profesor tiene que evitar que los alumnos se salgan de la estructura diseñada en la experiencia, ya que estas herramientas permiten una interacción totalmente libre y desestructurada. Estas herramientas genéricas tampoco presentan una interfaz de usuario uniforme, lo que dificulta la utilización de distintas herramientas de forma complementaria.

Debido a la extensión actual de la investigación en CSCL y el gran número de líneas de investigación y áreas de conocimiento que intervienen, nos centraremos a continuación en el estudio de las herramientas para la construcción de aplicaciones o entornos colaborativos de enseñanza.

2.3.1 Herramientas para el desarrollo de aplicaciones colaborativas

Como hemos visto en el apartado 2.2, el desarrollo de aplicaciones colaborativas es muy complejo, requiere programar interfaces multiusuario, surgen problemas de gestión y comunicación de procesos distribuidos, es necesario crear sistemas de control de los permisos y seguridad en red, mantener y controlar el acceso o modificación de los datos compartidos, gestionar las sesiones de trabajo, etc. Esta sobrecarga de trabajo hace necesaria la utilización de herramientas que simplifiquen el desarrollo de estas aplicaciones.

Podemos distinguir tres niveles de abstracción proporcionados por este tipo de herramientas:

1. Bajo nivel: En este nivel nos encontramos con herramientas básicas de ayuda a la hora de crear aplicaciones distribuidas, como Corba[15] o RMI[16], que evitan la necesidad de tener que programar todos los aspectos de las

[15] CORBA es un estándar del OMG que define los protocolos y modelos para permitir la interacción entre aplicaciones remotas, independientemente de la plataforma. http://www.omg.org/technology/documents/formal/corba_iiop.htm

[16] RMI es el método de invocación remota estándar de JAVA

comunicaciones, o JSDT[17] para simplificar la compartición de datos. Sin embargo en éste nivel no encontramos soluciones a los problemas específicos de las aplicaciones CSCW o CSCL.

2. Medio nivel: En este nivel tenemos los toolkits y frameworks específicos de aplicaciones CSCW, como Colabrary, [Boyle 2002], o COAST, [Schuckmann 1999], aunque también podemos encontrar toolkits que tratan aspectos específicos de CSCL.
3. Alto nivel: En el nivel más alto podemos situar las herramientas orientadas a los creadores de contenido. Estas herramientas permiten crear recursos CSCL fácilmente, reutilizando objetos educativos y creando un entorno de aprendizaje adaptado a una experiencia concreta. Cuando es necesario crear nuevos objetos educativos, porque no se puede conseguir el efecto deseado con una combinación de los existentes, suele ser necesario recurrir a programadores con experiencia en la herramienta de autor para crear dichos objetos nuevos. Ejemplos de estas herramientas los tenemos en Active Document System, [Verdejo 2002], o FLE3, [Chen 2003].

Las herramientas que más nos interesan en este estudio son las de medio/alto nivel. Comenzaremos viendo los toolkits más conocidos para la creación de aplicaciones CSCW y CSCL, y en el siguiente apartado veremos las aplicaciones de alto nivel más destacadas para la creación de contenido.

En los toolkits para el desarrollo de aplicaciones CSCW podemos distinguir cuatro componentes básicos, [Greenberg 1999]:

1. Arquitectura de ejecución: Este componente se encarga de la creación, interconexión y comunicación entre los procesos centralizados y/o distribuidos.
2. Abstracción de la programación groupware: Estos componentes nos abstraen de los problemas más corrientes en el desarrollo de groupware, como son la sincronización de eventos de interacción con el modelo de datos compartido y con las diferentes vistas presentadas a los distintos usuarios.

[17] JSDT es una biblioteca JAVA para simplificar la compartición de datos en las aplicaciones. Está disponible en http://java.sun.com/products/java-media/jsdt

3. Componentes reutilizables de interfaz de usuario: Mediante la incorporación de estos componentes visuales en las interfaces podemos construir interfaces colaborativas de forma sencilla.
4. Gestores de sesión: Estos componentes permiten a los usuarios finales crear, unirse, o abandonar las sesiones de colaboración o reuniones virtuales.

En función de cómo sean estos componentes el sistema ofrecerá una funcionalidad u otra. Analizando cómo son estos componentes podemos obtener la siguiente funcionalidad:

1. Según la arquitectura de ejecución:
   a) La arquitectura puede soportar el trabajo síncrono, asíncrono, o ambos.
   b) Las arquitecturas pueden ser centralizadas, distribuidas, o semidistribuidas. En el caso de las arquitecturas centralizadas el estado está en una sola máquina. Éste es el caso de sistemas como netmeeting, vnc, rdesktop o similares. La mayoría de los toolkits entran en la categoría de arquitectura semidistribuida, dejando centralizado el estado imprescindible, debido sobre todo al coste en complejidad que supone un sistema totalmente distribuido y a la dificultad de gestión que supone.
   c) La arquitectura ha de soportar algún tipo de control de la concurrencia, aunque en el caso totalmente centralizado esto no suele ser necesario. Según como sea este control podemos tener una granularidad de bloqueo y latencia distinta. La arquitectura también puede usar un control de concurrencia social, donde son los propios individuos los que se coordinan para realizar las acciones de forma ordenada.
2. Según la abstracción de la programación groupware:
   a) La abstracción groupware puede ofrecer distintos grados de separación entre el modelo de datos y las vistas, lo cual nos lleva a diferentes arquitecturas de compartición de la información, [Patterson 1995]. Podemos encontrarnos con sistemas que no ofrecen mecanismos para compartir los datos, sistemas que mantienen un modelo de datos compartido y sistemas con vistas compartidas, los cuales mantienen sincronizados tanto las vistas como el modelo de datos.

b) Para mantener la información compartida la abstracción de programación nos proporciona diferentes mecanismos, como las llamadas remotas multiproceso, eventos o la sincronización de modelos de datos o vistas de forma automática.

3. Según los componentes de la interfaz de usuario:

   a) El sistema puede proporcionar equivalentes groupware de los componentes de interfaz de usuario tradicionales. De esta forma podemos construir interfaces multi-usuario de forma sencilla, sin tener que programar los aspectos groupware. Sin embargo aparecen algunos problemas nuevos que no tenían sentido en los componentes mono-usuario, como son el control de acceso, quién puede realizar qué acción sobre qué componentes, o cómo el nivel de acoplamiento entre los distintos usuarios, es decir, si los usuarios ven lo mismo en todo momento, o sólo ven los cambios importantes.

   b) También podemos tener componentes de interfaz propios del groupware, por ejemplo para proporcionar cierto grado de consciencia sobre el resto de usuarios, para gestionar los permisos de cada usuario, las sesiones, formar grupos, etc.

4. Según la capacidad de gestión de las sesiones, el sistema de gestión puede permitir gestionar conferencias, permitir unirse a conferencias ya iniciadas, almacenar de forma persistente el estado de las conferencias, gestionar el control de acceso y las políticas, realizar búsqueda de conferencias, etc.

Una de las características de la mayoría de los frameworks para trabajo colaborativo síncrono es que la interfaz de usuario final es una aplicación, y no el navegador Web. En los sistemas CSCW asíncronos ocurre al contrario, ya que la Web se adapta perfectamente a estos sistemas, como veremos en el apartado 2.3.2. Esto es debido a que la Web no está diseñada para ofrecer un contenido muy dinámico, el cliente no sabe cuando ha cambiado el contenido y el servidor no conoce que es lo que está viendo el cliente. De esta forma la Web se limita a ser un medio para acceder a la aplicación, descargándola del servidor Web, o ejecutando la aplicación embebida en el navegador. También se pueden crear aplicaciones Web con mayores

características síncronas mediante la incorporación de Applets en las páginas que comuniquen los eventos en tiempo real con el servidor.

Recientemente se han llevado a cabo algunos esfuerzos para eliminar esta limitación de la Web. Por un lado tecnologías como Java Web Start[18] permiten distribuir aplicaciones java por la Web con una instalación y actualización transparente, consiguiendo una facilidad de distribución cercana a la conseguida por las aplicaciones Web. Por otro lado la aparición de tecnologías como DHTML[19], JavaScript[20] y AJAX, [Garrett 2005], soportadas por la mayoría de los navegadores, permite construir aplicaciones Web con un mayor dinamismo y con menores tiempos de respuesta, requisito imprescindible para la construcción de aplicaciones síncronas.

Acabamos de ver las características generales que nos podemos encontrar en la mayoría de los frameworks o toolkits para el desarrollo de groupware. A continuación veremos las características más destacadas de los principales toolkits o frameworks existentes en investigación. Todos los frameworks seleccionados presentan grandes parecidos, permiten la construcción de aplicaciones síncronas, presentan un modelo semidistribuido o distribuido, permiten compartir un modelo de datos, sin necesidad de compartir toda la interfaz de usuario, y están orientados a desarrolladores.

2.3.1.1 GroupKit

GroupKit[21] es una biblioteca Tck/Tk para crear prototipos y construir aplicaciones colaborativas síncronas. Actualmente se ofrece mediante licencia de código abierto, y puede funcionar en los principales sistemas operativos. Esta biblioteca fue creada en 1992 por Mark Roseman en la universidad de Calgary y fue una de las primeras bibliotecas que permitían incorporar procesos distribuidos y datos replicados [Roseman 1992]. El

[18] Para más información sobre Java Web Start se puede consultar http://java.sun.com/products/javawebstart

[19] DHTML no es realmente una tecnología sino un método para la creación o modificación de HTML dinámicamente en el cliente

[20] JavaScript es un lenguaje interpretado creado por Netscape y diseñado para ejecutarse en los navegadores web. Este lenguaje es una extensión del estándar ECMAScript. Para más información se puede consultar: http://www.mozilla.org/js

[21] GroupKit se encuentra disponible en http://www.groupkit.org

desarrollo de GroupKit parece estar detenido, ya que en los dos últimos años no ha aparecido ninguna versión nueva, desde la salida de la versión 5 en el 2003.

La mayor parte de la arquitectura es distribuida. Únicamente es necesario tener centralizado el proceso de registro, el cual permite saber a cada proceso cómo acceder al resto de procesos distribuidos. El control de la concurrencia es sencillo, permitiendo elegir entre sincronizar las vistas o el modelo de datos.

2.3.1.2 Collabrary

Collabrary[22] es de los mismos autores que GroupKit y surge de la necesidad de soportar aplicaciones con gran demanda multimedia, ya que GroupKit no se adaptaba bien a este tipo de aplicaciones [Boyle 2002].

Básicamente se trata de una biblioteca de componentes COM para el prototipado rápido de aplicaciones multimedia colaborativas. El principal componente es el diccionario compartido, que permite mantener un estado de la aplicación compartido, a la vez que simplifica la conexión de los datos con la interfaz de usuarios.

Los diccionarios compartidos asocian pares clave-valor. Además las claves siguen un modelo jerárquico, como en un sistema de archivos. Estos diccionarios notifican a la aplicación cuándo se produce algún cambio, lo que permite que el modelo de datos se comparta y que las interfaces de usuario reaccionen ante estos cambios. Los valores almacenados en estos diccionarios pueden ser números, cadenas de caracteres, tipos agregados, como vectores o registros, y tipos complejos como datos de audio, vídeo, o imágenes JPEG.

Una de las mejoras sobre GroupKit, para permitir aplicaciones multimedia, es el uso de señales. Las señales se utilizan en datos que sólo tienen sentido temporalmente, como por ejemplo un vídeo en directo. Collabrary soporta el envío del valor de una clave cada vez que cambia, como un flujo de información, sin importar si se pierden algunos valores, ya que el valor cambia rápidamente.

[22] Collabrary está disponible en http://grouplab.cpsc.ucalgary.ca/collabrary/

2.3.1.3 Clock

Clock es un lenguaje de alto nivel pensado para el prototipado de interfaces gráficas, y en particular para el desarrollo de interfaces multiusuario y groupware. Este lenguaje separa el modelo de las vistas y permite replicar ciertos componentes de la aplicación, es decir, es un modelo semidistribuido, aunque también permite usar el modelo centralizado. Una de las características más destacadas de este lenguaje es que los componentes se programan de forma declarativa, basándose en el lenguaje funcional Haskell [Hudak 2000]. La herramienta ClockWorks, [Graham 1996], permite desarrollar de forma visual la arquitectura abstracta de las aplicaciones escritas en Clock, usando un modelo basado en componentes que se distribuyen jerárquicamente, consiguiendo de esta forma simplificar el desarrollo de las aplicaciones escritas en Clock.

2.3.1.4 Habanero

Habanero es un framework escrito en Java que permite el trabajo colaborativo síncrono y asíncrono, de forma que se pueda compartir información entre uno y otro modo de trabajo. Las sesiones síncronas son grabadas en un documento que permite ser examinado de forma automática y reutilizables en otras comunicaciones síncronas o asíncronas [Jackson 1999].

Las aplicaciones escritas en Habanero comparten el modelo de datos, permitiendo así tener diferentes vistas sobre el mismo modelo. El modelo se mantiene sincronizado mediante la distribución de los eventos que ocasionan cambios en el modelo de datos. Estos eventos se han de replicar en el mismo orden en todas las instancias para que el estado sea el mismo en todas las réplicas.

Habanero dispone de un mago que permite convertir el código de aplicaciones java existentes, de forma que sean compatibles con el framework y puedan ser compartidas.

2.3.1.5 Disciple

Disciple es un framework que permite compartir aplicaciones basadas en JavaBeans en tiempo real. Las aplicaciones no tienen por qué diseñarse para

ser compartidas, de hecho Disciple permite compartir aplicaciones Java mono-usuario de forma transparente [Li 1999].

El requisito para compartir las aplicaciones es que la aplicación o el applet cumplan la especificación JavaBeans, lo que se cumple en la mayoría de las aplicaciones Java recientes. La colaboración se produce compartiendo de forma síncrona una serie de componentes JavaBeans entre un grupo de usuarios.

Si el componente JavaBean no está diseñado para la colaboración con Disciple, éste es compartido de forma transparente mediante la interposición de una ventana transparente entre la interfaz de usuario original y el usuario. Esta ventana captura todos los eventos de ratón y teclado y los envía al resto de usuarios. De esta forma las interfaces de usuarios se mantienen sincronizadas, ya que se realizan sobre ellas las mismas acciones.

Para permitir el uso de recursos compartidos, Disciple introduce el concepto de servidores de recursos. De esta forma todos los usuarios tienen acceso a los mismos recursos. El problema más complejo ocurre con las aplicaciones compartidas que no eran colaborativas. En este caso Disciple realiza modificaciones en el código binario de las aplicaciones, o Java Byte-Code, para que la aplicación crea que está trabajando en el servidor de recursos.

2.3.1.6 Coast

COAST[23] es un toolkit para el desarrollo de groupware síncrono escrito en Smalltalk. Algunos aspectos novedosos de este toolkit son el acceso a los objetos replicados mediante transacciones y un control de concurrencia totalmente optimista [Schuckmann 1999]. Como hemos visto anteriormente, apartado 2.2.2.1.1, el control optimista aumenta la usabilidad incluso en entornos con una latencia alta, siempre que no existan demasiados fallos en el control de la concurrencia.

El modelo de programación utilizado es el Model-View-Cotroller, mejorado con un sistema de control de dependencias, que permite mantener las vistas y el modelo actualizado. Este modelo de datos, o documento, se mantiene

[23] COAST está disponible de forma gratuita en http://www.opencoast.org/

replicado, siendo una de las réplicas la responsable de su almacenamiento de forma persistente.

2.3.1.7 AORTA y ANTS

AORTA es una arquitectura software que provee y mejora la coordinación a nivel de objeto y capacidades de consciencia de las aplicaciones colaborativas [Orozco 2004]. Actualmente se puede utilizar en aplicaciones escritas para el framework ANTS[24], [Lopez 2003], extendiendo de este modo la funcionalidad de este framework, aunque en teoría puede generalizarse para otros frameworks. Esta arquitectura se basa en coordinar el acceso a los objetos, definiendo políticas que deciden qué acciones se pueden llevar a cabo y cuáles no, dependiendo del objeto, el usuario, el tipo de acción y el tiempo. Del mismo modo que ocurre con las políticas de coordinación, también existen políticas de consciencia, que determinan qué acciones se han de notificar y a qué usuarios.

Por otra parte ANTS se basa en compartir las propiedades de objetos JavaBean, permitiendo suscribirse a eventos de cambio de dichas propiedades. La coordinación se realiza mediante un mecanismo similar al de propiedades con restricciones de Java. Además el sistema ofrece gran flexibilidad, permitiendo incorporar nuevos objetos JavaBeans y adaptarse a distintos servicios de transporte como multicast, Secure Socket Layer, etc.

2.3.1.8 Active Document System

Active Document System, [Verdejo 2002], es un entorno para el diseño y desarrollo de actividades de aprendizaje colaborativo. La especificación de las actividades se basa en la teoría de la actividad, ver apartado 2.1.1.

Este sistema tiene tres componentes principales. Dispone de una herramienta de autor para modelar las actividades de aprendizaje. Con este componente se crean los “documentos activos”, que definen tres aspectos fundamentales: en primer lugar el objeto de la actividad, en segundo lugar las las tareas, subtareas, y roles, y por último los recursos y herramientas que estarán disponibles para llevar a cabo la actividad.

[24] ANTS está disponible de forma gratuita en http://ants.etse.urv.es/

El segundo componente del sistema es el repositorio de objetos distribuido, de donde se obtienen los distintos recursos y herramientas disponibles.

Por último, el sistema dispone de una arquitectura que gestiona los documentos activos, y genera el entorno de usuario para llevar a cabo las actividades, junto con las herramientas y recursos definidos previamente por el diseñador de dichas actividades.

2.3.2 Sistemas Groupware orientados a usuarios

Actualmente existe un gran número de sistemas groupware, basados en componentes genéricos, que soportan el aprendizaje colaborativo en un gran número de ámbitos. La mayoría de estos sistemas son aplicaciones Web y su funcionalidad principal es la de compartir y permitir la construcción de conocimiento sobre un tema específico. También es habitual incorporar foros de discusión y otras herramientas de colaboración síncrona, como Microsoft Messenger[25], que permite chats, vídeoconferencia, pizarras compartidas y mensajería instantánea.

También existen algunos estándares, como los definidos por las organizaciones IMS[26] o ADL[27], creadores de SCORM[28], que permiten definir objetos didácticos de forma estándar, facilitando la incorporación de nuevos componentes didácticos y la interoperatividad entre sistemas. Estos estándares modelan como son los objetos didácticos, los repositorios de objetos, las secuencias de tareas y los estudiantes a los que va dirigido cada objeto. Las experiencias de aprendizaje colaborativo se pueden modelar definiendo grupos de estudiantes, en lugar de estudiantes individuales, aunque este mecanismo no es lo suficientemente flexible en el caso de los objetos didácticos colaborativos. Para solucionar estas limitaciones han

[25] Messenger es una aplicación de mensajería instantánea y comunicación multimedia incluida en las últimas versiones del sistema operativo Windows de Microsoft

[26] La organización IMS Global Learning Consortium es una organización sin ánimo de lucro para la definición de estándares que faciliten la interoperatividad de las distintas aplicaciones de aprendizaje. La web principal es: http://www.imsglobal.org

[27] Advanced Distributed Learning, ADL, es una iniciativa gubernamental de EE.UU., para la mejora y modernización del aprendizaje. La web principal es: http://www.adlnet.gov

[28] SCORM es un conjunto de estándares, especificado por ADL, para la definición de e-learning basado en web

surgido propuestas de mejora de algunos de estos estándares, como [Ip 2003] o [Hernández 2004].

Aunque la mayoría de los entornos para la colaboración asíncrona se basan en interfaces Web que usan el protocolo http, la aparición de WebDAV[29] permite desarrollar sistemas asíncronos basados en documentos sin necesidad de crear aplicaciones en el servidor para gestionar el contenido. WebDAV permite crear, o gestionar meta información sobre los documentos, agruparlos en colecciones, crear bloqueos para permitir la modificación concurrente de los documentos, etc. Este protocolo es soportado además por un gran número de aplicaciones de usuario, sistemas operativos, y los principales servidores web.

2.3.2.1 FLE3

FLE3 es un entorno asíncrono basando en Web donde el alumno aprende sobre un tema investigando sobre él, generando su propio problema, creando hipótesis, y buscando información de forma colaborativa [Muukkonen 1999]. El entorno ofrece un guía al alumno para que pueda seguir las distintas fases del proceso de investigación. Una serie de agentes inteligentes ofrecen información estadística sobre la colaboración y uso del entorno, detectando posibles problemas para avisar a los estudiantes o al profesor. Los agentes también son utilizados para fomentar la participación, avisando a los usuarios que han participado menos en las discusiones, [Chen 2003].

2.3.2.2 BSCW

BSCW[30] es una aplicación Web que facilita la colaboración mediante espacios de trabajo compartidos, [Appelt 1996]. El espacio de trabajo distribuye una serie de recursos de forma jerárquica. Estos recursos pueden ser documentos, imágenes, direcciones Web, foros, información de contacto, etc.

El sistema también permite acceder a los espacios de trabajo y los documentos mediante WebDAV, simplificando la edición de los documentos,

[29] WebDav es una extensión del IETF al protocolo http para permitir que se pueden leer y escribir objetos en la web. Para más información consultar http://webdav.org

[30] BSCW se puede usar gratuitamente en http://bscw.fit.fraunhofer.de/index.html

mantener varias versiones de un objeto, gestionar el control de acceso a los recursos e incluso crear flujos de trabajo sencillos. Existen dos tipos de flujos de trabajo, los basados en una carpeta con recursos que se va pasando de un miembro a otro, creando un flujo de trabajo lineal. El otro tipo de flujo de trabajo se basa en el concepto de tareas, que pueden verse como los pasos del flujo de trabajo.

Para facilitar la colaboración el sistema registra los eventos producidos sobre el espacio compartido. De esta forma, cuando alguien accede al espacio compartido, puede conocer los últimos cambios que han ocurrido desde la última vez que se conectó.

Esta aplicación también permite organizar la colaboración síncrona, usando objetos de tipo reunión, que incluye información sobre cuándo, dónde y quiénes se reunirán. La conferencia o reunión se realiza con otros programas, o por teléfono.

2.3.2.3 PHProjekt

PHProjekt[31] es una alternativa de software libre a BSCW, con una funcionalidad similar, aunque no permite definir flujos de trabajo ni acceder por WebDAV a los objetos. El trabajo con PHProjekt se centra entorno a la definición de grupos y proyectos.

2.3.2.4 OpenGroupware

OpenGroupware[32] es un entorno colaborativo gratuito que permite compartir calendarios, documentos, proyectos, correo, contactos y noticias. Este sistema destaca, además de por los servicios que ofrece, por la variedad de clientes que pueden usar la aplicación, ya que ofrece interfaces para el navegador Web, Microsoft Outlook, Ximian Evolution, clientes WebDAV, etc., y por sus prestaciones, rivalizando con soluciones comerciales como Microsoft Exchange o Novel GroupWise.

[31] PHProjekt es disponible de forma gratuita en http://phprojekt.com

[32] OpenGroupware está disponible de forma gratuita en http://www.opengroupware.org/

2.3.2.5 KnowCat

KnowCat, [Cobos 2002], es un sistema web que permite de gestionar, compartir y valorar el conocimiento de una comunidad. Este sistema permite crear bibliotecas o libros electrónicos que se gestionan automáticamente, sin necesidad de supervisión.

Para ello el sistema modela una comunidad de expertos que puede dar su opinión acerca de la calidad del contenido de los distintos documentos que aportan los usuarios. Estos votos sirven de entrada a un proceso de cristalización del conocimiento, que con el tiempo selecciona los documentos de mayor calidad o más relevantes para la comunidad.

2.3.2.6 Lotus Notes y Lotus SameTime

Lotus Notes es una aplicación comercial para el trabajo en grupo que ofrece una gran flexibilidad. Notes se basa en el concepto de base de datos documental compartida, donde cada base de datos se compone de documentos y otros objetos especiales.

Por otro lado, Lotus SameTime añade a Notes aspectos de colaboración síncrona, como la mensajería instantánea, vídeo-conferencias, y las pizarras compartidas.

Una de las características más potentes de Lotus Notes es que permite crear aplicaciones documentales de forma muy sencilla, creando formularios para introducir datos, vistas de resumen, agentes que realizan tareas automáticas con los documentos, o incluso se pueden programar nuevas funciones, usando distintos lenguajes. Aplicaciones como el correo electrónico o los flujos de trabajo se pueden crear desde cero utilizando bases de datos y agentes, aunque el sistema ya implementa un gran número de aplicaciones, en forma de tipos de bases de datos.

2.4 Programación interactiva de interfaces de usuario y tutores inteligentes

Las herramientas que hemos visto en los apartados anteriores permiten construir entornos colaborativos aplicables a la enseñanza a distancia, pero no resuelven el problema de la creación del contenido didáctico. Este

contenido didáctico se enfrenta a un importante reto, sustituir en la medida de lo posible las consultas al profesor por un tutor inteligente integrado en la propia aplicación. De esta forma el alumno consigue una mayor flexibilidad temporal en su aprendizaje, que viene a unirse a la flexibilidad espacial conseguida con las aplicaciones CSCL, y no se ve retrasado por las dudas que le pueden surgir y que pueden resolverse mediante los tutores inteligentes integrados en la aplicación.

Estos tutores inteligentes se pueden integrar en la aplicación mediante agentes inteligentes, que muestran ayuda cuando es necesario, o pueden integrarse en el propio flujo de diálogos de la aplicación. En cualquiera de los dos casos la tarea de crear un buen sistema tutor inteligente es muy costosa, y suele requerir conocimientos avanzados de programación.

Uno de los aspectos más complejos es la especificación de la interacción en la interfaz de usuario. Debido a esto se ha extendido la utilización de herramientas que automatizan gran parte del proceso, aunque la mayoría de estas herramientas están orientadas a programadores.

De estas herramientas las más utilizadas actualmente por los programadores son los constructores de interfaces orientados a componentes. Gracias a esto los programadores pueden especificar de forma sencilla la disposición de los componentes que componen la interfaz de usuario, los valores iniciales de dichos componentes, o relaciones sencillas entre las propiedades de los componentes. El problema es que estas herramientas no permiten especificar, sin programación, el comportamiento interactivo de los componentes, ni relaciones complejas entre las propiedades de dichos componentes.

También existen herramientas y técnicas para especificar la interacción con el usuario que no requieren grandes conocimientos de programación, lo que permite a los expertos en el contenido didáctico realizar la mayor parte del proceso. Por un lado, la programación por demostración, PBD en sus siglas en inglés, permite definir los aspectos de interacción de la interfaz de usuario mediante la manipulación directa con los objetos que componen la interfaz De esta forma el diseñador demuestra cómo se tienen que comportar los objetos ante las distintas acciones del usuario. Uno de los primeros

sistemas para la construcción de interfaces de usuario mediante la programación por demostración fue Peridot, [Myers 1990].

Otra técnica que ayuda a la creación de estas interfaces es la utilización de un sistema de dependencias entre las propiedades de los objetos de la interfaz. Por ejemplo, Amulet [Myers 1997], es una herramienta para crear interfaces de usuario en C++ que incluye una serie de widgets y permite restricciones o dependencias entre los objetos, de forma que cuando cambia el valor en las propiedades de un objeto, este valor se propaga automáticamente para modificar las propiedades de otros objetos. Esta técnica se utiliza de forma complementaria a la programación por demostración en algunos sistemas, como por ejemplo HandsOn, [Castells 1999], o Peridot, donde el desarrollador especifica las dependencias entre los componentes de la interfaz de usuario mediante técnicas de programación por demostración.

2.4.1 Programación por demostración

La programación por demostración, o programación mediante ejemplos, se basa en enseñar a un programa cómo realizar una acción, realizándola primero el programador mediante la manipulación directa de los objetos involucrados en dicha acción. De esta forma el programador no requiere aprender ningún lenguaje complejo de especificación, y puede realizar la programación de forma visual.

Otra de las ventajas de la programación por demostración es que, para un gran número de aplicaciones, se requiere menos tiempo de programación con esta técnica que el requerido cuando se ha de utilizar un lenguaje de programación. Según los datos de algunos experimentos en la creación de tutores inteligentes, [Koedinger 2004], se observa una relación entre 1:6 y 1:12 entre los tiempos de desarrollo de varios tutores inteligentes con programación por demostración y programación con un lenguaje de programación.

Un ejemplo básico de este tipo de programación es la creación de macros en los editores de texto avanzados, donde el usuario es capaz de programar pequeñas funciones para realizar tareas repetitivas, grabando primero los pasos para realizar dicha tarea.

El problema principal de esta técnica, en apariencia sencilla, desde el punto de vista del usuario, es la capacidad de generalización necesaria para que las tareas programadas funcionen correctamente cuando cambie el contexto.

Para resolver el problema de la generalización se pueden utilizar diversas técnicas:

- Utilización de inteligencia artificial: Mediante técnicas de inteligencia artificial se pueden crear entornos de desarrollo que generalicen automáticamente los ejemplos dados por el usuario. De esta forma el sistema busca el modelo más general que englobe a todos los ejemplos suministrados por el usuario. Para mejorar la búsqueda del modelo es importante que se puedan suministrar tanto ejemplos positivos como negativos. Esto se puede realizar de forma incremental, hasta que el usuario crea que el modelo final es el que busca.

  El problema de la utilización de inteligencia artificial es que el usuario puede perder con facilidad la sensación de tener control sobre el resultado, sobre todo si la trasformación para pasar de los ejemplos al modelo abstracto es compleja, o el modelo obtenido es difícil de entender [Myers 2001].

- Utilización de heurísticas: Éste es el método más utilizado en este tipo de sistemas, ya que sin demasiada complejidad se pueden lograr resultados similares a los obtenidos mediante inteligencia artificial, de forma mucho más eficiente. El problema es que las heurísticas requieren un contexto específico para poder aplicarse, lo que requiere programar heurísticas para los casos más habituales.

- Uso de una generalización asistida por el usuario mediante la eliminación del contexto. Esta técnica se utiliza por ejemplo en ToonTalk, un sistema para enseñar programación a niños, y que permite que el usuario borre el contexto que no debe influir en la generalización. ToonTalk se verá en el apartado 2.4.1.1.

Otro de los problemas que aparecen en los sistemas de programación por demostración es la modificación del programa que se acaba de crear. En los lenguajes de programación esta tarea es relativamente sencilla, al menos si la modificación la realiza un programador que conozca la implementación original. Simplemente es necesario tener disponible el código fuente original, modificarlo y volver a compilar el programa para comprobar que funciona. En el caso de la programación por demostración esto no es trivial. Existen varias alternativas para solucionar este problema, que dependen en parte del método de generalización utilizado. Algunas soluciones a este problema son:

- Mayor modularidad: Una forma sencilla de resolver este problema es dividirlo en problemas más pequeños. Esto no resuelve todo el problema, pero en la mayoría de los casos es suficiente con dividir la tarea de programación por demostración en tareas más pequeñas. De esta forma no será necesario repetir toda la programación para modificar algo, y puede ser suficiente con repetir la programación de las tareas que se desea modificar. En los sistemas basados en ejemplos se puede modificar el modelo eliminando o creando nuevos ejemplos. En este caso el nivel de granularidad está en los ejemplos, en lugar de en las tareas.
- Modelo de programación modificable: Los sistemas de programación por demostración convierten las acciones que interpretan del usuario en un modelo semántico. Este modelo suele ser sencillo de modificar, de forma análoga a la modificación del código fuente de un programa. El problema es que requiere disponer de conocimientos de programación y comprender el lenguaje usado en el modelo. Un ejemplo de está técnica se da en las macros de Microsoft Office, que son trasformadas al lenguaje de Visual Basic.

  En otros sistemas la trasformación a un lenguaje de programación sólo es necesaria para modificaciones complejas, no soportadas de forma visual. El problema en este caso es que una vez modificado en el lenguaje de programación no se pueden realizar más modificaciones mediante ejemplo. Un ejemplo de este caso lo encontramos en los entornos que soportan definir consultas a bases de datos mediante ejemplo, QBE[33]. En

[33] El sistema QBE, aunque su nombre indica que se basa en ejemplos, no es considerado por algunos autores como programación por demostración/ejemplos, aunque si es un

ese caso se pueden realizar consultas sencillas, pero para las modificaciones más complejas es necesario pasar al lenguaje SQL, y después no se puede volver al modo basado en ejemplos.

- Demostración modificable de forma visual: Para el caso de demostraciones complejas, en el que no interesa volver a repetir completamente la programación por demostración, es importante poder reutilizar parte de la demostración para evitar tener que repetir todo el proceso. Este método permite realizar las modificaciones de forma sencilla, en el mismo lenguaje y contexto en el que el usuario describe el ejemplo o demostración original.

  Este método presenta algunas dificultades técnicas difíciles de resolver que se detallan a continuación, lo que provoca que sólo se aplique en contextos muy concretos, donde su programación resulta sencilla. En general, sólo lo encontramos en sistemas donde el elemento a modificar de la demostración no incluye relación temporal con las acciones.

  El problema principal es que requiere poder reutilizar las acciones del usuario en un contexto distinto del original, ya que la modificación que realiza el usuario en una acción intermedia puede alterar el contexto de las acciones posteriores a la modificación. Este problema es similar al que ocurre en los sistemas colaborativos síncronos, cuando dos usuarios realizan distintas acciones sobre el mismo contexto, y el resultado de estas acciones depende del orden de realización de las mismas.

A continuación veremos algunos ejemplos de sistemas que soportan la programación por demostración.

2.4.1.1 ToonTalk

ToonTalk[34] es un software educativo que permite a los estudiantes crear sus propios programas mediante técnicas de programación por demostración. En ToonTalk los conceptos de programación se trasladan a conceptos de un mundo virtual donde los robots manipulan los datos, lo cual representa una función, construyen casas donde trabajan, es decir, se crea

método de programación visual, como muchos de los sistemas de programación por demostración.

[34] Más información sobre Toontalk en http://www.toontalk.com/

Figura 3: Interior de una casa en ToonTalk

un subproceso, o se envían cosas mediante pájaros que vuelan a sus nidos, Figura 3.

El usuario crea un programa entrenando a robots, es decir, creando métodos, que actúan sobre los datos. Este entrenamiento se realiza manipulando los datos de ejemplo realizando operaciones sencillas sobre ellos. De esta forma se crea un robot para cada tarea de programación, que se ejecutará cuando se de el contexto para el que se entrenó el robot. En ToonTalk la generalización se consigue eliminando detalles del contexto de ejecución, que se representa dibujando lo que está pensando el robot en un bocadillo de diálogo, para ello se utiliza la aspiradora, [Kahn 2001].

ToonTalk también soporta el trabajo colaborativo y las aplicaciones distribuidas con el concepto de vuelos largos de los pájaros, los cuales pueden llevar los datos a nidos alojados en otros ordenadores. Por ejemplo un niño puede darle lo que está haciendo a otro mediante un pájaro para que lo continúe.

Un ejemplo ilustrativo de que tipo de programas podemos realizar es la implementación de un algoritmo que encuentre números primos[35]. Para crear este ejemplo el usuario crea una casa con un robot dentro que se entrena para generar una serie de números consecutivos comenzando por el 2. Este robot le pasará los números a un pájaro que los lleva a su nido.

Para terminar de implementar el algoritmo se entrena al sistema para que cree una casa con un robot que filtre los números divisibles por el número

[35] El ejemplo de los números primos, junto a otros ejemplos y material educativo, está disponible de forma gratuita en la página Web de Toontalk

que acaba de llegar al final del proceso, en este caso el 2. De esta forma cada vez que un número alcance el final de la cadena de casas con robots, este número será primo, y se construirá una nueva casa para que los divisibles por ese número primo sean eliminados.

2.4.1.2 CTAT

CTAT, [Koedinger 2004], del inglés “Cognitive Tutor Authoring Tools”, es una herramienta para simplificar la creación de Pseudo Tutores inteligentes, mediante programación por demostración y diseño visual de interfaces de usuario basadas en componentes. Un Pseudo Tutor Inteligente es un sistema educacional que emula el comportamiento de un tutor inteligente pero sin usar inteligencia artificial. Los tutores creados permiten al alumno obtener ayuda y retroalimentación para la construcción del conocimiento, de forma sensible al contexto. Además ofrece flexibilidad para explorar distintas estrategias de solución durante el aprendizaje.

Para crear estos tutores no es necesario poseer grandes conocimientos de programación. La creación se divide en dos fases, en la primera se diseña el aspecto visual de las interfaces de usuario. Esta primera fase se puede realizar con una herramienta de programación visual para crear interfaces en Flash o en Java. Los componentes utilizados para la interfaz pueden ser los componentes disponibles normalmente en la plataforma, Flash o Java, o componentes especiales de CTAT, que permiten grabar la información sobre qué acciones se realizan sobre ellos.

En la segunda fase se diseña el comportamiento dinámico de la interfaz ante las acciones del usuario sobre los componentes de CTAT, que son los elementos de introducción de datos del formulario. Para diseñar este comportamiento el diseñador demuestra las distintas formas de solucionar un problema, o caminos de solución, interactuando con los componentes gráficos diseñados en la primera fase. Además de crear posibles caminos de solución, también crea posibles caminos incorrectos. Toda esta información es grabada por el “grabador de comportamiento”, para crear un grafo de comportamiento, con los distintos caminos grabados, llamado “Diagrama de Comportamiento”. Para completar el diseño del comportamiento dinámico el diseñador crea anotaciones en algunos nodos del “Diagrama de

Comportamiento", adjuntando mensajes de error a los nodos del diagrama que representan estados incorrectos, o donde el alumno puede requerir ayuda extra.

Las anotaciones sobre el grafo de comportamiento se pueden etiquetar y reutilizar. La reutilización de anotaciones ayuda a identificar las habilidades requeridas del estudiante para resolver un problema. Con esto se crea una matriz que indica para cada problema qué habilidades requiere.

La ventaja fundamental de este sistema es la disminución del coste de creación de los tutores inteligentes. El desarrollo de este tipo de tutores inteligentes, utilizando lenguajes de programación, tiene un coste de entre 100 y 1000 horas de desarrollo por cada hora de instrucción para el estudiante. Con este sistema el coste estimado, utilizando 4 proyectos diferentes, se reduce a entre 16 y 32 horas de desarrollo por hora de instrucción. También reduce el nivel de conocimiento de programación para desarrollar estos sistemas.

El "Grabador de Comportamiento" se puede reutilizar con otras aplicaciones para crear el sistema tutor. Se puede conectar una aplicación colaborativa a este componente, de forma que los alumnos sean quienes creen la mayor parte del grafo de comportamiento interactuando con la aplicación. Con un número suficiente de alumnos se consigue que exploren la mayor parte de los caminos, incluyendo los errores habituales y los caminos de solución correctos. Una vez conseguido el grafo inicial, el profesor sólo tiene que anotar los caminos correctos y los mensajes de ayuda, creando de esta forma un nuevo sistema tutor, [McLaren 2004].

2.4.1.3 Peridot, Gamut y Marquise

Estos tres sistemas, comenzando por Peridot, [Myers 1990], que fue una de las primeras aplicaciones basada en programación por demostración, representan cómo han ido evolucionando este tipo de sistemas.

Peridot permite crear interfaces de usuario genéricas, aunque está diseñado para el prototipado de aplicaciones. El diseño tiene lugar en dos fases: en la primera se dibuja la interfaz de usuario, y después se interactúa, moviendo el ratón y pulsando los botones, enseñando al sistema cómo se han de comportar los distintos elementos gráficos.

Peridot crea procedimientos parametrizados, de forma que parte de la interfaz depende de los valores de los parámetros, y permite dar valores de ejemplo para cada parámetro. Un ejemplo de esto son los menús, cuyo tamaño dependerá del número de cadenas que contiene.

Este sistema utiliza restricciones gráficas para ligar los objetos gráficos, de forma que cuando cambia uno los otros son actualizados, y restricciones de datos, que actualizan la parte gráfica cuando cambian los valores de variables especiales, llamadas “valores activos”.

El sistema infiere la abstracción de los ejemplos de interacción suministrados por el diseñador. Cada inferencia es notificada al diseñador para que vea el resultado y confirme que la inferencia es correcta. Estas inferencias pueden incluir iteraciones, cuando se usan dos componentes de una secuencia de datos de ejemplo asociadas a dos componentes gráficos. En este caso se infiere que es una iteración, y se crean automáticamente el resto de componentes gráficos asociados a la lista de ejemplo. De esta forma el sistema de restricciones relaciona los datos dinámicos con la interfaz de usuario. También se pueden crear objetos gráficos que se muestren de forma condicional, asociando una variable de control a su visibilidad.

Marquise, [Myers 1993], es el sucesor de Peridot, y está orientado a crear interfaces de usuario completas mediante demostración, en lugar de sólo el prototipado y el diseño de componentes, como ocurría con Peridot, aunque recoge gran parte de las ideas de éste.

En Marquise existen cuatro modos de trabajo distintos, que el diseñador distingue fácilmente mediante cambios en la forma del puntero del ratón. Estos modos son, modo construcción, ejecución, entrenamiento y modo de demostración.

En este sistema se ha mejorado el sistema de inferencia, y ya no es necesario pedir confirmación al diseñador de cada paso, sino que simplemente se muestra cada inferencia en una ventana aparte, para que el diseñador pueda conocer qué está infiriendo el sistema, eliminando las inferencias erróneas cuando sea necesario. También soporta restricciones que son mantenidas automáticamente por el sistema, al igual que ocurría con Peridot.

Por último, Gamut permite crear juegos y software educativo mediante demostración. Permite crear las reglas del juego, pero no permite crear oponentes, que requeriría especificar estrategias, [McDaniel 1999].

El sistema de inferencia está basado en el algoritmo utilizado en Marquise, mejorado al permitir realizar correcciones mediante demostración, ya que permite definir el comportamiento con ejemplos positivos y negativos. Los ejemplos negativos se dan diciendo al sistema cuándo un objeto se comporta de forma inesperada. También permite decir al sistema sobre qué objetos se centra la demostración que realiza el programador, ayudando de esta forma a que el sistema de inferencias conozca cuál es el contexto correcto de las acciones.

Otra de las mejoras son los objetos “fantasma”, que representan objetos que han dejado de existir o no son visibles, pero que pueden ser necesarios para el ejemplo de demostración actual, o simplemente para ver que ha cambiado en la interfaz.

En Gamut se ha conseguido poder realizar la programación de forma incremental, creando nuevas reglas, que se pueden probar y corregir sobre el contexto actual de la aplicación, sin tener que volver a un estado inicial.

2.4.1.4 HandsOn

HandsOn es un sistema para la programación de interfaces de usuario mediante manipulación directa de los componentes gráficos. Estos componentes pueden tener además restricciones que crean dependencias entre los valores de las propiedades de los distintos componentes, [Castells 1999].

El sistema se compone de:

- Un modelo de la parte visual que incorpora la funcionalidad dinámica.
- Un lenguaje visual de programación mediante ejemplo.
- Un módulo para crear el modelo de datos y los datos de ejemplo. Los modelos de datos pueden ser complejos, incluyendo tipos de datos compuestos, listas, grafos, etc.

- Un sistema de diseño gráfico de alto nivel que permite disponer los objetos gráficos con disposiciones visuales complejas.

En HandsOn la creación de las restricciones se realiza de forma visual, creando funciones lineales, fijando máximos, mínimos, etc. Estas restricciones ligan los datos de ejemplo con las propiedades de los objetos de la interfaz Puesto que el modelo de datos permite definiciones de datos recursivas, como las listas, o los grafos, el sistema crea de forma automática los componentes visuales asociados a cada objeto del modelo de datos. De esta forma se pueden crear de forma automática listas, o grafos de componentes gráficos, cuya relación visual, de posición relativa, anchura, etc., dependerá de los datos.

2.4.2 Programación interactiva mediante restricciones

En el apartado anterior hemos visto cómo se pueden utilizar técnicas de programación por demostración para la creación de aplicaciones. La mayoría de estos sistemas se ayudan de un sistema de dependencias, o de restricciones, para modelizar gran parte de la interfaz de usuario.

Los sistemas de restricciones también se pueden utilizar sin la programación por demostración. En este caso el diseñador especifica directamente el modelo abstracto, en lugar de dejar que el sistema lo infiera de los ejemplos, como ocurría en la programación por demostración.

En este apartado nos centraremos en sistemas que destacan por el sistema de restricciones utilizado, algunos de los cuales también utilizan técnicas de programación por demostración.

El sistema de restricciones se utiliza para el mantenimiento de la consistencia de la interfaz, modelizar la interacción con el usuario, para crear plantillas de interacción, definiciones de problemas en sistemas de enseñanza, etc.

2.4.2.1 SQL-Tutor

SQL-Tutor es un tutor inteligente para la enseñanza del lenguaje de bases de datos SQL, [Martin 2002]. El proceso general de aprendizaje consiste en la realización de una serie de preguntas al estudiante, que son elegidas por

el sistema de forma automática de acuerdo con su nivel actual de conocimiento.

El modelado del conocimiento del sistema se basa en un conjunto de restricciones del tipo, si <condición de activación> entonces <condición que se ha de satisfacer>. La primera condición indica cuándo se aplicará esta restricción. Si la restricción se puede aplicar pero no se cumple la segunda condición, entonces el sistema deduce que el alumno ha cometido un fallo. Cada una de estas restricciones está asociada a un concepto a estudiar del lenguaje SQL. Por lo tanto se puede conocer qué conceptos necesita repasar el estudiante, comprobando qué restricciones no se satisfacen.

Por otro lado el sistema dispone de un conjunto de problemas, con su respectiva solución ideal, en forma de sentencia SQL. Para elegir el problema que se presentará al estudiante se busca uno que permita practicar el concepto que más fallos está provocando. Para ello se utiliza la condición de activación de las restricciones.

La dificultad para que este algoritmo de entrenamiento funcione se centra en disponer de un número suficientemente alto de problemas, que permita practicar los diferentes conceptos del área de conocimiento, y que además se adapten a distintos niveles de dificultad.

Para superar esta dificultad se ha desarrollado un generador automático de problemas, que partiendo del conjunto de restricciones genera soluciones ideales. Posteriormente se ha de crear manualmente el enunciado del problema asociado, traduciendo a lenguaje natural la sentencia SQL generada. De esta forma se reduce considerablemente el tiempo necesario para crear el conjunto de problemas.

La generación automática de problemas que sean adecuados para estudiar los conceptos asociados a un conjunto de restricciones o conceptos, consiste en insertar el texto de la condición de activación de las restricciones, que básicamente es un reconocimiento de patrones. Estos patrones pueden contener variables, a las que se da un conjunto válido de valores, según el tipo de la variable o la expresión donde se utiliza, y se restringe posteriormente dicho conjunto de valores a los que cumplan la condición a satisfacer de la restricción. El resultado de esta fase es un conjunto de

fragmentos SQL, donde puede haber tanto fragmentos válidos como inválidos.

Para corregir el resultado anterior el algoritmo iterará también sobre un conjunto de restricciones que indica cuándo una solución es válida semánticamente. En este proceso se corrige la sentencia SQL cada vez que se encuentra una restricción que no se cumple. Básicamente se busca, en el conjunto de problemas existente, aquellos problemas que cumplan la restricción a solucionar, y se realiza un proceso de encaje de patrones para rellenar con fragmentos correctos la parte de la solución automática que no cumple la restricción. El algoritmo utilizado en esta parte de la generación automática se utiliza además para corregir errores sintácticos en la respuesta de los alumnos.

2.4.2.2 MathEdu

MathEdu es una herramienta de autor para crear cursos de Matemáticas, que permite al alumno crear problemas de forma aleatoria, y resolverlos siguiendo una serie de pasos, guiado por el sistema, [Díez 2003].

Esta herramienta está programada en Mathematica[36], utilizando la capacidad de cálculo simbólico de este software, y usa técnicas de programación por demostración para la especificación de los problemas tipo que el alumno puede resolver.

El proceso de diseño consiste en la especificación de una serie de clases de problemas, las condiciones iniciales de dichos problemas, y las diferentes estrategias de resolución. Esta especificación se realiza creando expresiones Matemáticas, que utilizan meta variables para la definición de dependencias entre dichas expresiones.

MathEdu permite al alumno resolver problemas del tipo de los especificados, para lo cual se pueden crear problemas de forma aleatoria, siguiendo el patrón expresado por el diseñador al especificar el tipo de problema.

[36] El sitio Web oficial de Mathematica está en la dirección: http://www.wolfram.com/products/mathematica

Estos problemas pueden ser complejos y requerir a su vez la división en subproblemas, los cuales han de ser analizados para comprobar a que clase de problema corresponde cada subproblema. Esta división en subproblemas no siempre es única, además cada subproblema puede a su vez pertenecer a varios de los tipos de problemas especificados. Por lo tanto, el alumno necesitará elegir entre distintas estrategias de resolución, para ello el sistema comprueba que la estrategia elegida por el estudiante es compatible con el problema actual, dejándole continuar cuando seleccione una estrategia correcta. Esta funcionalidad también es utilizada en MathEdu para enseñar al estudiante los pasos para resolver un problema creado de forma aleatoria, de entre los tipos de problemas especificados en la fase de diseño.

2.4.2.3 Amulet

Amulet es un entorno para el desarrollo de interfaces de usuario, [Myers 1997]. La principal aportación es la de ser uno de los primeros sistemas que incorporó un mecanismo de restricciones para facilitar la creación de las interfaces, y ha servido como base de algunos de los sistemas de programación por demostración que hemos visto en el apartado anterior.

Otra de las novedades es que permite crear nuevos objetos de una forma muy sencilla, mediante la técnica de prototipo-instancia, la cuál permite que cualquier objeto pueda ser usado como prototipo de nuevos objetos.

El sistema de restricciones utilizado permite que el valor de las propiedades de los objetos pueda calcularse como una fórmula que depende del valor de las propiedades de otros objetos, recalculándose automáticamente cuando sea necesario. El sistema de fórmulas es muy flexible, ya que pueden ser cualquier función C++. Esta flexibilidad se consigue detectando desde dónde se accede a las propiedades de los objetos. Cuando se accede desde una fórmula se crea una dependencia entre la propiedad del objeto accedido y la fórmula desde la que se accede, así el sistema de restricciones puede saber cuándo es necesario recalcular una fórmula para mantener una restricción.

Este sistema también permite que varios objetos compartan la misma fórmula, aunque la evaluación de la misma dependa del objeto que va a utilizar dicha fórmula, también se pueden usar varias fórmulas para calcular

el valor de una misma propiedad, crear referencias circulares, o incluso crear fórmulas que tengan “efectos secundarios”, como crear o destruir objetos, o asignar valores a otras propiedades. Todo esto se puede realizar de forma prácticamente transparente al programador, ya que el sistema detecta este tipo de situaciones especiales, sin requerir que el programador utilice una sintaxis diferente a cuando crea una fórmula normal.

Capítulo 3. Aprendizaje colaborativo guiado

El proceso de aprendizaje tradicional impartido en el aula de forma presencial permite una interacción entre profesor y alumnos bastante fluida para el caso de aulas con pocos alumnos por profesor. Mantener estas condiciones requiere de mayores recursos en el caso de las enseñanzas científico técnicas, donde los alumnos necesitan un mayor grado de experimentación. Además, en muchos casos dedican una gran cantidad de tiempo realizando prácticas o resolviendo problemas fuera del aula.

Para obtener una buena calidad del aprendizaje es necesario poder evaluarlo correctamente, tanto durante el desarrollo como a la finalización. Si esta evaluación se pudiera realizar de forma continua podríamos adaptar los contenidos a los alumnos, de forma que se maximice el ritmo de aprendizaje. En la enseñanza tradicional sólo podremos hacer este seguimiento correctamente con un número relativamente bajo de alumnos por profesor. Esto se debe a que tanto mantener un nivel de conocimientos homogéneo en cada grupo, como realizar una evaluación continua, será más costoso y complejo cuanto mayor sea el número de alumnos.

En este proceso de seguimiento y adaptación continua del proceso de aprendizaje, podemos ayudarnos de las técnicas de aprendizaje en grupo. Estas técnicas facilitan el aprendizaje, ya que los grupos de alumnos pueden resolver problemas más complejos que los que pueden resolver por separado, pudiendo aprender por sí mismos a resolver nuevos tipos de problemas. Como hemos visto en el apartado 2.1.1, esto permite al profesor descubrir la zona de próximo desarrollo de los alumnos, o ZPD, obteniendo una evaluación más precisa del proceso de aprendizaje, lo que permite a su vez una mejor adaptación a los alumnos, y por lo tanto mejorar el aprendizaje.

Para poder continuar el aprendizaje en grupo fuera del aula podemos recurrir a la enseñanza a distancia. De esta forma los alumnos no necesitan desplazarse para la mayoría de las actividades y pueden estudiar o practicar a cualquier hora. Sin embargo surgen nuevos problemas que pueden disminuir la calidad de la enseñanza. Uno de los problemas principales que surgen son las dificultades de seguimiento y evaluación del proceso, sobre todo por las limitaciones de comunicación entre profesor y alumnos. Para solucionar este problema hay que buscar mecanismos de comunicación que

permitan realizar adecuadamente las tareas de seguimiento y tutorización de los alumnos.

La enseñanza a distancia también permite aumentar el número de alumnos por profesor, ya que éstos pueden aprender a su propio ritmo, con el horario que deseen, y gracias a Internet los contenidos didácticos pueden llegar a cualquier número de alumnos. Lo que limita el número de alumnos en la enseñanza a distancia es la necesidad de seguimiento y tutorización que permita al profesor conocer cómo evolucionan los alumnos, y resolver las dudas y problemas planteados con un tiempo de respuesta breve, que permita al alumno mantener un ritmo de aprendizaje adecuado y no perder la motivación.

Para facilitar el seguimiento y tutorización en la educación a distancia surge el aprendizaje colaborativo guiado, que permite la revisión crítica del trabajo realizado por los alumnos. Esta revisión, incluyendo la propuesta de alternativas, la pueden realizar distintos actores del proceso de aprendizaje, incluyendo a los propios alumnos y al profesor. Parece que la mayor utilidad corresponda al caso de la revisión del trabajo del alumno por un profesor, que puede guiar a aquél mediante las propuestas que realiza. Este es el origen del término utilizado. Sería de interés estudiar las posibilidades de utilización de las herramientas de revisión crítica por parte de los propios alumnos, aunque este es un trabajo que excede a los objetivos de esta tesis.

El aprendizaje colaborativo guiado permite a alumnos y profesores, gracias a las herramientas colaborativas, compartir el mismo espacio de trabajo, y por lo tanto una información más completa sobre el tema a tratar. De esta forma se facilita enormemente la comunicación entre profesor y alumnos, y simplifica la evaluación, seguimiento y tutorización. Además soporta de forma natural el aprendizaje colaborativo, permitiendo a los alumnos que realicen las actividades en grupo y al profesor hacer un seguimiento de dichos grupos. Estas herramientas también permiten al profesor o profesores realizar un seguimiento simultáneo de los diversos grupos de trabajo, revisando el trabajo realizado o las tareas que realizan en cada momento, sin que sea necesario interrumpir a los alumnos. Además, el aprendizaje colaborativo guiado facilita la integración de agentes inteligentes con los profesores, permitiendo que éstos puedan atender a un mayor número de

alumnos. Esta integración puede facilitar la tarea del profesor en dos sentidos, primero, el alumno puede recibir ayuda automáticamente para tareas sencillas y, en segundo lugar, el profesor puede recibir informes sobre qué grupos de trabajo necesitan ayuda en cada momento, o resúmenes del trabajo realizado.

La revisión del trabajo realizado por los alumnos es una parte fundamental del aprendizaje colaborativo guiado. Esta revisión puede llevarse a cabo por el propio alumno, por un grupo de alumnos o por los profesores, junto con el alumno o grupo de alumnos. La revisión se completa con el análisis de propuestas para mejorar los resultados obtenidos y superar las dificultades y errores cometidos, estudiando las alternativas existentes. De esta forma los alumnos pueden estudiar los errores más comunes que han cometido otros alumnos en situaciones parecidas, y las sugerencias de sus compañeros o los profesores. Como resultado obtenemos una base de conocimiento que va aumentando con las nuevas aportaciones de profesores y alumnos.

El aprendizaje colaborativo guiado se fundamenta por tanto en la integración de las capacidades de supervisión y análisis con la capacidad de trabajo o aprendizaje colaborativo. Con esta integración obtenemos aplicaciones colaborativas con capacidades para el aprendizaje. Las aplicaciones resultantes permiten supervisar el trabajo o aprendizaje a distancia, tanto síncrona como asíncronamente, y pasar de la tarea de supervisión o revisión a la de trabajo activo sin la necesidad de realizar ninguna acción especial.

La revisión permite ver paso a paso las tareas realizadas, como si fuera un vídeo, y añadir en cualquier momento alternativas al trabajo realizado mediante la simple ejecución de las mismas, creando así una nueva rama en la historia. El proceso de revisión también permite añadir comentarios a la historia revisada. De la misma forma se puede estar trabajando colaborativamente y en cualquier momento revisar cómo otro grupo ha realizado una tarea similar. Esto permite por ejemplo que un profesor pueda supervisar el trabajo de varios grupos de alumnos, y en cualquier momento colaborar activamente en alguna de las tareas asignadas.

Gran parte de las tareas necesarias para crear las aplicaciones que soporten el aprendizaje colaborativo guiado se pueden simplificar

construyendo un *framework* que implemente los aspectos generales y facilite la integración de los aspectos específicos de las aplicaciones. Por ejemplo, todos los aspectos relativos a las comunicaciones se pueden generalizar, permitiendo que las aplicaciones se puedan crear como aplicaciones monousuario, no colaborativas, mucho más sencillas de implementar. También se pueden generalizar los aspectos de revisión o análisis, por lo que la complejidad para convertir una aplicación normal en una herramienta que soporte el aprendizaje colaborativo guiado se minimiza.

Gracias a este framework podemos por ejemplo trasformar una aplicación muy sencilla que implemente un tablero de ajedrez en una aplicación colaborativa que permita jugar a distancia, y que además pueda ser usada para la enseñanza del ajedrez, permitiendo la revisión de las partidas, añadiendo comentarios o alternativas, tanto por parte del profesor como de otros alumnos.

En el siguiente punto se expondrán con más detalle los requisitos para el aprendizaje colaborativo guiado así como aspectos deseables en el mismo. Posteriormente se describirá un framework que implementa estos requisitos.

3.1 Requisitos y aspectos deseables en el aprendizaje colaborativo guiado

En lo que sigue describimos los requisitos que tiene que satisfacer un sistema de enseñanza asistida por ordenador para potenciar el aprendizaje colaborativo guiado. En primer lugar describiremos los requisitos y aspectos deseables que puede percibir un usuario final de la aplicación, profesor o estudiante, sin entrar en ningún tipo de aspectos técnicos relacionados con la arquitectura o funcionalidad interna del sistema. Posteriormente, basándose en los requisitos obtenidos al nivel de usuario, describiremos los requisitos funcionales que de aquéllos se difieren.

3.1.1 Requisitos de usuario

Supondremos que el contexto de trabajo consiste en la ejecución por parte del estudiante de determinadas tareas específicas, como puede ser la elaboración de un diagnóstico y un tratamiento clínico o de un presupuesto

de una entidad virtual, la realización de un experimento en un laboratorio virtual o la resolución de un problema de Cálculo Infinitesimal. Damos por supuesta la necesidad de que la labor de aprendizaje se pueda llevar a cabo sin la presencia física constante del profesor. Según se ha dicho en la introducción a este capítulo, una parte fundamental en el aprendizaje colaborativo guiado es que el alumno pueda recurrir mientras trabaja a un profesor que revise su trabajo y le aconseje acerca de los errores que comete y la forma de subsanarlos y el profesor a su vez pueda interrumpir al alumno para encaminar mejor los esfuerzos que está llevando a cabo dentro de su tarea de aprendizaje. Esta tarea puede ser sencilla en los casos en que este trabajo se puede mostrar a través de un documento, como cuando el alumno está redactando unas notas. En otros casos, como en el de un experimento que se realiza en un laboratorio virtual, esto puede requerir un enfoque totalmente diferente. En estos casos en que el trabajo del estudiante consiste en realizar acciones de forma interactiva (valga la redundancia) sobre la interfaz de un programa, modificando con ello el estado de la misma, la revisión del trabajo realizado consistirá en la simulación paso a paso de las acciones llevadas a cabo por parte del alumno, de forma análoga a como se muestra el contenido de un vídeo que tiene grabada una escena. Esta revisión se deberá poder realizar también en sentido inverso al de la realización de las acciones por el usuario o partiendo de un determinado momento, de forma que no sea necesario ejecutar todas las acciones una a una, permitiendo así mayor flexibilidad al profesor a la hora de revisar el trabajo de los alumnos.

Algunos sistemas que permiten grabar vídeos interactivos con las acciones del usuario, como CEVA [Cockburn 1997] o TeleTeaching [Ziewer 2002], permiten obtener un vídeo con las acciones realizadas. Uno de los problemas con este tipo de sistemas es que generan demasiados datos, lo cual limita su uso práctico en la educación a distancia, al requerir un ancho de banda demasiado alto para obtener vídeos de alta calidad que permitan ver con claridad el trabajo del alumno.

Además de la revisión del trabajo previo, un sistema para el trabajo colaborativo guiado debe de permitir que el profesor proponga soluciones alternativas. Estas alternativas pueden consistir en resoluciones parciales o totales del problema que pretende resolver el alumno, y es deseable que el

profesor las pueda especificar simplemente llevando a cabo en la misma interfaz de usuario del programa las acciones correspondientes. Al igual que el profesor debe poder reproducir el trabajo del estudiante, éste debe tener la posibilidad de reproducir las alternativas propuestas por aquél. Este requisito tampoco se puede cumplir con los sistemas que generan vídeos citados anteriormente, y la única posibilidad sería que el profesor comenzase el ejercicio o problema desde cero para generar un nuevo vídeo con la solución o alternativa propuesta.

Por último, el profesor debe poder añadir comentarios a los distintos pasos del trabajo del estudiante, y del mismo modo el estudiante debería poder añadir anotaciones e incluso nuevas alternativas a las alternativas propuestas por el profesor. Las anotaciones de alumnos y profesores deben poder incluir casos de trabajo, es decir resoluciones de otros problemas relacionados que sean relevantes para el trabajo que se está llevando a cabo.

Un segundo nivel de requisitos de usuario relacionados con el anterior se refiere al grado de sincronización entre los distintos actores del proceso. No solamente es preciso que el estudiante pueda trabajar sin la presencia del profesor, sino también que lo pueda hacer en momentos en que éste no está accesible. Así pues, tanto la revisión del trabajo del alumno como la de las alternativas planteadas por el profesor, anotaciones, etc., e incluso la creación de alternativas y anotaciones se deben poder llevar a cabo de forma síncrona o asíncrona. En el primer caso, debe ser el profesor el que decida quién tiene la iniciativa en cada momento en el desarrollo de la sesión de trabajo; en el segundo, cabe la posibilidad de que ésta se lleve a cabo por decisión independiente del propio actor o inducido por un mensaje o alerta que active la sesión de revisión. Estos requisitos admiten una extensión que consiste en la posibilidad de que varios estudiantes realicen conjuntamente la revisión colaborativa del trabajo de uno de ellos o, en caso de que la aplicación subyacente lo permita, la revisión colaborativa del trabajo.

También debería ser posible soportar la revisión del trabajo realizado de forma colaborativa por varios alumnos, ya que, al igual que ocurre en el aula, es importante que los estudiantes aprendan a trabajar en grupo, y por lo tanto las herramientas de revisión y análisis han de suministrar información

sobre la interacción entre los miembros de cada grupo de trabajo, y quién ha realizado cada acción, para de esta forma permitir al profesor ayudar a cada miembro del grupo de forma individual.

Relacionada con lo anterior está la necesidad de que las aplicaciones para el aprendizaje colaborativo guiado permitan una comunicación adecuada entre sus actores. Las necesidades esenciales se pueden cubrir mediante servicios complementarios como el chat escrito y hablado, la vídeo-conferencia y el correo electrónico.

De lo comentado anteriormente se deduce así mismo la necesidad de contar con diversos roles, entre los que se encontrarán los de profesor y alumno. Además, es útil disponer del rol de espectador, que será el rol que tome un grupo de alumnos que pueden ver cómo se desarrolla o ha desarrollado una actividad, pero no pueden alterarla. Esto puede ser especialmente útil cuando se expone el trabajo en un aula virtual o real. Cada rol tendrá asignadas determinadas capacidades, que pueden referirse a cursos concretos o al trabajo realizado por determinados usuarios.

Otro requisito relacionado con los anteriores es la posibilidad de que los usuarios finales (estudiantes, profesores y espectadores) puedan incorporarse o abandonar el trabajo síncrono con otros usuarios de forma simple y transparente para los demás usuarios. Esto incluye la posibilidad de dejar de seguir síncronamente el trabajo de un estudiante para revisar paso a paso su trabajo previo o, al revés, incorporarse directamente tras un recorrido por el trabajo realizado al seguimiento síncrono de su realización. Ambos tipos de cambios se deben poder realizar de forma fluida, sin solución de continuidad. Igualmente, los usuarios deben de poder pasar de trabajar síncronamente con otros usuarios a hacerlo de forma individual.

Otro aspecto de interés, especialmente en el estudio de materias que requieren la utilización de técnicas dispares, es que el sistema colaborativo permita reproducir sesiones de trabajo que involucran la utilización de más de una herramienta. Por ejemplo, un estudiante que resuelve problemas de estructuras podría utilizar sucesivamente un editor gráfico y una hoja de cálculo, y podría ser de interés seguir la forma en que lo hace. Al reproducir el trabajo realizado las acciones realizadas sobre ambas interfaces se reproducirían en el mismo orden que en su ejecución inicial.

Independientemente de lo anterior, como ya hemos dicho, puede resultar útil para el aprendizaje la asistencia al alumno por parte de agentes interactivos que, siguiendo su trabajo, le den indicaciones acerca de la mejor forma de enfocarlo, evitando problemas que surjan, o incluso le pongan automáticamente en contacto con el profesor en caso de necesidad. Estos informes pueden dirigirse también al profesor. Además de seguir cada uno de los pasos dados por el alumno y analizarlos en su contexto, estos agentes deberán poder detectar la existencia de contactos con el profesor al respecto, para evitar actuar de forma intrusiva cuando sea innecesario. Uno de los beneficios de la utilización de agentes de esta forma es permitir que en cada momento el profesor atienda a los estudiantes que más lo necesitan, optimizando sus posibilidades de ayuda al grupo de estudiantes que dependen de él.

3.1.2 Requisitos funcionales

A continuación expondremos los requisitos funcionales que surgen para satisfacer los requerimientos de usuario descritos anteriormente. Comenzaremos con el requisito mencionado en el apartado anterior referente a que el diseñador pueda adaptar una aplicación inicial a la enseñanza colaborativa guiada mediante un componente software genérico reutilizable. La solución más potente disponible hoy día para este tipo de tarea es la utilización de un framework orientado a objetos.

La necesidad de que la revisión se pueda realizar de forma directa o *deshaciendo* las acciones del usuario en forma inversa exige que las aplicaciones de partida satisfagan el requerimiento funcional de permitir deshacer y rehacer acciones. El mecanismo de deshacer y rehacer acciones es también una pieza fundamental para soportar el trabajo y el aprendizaje de forma asíncrona.

El framework que hemos desarrollado, FACT, el cual veremos en detalle en el capítulo 4, utiliza el mecanismo de deshacer y rehacer acciones como abstracción básica de implementación para permitir que las aplicaciones colaborativas se puedan integrar, satisfaciendo así los requisitos funcionales y de usuario que se describen en este capítulo. Para ello, las historias de colaboración en FACT incluyen información sobre las acciones de deshacer y

rehacer. De esta forma todos los usuarios que acceden a una historia determinada pueden trabajar colaborativamente, tanto síncrona como asíncronamente.

La abstracción de deshacer y rehacer puede requerir un gran ancho de banda, en ciertas aplicaciones colaborativas, a la hora de navegar por la historia. Este ancho de banda se puede disminuir enviando los segmentos completos de historia que se requieren para pasar de un punto a otro, fusionando eventos antes de ser enviados, comprimiendo la información y mediante eventos especiales que contengan todo el estado. La fusión de eventos y el envío de eventos con todo el estado requieren codificar estos mecanismos para cada aplicación específica. Métodos que lean o escriban el estado completo de la aplicación suelen estar implementados como parte de la funcionalidad de la aplicación, por lo que esta optimización en general no será muy costosa. Por lo tanto, en la comunicación entre el servidor que almacena la historia y las aplicaciones, se podrá enviar un solo evento con el estado completo más los eventos creados desde ese punto hasta el punto de la historia al que se quiere ir. De esta forma la información enviada por el servidor disminuye considerablemente, lo cual disminuye la espera cuando un usuario se incorpora a la sesión de trabajo, o cuando se intenta saltar a otro punto de la historia.

Cuando las aplicaciones son utilizadas de forma síncrona por varios alumnos, o en la interacción profesor/alumno, cada usuario ha de poder ver los cambios producidos por otros usuarios lo antes posible, evitando además los problemas típicos de concurrencia de estos sistemas. Por consiguiente, se deben tratar los problemas de sincronización de los cambios y de latencia. La forma más sencilla de tratar los problemas de sincronización es mediante un componente centralizado que gestione la concurrencia.

Relacionado con la gestión de los grupos de trabajo está la necesidad de permitir la modificación de los usuarios que están trabajando síncronamente, incorporando nuevos usuarios o eliminándolos temporalmente, de forma simple y sin perturbar a los demás usuarios, el cual es otro de los requisitos detectados a nivel de usuario. Para ello es necesario crear sesiones que representan el trabajo que realiza un grupo de forma síncrona. Es necesario entonces permitir la creación y modificación dinámica de sesiones,

incluyendo su separación en varias sesiones diferentes, que podrían tener usuarios en común para permitir que el tutor pueda conectarse a varias sesiones simultáneamente o que un estudiante repase sus acciones previas mientras mantiene activa su sesión de trabajo. La gestión de estas sesiones es también parte de las funcionalidades que se han de proporcionar en el framework.

La necesidad de contar con diversos roles, incluyendo los de tutor, alumno y espectador, se traduce en términos funcionales en la necesidad de definir los roles básicos de usuario de un sistema tutor, usuario dentro de una sesión, revisor y espectador de una sesión y administrador de roles, así como de desarrollar mecanismos para la asignación a profesores y alumnos de los roles básicos que les corresponda en cada caso. Cada rol ha de tener determinadas capacidades dentro de un ámbito, que será el sistema tutor, sesión o el conjunto de usuarios a que hace referencia el rol en concreto.

El requisito de permitir la revisión del trabajo previo da lugar a la necesidad de considerar las historias de trabajo como entidades básicas del sistema. Estas historias contendrán la información necesaria para revisar todos los aspectos del trabajo realizado, incluyendo la colaboración que haya tenido lugar. Las historias han de ser dinámicas en dos aspectos: por una parte, se deben actualizar constantemente en base a las nuevas acciones de los usuarios; por otra parte, deben permitir incorporar alternativas en cada uno de los pasos que incluyen. Esto confiere a las historias una estructura conceptual de árbol dinámico cuyas modificaciones posibles consisten en la creación de nuevas ramas y el crecimiento de las mismas por las hojas. Este aspecto, inherente al hecho de que las historias acumulan todos los hechos relevantes para ellas, que nunca se modifican ni desaparecen, es fundamental para permitir de manera segura la revisión asíncrona por diversos actores de las historias, incluyendo sus diversas ramas.

Las anotaciones en la historia que permiten comunicar de forma asíncrona, por ejemplo, cuál es la alternativa que parece más apropiada en la resolución de un problema han de soportar la inclusión de enlaces a otros puntos de historias que puedan utilizarse como ejemplos o como parte de la explicación contenida en la anotación. Además, en ocasiones es preciso que las propias interfaces de las aplicaciones referidas aparezcan incluidas en la anotación,

para ayudar a clarificar el texto de la anotación. Esto se puede hacer por ejemplo en forma de imagen, que represente el estado de la interfaz de usuario en un estado dado, junto con un enlace que permita ir al punto de la historia del que se está hablando.

La necesidad de crear un entorno que permita el aprendizaje a distancia y la colaboración síncrona, implica también una arquitectura que ofrezca unos tiempos de respuesta mínimos y una gran flexibilidad en cuanto a los distintos tipos de interfaces de usuario que se crearán. La forma de conseguir esto es creando aplicaciones cliente que procesen la mayor parte de la interacción del usuario localmente, minimizando así la cantidad de información que es necesario intercambiar a través de la red. Las aplicaciones cliente deberán poder ser soportadas por el mayor número de plataformas hardware o software posible y permitir el trabajo a través de Internet. Estos requisitos podemos satisfacerlos actualmente mediante el uso de sistemas multiplataforma, como Java o .NET.

Recientemente están surgiendo alternativas basadas únicamente en navegador Web, como las aplicaciones que usan la técnica AJAX [Garrett 2005], que combinan el uso de lenguajes interpretados por el navegador con el envío de mensajes xml para la actualización parcial de la interfaz de usuario. De esta forma se consiguen tiempos de respuesta cortos y aplicaciones ligeras para el navegador Web con una funcionalidad similar a la conseguida con aplicaciones cliente pesadas, sin necesidad de realizar ninguna instalación. El problema con estas aplicaciones es que son difíciles de implementar y no permiten reutilizar el código de aplicaciones existentes, aunque esto se solucionará en parte con la aparición de herramientas de desarrollo Web más potentes.

En cuanto al soporte sistemático a la comunicación entre los usuarios de las aplicaciones mediante chateo, vídeo-conferencia y de envío de correo electrónico se deben incorporar módulos o aplicaciones genéricas.

Para permitir la integración de agentes en la enseñanza colaborativa guiada, incorporando características de los tutores *inteligentes* a las aplicaciones, el sistema deberá incorporar un módulo interactivo de especificación de estados de alarma y de acciones a tomar, que conceptualmente corresponden a reglas de producción. Existen trabajos de

investigación, realizados en contextos próximos como el de las herramientas para la creación de sistemas de ayuda al usuario en interfaces interactivas, que son muy relevantes para esta cuestión, [Moriyon 94].

Capítulo 4. FACT

FACT es un framework que facilita la construcción de aplicaciones colaborativas con soporte de aprendizaje guiado. El nombre proviene del acrónimo en Ingles “Framework for Advanced Collaborative Tutoring”, y consiste en una serie de bibliotecas Java y varias aplicaciones cliente/servidor. Las aplicaciones creadas con FACT cumplen los requisitos estudiados en el capítulo anterior, con unas condiciones mínimas para su integración con el framework.

El resultado son aplicaciones que se pueden registrar en el servidor de FACT para que puedan utilizarse por cualquiera que tenga acceso al servidor, mediante el cliente genérico de FACT. El entorno de ejecución cliente permite el trabajo colaborativo con dichas aplicaciones, el análisis del trabajo realizado, la creación de alternativas desde cualquier punto en la historia de colaboración, crear anotaciones, etc.

La ventaja principal de este framework es que toda la funcionalidad de análisis y trabajo colaborativo se añade de forma transparente a la aplicación integrada, al ejecutar las aplicaciones desde el cliente de FACT. De las aplicaciones sólo se requiere que implementen un mecanismo de hacer o deshacer acciones, y que toda la información necesaria para ello sea notificada en forma de acciones individuales al framework. Puesto que el poder hacer y deshacer acciones se considera actualmente una funcionalidad básica en las aplicaciones interactivas, y que el lenguaje Java, en el que están escritas dichas aplicaciones, ya prevé un mecanismo estándar para ello, la adaptación de las aplicaciones existentes a este framework resulta una tarea muy sencilla en la mayoría de los casos.

Normalmente estas acciones son de alto nivel, que representan por ejemplo que un usuario ha escrito una frase en un documento, o que ha movido una pieza en un tablero de ajedrez. También se puede crear una aplicación que envíe solo información de bajo nivel, como el movimiento del ratón o las pulsación de teclas. Sin embargo, esto obliga a que los usuarios de las distintas aplicaciones vean exactamente la misma interfaz, limitando la usabilidad de la aplicaciones colaborativas. Además, las tareas de análisis son más sencillas para el usuario si éste trabaja con acciones de alto nivel.

Otra de las ventajas es que el análisis está integrado con el trabajo normal de la aplicación, permitiendo que ambas tareas puedan ser llevadas a cabo

de forma simultánea por los distintos colaboradores, sin necesidad de realizar un cambio de contexto ni alguna tarea especial para realizar el cambio entre el trabajo colaborativo normal y el análisis de la historia. Para permitir este nivel de integración, el análisis del trabajo colaborativo se lleva a cabo utilizando la propia aplicación, para reproducir la historia tanto hacia delante como hacia atrás. Esto es posible ya que el mecanismo para reproducir la historia es el mismo que el utilizado en la colaboración para reproducir los eventos remotos que producen los distintos usuarios, reenviando los eventos capturados o enviando eventos del tipo deshacer. Esto es así aunque dicho análisis implique revisar el trabajo de varias aplicaciones colaborativas que se ejecutan simultáneamente en una misma sesión, y que por lo tanto se han de cerrar y abrir automáticamente durante la navegación por la historia.

Respecto a la implementación del framework, hemos seleccionado Java como lenguaje de desarrollo, en lugar de otras alternativas multiplataforma, ya que en el momento en que se inició el desarrollo era la plataforma en la que resultaba más sencillo desarrollar aplicaciones interactivas complejas, y aún sigue siéndolo, aunque recientemente han aparecido nuevas opciones. También es sencillo encontrar multitud de aplicaciones educativas para integrarlas en el framework. Además, las aplicaciones escritas en este lenguaje pueden ejecutarse en distintas plataformas y los mecanismos de comunicación son potentes y fáciles de utilizar.

Hasta ahora hemos visto a grandes rasgos como es FACT. En el resto del capítulo veremos en profundidad su estructura, los distintos componentes de la misma, cómo se integran las aplicaciones colaborativas en FACT, y las experiencias realizadas con el mismo.

4.1 Arquitectura

Este framework se ha diseñado para facilitar la construcción de aplicaciones que soporten el aprendizaje colaborativo guiado, intentando abarcar la mayor diversidad de aplicaciones posible, lo que obliga a una arquitectura suficientemente adaptable a las necesidades de las distintas aplicaciones.

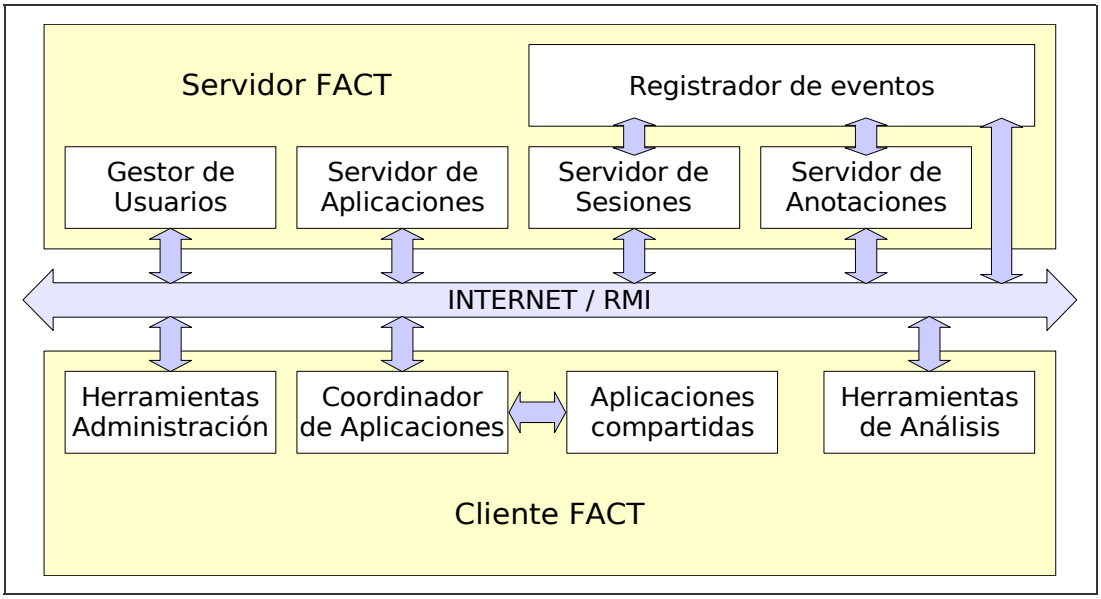

Figura 4: Arquitectura de FACT

Como se puede observar en la Figura 4, en la arquitectura se distinguen claramente dos tipos de elementos, los servidores y las aplicaciones cliente. En esta figura se representa una única instancia del servidor FACT, y un cliente FACT. Sin embargo el caso típico se da con un servidor y varios clientes.

En el lado servidor, tenemos el registrador, servidor de anotaciones y servidor de sesiones. Estos servicios se encargan de gestionar la historia de colaboración, las anotaciones y el trabajo síncrono entre los grupos de usuarios, respectivamente. Los otros dos servicios, el Gestor de Usuarios, y el Servidor de Aplicaciones, se encargan de la gestión de permisos y usuarios, y de las aplicaciones colaborativas disponibles.

Por otro lado tenemos el cliente FACT, que es la aplicación desde la que el usuario accede a todos los servicios de FACT. Es de destacar que las aplicaciones compartidas no se comunican directamente con el servidor, sino que lo hacen a través del Coordinador de Aplicaciones. Este componente es el único que se comunica con las aplicaciones colaborativas, minimizando así la interfaz entre las aplicaciones colaborativas y el framework, permitiendo una gran independencia de aspectos técnicos, como el sistema de comunicación utilizado, o la versión del resto de componentes del framework. Este componente se encarga principalmente de coordinar la información de

varias aplicaciones colaborativas, de forma que se comporten como una sola macro-aplicación.

La arquitectura ofrece una gran flexibilidad, permitiendo adaptarse a distintas configuraciones, aparte de la configuración típica de varios clientes/ un servidor. Por ejemplo es posible tener distribuidos todos los servicios de FACT, actuando el servidor FACT como servidor de directorio, de forma que la arquitectura distribuida sea transparente a los clientes. Algunos servicios, como el registrador, que es el que más carga de red soporta, también funciona en modo semi-replicado en los clientes, permitiendo de esta forma una navegación fluida por las historias de colaboración.

Otro de los aspectos importantes del framework es que también permite experimentar con distintas técnicas y modos de trabajo. Por ejemplo, FACT permite trabajar de forma simultánea usando colaboración síncrona y asíncrona sobre las mismas historias de colaboración. Además todos los servicios del framework son accesibles de forma remota a través de RMI[37], lo cual hace muy sencillo extender el framework para implementar nuevos módulos o herramientas.

Aunque todos los servicios están accesibles de forma remota, cuando éstos interactúan entre ellos, por ejemplo el servidor de sesiones con el registrador, la comunicación se realiza de la forma más eficiente posible. Esto se consigue de forma transparente gracias al polimorfismo que ofrece la utilización de RMI. Este polimorfismo permite intercambiar objetos remotos con locales, por lo tanto, si ambos servicios se ejecutan en máquinas virtuales diferentes la comunicación se realizará remotamente, mientras que si se ejecutan en la misma máquina virtual la comunicación será mediante llamadas directas a los métodos.

4.2 Descripción jerarquizada de los componentes

Otra forma de ver cómo están relacionados los distintos componentes de FACT es a través de una vista por niveles, Figura 5. En general podemos hablar de tres niveles fundamentales, el de aplicación, colaboración y el de análisis.

[37] RMI es el sistema utilizado en Java para la invocación remota de métodos.

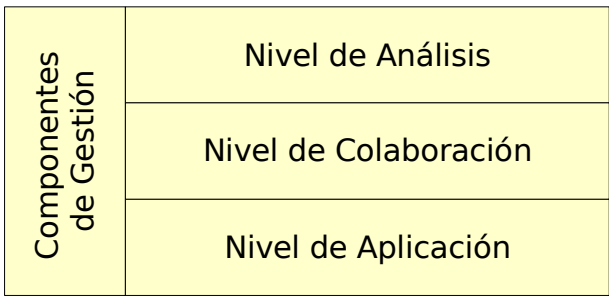

Figura 5: Niveles conceptuales de FACT

En el nivel de colaboración tenemos los componentes básicos del trabajo colaborativo, las sesiones de colaboración y el registrador de eventos. Por último, en el nivel de análisis, tenemos las distintas herramientas de análisis, como son el navegador de la historia, el editor de anotaciones y el servidor de anotaciones.

En la columna de la izquierda tenemos los distintos componentes de gestión, representados por las herramientas administrativas y de supervisión en el cliente de FACT, y el módulo de gestión de usuarios del servidor, principalmente. Los componentes de gestión permiten configurar el comportamiento o ver el estado de los componentes que están en los distintos niveles de sistema.

A continuación veremos en detalle los componentes de FACT, tanto del lado servidor como del cliente, siguiendo para ello la descripción conceptual por niveles que hemos visto en este apartado.

4.2.1 Nivel de Aplicación

El subconjunto de componentes de FACT relacionados con el nivel de aplicación es el que se aprecia en la Figura 6. También se ha representado sombreado el servidor de sesiones, que pertenece al nivel de colaboración, para ayudar a entender cómo se relaciona este nivel con el siguiente.

Como se puede apreciar, el elemento central en este nivel es el coordinador de aplicaciones, que es responsable de la integración de las aplicaciones en FACT, de forma que éste pueda suministrar la infraestructura de colaboración y análisis, de forma totalmente transparente a las

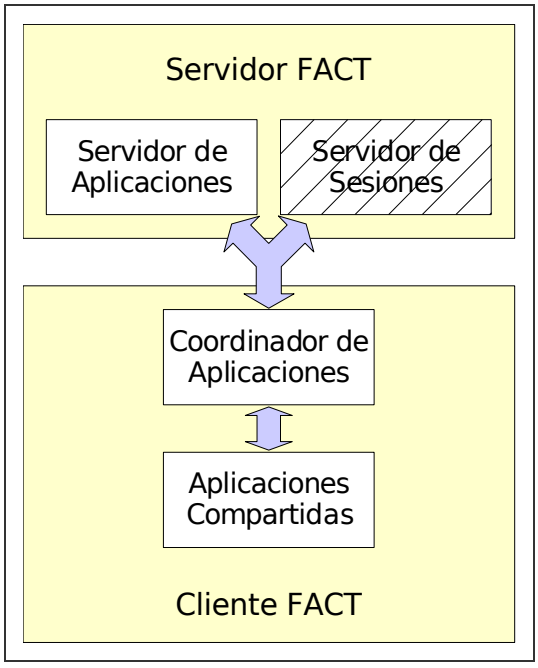

Figura 6: Nivel de aplicación

aplicaciones integradas. Este elemento también se encarga de incorporar nuevas aplicaciones en la interfaz del cliente, obteniéndolas del servidor de aplicaciones.

4.2.1.1 Servidor de aplicaciones

El servidor de aplicaciones se encarga básicamente de suministrar la lista de aplicaciones colaborativas a las que tiene acceso cada usuario. Cuando un usuario quiere iniciar una aplicación nueva, para su integración en la sesión de trabajo activa, y una vez consultada la lista de aplicaciones disponibles, el servidor de aplicaciones envía la aplicación a todos los usuarios que comparten la misma sesión.

En la implementación actual, donde las aplicaciones están escritas en Java, esto se consigue enviando la clase principal de la aplicación, de forma que cada cliente pueda crear una instancia de dicha aplicación en su máquina virtual. Aunque las clases necesarias para ejecutar las aplicaciones no se encuentren en el cliente, estas clases se pueden obtener automáticamente del servidor mediante RMI.

Sin embargo, este mecanismo automático presenta algunas limitaciones, y es recomendable utilizar un sistema de distribución de clases más avanzado,

compatible con el mecanismo de distribución del servidor de aplicaciones de FACT, y que garantice que todas las instancias utilizan la misma versión de las aplicaciones colaborativas. Esto se puede conseguir, por ejemplo, utilizando Java Web Start[38], en adelante JWS. Con JWS se puede hacer una carga flexible de recursos y aplicaciones, de forma que una parte del sistema se cargue al comienzo y otra se realice bajo demanda. Con este sistema también se garantiza que todos los clientes ejecutan la última versión disponible de las aplicaciones, lo cual es un aspecto crucial cuando se trata de aplicaciones colaborativas. Para distribuir FACT con esta técnica sólo es necesario publicar en la web el fichero de configuración de JWS, que contiene la información sobre cómo acceder a los distintos paquetes del framework y las aplicaciones colaborativas.

4.2.1.2 Coordinador de aplicaciones

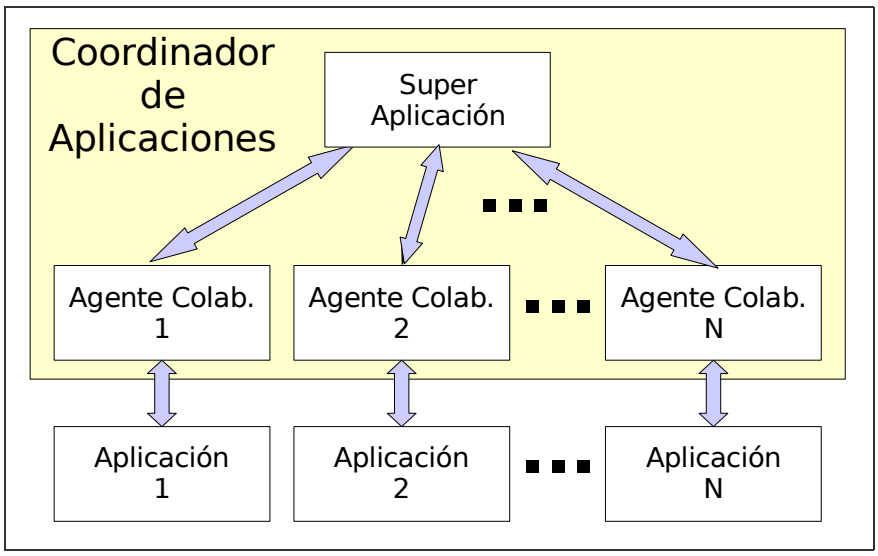

Figura 7: Coordinación de aplicaciones en un cliente de FACT

Las aplicaciones cargadas del servidor no son las aplicaciones finales que ejecutará el usuario. El último paso de la carga de la aplicación es embeberla en FACT para aumentar su funcionalidad y convertirla en una aplicación que soporte el aprendizaje colaborativo guiado. Para lograr esto, y una vez cargada la aplicación deseada en todos los miembros de la sesión

[38] Para más información sobre Java Web Start se puede consultar http://java.sun.com/products/javawebstart

colaborativa, cada instancia es asociada con un agente de colaboración, que forma parte del coordinador de aplicaciones, Figura 7.

Este agente es el único contacto que tiene la aplicación con el resto del framework, y tiene dos tareas fundamentalmente, en primer lugar aisla a la aplicación de los posibles problemas de comunicación que puedan ocurrir, minimizando además la interfaz entre las aplicaciones y FACT. En segundo lugar el agente almacena el estado colaborativo de la aplicación, es decir, su identificador único, y una cola de eventos ejecutados. Esta cola de eventos del agente de colaboración tiene como principal cometido el permitir implementar un mecanismo de ejecución optimista de eventos en las aplicaciones colaborativas. Este mecanismo permite que las instancias que producen los eventos puedan ejecutarlos en su interfaz de usuario antes de asegurarse que no hay otro evento, producido en otra instancia, que deba ejecutarse antes que el producido localmente.

Las aplicaciones por tanto no implementan el soporte de la colaboración, sino que delegan en el framework todo el trabajo, a través del agente de colaboración. Lo único que tienen que hacer las aplicaciones es comunicar al agente los eventos producidos y esperar que el agente le indique que ejecute eventos del mismo tipo, o los deshaga. Estos eventos pueden provenir de la misma instancia o de otras instancias de la misma aplicación, por lo tanto los eventos han de contener toda la información necesaria y no depender de objetos externos, que sólo podrían encontrarse en la instancia que los produjo. En el apartado 4.3 veremos en más detalle los requisitos que han de cumplir las aplicaciones para ser incorporadas en FACT.

Toda la información remitida por los distintos agentes de colaboración es coordinada a través de un único componente, que crea la ilusión de que las distintas aplicaciones que se ejecutan en paralelo se comportan como una única macro aplicación, de ahí el nombre de super-aplicación.

El algoritmo empleado en el agente de colaboración, para resolver los posibles conflictos que plantea la ejecución optimista de eventos, no se ve afectado por la existencia de varias aplicaciones que forman parte de una misma macro aplicación, ya que los eventos de las distintas aplicaciones se consideran sucesos independientes. Esta estrategia de separación de eventos independientes en la macro aplicación se puede emplear para crear

series de sucesos dependientes, que son independientes entre series, dentro de una misma aplicación. Para ello sólo es necesario asignar varios agentes de colaboración a una misma aplicación. El algoritmo optimista de ejecución de eventos y resolución de conflictos se explicará en el apartado 4.2.2, cuando hablemos del nivel de colaboración.

El coordinador de aplicaciones, además de coordinar los eventos, también se encarga de añadir o eliminar nuevas aplicaciones de la macro aplicación dinámicamente, creando para ello eventos especiales que son compartidos por todos los clientes de la misma sesión colaborativa, de forma que en la misma sesión siempre se estén ejecutando las mismas aplicaciones y en el mismo estado. La identificación de a qué aplicación corresponde cada evento se consigue asignando un número a cada aplicación en el momento de su instanciación. Esté número es el primer natural no asignado, que será el mismo en todos los clientes de la sesión, ya que FACT garantiza que el orden de aplicación de los eventos es el mismo en todas las instancias con el estado compartido, y por lo tanto el orden en que se crean o destruyen aplicaciones será el mismo en todos los clientes de la sesión.

4.2.2 Nivel de colaboración

En el nivel de colaboración tenemos los componentes encargados de soportar el trabajo colaborativo, tanto síncrono como asíncrono. De la sincronización se encargan el servidor de sesiones junto con los agentes de colaboración, mientras que del soporte del trabajo asíncrono se encargan fundamentalmente los componentes del nivel de análisis y el registrador.

El registrador proporciona la estructura de datos básica para el funcionamiento de gran parte del framework, las historias de colaboración, las cuales consisten en un árbol donde cada rama corresponde a una serie de eventos que ocurren de forma consecutiva, siguiendo un orden temporal, donde los primeros eventos están más cerca de la raíz y los últimos ocurren en las hojas. El registrador gestiona estas historias de colaboración, que a su vez pueden integrar los eventos de varias sesiones colaborativas.

Por otra parte, el servidor de sesiones es el encargado de la gestión de las sesiones de colaboración síncronas. Cada una de estas sesiones está

relacionada en cada momento con una historia de colaboración, integrando en ella todos los eventos que ocurren en cada sesión colaborativa.

A continuación veremos en más detalle cómo FACT implementa el soporte del trabajo colaborativo síncrono, y qué papel juegan las historias de colaboración y el registrador en el soporte básico de la colaboración asíncrona.

4.2.2.1 Trabajo colaborativo síncrono

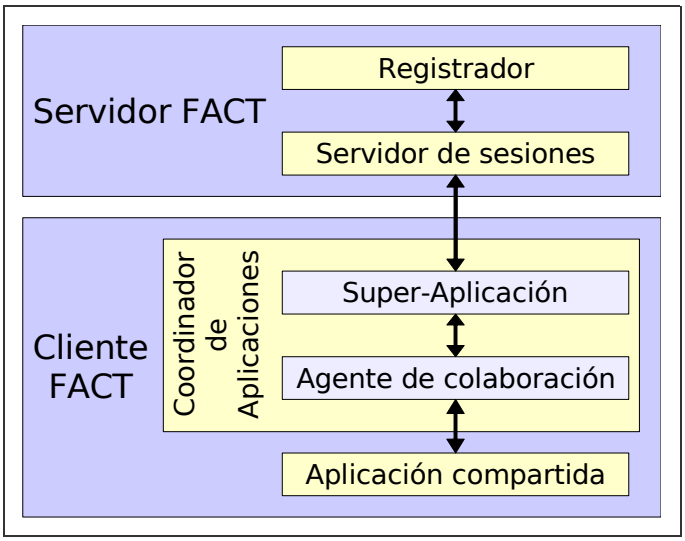

Figura 8: Flujo de información en el nivel de colaboración

El soporte del trabajo colaborativo síncrono se realiza a través de los agentes de colaboración, como hemos visto en el apartado 4.2.1.2. El flujo de información en el nivel de colaboración es el que se aprecia en la Figura 8. La información de los agentes de colaboración es coordinada en cada cliente, creando la super-aplicación, y enviada al servidor de sesiones, el cual sincroniza los eventos recibidos por los distintos clientes y finalmente los envía al registrador para su archivo.

El servidor de sesiones se encarga de gestionar las distintas sesiones de trabajo colaborativo, manteniendo sincronizados los distintos grupos de usuarios. La sincronización es un aspecto muy importante del trabajo colaborativo, ya que la ejecución de dos acciones en distinto orden puede llevar a estados distintos de la aplicación colaborativa, produciendo efectos indeseables. Esta sincronización se consigue creando un orden en los

eventos que se reciben, para que éstos puedan ser ejecutados en el mismo orden en todos los clientes de una sesión. En este mismo orden se envían los eventos al registrador, para que los inserte en la rama de la historia asociada a la sesión colaborativa. Simultáneamente a esto, los eventos son reenviados a todos los clientes, incluso al que envió el evento original, de forma que los clientes puedan conocer el orden correcto de ejecución y mantener coherente el estado compartido de las aplicaciones colaborativas.

Cada sesión tiene dos formas de funcionamiento distintas, la navegación colaborativa y el trabajo normal con las aplicaciones. La navegación consiste en desplazar la sesión a otro punto de la historia de colaboración, mientras que durante el trabajo normal con las aplicaciones se añade información a la rama actual de la historia. En este último caso también hay una navegación implícita, ya que la sesión se desplaza automáticamente siguiendo el punto de trabajo actual.

Las sesiones colaborativas han de permitir una configuración dinámica de los usuarios involucrados, de forma que se puedan añadir o eliminar clientes en cualquier momento. De estas operaciones, la más compleja ocurre cuando se añade un nuevo usuario a una sesión ya iniciada. Para solucionar este problema se realiza la incorporación en dos fases. La primera fase consiste en poner al día el cliente, enviando todos los eventos desde la raíz de la historia hasta el punto en el que se encuentra la sesión actual, dejando en espera de ejecución en el cliente todos los eventos nuevos que reciba de la sesión. La segunda fase ocurre únicamente en el cliente, donde se ejecutan los posibles eventos que estaban pendientes y se desbloquea la ejecución y producción de eventos.

En la ejecución normal, los eventos son enviados al servidor de FACT antes de su ejecución, para que este los distribuya en el mismo orden a todas las instancias. Además, como ya hemos visto en el apartado 4.2.1.2, las aplicaciones también pueden ejecutar los eventos antes de esperar la confirmación del orden correcto de ejecución, mediante el sistema de ejecución optimista. El agente de colaboración permite que estos dos modos de trabajo se mezclen, de forma que unos eventos usen ejecución optimista y otros ejecución normal. A continuación, detallaremos el algoritmo de ejecución optimista.

Este algoritmo se ejecuta en los agentes de colaboración, y tiene dos entradas de información, por un lado recibe los eventos enviados desde las aplicaciones a los agentes de colaboración, en el orden en que se han ejecutado en la aplicación, y por otro lado recibe los eventos que llegan desde el servidor, en el orden correcto en que se deben ejecutar. Este algoritmo realiza dos tareas fundamentalmente. Primero se encarga de comprobar que el orden de ejecución en las aplicaciones es el mismo que el orden que dicta el servidor. Por último, si se detecta que se han ejecutado en un orden distinto al fijado por el servidor, es decir, una ejecución fuera de orden, se deshacen los eventos que se ejecutaron incorrectamente y se ejecutan de nuevo en el orden correcto.

Para detectar si se han ejecutado los eventos en el orden correcto el agente de colaboración incorpora una cola de ejecución optimista, que se va llenando únicamente con los eventos que usan la ejecución optimista. Los eventos se van eliminado de la cola cuando se comprueba que se han ejecutado en el orden correcto. Esta tarea es muy sencilla, simplemente hay que comprobar, cada vez que llega un evento desde el servidor, que o bien la cola está vacía o este evento coincide con el evento más antiguo de la cola, en cuyo caso se elimina.

Para distinguir unos eventos de otros, el agente de colaboración los marca con un identificador global único, independiente de la instancia de la aplicación que los produjo.

Una vez que se ha detectado una ejecución fuera de orden, lo único que hay que hacer para solucionarlo es deshacer todos los eventos pendientes que están en la cola de ejecutados, ya que el evento que acaba de llegar es anterior a todos los que están en la cola de ejecución. Esto es así porque en la cola de ejecutados sólo están los eventos producidos localmente, éstos son apilados en el mismo orden en que son enviados a la sesión, y llegarán de vuelta en dicho orden. Si el evento que llega no coincide con el primero, significa que es anterior a este, y en consecuencia anterior a todos los de la cola de eventos ejecutados, que estaban pendientes de confirmar.

También será un conflicto si el servidor envía una orden de deshacer un evento y la cola de ejecutados no es vacía. Esto es así ya que la orden de deshacer un determinado evento nunca puede llegar antes de la orden de

ejecutar dicho evento. Gracias a esta premisa, cuando llega una orden de deshacer y hay eventos en la cola de ejecutados, se puede asegurar que el evento no estará en la cola de ejecutados, y que los que están en la cola se habían ejecutado antes de tiempo, ya que se pretende deshacer un evento anterior. En este caso el tratamiento del conflicto es el mismo que antes, deshacer los eventos de la cola de ejecutados.

4.2.2.2 Historias de colaboración

El registrador es el componente encargado de almacenar en estructuras de árboles las historias de colaboración. Cada nodo del árbol es un objeto que contiene información sobre el usuario y la sesión que ha creado esa información, el tipo de información, un identificador único del nodo en el árbol, una descripción del objeto, el momento en que se creo el nodo y los datos privados codificados por la aplicación que generó el contenido del nodo. La descripción del objeto no es necesario almacenarla en el nodo, ya que se puede generar dinámicamente con la llamada a un método del objeto.

En los sistemas colaborativos que han de poder acomodar nuevos usuarios en las sesiones de colaboración, es necesario un componente que almacene la información necesaria para poner al día el estado compartido de la nueva instancia de la aplicación [Chung 1998]. La mayoría de estos componentes recrean el estado actual mediante una serie de eventos, igual que en FACT, o directamente copian el estado de la aplicación. En nuestro sistema la información almacenada es mucho más rica, ya que permite guardar no sólo los pasos desde el comienzo de una sesión de colaboración hasta el final, sino todas las ramas de la interacción que han sido exploradas por varios grupos de trabajo en sesiones colaborativas distintas.

Aunque FACT no impone restricciones sobre el formato de los datos privados almacenados en los eventos generados por las aplicaciones, estos eventos han de contener la información necesaria para poder rehacerse o deshacerse en distintas instancias de la misma aplicación. De esta forma el usuario puede navegar por el árbol de la historia sin ninguna limitación, pudiendo observar las historias producidas por todas las sesiones.

El registrador también se encarga de etiquetar los eventos, de forma que podemos identificar cada punto de la historia. La identificación de los puntos

de la historia es necesaria para poder crear anotaciones asociadas a puntos concretos de una historia de colaboración, o simplemente para conocer en qué punto de la historia está cada usuario.

4.2.3 Nivel de análisis

En este nivel tenemos los distintos componentes que permiten analizar de forma dinámica las sesiones colaborativas. Decimos que es dinámico ya que el análisis se puede realizar mientras las sesiones a analizar están aún activas, y por lo tanto la información puede cambiar. Además, durante el análisis podemos influir en las sesiones colaborativas activas, o incluso en las futuras sesiones de colaboración que utilicen la misma historia de colaboración.

El análisis permite por ejemplo aprender cómo realizar una determinada tarea paso a paso, buscar errores, conocer el estado o las conclusiones de un grupo de trabajo, etc. Este análisis es a su vez colaborativo, síncrono si se comparte una sesión con más colaboradores, o asíncrono si sólo se comparten las anotaciones o las historias de colaboración.

Una característica destacable de FACT es que el análisis no supone un estado distinto a la colaboración. Esto implica que el usuario utiliza la misma interfaz para el análisis de una historia de colaboración y para compartir una aplicación, pudiendo pasar de una tarea a otra sin necesidad de ninguna acción especial. Por ejemplo un usuario puede estar viendo una partida de ajedrez, y en cualquier punto de la partida puede proponer una alternativa a la jugada que están realizando los jugadores. Lo único que tiene que hacer es mover una pieza del tablero, igual que si estuviera jugando él mismo la partida. La alternativa propuesta creará una nueva rama en la historia de colaboración, lo que permitirá a otras personas ver las distintas alternativas a la jugada original. Además se pueden crear anotaciones asociadas a un punto de la historia, lo que permitirá comentar la jugada, sugerir alternativas, etc.

4.2.3.1 Servidor de anotaciones

El servidor de anotaciones se encarga de almacenar las anotaciones y proporcionarlas al resto de usuarios. Cada anotación tiene un tipo, un

nombre y está asociada a uno o varios nodos de la historia de colaboración. La asociación con la historia permite por ejemplo que los usuarios puedan ver las anotaciones relevantes al punto de la historia donde se encuentran.

Las anotaciones son objetos y pueden ser de cualquier tipo, ya que serán las distintas herramientas de análisis de FACT, como el cliente de anotaciones o el navegador, los encargados de crear e interpretar dichos objetos.

Además de mantener las anotaciones, el servidor también implementa un mecanismo de notificación, lo que permite al cliente de anotaciones mostrar siempre a los usuarios la última versión de las anotaciones, informándole cuándo ha cambiado una anotación o cuándo hay nuevas anotaciones que le puedan interesar.

4.2.3.2 Editor de anotaciones

El editor de anotaciones se puede usar como visor de las anotaciones producidas por el resto de usuarios y como editor de las mismas. Las anotaciones pueden ser objetos de cualquier tipo, aunque actualmente sólo está implementada la funcionalidad de un editor de documentos con formato, por lo que las anotaciones serán documentos.

Este editor forma parte de las herramientas colaborativas de análisis, aunque en este caso el grado de acoplamiento entre distintas instancias del editor en la misma sesión es mínimo. Esto se debe a que los usuarios no comparten las anotaciones mientras se editan, por lo que varios usuarios de la misma sesión pueden editar o ver anotaciones diferentes. Lo que si ven todos los usuarios de la misma sesión colaborativa es la lista de anotaciones relevantes en el punto de la historia en que se encuentran. Si el usuario no decide ver alguna anotación en concreto el sistema le mostrará la primera de la lista de anotaciones.

4.2.3.3 Navegador de la historia

Hay dos mecanismos de navegación relevantes para el usuario, lo que nos lleva a dos versiones del navegador de la historia. La primera de ellas muestra la historia de colaboración, permitiendo que los usuarios se muevan por ella simplemente pinchando con el ratón en el punto al que quieren ir. La

otra forma de movernos por la historia es mediante un navegador que sólo nos muestra el punto actual de la historia en el que estamos y nos permite movernos hacia atrás o escoger una de las ramas que se abren en el punto actual de la historia.

Cuando el usuario navega por la historia todos los usuarios de la misma sesión lo hacen simultáneamente, ya que se modifica el punto de la historia donde se encuentra la sesión. La navegación se consigue deshaciendo o rehaciendo los eventos necesarios para que las aplicaciones actualicen su estado con el del punto de destino.

Si por algún motivo el usuario quiere navegar sin el resto de usuarios de la misma sesión, simplemente tiene que cambiar a una sesión privada. Al crear la sesión privada se puede elegir entre partir del punto de la sesión anterior, o cambiar al inicio de cualquier historia de colaboración. El usuario puede cambiar de sesión en cualquier momento, con lo cual puede alternar de una forma sencilla entre navegar junto con otros usuarios o hacerlo independientemente.

La versión del navegador que muestra la historia requiere una gran cantidad de información, por lo que para disminuir la latencia cada instancia del navegador implementa una caché de la historia de colaboración que se está mostrando. Esta caché representa la historia explorada, que se actualiza automáticamente cuando cambia la copia original de la historia de colaboración. Por otro lado, la versión reducida requiere una mínima cantidad de información y es por lo tanto ideal en situaciones en las que no se dispone de un gran ancho de banda, el árbol de la historia no es relevante, o simplemente hay poco espacio para la interfaz de usuario.

Los navegadores implementan también un mecanismo de navegación asistida, que permite por ejemplo seguir el camino que conduce a la posición actual de una sesión concreta o seguir un camino marcado previamente. Los caminos se definen con el propio navegador, indicando el inicio y final del mismo. Una marca especial de camino es el camino principal de la historia. Este camino principal ayuda al navegador a escoger que rama seguir cuando el usuario avanza por la historia y hay varias alternativas posibles. Los caminos están asociados a la historia, como información adicional, comportándose como anotaciones. Por lo tanto, será el servidor de

anotaciones el encargado de almacenar esta información. Para añadirla, el navegador sólo tiene que asociar la definición del camino a la historia, mediante un objeto que represente dicho camino.

4.2.4 Gestión del sistema

Hasta ahora hemos visto los distintos niveles conceptuales de FACT. La gestión del sistema es un nivel transversal, que se relaciona directamente con el resto de niveles y engloba todas las tareas relacionadas con la gestión del sistema.

En este nivel tenemos todas las tareas de configuración y mantenimiento del sistema, aunque destacaremos en este apartado la gestión de usuarios y permisos, dada su importancia en el trabajo colaborativo con la herramienta.

Otro aspecto importante de la gestión del sistema es la persistencia de los datos y configuraciones. Es necesario poder guardar la información de las historias, anotaciones, permisos y usuarios, de forma que se puedan recuperar estos datos en casos de fallos del servidor. También puede ser interesante guardar las configuraciones de las sesiones, aunque esta información no es imprescindible pero ayuda, en caso de un fallo, a poder restaurar los grupos de trabajo en el menor tiempo posible.

En la versión actual de FACT la persistencia se consigue guardando los datos en ficheros en el sistema de archivos del servidor. Los datos de las anotaciones y las historias de colaboración se guardan usando el mecanismo de serialización de Java, ya que de esta forma se puede utilizar el mismo mecanismo para el almacenamiento persistente y para la comunicación de estos mismos datos entre los clientes y el servidor. Además de ser compatible con las aplicaciones colaborativas de FACT, es el mecanismo más eficiente y que impone menos requisitos a los datos. Para el resto de datos de gestión se utiliza la persistencia XML, ya que ofrece mayor flexibilidad y permitirá mayores grados de compatibilidad entre distintas versiones del framework.

En cuanto a la gestión de permisos, en FACT se ha creado un sistema sencillo basado en roles y unidades de gestión, o ámbitos, jerarquizados. Las unidades básicas son las sesiones, las historias de colaboración y el sistema

tutor, de forma que sobre una historia de colaboración puede haber varias sesiones, y el sistema tutor pueda albergar varias historias de colaboración.

De los ámbitos anteriores, el sistema tutor hace referencia a todos los componentes de un servidor de FACT y la historia de colaboración denota una única historia. En el caso de la sesión este ámbito se refiere al trabajo realizado durante una sesión, que no tiene por qué continuar activa. Esto incluye la parte de la historia y las anotaciones creadas durante la sesión.

A su vez, en cada actividad de tutorización, sea del ámbito que sea, podemos tener dos roles distintos, usuario de la aplicación y observador. Además, tanto los usuarios como los observadores pueden ser a la vez administradores de la actividad en cuestión, por lo que existen cuatro combinaciones diferentes de tres posibles roles.

Estos tres roles también están jerarquizados. Así pues, si un usuario tiene asignado el rol de usuario de la aplicación, este tendrá más privilegios que uno del rol observador. A estos permisos se añadirán los que otorga el rol de administrador en caso de pertenecer a este grupo. Las distintas acciones que puede realizar cada usuario dependerán del ámbito donde desempeña ese papel. Además, un usuario puede desempeñar distintos roles en cada actividad de tutorización. Por ejemplo, un mismo usuario puede ser usuario de las aplicaciones en una sesión y puede desempeñar el papel de observador para una determinada historia de colaboración. También puede ocurrir que no tenga permisos de ningún tipo en un ámbito concreto.

Rol \ Ámbito	***Sesión***	***Historia***	***Sistema***
Observador	-Explorar -Observar en tiempo real	-Navegar -Unirse como observador de las sesiones vinculadas -Crear sesiones	-Unirse como observador de cualquier historia o sesión del sistema
Usuario	-Crear anotaciones -Trabajar -Navegación síncrona	-Navegar -Crear anotaciones en la historia -Unirse como usuario de las sesiones vinculadas -Definir caminos en la historia -Crear sesiones	-Unirse como usuario de cualquier historia o sesión del sistema
Administrador	-Invitar/excluir a otros -Borrar la sesión	-Administrar cualquier sesión vinculada a la historia -Especificar el camino principal en la historia	-Crear o borrar historias o sesiones -Administrar todas las sesiones e historias

Tabla 2: Permisos de los distintos roles en cada ámbito del sistema

En la Tabla 2 se muestran las acciones más relevantes que puede realizar un usuario en cada actividad, dependiendo del rol asignado. En el caso de las sesiones no se ha de confundir el ámbito de sesión, sobre el que el usuario tiene unos determinados permisos, con la sesión propia a la que está conectado el usuario. Por ejemplo, un observador puede tener derecho para navegar por el conjunto de la historia correspondiente al ámbito de una sesión determinada, aunque no esté conectado a dicha sesión. Si el observador quiere ver en tiempo real lo que ocurre en una sesión, entonces

sí se conectará a la sesión, aunque no podrá realizar ninguna acción al ser un observador, siendo en esta caso la sesión propia igual a la observada.

Las acciones están restringidas al ámbito sobre el que se definen. Por ejemplo, un usuario de una historia puede crear nuevas sesiones, pero sólo si están asociadas a alguna historia donde tenga permiso para ello. En el caso de un usuario cuyos permisos estén restringidos al ámbito de una única sesión, sin permisos sobre el total de la historia u otras sesiones, éste sólo podrá crear nuevas ramas de la historia si parten de la posición actual o de un nodo perteneciente al trabajo anterior de la sesión. Las anotaciones o la navegación también sufren esta restricción. En el caso de la navegación, puede ocurrir que un usuario tenga permiso para guiar a otros, gracias a la navegación síncrona, a una zona de la historia donde no pueden ir por sí mismos.

Para administrar los distintos permisos sobre cada ámbito un usuario puede ser además administrador. Normalmente los creadores de las historias y las sesiones también serán administradores de las mismas. Por ejemplo, los profesores pueden ser usuarios del sistema y administradores del mismo, por lo que automáticamente tendrán derechos de administración en todos los ámbitos.

La única limitación de los administradores a la hora de asignar un rol a un usuario es que no podrá superar el nivel de permisos que tiene el propio administrador en la actividad que administra. Por ejemplo, un observador de la historia puede crear una sesión, de la que será su administrador, sin embargo, este administrador no podrá dar permisos de usuario de aplicación sobre su propia sesión, por lo que en principio la sesión estará limitada a la observación, salvo que permita que alguien con permiso de usuario de aplicación en la historia se conecte a esta sesión.

4.3 Creación de aplicaciones

FACT está orientado a la creación de aplicaciones que requieran integrar colaboración síncrona y asíncrona. Uno de los campos que mejor se adaptan a este esquema es el de la enseñanza, donde podemos encontrar multitud de aplicaciones que pueden aprovechar esta capacidad de trabajo colaborativo, como hemos visto en el apartado 2.1. Para permitir este tipo de

aplicaciones de enseñanza hemos creado un modelo representativo del uso y funcionamiento de las mismas, llamado Aprendizaje Colaborativo Guiado, descrito en el capítulo 3, aunque FACT no está limitado a la creación de aplicaciones que sigan este modelo, ya que permite crear aplicaciones colaborativas aunque no sean de aprendizaje.

A la hora de crear aplicaciones que se puedan integrar en FACT hay que tener en cuenta que el resultado serán aplicaciones colaborativas. De las cuestiones técnicas relacionadas con la colaboración, y que hemos visto en el apartado 2.2, se encarga el framework, pero quedan otras, como la facilidad de uso por parte de varios usuarios, que hay que tener en cuenta al diseñar la aplicación colaborativa.

Una de las decisiones más importantes al respecto es decidir cuál será el estado compartido de la aplicación. Para mejorar la facilidad de uso de la aplicación es preferible que el estado compartido sea el mínimo y con el mayor nivel de abstracción posible, así ganaremos en flexibilidad, permitiendo que cada usuario pueda personalizar su aplicación sin afectar al resto. Al aumentar el nivel de abstracción será más sencillo codificar las acciones para que representen mejor la intencionalidad de los usuarios. Esto será fundamental para que no se produzcan resultados inesperados cuando ocurra un conflicto, al producirse varias acciones simultáneamente, por parte de distintos usuarios.

Desde el punto de vista del framework, el único requisito es que la aplicación tenga un método donde recibirá como argumento el agente de colaboración asociado a esa instancia. Gracias a este agente la aplicación podrá enviar los eventos que necesita comunicar al resto de instancias de la sesión colaborativa con el fin de mantener el estado compartido de la aplicación. Estos eventos pueden ser de cualquier tipo, siempre y cuando sean serializables, es decir, se puedan transmitir por red.

Hay dos modelos de comunicación de eventos entre las aplicaciones y el agente de colaboración. La más simple requiere que la aplicación envíe todos los eventos al agente y que escuche los eventos enviados por el servidor en dos métodos, uno para recibir eventos que tiene que ejecutar localmente, y otro donde recibe los que tiene que deshacer. Este modelo

será apropiado para aplicaciones sencillas que no presenten un flujo complejo de eventos.

Para aplicaciones más complejas se recomienda la utilización del modelo basado en acciones, idéntico al utilizado normalmente para simplificar la programación de las interfaces de usuario. En este caso son los propios eventos los que tienen la lógica de la acción y la información necesaria para identificar el objeto sobre el que han de actuar. El requisito en este caso será que las propias acciones tengan los métodos para ejecutar y deshacer la acción.

El ciclo de vida de la acción es el siguiente:

1. Se crea un objeto acción asociado a distintos componentes de la interfaz gráfica. Este mecanismo de asociación es muy habitual en el desarrollo de interfaces gráficas en Java, y está soportado en la mayoría de entornos de programación, de forma que esta tarea se puede realizar gráficamente, sin necesidad de programar. Además, esta instancia de la acción recién creada se podrá reutilizar las veces que sea necesario, por ejemplo para ligar la misma acción a un botón de acceso rápido y a un elemento de menú.
2. Cuando el usuario active el componente gráfico que produce la acción, ésta se ejecutará. En el caso de las acciones colaborativas, la ejecución real no ocurre inmediatamente, sino que la acción será enviada al servidor de FACT mediante el agente de colaboración.
3. La acción se pasará desde el servidor a los clientes para que sea procesada. En este caso el agente de colaboración llamará a los métodos que ejecuten o deshagan realmente la acción.

El ciclo anterior describe la ejecución normal de las acciones. También se puede conseguir una ejecución optimista, indicándole al agente de colaboración que ejecute las acciones primero en local.

Cuando hay pocos usuarios en la sesión colaborativa es raro que ocurran conflictos durante la ejecución optimista, lo que permite que mejoremos el tiempo de respuesta de las aplicaciones. A veces también es necesario elegir la ejecución optimista cuando los componentes gráficos que queremos reutilizar en nuestra aplicación no permitan capturar los eventos antes de la

ejecución de la acción que desencadenan, o simplemente porque estamos reutilizando una aplicación existente y resulta más sencillo no alterar el flujo de ejecución de eventos.

Si la aplicación que estamos portando a FACT tiene previsto un mecanismo de deshacer las últimas acciones, por ejemplo utilizando el "UndoManager" de Java, será muy sencillo adaptarla al framework, ya que estará implementado el mecanismo de deshacer y rehacer los distintos tipos de acciones. En la mayoría de los casos simplemente es necesario enviar estas acciones al agente de colaboración. Además no es necesario almacenar los eventos en local, ya que FACT almacena la información necesaria en las historias de colaboración, y la envía a las aplicaciones cuando la necesitan.

Una característica que tienen que cumplir los eventos o acciones enviados a través del agente de colaboración es que se han de poder ejecutar de nuevo en otras instancias de la aplicación. Esto será un problema si se utilizan punteros para identificar los componentes de la aplicación relacionados con los eventos colaborativos, ya que los punteros únicamente son identificadores válidos en la instancia que origina los eventos. Para solucionar este problema se ha de traducir los punteros en un identificador válido para todas las instancias de la aplicación.

Algunos autores proponen mecanismos automáticos que pueden funcionar relativamente bien si las aplicaciones compartidas tienen la misma estructura de componentes gráficos en todo momento, [Li 1999], [Marsic 1999]. Sin embargo el problema resulta demasiado complejo para objetos en general, donde no existe una estructura jerárquica, como ocurre con los componentes gráficos. En FACT la mayoría de eventos están relacionados con el modelo de datos y no con los objetos gráficos. Por este motivo se ha elegido crear un mapa, que gestiona la propia aplicación, y que permite dar un nombre a los objetos de la aplicación que lo requieran. A este mapa se puede acceder a través del agente de colaboración y permite a la aplicación transformar los punteros en nombres, y así enviar eventos colaborativos que puedan ser reproducidos sin problemas en otras instancias.

También será muy útil que los eventos enviados implementen un método para convertir el evento en una cadena de texto que describa la acción que

produce, de esta forma el navegador de la historia podrá mostrar al usuario dicha cadena cuando se visualice la historia de colaboración o la plantilla de interacción.

Un último problema que hay que tener en cuenta a la hora de crear la aplicación colaborativa es el tratamiento de los recursos compartidos. En primer lugar, éstos deberían estar centralizados, o al menos ser accesibles con una dirección única, que sea independiente de la instancia de la aplicación que requiera dicho recurso. Una buena forma de hacerlo es con WebDAV[39]. También es muy sencillo adaptar la aplicación a este protocolo gracias a Slide[40], la cual permite acceder a recursos webdav a través de su URL como si fueran ficheros normales.

El mayor problema del tratamiento de los recursos compartidos aparece cuando es necesario modificarlos, ya que si no se trata adecuadamente todas las instancias de la aplicación podrían intentar modificar el recurso simultáneamente. La solución sin embargo es muy sencilla. En la mayoría de los casos se puede resolver limitando la modificación del recurso a la instancia que produjo inicialmente el evento. Este es uno de los motivos por los que el agente de colaboración añade a los eventos colaborativos información sobre la aplicación y el usuario que ha creado cada evento.

4.4 Pruebas realizadas

FACT es una herramienta orientada al desarrollo de aplicaciones con soporte de aprendizaje colaborativo guiado. Para probar que el framework permite crear dichas aplicaciones con un mínimo esfuerzo de desarrollo, respecto a crear una aplicación similar monousuario, se han elegido y portado varias aplicaciones ya existentes.

En este apartado veremos cómo eran las aplicaciones antes de ser portadas a FACT y explicaremos los cambios que fueron necesarios para su incorporación al framework. El beneficio de dicha incorporación se puede apreciar claramente al comparar la aplicación original con el resultado, una vez que soporta su uso colaborativo y se integra en el framework. Como veremos a continuación, el esfuerzo necesario para portar una aplicación a

[39] En http://webdav.org se puede encontrar la especificación del protocolo y otros recursos.

[40] La biblioteca Slide se encuentra disponible en http://jakarta.apache.org/slide

FACT es el equivalente al esfuerzo necesario para implementar un mecanismo de deshacer y rehacer las acciones del usuario en dicha aplicación.

4.4.1 ColPuzzle

Una de las aplicaciones sencillas que se ha portado a FACT es ColPuzzle. Esta aplicación está basada en el código de Mihai Munteanu[41], que consiste en una Applet Java que presenta una rejilla de 16 imágenes, correspondientes a los 16 trozos en los que se ha dividido una imagen mayor, Figura 9. El objetivo del juego es mover el cuadrado negro para colocar las imágenes en su posición correcta.

Figura 9: Aplicación Puzzle original

Los cambios necesarios para portar esta aplicación a FACT se limitaron a crear unas 30 líneas de código, las necesarias para añadir una variable que almacenase el agente de colaboración, los métodos que reciben los eventos desde FACT y la creación de dos clases que representan los eventos de mover el cuadrado de una posición a otra y el evento de comienzo con un puzle nuevo.

[41]La aplicación Puzzle original se puede encontrar en: http://hp-h.com/p/munte/java_puzzle

Lo primero que se hizo fue localizar el método que movía el cuadrado negro. Este método se sobrecargó por otro que simplemente creaba el evento de mover el cuadrado y se lo pasaba al agente de colaboración. Cuando el evento llegaba de nuevo a la aplicación se llamaba al método original para mover realmente el cuadrado. Si lo que llegaba era un evento de deshacer se llamaba al método original de mover, pero cambiando el sentido del movimiento.

La parte más compleja resultó ser la creación de un puzle nuevo, ya que esto es un proceso aleatorio, y todas las instancias de la aplicación debían crear el mismo puzle. Para ello se creo un tipo de evento colaborativo que almacenaba el estado del puzle antes y después de crear el nuevo. De esta forma, cuando se pulsa el botón de crear el puzle nuevo se decide las posiciones que ocuparán las imágenes y se enviará un evento colaborativo con las posiciones actuales y las nuevas, las instancias simplemente tendrán que fijar las imágenes en dichas posiciones, cuando reciban el evento colaborativo. En este caso es necesario almacenar el estado anterior a la creación del puzle para poder implementar la operación de deshacer.

También se han realizado otros cambios para convertir el Applet Java en aplicación y se mejoró para permitir usar imágenes distintas a la proporcionada originalmente, dividiéndolas automáticamente para adaptarlas a puzles de diferentes tamaños.

4.4.2 ChessEdu

Para la creación de ChessEdu se ha partido de TkChess, creado por James Aspnes[42]. En TkChess lo único que está escrito en Java es la interfaz de usuario, en forma de Applet, Figura 10, mientras que el resto del programa está escrito en C y tcl/tk. De este programa sólo se ha aprovechado la parte escrita en Java, eliminando el motor que permitía jugar contra el ordenador. También fue necesario escribir en java el sistema que implementaba las reglas del juego, ya que éste estaba escrito en C originalmente.

[42]TkChess está disponible en http://pine.cs.yale.edu/java/chess.html

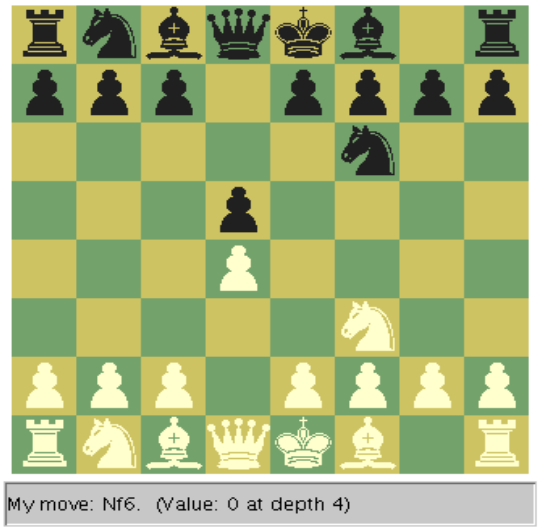

Figura 10: Applet para jugar al ajedrez con tkChess

Por lo tanto se ha partido realmente de un tablero de ajedrez al que se ha incorporado las reglas del juego y el código para convertirlo en una aplicación colaborativa. En total se añadieron o modificaron unas 36 líneas de código, sin contar las que fueron eliminadas de la aplicación original, ni las que implementan las reglas del juego. Las modificaciones necesarias fueron muy parecidas al caso del ColPuzzle. Se sustituyó el método que mueve las fichas por uno que crea un evento representando dicho movimiento y lo envía a FACT a través del agente de colaboración. Cuando la aplicación recibe el evento de mover una ficha simplemente llama al método correspondiente de la aplicación original.

En esta aplicación lo más complejo era la orden de deshacer los eventos, ya que, por ejemplo, cuando se mueve una ficha otra puede desaparecer. La solución consiste en transmitir el estado del tablero en cada evento. Ésta no es la mejor solución, pero sí la más sencilla de implementar, ya que la aplicación original transmitía también el estado completo del tablero al cliente Java para indicar el resultado de los movimientos, y por lo tanto ya estaban programados los métodos que guardaban y restauraban el estado de la aplicación en una representación compacta.

El resultado de portar este tablero a FACT es una aplicación de ajedrez que permite jugar en red, grabar partidas con su evolución paso a paso, incluyendo movimientos alternativos, comentarios, etc. Esto permite usar

ChessEdu para cursos de enseñanza del ajedrez, donde el tutor u otros alumnos pueden monitorizar las partidas en tiempo real, unirse a ellas, analizar movimientos anteriores de cualquier partida, activa o no, etc.

4.4.3 Editor WYSIWYG colaborativo

Para esta aplicación se ha seguido una estrategia diferente a las anteriores, en este caso se ha construido un editor colaborativo partiendo de componentes estándar, en lugar de a partir de una aplicación existente.

En este caso la tarea de construir la aplicación colaborativa presenta dificultades añadidas, ya que el editor ha de admitir texto con diferentes estilos, y el componente utilizado, JtextPane, representa los cambios en el texto mediante un evento con un gran número de dependencias con objetos creados por la aplicación, y que por lo tanto no puede transmitirse de una instancia a otra. El problema de las dependencias se debe a que el modelo de datos de este componente usa objetos para representar cada elemento, caracteres, párrafos, estilos, etc., y los eventos que representan los cambios usan punteros a estos objetos, que si fueran transmitidos dejarían de ser una referencia a una parte del modelo, para pasar a ser una copia independiente.

Una estrategia plausible ante este tipo de problemas consiste en capturar eventos de más bajo nivel, como movimientos del cursor, o tecleo, que pueden ser reproducidos posteriormente en otras instancias, en lugar de trabajar directamente con el modelo de datos. Sin embargo esto requiere identificar todas las posibles causas de modificación del texto, analizarlas, y transmitirlas con la codificación apropiada para su correcto funcionamiento como aplicación colaborativa. Por ejemplo, el usuario podría pegar un trozo de texto con formato, proveniente de una página Web o de otro documento. Además el texto pegado puede estar codificado en multitud de formatos.

La solución elegida ha sido analizar los cambios en el modelo de datos del editor, de esta forma se puede crear un objeto que representa el cambio sin dependencias con otros objetos del editor, y por lo tanto reutilizables en otras instancias. El código necesario para analizar los cambios en el modelo de datos, creando un evento colaborativo, y que es independiente de la implementación del componente de edición, JtextPane, o de si el texto está en formato HTML, RTF, etc., más los métodos que rehacen y deshacen estos

eventos en las distintas instancias de la aplicación, requiere unas cien líneas de código Java. La independencia de los formatos y del componente utilizado para el editor se ha conseguido gracias a que la biblioteca estándar de Java ofrece un API[43] que posee un grado de abstracción muy alto, orientado a incorporar las tecnologías de accesibilidad en esta plataforma.

Este editor colaborativo produce tres tipos de eventos diferentes, de borrado, inserción y modificación. Este último es el más completo, ya que equivale a un borrado y una inserción. Mientras que los eventos de borrado e inserción son similares, la única diferencia es que el método de deshacer una inserción es el de rehacer en el borrado, y viceversa. Para representar cada cambio hay que tener en cuenta que éste puede afectar a una región de texto con diferentes estilos, por lo que los cambios son en realidad una lista de cambios más pequeños, uno por cada región de texto con las mismas características.

El hecho de representar de la forma más precisa posible la intención del usuario permite crear aplicaciones colaborativas que funcionen mejor en todos los casos, incluso cuando hay un conflicto porque un usuario ha realizado una acción después que otro usuario y antes de que el estado de su aplicación se actualice, como hemos visto en el apartado 2.2.2.1.1. En el caso del editor, si dos usuarios escriben casi simultáneamente las letras A y B, y además el servidor de FACT determina que primero se ha escrito A y después B, lo razonable sería deducir, al ver la historia, que los dos usuarios verán el texto AB. Sin embargo, si los eventos se basan en indicar posición y letra, ambos terminarán con las letras BA, o incluso con solo B, ya que primero se escribirá A y después se escribirá B en la posición que ahora ocupa A, desplazando o borrando la letra A, según sea la implementación.

Para solucionar esto se modificó la forma en la que se crean los eventos de introducción de texto, de forma que éstos no incluyan la posición absoluta donde insertar las letras. Una solución es modelar la intención del usuario con dos eventos distintos, uno para representar cuando el usuario desea cambiar el cursor a otro sitio, en cuyo caso el desplazamiento del cursor sería relativo a su posición original. El otro evento sería el de introducción de texto, que lleva implícito un avance en la posición del cursor. Se puede

[43] El paquete Java que proporciona el API de accesibilidad es javax.accessibility

comprobar que esta solución representa mejor la intencionalidad del usuario, minimizando la información transmitida.

La solución que se ha adoptado en la implementación actual simplemente transmite el texto con una posición relativa al final del documento. De esta forma se respeta mejor la intencionalidad que en la implementación original, y además no requiere introducir el concepto de cursor. La ventaja de esta solución frente a la anterior es que no limita el editor colaborativo a un solo cursor, y cada usuario puede tener el suyo propio.

Otra mejora incorporada al editor es la posibilidad de añadir componentes gráficos y campos de formulario. Dichos componentes en la versión actual se comportan como un único carácter, y sus atributos, en lugar de especificar el estilo del texto, especifican el componente gráfico y sus propiedades. Para conseguir reaccionar a los cambios en los componentes insertados, y no sólo en el texto o su estilo, se utiliza un sistema de dependencias basado en la especificación JavaBeans[44], que permite a la aplicación reaccionar a los cambios en las propiedades de los componentes, y transmitir dichos cambios de forma colaborativa al resto de instancias.

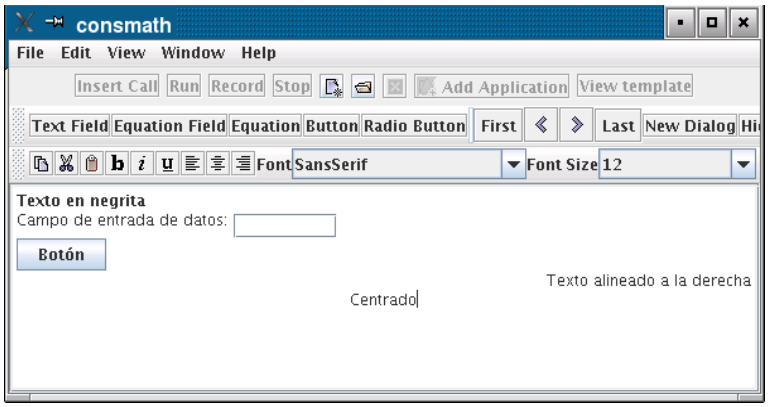

Figura 11: Editor colaborativo en ConsMath

Este componente, una vez mejorado para ser incorporado en FACT, es uno de los elementos principales en la interfaz de usuario de ConsMath, Figura 11. En ConsMath se ha añadido además la capacidad de crear dependencias entre los propios componentes, mediante funciones y

[44] La especificación JavaBeans se puede obtener en: http://java.sun.com/products/javabeans

expresiones Matemáticas, junto con la posibilidad de crear dinámicamente diálogos dentro del propio documento.

Capítulo 5. Diseño por demostración de aplicaciones colaborativas

En el capítulo anterior hemos estudiado FACT y hemos visto cómo se crean nuevas aplicaciones con este framework. Aunque el framework simplifica enormemente el desarrollo de una aplicación colaborativa, no ocurre lo mismo con los conocimientos necesarios para crear dichas aplicaciones, ya que siguen siendo necesarios amplios conocimientos de programación. En el caso de las aplicaciones educativas esto significa en la mayoría de los casos la creación de equipos de trabajo que integren pedagogos, programadores, etc.

Para simplificar este proceso al máximo se ha creado una nueva herramienta, DFACT, que permite diseñar por demostración nuevas aplicaciones colaborativas a partir de aplicaciones existentes basadas en FACT. DFACT tiene dos modos de funcionamiento, modo diseño y modo interpretación. En el modo diseño permite especificar el comportamiento de la aplicación que se está diseñando, mientras que en el modo interacción ejecuta la aplicación diseñada. Esta herramienta es realmente un extensión de FACT, ya que aprovecha la infraestructura de este framework para dar soporte al diseño y ejecución de nuevas aplicaciones. Como consecuencia de esto, DFACT incorpora todas las características de FACT.

Según hemos visto en el apartado 2.4.1, la programación por demostración consiste en especificar interactivamente cómo responden las aplicaciones a las distintas acciones del usuario o a los eventos generados por las propias aplicaciones. Esto se consigue mediante la simulación por parte del diseñador de las acciones de usuario que se desean modelar, y la especificación de las distintas respuestas de las aplicaciones ante dichas acciones.

El mecanismo de programación por demostración empleado en DFACT es similar al usado en CTAT, [Koedinger 2004], una herramienta desarrollada en la universidad de Carnegie Mellon para simplificar la creación de tutores inteligentes, y que hemos visto en el apartado 2.4.1.2. DFACT, además de poder crear tutores similares a CTAT, permite diseñar por demostración aspectos interactivos arbitrarios de las aplicaciones, en lugar de limitarse a diálogos sencillos que responden ante acciones de los alumnos mediante mensajes de texto, e incorpora aspectos colaborativos.

DFACT modela las nuevas aplicaciones partiendo de aplicaciones existentes, por lo tanto el conjunto de aplicaciones base, sobre el cual el diseñador realiza las demostraciones, es un elemento clave en este proceso. Las aplicaciones base han de cumplir los mismos requisitos necesarios para la integración de las aplicaciones colaborativas en FACT, los cuales hemos estudiado en el apartado 4.3. Por lo tanto, cualquier aplicación colaborativa que funcione en el framework puede usarse como base del diseño por demostración. Las demostraciones, con las distintas respuestas de las aplicaciones a las acciones del usuario, conforman el comportamiento interactivo de la nueva aplicación, al que llamaremos plantilla de interacción. Esta plantilla es similar a un sistema de reglas que se disparan ante situaciones determinadas, cuya acción es la modificación del entorno, lo cual posibilita a su vez que se active un nuevo conjunto de reglas. Las plantillas de interacción representan realmente una nueva aplicación colaborativa que puede ser ejecutada en DFACT como otra aplicación más.

Por ejemplo, si usamos el editor de documentos descrito en el apartado 4.4.3, podemos crear un sencillo tutor, que realice al alumno una serie de preguntas, ayudándole en caso de que se equivoque con las respuestas. Para programar este tutor el desarrollador sólo tiene que escribir cada pregunta en el editor de textos, e interactuar con la aplicación para ir simulando la introducción por parte del alumno de repuestas correctas e incorrectas. Cada vez que escribe una respuesta el diseñador escribe la reacción del sistema tutor, ofreciendo las explicaciones oportunas en caso de fallo, o la siguiente pregunta en caso de acierto. El resultado de esta interacción es un árbol con las alternativas correctas e incorrectas, y la respuesta del editor a cada una de estas alternativas. Este ejemplo sólo es factible si el conjunto de respuestas a cada pregunta es reducido, ya que la aplicación base es un editor de textos, que no está diseñado para hacer este tipo de tareas.

En base a esta herramienta de diseño, y para facilitar específicamente el desarrollo por demostración de aplicaciones educativas con un alto contenido en matemática simbólica, se ha desarrollado una nueva herramienta especializada, llamada ConsMath. Esta última herramienta permite diseñar ejercicios interactivos complejos y reutilizables, sin necesidad de programar.

ConsMath permite además la generación automática de problemas que pueden formar parte de ejercicios más complejos.

A continuación, estudiaremos en detalle DFACT. Por último veremos como se ha utilizado esta herramienta para crear ConsMath, lo que permite aplicar la programación por demostración a la creación de cursos y problemas de Matemáticas.

5.1 DFACT

El componente principal de DFACT es el agente de seguimiento. Éste es el responsable de asistir al diseñador en el diseño de las aplicaciones. Para representar el diseño de dichas aplicaciones se han de crear una serie de plantillas de interacción, cuyo número dependerá de la complejidad del diseño. Estas plantillas de interacción son elementos de diseño reutilizables, de forma similar a lo que ocurre con una biblioteca de funciones en programación estructurada.

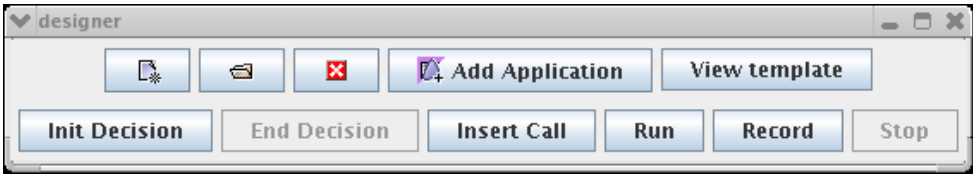

Figura 12: Agente de seguimiento. Interfaz para el diseño de aplicaciones

La interfaz de usuario del agente, Figura 12, sólo se muestra al diseñador, para controlar el proceso de diseño de las plantillas. Durante la ejecución de las aplicaciones el agente permanece oculto, encargándose únicamente de interpretar estas plantillas, que representan el código de la aplicación, y de realizar un seguimiento de las acciones del usuario.

DFACT por sí solo no permite diseñar nuevas aplicaciones, sino que requiere otras aplicaciones basadas en FACT que sirvan de base para el nuevo diseño. Como ocurre en todos los sistemas de programación mediante demostración, se necesita un soporte básico para poder realizar las demostraciones. Este proceso general se puede resumir con la fórmula ilustrada en la Figura 13.

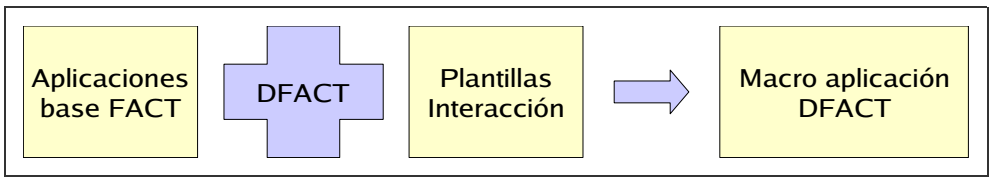

Figura 13: Proceso de diseño en DFACT

La diferencia fundamental con el resto de sistemas de programación por demostración, cuyos ejemplos más relevantes hemos visto en el apartado 2.4.1, es que hemos independizado el proceso de programación de las aplicaciones sobre la que se realizan las demostraciones.

En DFACT también podemos tener varias aplicaciones base basadas en FACT que actúan de forma coordinada en lugar de una sola aplicación base. Siguiendo con el ejemplo del examen con el editor, podemos crear un ejercicio sobre ajedrez, que muestre al alumno una posición de una partida, usando ChessEdu, y le pregunte al alumno por el siguiente movimiento a realizar. Se puede crear el ejercicio de tal forma que se muestre una pregunta o comentario, en el editor de textos, cuando el alumno realice un movimiento en el tablero, o se le pida resolver un ejercicio relacionado más sencillo.

Gracias a esta independencia de la aplicación base, DFACT se puede considerar una herramienta que al integrarse con una aplicación basada en FACT da lugar a otra herramienta para la construcción de ejercicios basados en aprendizaje colaborativo guiado.

Como podemos ver en la Figura 14, DFACT aprovecha gran parte de la infraestructura de FACT. Por ejemplo, las historias de colaboración se utilizan como formato para guardar las plantillas de interacción en el registrador. Básicamente la información contenida en las plantillas de interacción es del mismo tipo que el de las historias de colaboración normales, si no tenemos en cuenta la información extra introducida por el agente de seguimiento. Este agente sólo inserta la información mínima necesaria para poder interpretar posteriormente las acciones del usuario, y seguir así las plantillas de interacción programadas. También se aprovecha el sistema de anotaciones, esta vez como mecanismo para crear la documentación de las plantillas de interacción.

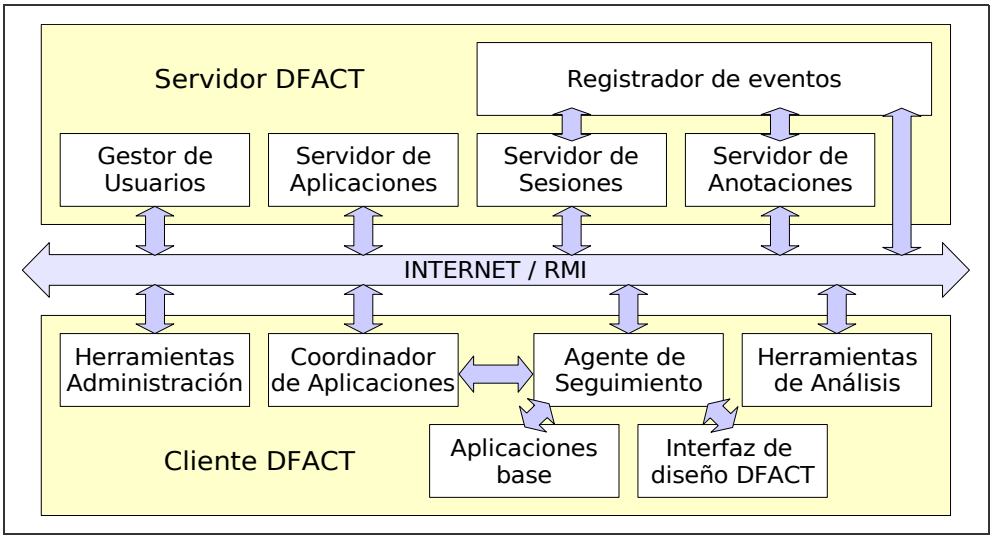

Figura 14: Arquitectura de DFACT

Una ventaja añadida de aprovechar la estructura de las historias de colaboración es que, al estar diseñadas para que varios grupos de usuarios colaboren asíncronamente sobre una misma historia, se obtiene esta misma ventaja en las plantillas de interacción, permitiendo que varios diseñadores trabajen simultáneamente diseñando o probando la misma aplicación interactiva.

El resto de elementos del framework (servidor de sesiones y herramientas de análisis) se utilizan durante la ejecución de las aplicaciones diseñadas, permitiendo así que éstas se puedan utilizar de forma colaborativa, junto con otras aplicaciones normales de FACT.

En el resto de este apartado veremos que requisitos se han de cumplir para poder implementar esta forma de programación por demostración, de manera que sea independiente de las aplicaciones sobre las que se basa el diseño. Veremos en detalle el diseño del agente de seguimiento. Por último, en el aparatado 5.1.3, veremos los resultados de las pruebas realizadas utilizando DFACT con tres aplicaciones colaborativas, las cuales sirven de base para las demostraciones a la hora de diseñar las plantillas de interacción.

5.1.1 Requisitos para DFACT

Un requisito fundamental a la hora de crear DFACT es que el proceso de diseño por demostración ha de ser independiente de las aplicaciones base utilizadas. Esta independencia no implica necesariamente que el proceso de diseño esté limitado a modelar reacciones interactivas simples, ante eventos comunes a todas las aplicaciones, o con respuestas predefinidas o basadas únicamente en la generación de dichos eventos comunes. Esto es así ya que DFACT no se basa en eventos de bajo nivel, como pueden ser los eventos de teclado o ratón, sino en eventos de alto nivel, definidos por las propias aplicaciones.

Esta independencia significa que cualquier aplicación que funcione en FACT puede utilizarse, a priori, como aplicación base. Simplemente no hay requisitos añadidos a los que impone FACT para las aplicaciones colaborativas normales, ya que toda la lógica se basará en los eventos colaborativos producidos por las aplicaciones base.

Gracias al diseño de FACT, no es estrictamente necesario cambiar nada en este framework para integrarlo con el agente de seguimiento en DFACT. El agente se comporta, desde el punto de vista del framework, como una aplicación colaborativa más. Por lo tanto, para estudiar DFACT nos centraremos en el agente de seguimiento y su interacción con el resto del framework. El resto de componentes, aunque ahora se empleen con nuevos fines, siguen siendo los mismos que hemos estudiado en el capítulo 4.

Un requisito que se ve simplificado, gracias a la reutilización de otro componente de FACT, es la gestión de varias aplicaciones base que se comportan como una sola. Este requisito es precisamente el objetivo principal del coordinador de aplicaciones de FACT, por lo que sólo será necesario que el agente de seguimiento incluya un coordinador propio de aplicaciones que herede dicha lógica. Esto implica también que el agente de seguimiento se conectará a las aplicaciones base del diseño a través de la interfaz ofrecida por los agentes de colaboración, igual que lo hace el coordinador de aplicaciones, como hemos visto en el apartado 4.2.1.2.

En cuanto a la funcionalidad del agente de seguimiento, éste ha de tener un mecanismo de decisión lo más expresivo posible, que permita, una vez

creadas las aplicaciones base, tomar decisiones complejas sobre las respuestas interactivas a las distintas situaciones que se puedan dar. Esta lógica se ha de poder programar mediante demostración, para facilitar su diseño. De esta forma se puede dar un ejemplo de cada situación que incluirá la respuesta de las aplicaciones base para cada caso. El número de casos que hay que programar por demostración se puede reducir considerablemente incluyendo los operadores lógicos, Y y Ó, para combinar distintas situaciones, a modo de expresiones que representan un patrón de situaciones, lo que permite que el diseño se realice dando muchos menos ejemplos.

Otro requisito fundamental para una herramienta de este tipo es la posibilidad de utilizar el mismo entorno para el diseño y para las pruebas. Esto permite un proceso de creación más natural, donde se puede probar inmediatamente cada elemento interactivo recién diseñado, sin tan siquiera ser necesario comenzar la prueba desde el comienzo, en la mayoría de los casos. Con el fin que este proceso integre las capacidades de diseño y prueba, será también el agente de seguimiento el encargado de interpretar las plantillas de interacción a la hora de la ejecución de las nuevas aplicaciones por parte del usuario, aunque en este caso no se requiera la capacidad de diseño.

Independientemente de lo anterior, una parte importante en todo diseño, es su capacidad de reutilización. Esto es necesario para poder crear una biblioteca de diseños, o para implementar elementos interactivos recursivos. Este es uno de los aspectos de mayor complejidad en la programación mediante demostración, ya que reutilizar un diseño significa poder aplicarlo en contextos distintos a los previstos inicialmente. En parte este problema requiere que las aplicaciones base estén diseñadas para que sean lo más independientes posible del contexto. Un ejemplo de aplicación base diseñada para maximizar la reutilización lo veremos al estudiar ConsMath, en el apartado 5.2.

Por último, la inclusión en FACT de mecanismos que faciliten y enriquezcan la integración de las aplicaciones en DFACT es un requerimiento importante. A la satisfacción de este requisito contribuye también la disponibilidad de aplicaciones específicas basadas en FACT,

como el editor de documentos WYSIWYG que puede incluir componentes de diálogo y que hemos visto en el apartado 4.4.3. Con este editor como aplicación base se pueden crear aplicaciones y pseudo tutores inteligentes sencillos, o se puede emplear como parte de una aplicación más compleja, como en el caso de ConsMath. En el siguiente apartado veremos cómo se ha diseñado el agente para solucionar todos estos requisitos.

5.1.2 Diseño del agente de seguimiento

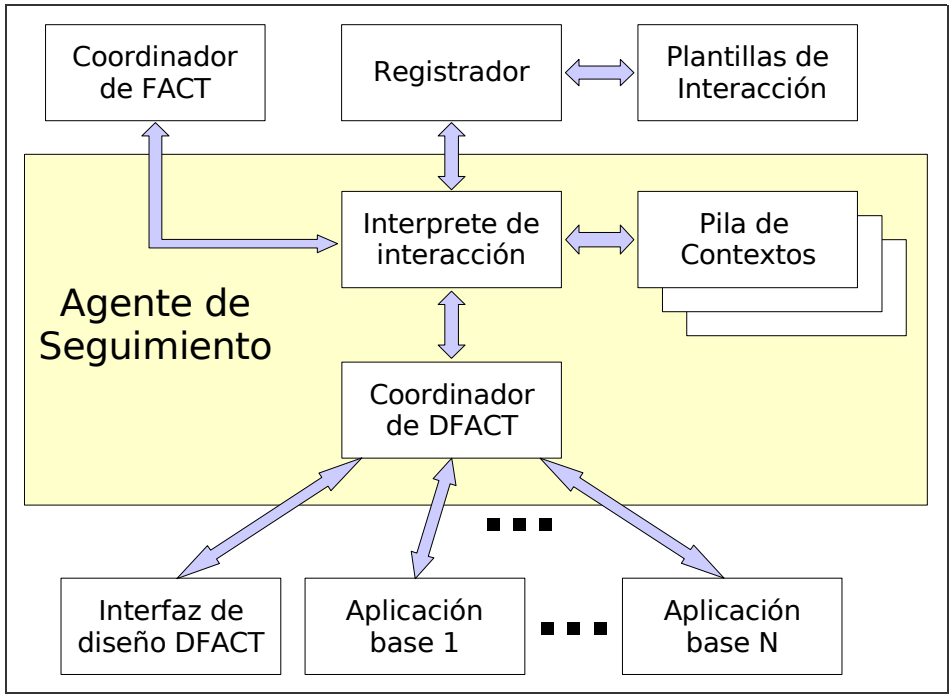

Figura 15: Agente de seguimiento

Como hemos visto en el apartado anterior, el agente de seguimiento se comporta como el coordinador de aplicaciones, y por lo tanto contiene un coordinador que su arquitectura, ampliándola para incorporar las plantillas de interacción. Además, durante la ejecución, el agente, que interpreta el diseño realizado, se comportará como una aplicación de FACT, por lo que se ha de conectar con el coordinador de aplicaciones de FACT, el encargado de crear la macro aplicación colaborativa de la sesión de trabajo. Por lo tanto este agente se comunicará directamente con tres elementos fundamentalmente: el coordinador de aplicaciones de FACT, para los aspectos colaborativos, con las distintas aplicaciones base, y con el registrador, para trabajar con las

plantillas de interacción. Esta arquitectura y su relación con los elementos externos se pueden ver en la Figura 15.

En este apartado nos centraremos en el funcionamiento del componente principal del agente, el intérprete de interacción. Este componente tiene como entrada de datos principal los eventos producidos en las aplicaciones base, tanto de las instancias locales, a través del coordinador de DFACT, como de las instancias remotas, ya que se trata de aplicaciones colaborativas. Este último flujo de eventos lo recibe a través del coordinador de FACT. También puede recibir eventos de control, producidos a través de la interfaz de usuario del agente. Esta interfaz, que sólo aparece cuando se utiliza el agente de seguimiento en la fase de diseño, está conectada al coordinador de DFACT, como cualquier otra aplicación base, con la diferencia de que se utiliza para controlar el diseño, introduciendo elementos de control en las plantillas de interacción.

El proceso de diseño de las plantillas es relativamente sencillo, tanto desde el punto de vista del diseñador como del intérprete de interacción. Podemos distinguir dos fases del diseño, que se van alternando durante el mismo, por un lado está la creación de las zonas de decisión, y por otro la de las zonas de acción. Las zonas de decisión tienen la información que indica al agente que camino seguir en función de los eventos que provoque el usuario, mientras que las zonas de acción representan la reacción de las aplicaciones base una vez que se determina el camino a seguir. Durante el diseño, el interprete de interacción envía al registrador las instrucciones para crear nuevas ramas en las plantillas de interacción, con el contenido de las distintas zonas de acción, y decisión. La estructura de estas plantillas la veremos con más detalle en el apartado 5.1.2.1.

El proceso de ejecución consiste en un seguimiento de las plantillas de interacción. En cada instante está activo uno de los nodos de una plantilla. Cuando la plantilla entra en una zona de acción, el agente reproduce en las aplicaciones base los eventos que siguen al nodo activo, hasta llegar a la zona de decisión, donde se detiene para esperar que la interacción de los usuarios con las aplicaciones produzca una serie de eventos que determine el camino a seguir hasta la siguiente zona de decisión.

También puede ocurrir que una de las acciones previstas sea ejecutar otra plantilla de interacción, como si fuera una llamada a función en programación clásica. Esta llamada se puede realizar creando nuevas instancias de las aplicaciones o reutilizando las instancias existentes. En cualquiera de los casos, se apila la información de contexto actual y se ejecuta la nueva plantilla hasta que termine su ejecución, momento en el que se volverá a recuperar el contexto anterior. La ejecución global de la aplicación terminará cuando la pila de contextos se quede vacía.

La información de contexto que se necesita guardar en cada instante incluye, entre otros datos, la posición actual de la plantilla en la que se encuentra el programa, y una tabla de correspondencias entre aplicaciones base en el espacio de diseño de la plantilla y el espacio real. Para identificar a que aplicación va dirigido cada evento, éstos llevan el identificador de la aplicación. El espacio de diseño lo forman los identificadores de aplicación durante el diseño, mientras que el espacio real lo forman los identificadores de las aplicaciones en la ejecución. Esta tabla permite que podemos tener varias instancias de la misma aplicación durante la ejecución, que no tienen por qué coincidir con las instancias utilizados durante el diseño. Por ejemplo, al ejecutar una plantilla de interacción anidada podemos controlar, mediante la tabla de correspondencias, si ésta plantilla se ejecutará sobre una instancia existente de la aplicación o creando una nueva instancia.

Para modelar el funcionamiento del agente se han creado tres estados en los que se puede encontrar: ejecución, grabación, y parada. En cualquier momento el diseñador puede pasar de un estado a otro. En el estado de grabación el agente graba los eventos producidos por el diseñador para crear las plantillas de interacción, mientras que en el estado de ejecución interpreta la plantilla actual desde el punto en el que se encuentra. Además, similarmente a la navegación por las historias de colaboración en FACT, durante el diseño podemos movernos tanto por la plantilla de interacción como por la historia de colaboración asociada a la sesión de trabajo, cambiando el estado de las aplicaciones base, sin borrar el diseño realizado en las plantillas de interacción, lo que facilitará que podamos diseñar y probar incrementalmente cada parte de la aplicación.

5.1.2.1 Estructura de las plantillas de interacción

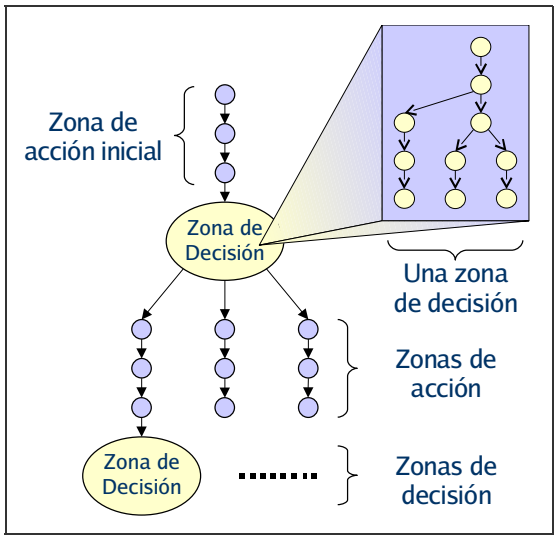

Figura 16: Estructura de las plantillas de interacción

La idea en la que se basa todo el proceso de diseño es en crear un árbol, la plantilla de interacción, donde cada rama representa un camino posible de interacción, si no tenemos en cuenta elementos complejos como los bucles, la recursividad, etc. La estructura de este árbol la podemos ver en la Figura 16. En cada camino se comienza normalmente por una zona de acción, donde están grabados los eventos necesarios para crear el escenario de interacción inicial. Estos eventos son generados interactuando con las aplicaciones base, lo que generará la rama principal en la plantilla de interacción.

Posteriormente se le indica al agente de seguimiento que pasamos a crear una zona de decisión. Para ello crearemos varias ramas, que parten del punto final de la zona de acción recién creada. Cada una de estas ramas contendrá los eventos que identifican cada uno de los posibles caminos de interacción que se pueden seguir a partir de este punto. Estas ramas se crearán de la misma forma que en la zona de acción, interactuando con las aplicaciones base, para generar las secuencias de eventos que el usuario ha de provocar para que la aplicación resultante siga cada uno de los distintos caminos de interacción.

Las zonas de decisión pueden tener secuencias de eventos que se han de dar en un orden determinado para seguir una rama, o ser más complejas, gracias a los operadores lógicos y las expresiones anidadas. Igualmente, las zonas de acción, que son ramas sin bifurcaciones, pueden tener acciones especiales, como incluir otra plantilla de interacción, como si fuera una llamada a un subprograma, lanzar o cerrar aplicaciones, etc.

Para delimitar unas zonas de otras se usan dos marcadores, uno de inicio de decisión, y otro de fin de decisión. El de inicio de decisión indica además el tipo de zona. Este tipo indica la lógica que se utilizará para evaluar la decisión a tomar en las ramas que parten de dicho inicio de decisión. Además, el uso de un marcador de inicio y otro de final permite representar zonas de decisión anidadas, permitiendo así tomar decisiones complejas. Existen tres tipos de zonas de decisión, que afectan a todos los caminos posibles en el árbol de decisión, desde la marca de inicio a la de fin correspondiente. Si tenemos un árbol de decisión anidado éste se considerará, desde el punto de vista de la lógica, como si fuera un único nodo de la zona de decisión donde está embebido. Según esto tenemos los siguientes tipos de caminos de decisión:

a) Camino tipo “S”, o “Secuencia”: Este es el tipo de segmento utilizado por omisión, y significa que se seguirá este camino si la secuencia de eventos producidos coincide con los eventos almacenados en los nodos de dicho segmento.

b) Camino tipo “Y”: Es igual que el de secuencia, con la diferencia de que no importa el orden en el que coincidan los nodos.

c) Camino tipo “O”: En este caso sólo será necesario que coincida uno de los nodos para que se siga este camino.

El caso más sencillo de árbol de decisión se da cuando todos los caminos son de tipo secuencia. Este árbol tendrá un nodo final en cada hoja, que se corresponden con las distintas alternativas que puede seguir el agente. Los nodos que hay desde la raíz de la decisión hasta un nodo final, representan los distintos eventos que se han de dar, fruto de la interacción del usuario con la aplicación, para que el agente decida seguir el camino que hay a partir de dicho nodo final.

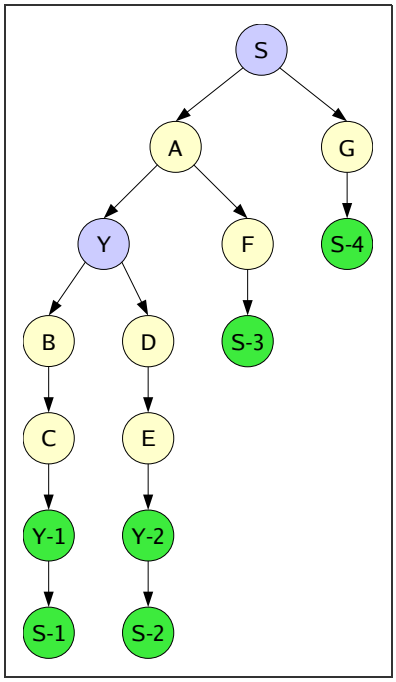

Figura 17: Ejemplo de árbol de decisión

Un ejemplo de árbol de decisión ilustrativo es el que tenemos en la Figura 17. En este ejemplo hay dos nodos que marcan el inicio de decisión, el principal, de tipo “S”, y dentro de esta decisión otra de tipo “Y”, estando ambos nodos en color gris. Los nodos finales, etiquetados con números, delimitan cada zona de decisión. Hay dos nodos finales que terminan los dos caminos posibles de la decisión interna de tipo “Y”, Y-1 e Y-2, y cuatro nodos finales de la decisión principal, de S-1 a S-4. Éstos están numerados únicamente para facilitar el seguimiento de la explicación. El resto de nodos del árbol, en un color claro, son los que se corresponden con los distintos eventos que puede producir la aplicación. Están etiquetados con una letra para distinguir unos eventos de otros.

Los cuatro caminos del ejemplo se corresponden con las siguientes secuencias de eventos:

1. El primer camino, S-1, se seguirá si se produce la secuencia de eventos “A (B y C)”, que equivale a las secuencias “ A B C” o “A C B”.
2. S-2 se seguirá si se producen las secuencias de eventos “A D E” o “A E D”.

3. S-3 se seguirá con la secuencia “A F”.
4. El último camino se seguirá si se produce el evento “G”.

Cualquier otro evento que produzca la aplicación, mientras se está en la zona de decisión, será ignorado por el agente de seguimiento, en su configuración por omisión.

Se pueden diseñar aplicaciones basadas en FACT que prevean su utilización con DFACT, ofreciendo un comportamiento más rico. Por ejemplo, se pueden limitar las acciones que puede realizar el usuario final mientras se está en una zona de decisión. Para ello lo más sencillo es diseñar la aplicación base para que ofrezca dos modos de trabajo, uno de diseño y otro de ejecución. De esta forma la propia aplicación base impedirá al usuario realizar acciones que están destinadas al diseñador. En el ejemplo del editor de documentos, podemos añadir campos de formulario al examen, de forma que en tiempo de ejecución el usuario sólo pueda modificar los valores en dichos campos. En este caso, los únicos eventos que se podrán producir en tiempo de ejecución, y por lo tanto los únicos que tendrá en cuenta el agente al evaluar las distintas zonas de decisión, serán los que representen el cambio de valor en los campos.

En cuanto a la implementación de un sistema de decisión basado en árboles, ésta es similar a crear un autómata finito no determinista. Al ser no determinista hay que decidir qué camino escoger en el caso en que se llegue simultáneamente a varias soluciones posibles del algoritmo. La solución más sencilla, y la que se sigue en el implementación actual, es salir por el primer camino de decisión encontrado que llegue al estado de aceptación.

Este modelo de programación no está limitado a un solo árbol, ya que permite llamadas a otras plantillas de interacción, incluso de forma recursiva, de forma que se pueden programar mediante ejemplo interfaces de usuario de complejidad arbitraria. Un aspecto importante de las llamadas a otras plantillas es decidir si se intentarán reutilizar las instancias de las aplicaciones base ya existentes, o si se ejecutarán en nuevas instancias.

Una de las ventajas de reutilizar las instancias existentes es que la nueva plantilla comparte el estado de la aplicación en la plantilla actual, lo que permitirá condicionar el desarrollo de la interacción. Por ejemplo, se puede

programar una aplicación base que tenga el concepto de variables. Estas variables se podrán usar entonces para pasar información de la plantilla de interacción actual a la llamada, y viceversa, simulando lo que ocurre cuando se llama a un procedimiento con variables de entrada y salida.

5.1.3 Pruebas de diseño por demostración

Se han realizado dos tipos de pruebas, por un lado se han diseñado nuevas aplicaciones creando plantillas de interacción con distintas aplicaciones base previamente portadas a FACT. Por último se ha creado desde cero una herramienta de diseño de ejercicios, ConsMath, basada en una nueva aplicación base creada para aprovechar al máximo DFACT en el campo de las Matemáticas simbólicas. Esta herramienta la estudiaremos en detalle en el apartado 5.2.

Una de las primeras pruebas con el agente de seguimiento se realizó con el editor colaborativo que ya funcionaba en FACT, y que hemos visto en el apartado 4.4.3. Este editor no se había creado expresamente para ser utilizado como aplicación base. Aún así se pueden diseñar ejercicios muy sencillos. Por ejemplo, se puede crear fácilmente un pequeño examen colaborativo, donde un grupo de alumnos tienen que rellenar varios formularios con preguntas, cada uno de los cuales se puede rellenar en cualquier orden. Para permitir que las respuestas se puedan dar en cualquier orden se pueden crear condiciones mediante decisiones de tipo “Y”. Los formularios que se muestran en cada paso pueden depender de la respuesta anterior y el sistema no mostrará el siguiente formulario hasta que no se rellene el anterior.

Este editor crea eventos con información de alto nivel, que respeta al máximo la intencionalidad del usuario. Este aspecto es muy importante en el caso de las aplicaciones base en DFACT, ya que cuando se hacen llamadas a otras plantillas reutilizando las instancias de las aplicaciones, estas plantillas anidadas se ejecutarán en un contexto distinto al de su diseño. En el caso del editor, se pueden insertar plantillas anidada sin que se sobrescriba el texto ya escrito, ya que los eventos codifican las posiciones de inserción o borrado de forma relativa al final del documento.

Otro ejemplo en el que se puede utilizar el editor colaborativo como aplicación base es al crear una historia interactiva. El editor admite incluir texto con formato e imágenes, además de las operaciones típicas de copiar y pegar, por lo que se podrá crear una historia en la que dependiendo de las decisiones del usuario en cada paso aparezca un texto u otro.

Algo parecido se puede hacer con otras aplicaciones existentes de FACT, como el ajedrez colaborativo, ChessEdu. En este caso partimos de un tablero con una situación interesante para que el alumno consiga un objetivo determinado, como dar mate o comer una pieza, en unos pocos movimientos. El tutor podrá crear árboles de decisión que realicen un seguimiento de la evolución de la partida, de forma que el alumno obtenga ayuda cuando comete un fallo. Esta ayuda se puede realizar mediante ventanas emergentes con el editor de textos como aplicación base adicional.

En el ejemplo del ajedrez y otros similares, DFACT puede ser útil siempre y cuando las posibles acciones a realizar por el usuario sean reducidas. De otra forma el árbol de decisión sería demasiado complejo de diseñar. Por ejemplo, se puede utilizar DFACT con la aplicación de ajedrez en estudios de finales de partidas con una pieza. En general, DFACT se puede utilizar razonablemente para cubrir aquellos ejemplos que se pueden explicar en un libro o manual.

Para facilitar la utilización de las aplicaciones base en la etapa de diseño, se ha creado un panel con la interfaz de usuario para el control del diseño en DFACT. Gracias a este panel, en ConsMath, que es la aplicación más compleja que utiliza actualmente DFACT, se puede acceder a todas las características de DFACT desde la propia interfaz de usuario de está aplicación, como si fuera una sola aplicación. ConsMath permite componer texto con elementos interactivos, creando dependencias entre unos elementos y otros, delegando en DFACT el diseño del flujo de interacción de los cursos o ejercicios creados.

5.2 ConsMath

ConsMath es un editor de cursos sobre Matemáticas simbólicas basado en DFACT, donde el diseñador puede crear textos con gráficas interactivas, fórmulas dinámicas, etc. Los cursos permiten la generación de problemas de

forma aleatoria, lo cual impide conocer a priori el camino de resolución correcto de los ejercicios. Esto supone un reto para el agente de seguimiento, ya que el usuario interactúa libremente con la aplicación. Además, el camino de resolución seguido y las respuestas del alumno dependen de variables o expresiones aleatorias, cuyo valor es por lo tanto desconocido en tiempo de diseño. La solución elegida para poder modelizar una interacción tan compleja consiste en que el agente tome las decisiones en función de los eventos de alto nivel que produce ConsMath. Estos eventos pueden ser tan sencillos como decir que la expresión introducida es correcta, o eventos más complejos que requieren evaluar una función dependiente de varias expresiones introducidas por el usuario, para terminar generando un valor que permita discriminar la decisión a tomar por el agente.

Con ConsMath se pretende crear una herramienta para el diseño de material didáctico con un alto grado de interacción y que esté al alcance de los profesores, al no requerir conocer un lenguaje de programación. Además todo el material didáctico creado se podrá utilizar de forma colaborativa a través de DFACT.

Para poder afrontar este objetivo nos hemos centrado en resolver los problemas que surgen a la hora de crear aplicaciones didácticas sobre materias científico-técnicas, aunque la infraestructura creada en ConsMath no limita su uso a este tipo de aplicaciones.

En el caso de las aplicaciones educativas a las que está orientado ConsMath el aprendizaje consiste normalmente en la resolución de problemas generados aleatoriamente, siguiendo un patrón tipo. La resolución típica consiste en seguir una serie de pasos, realizando transformaciones o hallando los valores de las distintas incógnitas del problema. Estos pasos se realizan de forma supervisada por la aplicación, por ejemplo ofreciendo una lista de alternativas en cada paso de resolución, o mostrando mensajes de ayuda cuando el alumno comete un error.

Este tipo de aplicaciones educativas son similares a las que se pueden crear con otras herramientas especializadas, como MathEdu, [Díez 2003]. La mejora fundamental respecto a esta herramienta es que ConsMath produce aplicaciones colaborativas y todos los pasos del diseño se realizan mediante programación por demostración.

En este tipo de aplicaciones tendremos una mezcla de dos tipos de contenidos, los estáticos y los dinámicos. Los contenidos estáticos suelen estar asociados con texto u otros elementos fijos; por ejemplo para crear los enunciados de los problemas, mientras que el contenido dinámico tiene que ver con valores y funciones que se evalúan dinámicamente ante cambios en las variables o en los datos introducidos por el usuario. Este contenido dinámico permite crear una interacción limitada en la aplicación. Para crear elementos más complejos podemos añadir comportamiento interactivo, que permite por ejemplo hacer aparecer o desaparecer nuevo contenido en la aplicación.

La forma de facilitar la creación del contenido es mediante una interfaz de usuario que resulte familiar al diseñador. Para ello se ha escogido una interfaz basada en el editor de documentos WYSIWYG, que hemos visto en el apartado 4.4.3.

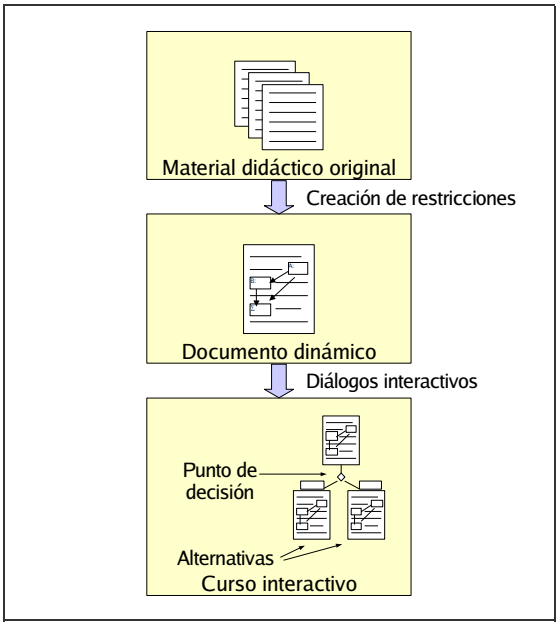

Figura 18: Proceso de diseño del material didáctico en ConsMath

Los pasos que se siguen para la creación del material didáctico interactivo son los que se describen en la Figura 18. Normalmente se comienza reutilizando el material didáctico disponible. Por ejemplo, a partir de un documento que describe la resolución de un problema. Este material se

enriquece para añadir dinamismo a las distintas partes del documento. Para el contenido dinámico se usa un mecanismo similar al de las hojas de cálculo, pero esta vez las celdas de las hojas de cálculo no están en posiciones fijas, sino que son componentes gráficos que se pueden intercalar con el resto de elementos del documento. Para editar el valor el diseñador simplemente tiene que seleccionar el componente e introducir una función o un valor fijo. En las funciones el diseñador puede hacer referencia a los valores de otros componentes, a los cuales puede dar el nombre que desee.

Por ejemplo, podemos crear una pequeña aplicación consistente en un único diálogo que muestre al alumno el enunciado del problema donde tiene que despejar las incógnitas para poder resolverlo. Para ayudarle en la resolución se le muestran los pasos de resolución, pero de forma incompleta, de forma que el alumno tenga que introducir la información que falta. Junto a cada campo de entrada podemos añadir un componente gráfico, por ejemplo un Checkbox, que muestre un símbolo de confirmación si la respuesta es correcta. En este ejemplo podemos conseguir que la aplicación evalúe de forma dinámica las respuestas, creando una restricción en el Checkbox que asocie su estado a un función que se evalúa a cierto si la respuesta es correcta.

Los componentes interactivos que se pueden insertar pueden ser etiquetas, botones, campos de entrada de datos, ecuaciones, gráficas interactivas, etc. Todos ellos tienen una o varias propiedades cuyo valor puede ser fijo, calculado, generado aleatoriamente y, en el caso de los componentes que lo admitan, introducido por los usuarios finales. Por ejemplo, una etiqueta puede mostrar datos calculados, pero no permite que el usuario interactúe con ella, mientras que si añadimos un campo de entrada de datos el usuario podrá escribir en él y además podrá tener un valor inicial.

Otro de los aspectos a destacar de ConsMath es su flexibilidad, ya que permite incluir en los documentos interactivos cualquier componente gráfico Java y crear dependencias con otros componentes para añadir un comportamiento dinámico. El único requisito es que el componente gráfico cumpla la especificación JavaBeans, lo que en la práctica significa que se puede incluir cualquier componente estándar y la mayoría de los componentes gráficos de terceros.

El último paso consiste en añadir el comportamiento interactivo de alto nivel, el cual coordina cómo y cuándo aparecen o desaparecen los distintos documentos dinámicos que se han creado en el paso anterior. Esto se hace mediante programación por demostración, gracias a que ConsMath está basado en DFACT. Para ayudar al agente de seguimiento a tomar las decisiones se pueden añadir predicados que permiten discriminar por ejemplo si el usuario ha respondido bien a un problema generado de forma aleatoria.

En el ejemplo anterior la aplicación consistía en un único diálogo. Para aumentar la capacidad de interacción de la aplicación podemos dividir este diálogo en varios, uno por cada paso de resolución, de forma que podamos mostrar al alumno una pequeña explicación cada vez que cometa un fallo. Para programar esta funcionalidad añadida hay que crear predicados que evalúen en cada paso si la respuesta del alumno es correcta o no. Una vez hecho esto el diseñador ha de enseñar al sistema qué hacer en las distintas situaciones, usando el mecanismo de programación por demostración de DFACT. En este caso el diseñador se hace pasar por el alumno, y cuando éste introduzca la respuesta correcta pasará al siguiente paso de resolución, mientras que cuando introduzca una respuesta incorrecta creará un mensaje para el alumno que le de sugerencias sobre cómo seguir o un enlace a un ejemplo relevante que puede resolver.

Esta división en dos fases no significa que sea necesario completar la primera fase del diseño para pasar a la segunda. Lo normal en el proceso de diseño del material didáctico es seguir estos pasos de forma incremental, por ejemplo diseñando una rama del curso interactivo y volviendo al punto de decisión anterior para diseñar la siguiente rama de la plantilla de interacción desde el primer paso descrito anteriormente.

El material creado en ConsMath posee un alto grado de abstracción y gracias al sistema de dependencias se pueden crear, por ejemplo, ejercicios personalizables, que dependan de fórmulas o valores aleatorios. Además se pueden crear ejercicios complejos que incluyan llamadas a otros ejercicios para resolver una parte del problema. De esta forma se puede crear una biblioteca de material didáctico reutilizable.

Una vez terminado el proceso de diseño del material didáctico, éste se podrá utilizar de forma colaborativa. Para ello el usuario ejecutará ConsMath y cargará el curso o ejercicio que desea ejecutar. En este caso la interfaz de usuario de ConsMath habrá cambiado, ya que esta aplicación presenta una interfaz distinta para el diseñador que para el usuario final, al desaparecer la posibilidad de añadir nuevo contenido o comportamiento interactivo. El usuario final puede interactuar con los componentes gráficos que lo permitan, como los botones, o las cajas de introducción de datos o fórmulas.

En los siguientes apartados veremos en detalle cómo está diseñado ConsMath, cómo se crean los cursos, la utilización por parte del usuario final y las pruebas que se han realizado.

5.2.1 Diseño de ConsMath

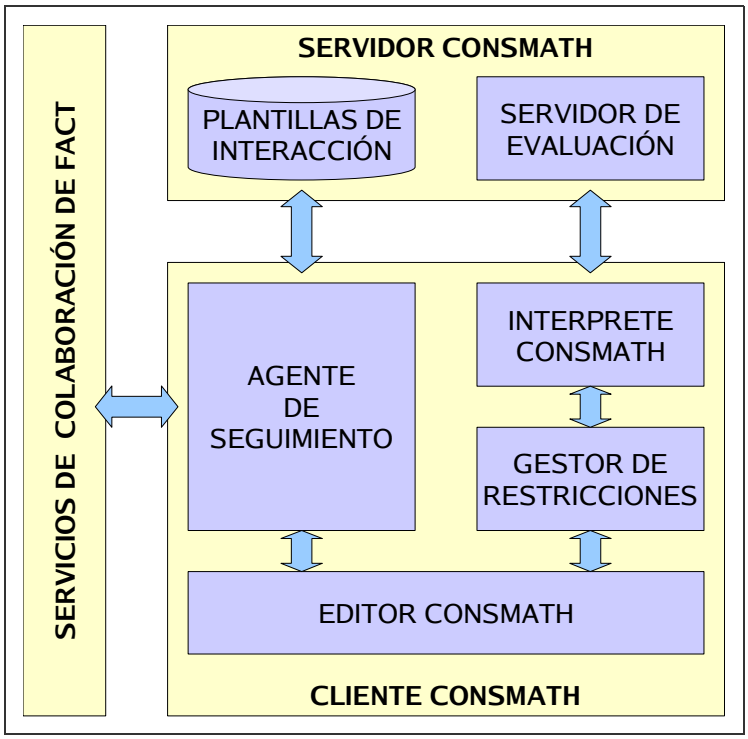

Figura 19: Arquitectura de ConsMath

La arquitectura de ConsMath es la que se representa en la Figura 19. Como acabamos de ver, ConsMath se basa en un editor que permite mezclar texto con componentes dinámicos. A esto se añade la capacidad de diseñar

el flujo de interacción mediante demostración, gracias al agente de seguimiento.

El elemento principal con el que interactúa el diseñador es el editor, desde el cual también accede a la interfaz de usuario de diseño, al estar integrada en el propio editor de ConsMath. La información producida llega al agente de seguimiento, y desde éste se envía a FACT, a través del coordinador de aplicaciones de FACT. De esta forma ConsMath se puede utilizar de forma colaborativa tanto en modo diseño como en ejecución. El agente de seguimiento también está conectado con el servidor de ConsMath, donde el registrador almacena las plantillas de interacción, o cursos diseñados.

El resto de componentes de la arquitectura son los encargados del comportamiento dinámico. El gestor de restricciones se encarga de mantener actualizados los valores de los componentes, propagando los cambios y evaluando las funciones. Esta evaluación la realiza un intérprete que reconoce las funciones propias de ConsMath y calcula en un evaluador externo el resto de expresiones Matemáticas.

Con el fin de evitar la necesidad de instalar el evaluador en todos los clientes se utiliza el servidor de evaluación de ConsMath, que será el único que conecte directamente con el evaluador externo. La interfaz que proporciona el servidor de evaluación es independiente del evaluador externo empleado, aunque en la implementación actual sólo se ha implementado una interfaz de evaluación con Mathematica[45], de la cual únicamente se utiliza su capacidad de evaluación de expresiones.

A continuación veremos en detalle cómo funcionan e interactúan entre sí los principales componentes que integran ConsMath. Primero veremos el editor y el contenido de los documentos, para a continuación estudiar los componentes responsables del comportamiento dinámico, como es la interacción de alto nivel, y por último los aspectos colaborativos.

5.2.1.1 Editor ConsMath

El editor es el componente central de esta herramienta, ya que la mayor parte de las tareas tanto de diseño como de resolución de las plantillas de

[45] El sitio Web oficial de Mathematica está en la dirección: http://www.wolfram.com/products/mathematica

interacción se realizan interactuando con este editor. Por este mismo motivo también es el componente con el que se comunica el agente de seguimiento. En este apartado nos centraremos únicamente en los aspectos del editor relacionados con el diseño de las plantillas de interacción desde un punto de vista estático o de su contenido.

La estructura principal que gestiona ConsMath son las plantillas de interacción, apartado 5.1.2.1. Esta estructura principal es un árbol, sin embargo el modelo sobre el que trabaja el editor directamente es un documento. La relación entre ambos modelos es que en cada instante el editor muestra un escenario, el documento, que corresponde al estado del entorno de resolución en un paso concreto de la resolución del problema. Las acciones de edición para crear este escenario son las contenidas en las zonas de acción, desde la raíz hasta el punto de la plantilla correspondiente a dicho instante. Por lo tanto, la plantilla de interacción contiene todos los escenarios posibles de la interacción, mientras que en el editor sólo se muestra el escenario actual de trabajo. Para cambiar de escenario el diseñador puede utilizar la interfaz de usuario del agente de seguimiento, que se encargará de realizar las acciones necesarias en el documento para que éste refleje el nuevo escenario.

Cuando se pasa de un escenario a otro los cambios en el documento pueden suponer ver o editar un documento totalmente diferente, o simplemente añadir u ocultar información, manteniendo inalterado parte del escenario anterior. En este último caso el diseñador puede simplemente añadir o eliminar el contenido que desee editando el propio documento. Sin embargo esto es más sencillo si se divide el documento en segmentos. Así pues el diseñador puede ocultar o mostrar de nuevo segmentos anteriores de forma mucho más sencilla.

En las aplicaciones educativas a las que está orientado ConsMath resulta muy útil la segmentación de los documentos, de forma que al alumno se le pueda presentar nuevos diálogos en el mismo documento de trabajo y que éste tenga toda la información de los pasos de resolución anteriores disponible, sin ni tan siquiera tener que cambiar de ventana. Además la segmentación permite ocultar de forma sencilla la información que ya no es necesaria a la vez que se permite limitar la interacción a los elementos de los

últimos segmentos del documento. Por ejemplo, en la resolución de un problema complejo, se pueden crear segmentos separados para los distintos pasos intermedios de resolución. De esta forma cuando se termine algún cálculo intermedio se podrán ocultar los detalles, dejando únicamente los segmentos que muestran los datos necesarios para continuar la resolución del problema.

Por lo tanto, en el contenido del material didáctico creado con ConsMath podemos distinguir tres niveles de detalle. El primero, la estructura principal, es la plantilla de interacción. La unión de las zonas de acción de la plantilla que están en el camino de interacción actual, desde la raíz de la plantilla hasta el punto actual de edición, se corresponde con el documento que se está editando, el cual representa el segundo nivel de detalle. El último nivel de detalle lo representan los segmentos, en los que está dividido el documento. Algunos segmentos pueden estar activos, con los cuales el usuario puede interactuar. Otros segmentos pueden estar inactivos, y otros pueden estar ocultos al usuario.

5.2.1.2 Contenido de los documentos

Como acabamos de ver, el elemento básico de contenido es el segmento de documento. A este nivel el diseñador puede introducir texto con estilo y distintos componentes gráficos. Los componentes insertados siguen el flujo del texto, comportándose como el resto de caracteres del texto.

Mediante los componentes y las relaciones entre ellos el diseñador puede especificar el dinamismo a nivel de documento. Este dinamismo consiste básicamente en crear dependencias entre las propiedades de los componentes, lo cuales pueden ser JavaBeans genéricos.

La mayoría de componentes tienen una propiedad de texto principal. Por ejemplo en los botones es el texto que muestran, en un campo de entrada de datos es el texto que escribe el usuario. Además tienen otras propiedades a las que puede ser interesante asignar un valor, tanto estático como dinámico.

En el caso de las propiedades de texto principales se les puede dar valor simplemente escribiéndolo en el editor y seleccionándolo antes de insertar el componente. El componente sustituye al texto, y el valor de la propiedad de texto principal pasa a ser el del texto anteriormente seleccionado.

Algunos componentes, como el campo de entrada, o las fórmulas, permiten que ese valor se cambie utilizando el propio componente. Sin embargo no todos los componentes permiten una edición directa de sus propiedades. Para ello es necesario utilizar un editor de propiedades.

También puede ocurrir que el valor de la propiedad principal de un componente no sea texto, como en el caso de los campos de elección múltiple. En este ejemplo el valor será el número de la opción seleccionada, o 0 si no se ha seleccionado ninguna, mientras que el texto principal del componente representa la lista de opciones, una por línea.

Además de texto y números, podemos tener componentes cuya propiedad principal es una ecuación matemática, como es el caso del editor de ecuaciones. Este componente es un editor WYSIWYG de ecuaciones. Por lo tanto el usuario puede editar la ecuación directamente, sin necesidad de usar el editor de propiedades. Desde el punto de vista de ConsMath el formato de las ecuaciones es MathML[46] de contenido, aunque ni el diseñador ni el usuario final requieren conocer este lenguaje para su uso en ConsMath.

En cualquier propiedad de un componente podemos crear una restricción, que consiste en una expresión matemática que depende de otras propiedades o valores, en lugar de utilizar un valor constante, de forma que éste sea calculado dinámicamente. En algunos casos estas expresiones pueden introducirse directamente en el componente, como acabamos de ver. Por lo tanto, en un editor de ecuaciones la expresión terminará escrita en MathML, mientras que en campo de entrada de texto la expresión se escribirá en una notación lineal, en lugar de gráfica. Con el editor de propiedades podemos usar el editor gráfico de ecuaciones o la notación lineal para dar valor a cualquier propiedad, de forma que el diseñador utilice la notación que le parezca más conveniente.

Las expresiones permiten acceder al valor de la propiedad principal de otros componentes, o incluso de alguna propiedad concreta, en lugar de la principal. También permiten definir variables, que se podrán reutilizar en la misma u otras expresiones. Además se pueden hacer todo tipo de cálculos matemáticos, tanto numéricos como simbólicos, pudiendo elegir en todo

[46] La especificación MathML la creó y mantiene el W3C en la dirección: http://www.w3.org/Math/

momento el diseñador cuándo una expresión se ha de evaluar a la forma más simple posible, o cuándo sólo se han de simplificar partes concretas de la expresión. Si no existen incógnitas por resolver la forma más simple de la expresión será un valor numérico.

Como hemos visto las propiedades de los componentes puede ser de distintos tipos, al igual que las expresiones. ConsMath permite mezclar en una misma expresión valores de distintos tipos, realizando automáticamente y de forma coherente las conversiones de tipo que sean necesarias.

Por ejemplo, una expresión escrita en MathML para indicar el valor a mostrar en una fórmula, puede ser la expresión que indique la suma sin evaluar de los valores de las propiedades principales de los componentes A y B. Uno de estos componentes puede por ejemplo tener una propiedad principal de tipo “cadena de texto”, mientras que el otro puede tener una propiedad de tipo numérico. ConsMath se encargará de traducir el valor de A en una expresión matemática en MathML, al igual que el número de B, para finalmente crear la expresión resultante.

Otra problema que puede ocurrir a la hora de evaluar las expresiones es qué hacer cuando varios componentes o variables tienen el mismo nombre. Esto puede ocurrir por ejemplo cuando dentro de una plantilla de interacción se inserta otra distinta. En este caso puede ocurrir que la nueva plantilla utilice variables con el mismo nombre que en la plantilla original. Para solucionar este problema se crean distintos ámbitos de las variables, uno por cada segmento del documento actual. Primero se buscará el componente o variable en el segmento actual, y después se busca en orden hacia segmentos anteriores. Esto será muy útil a la hora de incluir ejercicios que requieren una resolución por partes, usando plantillas adicionales, o se resuelven de forma recursiva.

En tiempo de ejecución la interfaz de usuario del editor cambiará para limitar las acciones del usuario a los componentes que admitan interacción en los segmentos que estén activos en cada momento. En este caso desaparecerá la paleta de componentes, las opciones para modificar el estilo del texto, el editor de propiedades, la interfaz de usuario del agente de seguimiento, etc. La única opción es interactuar con los componentes que permitan introducir datos o ser manipulados directamente, como los campos

de entrada, de texto o ecuaciones, elementos de elección múltiple, o los botones.

5.2.1.3 Interacción de bajo nivel y gestión de restricciones

El sistema de gestión de restricciones es el encargado de mantener actualizados los valores de las componentes del documento, de forma que se cumplan las restricciones impuestas por el diseñador.

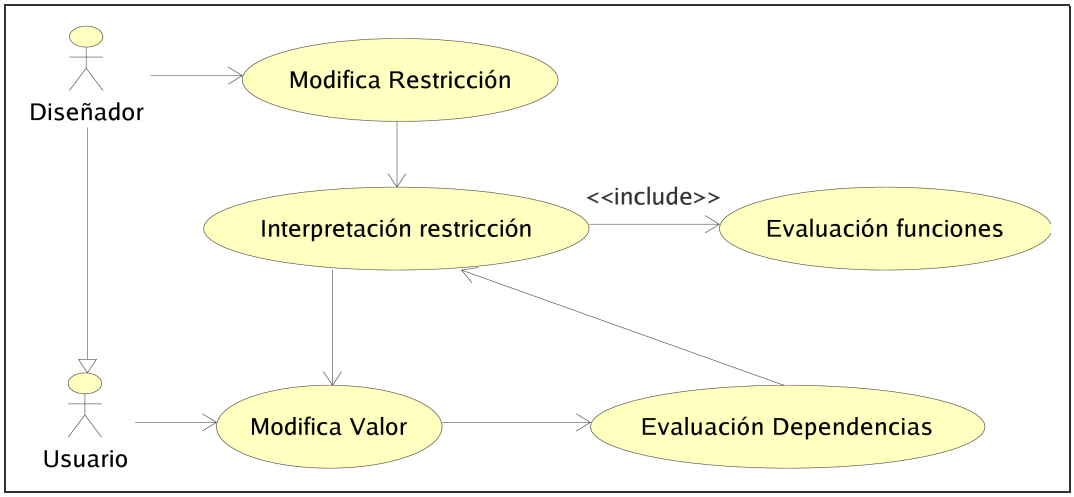

Figura 20: Principales casos de uso del sistema de restricciones

Los casos de uso más habituales y que son representativos de la mayoría de situaciones que se pueden dar al trabajar con el sistema de restricciones están representados en la Figura 20. Estos son la introducción o modificación de restricciones por parte del diseñador, y la modificación de los datos de entrada por parte del usuario.

Como hemos visto los componentes del documento tienen propiedades, y a su vez éstas pueden tener asociada una expresión que permite calcular su valor de forma dinámica. Para facilitar la creación de estas expresiones, el diseñador puede dar el nombre que desee a los componentes presentes en el documento. Si no lo hace el editor asignará un nombre de forma automática al componente.

Cuando se crea una restricción ésta se ha de interpretar. En la implementación actual el intérprete de ConsMath reconoce un conjunto reducido de funciones propias. Sin embargo el repertorio de funciones que se

pueden emplear no se limita a éstas, ya que mediante el servidor de evaluación ConsMath permite que se utilice cualquier función soportada por el evaluador externo.

Las funciones propias de ConsMath pueden encontrarse intercaladas con expresiones Matemáticas, y tienen a su vez expresiones como argumento. En la versión actual las funciones propias son:

- defVar(variable, expresión): Esta función define la variable indicada en el primer argumento, asignando como valor la expresión del segundo argumento.
- val(variable o propiedad): Esta función simplemente devuelve la expresión asociada a la variable o propiedad del componente indicado.
- eval(expresión): Esta función evalúa la expresión pasada por argumento, realizando los cálculos que sean necesarios para obtener una expresión simplificada. Antes de ser simplificada, esta expresión ha de ser evaluada por el intérprete de ConsMath para ejecutar las funciones propias que pueda contener. Por último se envía al servidor de evaluación para su simplificación.
- like(patrón, expresión): Indica si la expresión cumple el patrón indicado.
- randomInteger(mínimo, máximo): Genera un número entero aleatorio entre el mínimo y máximo indicado.
- randomReal(mínimo, máximo): Genera un número real aleatorio entre el mínimo y máximo indicado.
- randomPol(grado, incógnita): Genera aleatoriamente un polinomio, con el grado e incógnita indicados.

ConsMath permite varios niveles de anidamiento en los componentes, de forma que el editor puede contener un componente con propiedades que son componentes, o incluso contener otro editor con nuevos componentes. Esto permite por ejemplo insertar un editor de formas geométricas dentro del documento, y que desde otros componentes podamos realizar cálculos que dependen de los objetos gráficos dentro del editor de formas.

Para poder hacer referencia a una propiedad concreta de un componente es necesaria una notación que permita especificar el camino en la jerarquía de componentes. ConsMath utiliza para ello una notación del tipo “componente[.propiedad]”, donde el nombre del componente a su vez puede hacer referencia a varios subcomponentes, separando todos ellos por un punto. El nombre de la propiedad también puede ser compuesto, ya que algunos componentes tienen propiedades que son objetos, y a su vez tendrán propiedades. Si la propiedad se omite se usará la propiedad principal del componente. Las variables se comportan como un componente más, cuya propiedad principal es el valor de la variable.

Con la finalidad de no limitar la reutilización y simplificar la tarea de diseño de las plantillas de interacción, ConsMath permite hacer referencia a las propiedades o variables mediante el camino relativo a la definición de la expresión. Para conseguir esta funcionalidad es necesario introducir el concepto de ámbitos de definición. Existen diferentes ámbitos, desde el más interno, el componente donde se define la restricción, hasta el más externo, el del documento actual, pasando por los distintos niveles de anidamiento de componentes dentro de componentes, y los distintos segmentos del documento. En este último caso, se considera de un ámbito más cercano los componentes del segmento actual, y de un ámbito más alejado cuanto más cerca del comienzo del documento está el segmento. Por lo tanto, el sistema de búsqueda de propiedades comenzará por el segmento actual e irá subiendo hasta llegar al primer segmento del documento. En el caso de las variables, su ámbito será el mismo que en el componente donde está definida.

Con el fin de implementar el mecanismo de búsqueda en ámbitos de forma eficiente, se ha representado el esquema de ámbitos mediante un árbol optimizado para las búsquedas, donde cada nodo es un ámbito, y sus descendientes son ámbitos internos. Se puede comprobar que cada ámbito coincide con un objeto del sistema, siendo el documento principal el objeto que se corresponde con el ámbito que está en la raíz de este árbol. El sistema de búsqueda por prioridad según el segmento donde se define la restricción también está definido como parte de este árbol. El primer segmento del documento desciende directamente de la raíz del árbol, el documento, y el resto de segmentos van descendiendo del segmento

anterior. Los componentes que están en cada segmento descienden directamente del nodo del árbol correspondiente al segmento donde se encuentran, mientras que los subcomponentes descienden del nodo asociado al componente que los engloba.

El algoritmo de búsqueda comienza en el nodo o ámbito correspondiente al componente donde está definida la restricción, e irá subiendo hacia la raíz mientras no encuentre la propiedad buscada, lo que equivale a buscar de forma sucesiva en ámbitos anteriores más alejados. Dentro de cada nodo, o ámbito, se realiza una busca de forma descendiente en el árbol.

Como hemos visto, ConsMath permite mezclar en una misma expresión propiedades cuyos valores pueden ser de diferentes tipos. Para conseguirlo se han de convertir los valores a un lenguaje interno común antes de poder operar con ellos. Este lenguaje es una representación en texto lineal de las expresiones Matemáticas.

El diseñador o el usuario final pueden introducir las expresiones tanto en texto lineal, como mediante un editor gráfico, y ConsMath detectará y transformará automáticamente al lenguaje interno todas las expresiones en otros lenguajes antes de operar, volviendo a transformar los resultados al lenguaje o tipo de dato utilizado en la propiedad que muestre el resultado.

ConsMath permite añadir en tiempo de ejecución nuevos objetos gráficos JavaBeans genéricos, que a su vez pueden tener propiedades de tipos no previstos. Para poder usar nuevos tipos de propiedades se ha extendido el sistema de restricciones con un mecanismo de introspección basado en el paquete de reflexión de Java. Básicamente hay dos alternativas para permitir una conversión del tipo A al B, proporcionar el constructor “B(A a)”, o el método “B A.toB()”.

Como hemos visto, algunas funciones requieren evaluarse en el servidor. En la implementación actual este servidor envía las expresiones al motor de cálculo de Mathematica, y devuelve los resultados en tres formatos distintos, MathML de presentación, texto, o como una imagen. El formato de imagen, además de permitir representar fielmente las fórmulas, permite mostrar gráficas con los resultados, que de otra forma requeriría disponer de un visor especializado. Para evitar sobrecargar el servidor, se ha creado un sistema caché de expresiones evaluadas. Esto disminuye la carga del servidor, ya

que ConsMath es una aplicación colaborativa, y si no se implementa este u otro mecanismo similar todos los clientes de la sesión enviarían simultáneamente las mismas expresiones de cálculo al evaluador externo.

El resultado de la interpretación de las expresiones, además de la expresión final calculada, informa sobre las dependencias que se crean con los distintos componentes o variables del documento. Cuando se crea una restricción el gestor de dependencias utiliza esta información para suscribirse a futuros cambios en las propiedades o variables. También es necesario comparar las nuevas dependencias con las anteriores cuando se modifica o borra una restricción, para así eliminar las suscripciones obsoletas.

5.2.1.4 Interacción de alto nivel

Como hemos visto en el apartado 5.1, DFACT permite diseñar la interacción de alto nivel utilizando una aplicación base. ConsMath utiliza DFACT para diseñar y ejecutar los cursos, integrando su funcionalidad como parte de la interfaz de ConsMath. En este caso la aplicación base es sólo una parte de ConsMath, el editor, en el cual nos centraremos en este apartado.

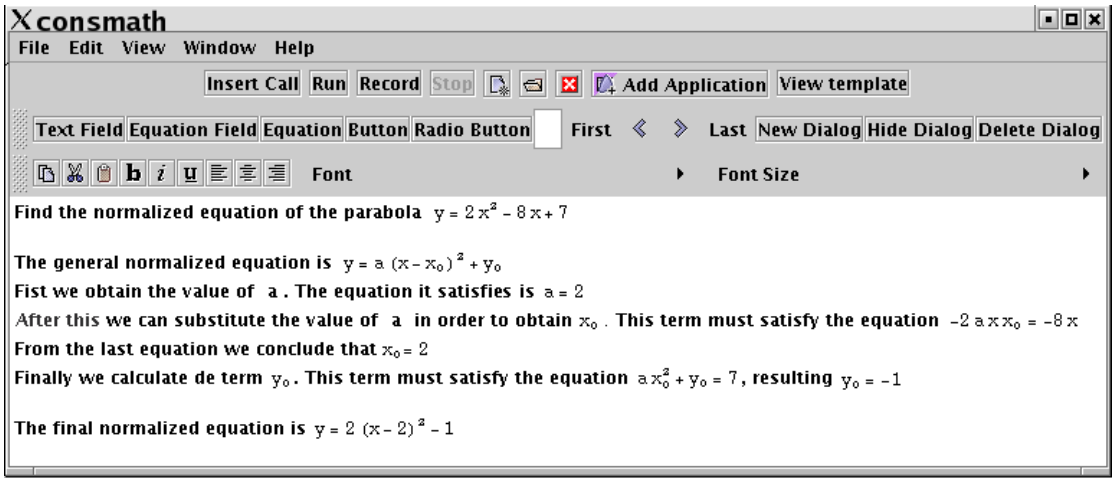

Figura 21: Documento inicial para el diseño de interacción

El diseño de la interacción de alto nivel, Figura 18, se puede comenzar con un documento en blanco, o se puede reutilizar un documento que plantee la resolución completa de un ejercicio, como el de la Figura 21. En este caso el trabajo del diseñador consiste en generalizar las expresiones y partir el documento en segmentos de forma que se puedan ir mostrando a los

alumnos consecutivamente conforme va resolviendo el problema. Por lo tanto, cada uno de estos segmentos se corresponderá con un diálogo interactivo distinto.

La generalización de los documentos se realiza mediante expresiones que dependen de los datos iniciales del problema, en este ejemplo la ecuación de la parábola. Gracias a esto la plantilla se podrá reutilizar para cualquier ecuación que cumpla las condiciones para las que está diseñada la plantilla.

Una vez generalizado el documento el diseñador ha de especificar los diálogos entre el sistema y el estudiante. En estos diálogos el estudiante introduce las respuestas al sistema. Como reacción a esta interacción el sistema le responde presentando nuevos diálogos o mostrando información sobre las respuestas anteriores.

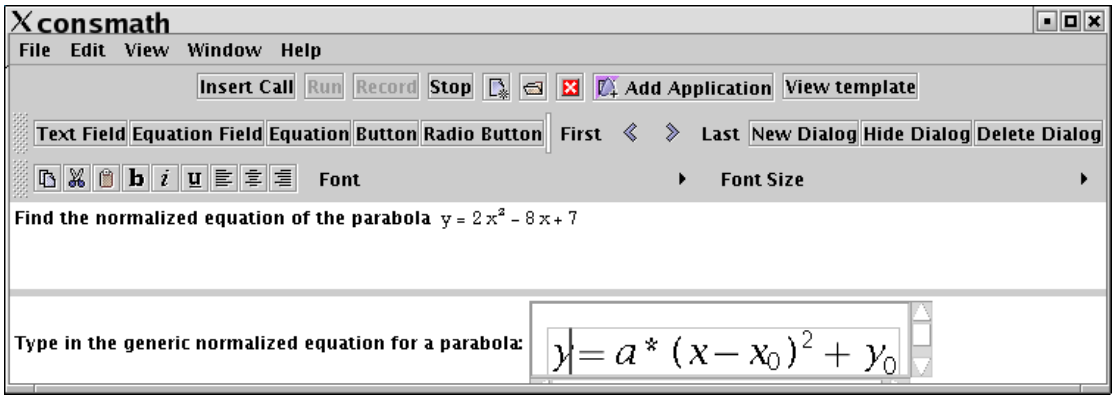

Figura 22: Diseñador Introduciendo una respuesta en ConsMath

El diseñador especifica las distintas respuestas del estudiante simplemente interactuando con los diálogos recién creados. Por ejemplo, en la Figura 22, el diseñador introduce la respuesta correcta a la pregunta formulada en el diálogo. Una vez introducida la respuesta el diseñador creará el siguiente diálogo interactivo, que producirá un nuevo segmento en el documento. Durante la ejecución, cuando el usuario esté en el diálogo anterior, podrá pasar al diálogo siguiente cuando introduzca una fórmula correspondiente a la introducida por el diseñador como ejemplo de dicha situación.

En la Figura 23 el diseñador ha modificado el diálogo anterior para sustituir la pregunta por una afirmación y a continuación ha creado un nuevo diálogo, para preguntar al alumno por dónde continuar con la resolución de las

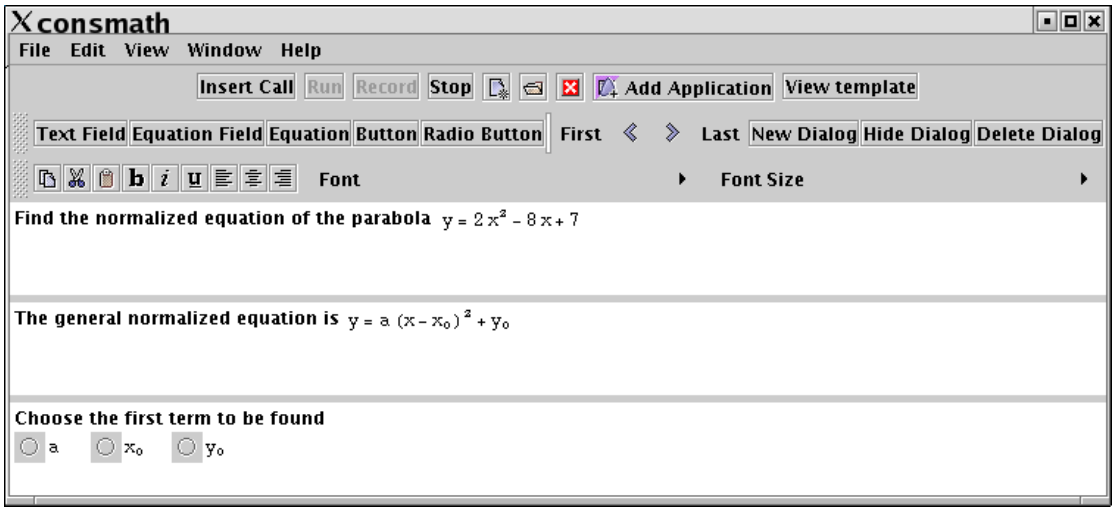

Figura 23: Especificación de una pregunta de respuesta múltiple

incógnitas. En este ejemplo el alumno puede comenzar a resolver el problema por varios caminos distintos, aunque lo correcto es comenzar por despejar *a*, después x_0, y por último y_0. Por lo tanto el profesor mostrará el mismo mensaje de advertencia al alumno si no elige resolver primero *a*, ya que es el único termino que no depende del valor del resto de variables.

En el ejemplo anterior el valor de la respuesta correcta era conocido por el diseñador del curso, sin embargo esto no será siempre así, ya que una misma plantilla puede utilizarse para generar problemas diferentes, y por lo tanto se pueden plantear preguntas a los alumnos cuya respuesta dependa del problema a resolver.

Para poder solucionar este tipo de casos el diseñador del curso dispone de los predicados. Los predicados son funciones Matemáticas que evalúan la respuesta del alumno para comprobar si es correcta o no. De esta forma, cuando el alumno introduzca una respuesta, el predicado se evaluará, de la misma forma que las restricciones que hemos visto en el apartado anterior. Sin embargo, a diferencia con las restricciones, el resultado de esta evaluación generará un evento especial que será interpretado por el agente de seguimiento. Gracias a este mecanismo, el diseñador puede abstraerse de la respuesta concreta del alumno y de las condiciones particulares del problema, ya que las decisiones se tomarán en función de una información de más alto nivel.

Para simplificar la edición de los documentos se siguen algunas heurísticas, como la segmentación automática cada vez que se crea una nueva zona de acción en la plantilla de interacción. Una vez en la zona de acción el diseñador puede crear nuevos segmentos en el documento, o eliminar u ocultar el segmento recién creado o incluso segmentos anteriores.

Otra heurística consiste en inhabilitar los segmentos anteriores cuando se pasa de una zona de decisión a una de acción, ya que eso significa que se desea crear un nuevo diálogo, por lo que en la mayoría de los casos es razonable limitar la interacción al nuevo diálogo creado.

Uno de los últimos aspectos que es necesario especificar es cómo se convierte una plantilla de interacción en un ejercicio concreto, lo que equivale a especificar las condiciones iniciales del problema a resolver por el alumno. En este punto existen varias alternativas. Se pueden crear condiciones iniciales constantes, aleatoriamente, o se puede dejar que las introduzca el alumno. En cualquier caso, el diseño de los diálogos ha de incorporar algún mecanismo que limite los parámetros iniciales a la clase de problemas para la que está diseñada la plantilla de interacción. En el caso de los problemas generados aleatoriamente esto se consigue mediante la especificación de qué partes de la expresión son aleatorias, y bajo qué condiciones se generan éstas. De esta forma se pueden generar expresiones arbitrarias dentro de las condiciones impuestas por el diseñador del curso.

Esta tarea debe realizarse al final del diseño de la plantilla por un motivo práctico de diseño, ya que lo más adecuado es crear una plantilla específica para generar las condiciones iniciales, y después incluir la plantilla de interacción abstracta, que heredará los valores iniciales de la plantilla de inicialización. De esta forma la plantilla abstracta se podrá reutilizar fácilmente, incluyéndola en otras plantillas, por ejemplo para crear otro tipo de valores iniciales, o simplemente como parte de un subproblema.

5.2.1.5 Aspectos colaborativos

La colaboración en ConsMath puede ser síncrona o asíncrona. En este apartado estudiaremos los aspectos colaborativos relativos a la integración de esta aplicación en FACT. Como hemos visto, dicha integración se hace a

través del agente de seguimiento de DFACT, el cual se encarga además de gestionar las plantillas de interacción.

El estado compartido de ConsMath consiste en el estado del documento que se está editando o ejecutando, junto con el contenido de las plantillas de interacción. Por lo tanto el problema se reduce a notificar al agente de seguimiento los cambios en los documentos, para que éste pueda enviarlos a FACT junto con los cambios en las plantillas.

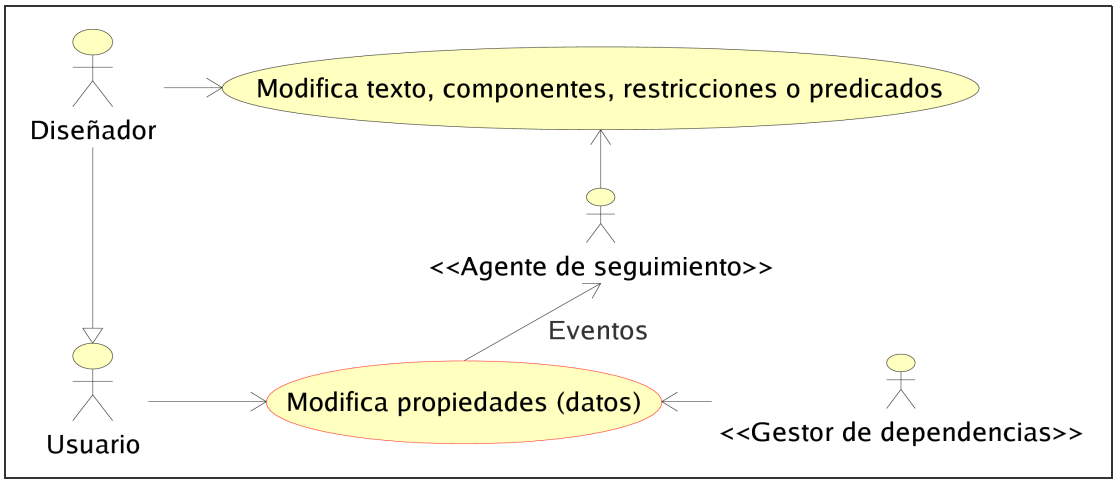

Figura 24: Actores que pueden modificar los documentos en ConsMath

Estudiando con más detalle cómo mantener el documento compartido, la primera opción es replicar todos los cambios, modificaciones en el texto, objetos insertados y borrados, cambios en las propiedades de los componentes, restricciones y predicados. Los actores que producen dichos cambios son los diseñadores, los usuarios finales, el propio agente de seguimiento y el gestor de restricciones, Figura 24. De estos actores, el agente de seguimiento y el gestor de restricciones realizan modificaciones en el documento de una forma predecible, que sólo depende del resto del estado compartido de la aplicación y de las acciones de los otros actores. Por lo tanto, sólo será necesario, para mantener el estado compartido de la aplicación, replicar las acciones que producen directamente los diseñadores y usuarios finales sobre el documento, pudiendo ignorar los cambios que se producen en las propiedades de forma automática mediante el gestor de restricciones, o los cambios que produce el agente de seguimiento cuando el

alumno responde a las preguntas y se crean nuevos segmentos o se modifica el documento.

En cuanto a la implementación, los aspectos colaborativos se reducen al tipo de acciones que realizan los diseñadores, ya que las acciones realizadas por los usuarios finales son un subconjunto de las que pueden realizar los diseñadores, al ser necesario que los diseñadores imiten las acciones de los usuarios finales, cuando programan por demostración las plantillas de interacción. Las acciones de modificación del texto o los componentes mediante el editor son las mismas que hemos visto en el editor colaborativo de FACT, apartado 4.4.3.

En ConsMath hay elementos nuevos en el estado compartido de los documentos respecto al editor colaborativo, como son: los segmentos de documentos, las dependencias entre componentes y los predicados. En el caso de las dependencias, éstas permiten que el valor de las propiedades de los componentes dependan del valor de otros componentes. Por lo tanto se producirán cambios automáticamente en dichas propiedades cuando cambien los valores de los que dependen. Con el fin de mantener el estado colaborativo no es necesario replicar todos los cambios en los componentes, ya que los cambios automáticos se reproducirán en todas las instancias de la aplicación, transmitiendo únicamente los cambios introducidos manualmente por los usuarios.

También hay que tener en cuenta que el agente de seguimiento se basa únicamente en los eventos de colaboración para tomar las decisiones o para crear las plantillas de interacción. La única excepción en este caso son los predicados, ya que, aunque son generados automáticamente en respuesta a una acción del usuario, el agente de seguimiento necesita los resultados de la evaluación de los predicados, pero no la respuesta introducida por el usuario. En este caso la estrategia consiste en generar un evento compuesto, con la acción del usuario que desencadenó la evaluación del predicado y el resultado de dicho predicado. Desde el punto de vista colaborativo el predicado se ignora, mientras que desde el punto de vista del agente de seguimiento ocurre lo contrario, y sólo se tiene en cuenta el predicado.

5.2.2 Diseño avanzado de plantillas de interacción

Como ya hemos visto, el diseño de las plantillas consiste en crear una estructura en forma de árbol que indica el orden de aparición de los distintos diálogos interactivos, junto con las condiciones de transición y selección del camino a seguir en el árbol. Los diálogos consisten normalmente en segmentos del documento, que se van añadiendo o eliminando del mismo, facilitando de esta forma que el alumno tenga acceso a la información anterior. Cada segmento a su vez está compuesto de texto y componentes para mostrar datos o para la introducción de información. Las propiedades de dichos componentes están relacionadas mediante restricciones que permiten además crear variables, predicados, o generar expresiones que inicializan los valores de los distintos elementos.

En este apartado nos centraremos en cómo utilizar los elementos anteriores para sacar el mayor partido a las plantillas de interacción, aumentando las posibilidades de reutilización y creando materiales didácticos complejos.

Uno de los aspectos más importantes al diseñar un ejercicio complejo o un curso es la división del diseño en varias plantillas de interacción. Esto aumentará las posibilidades de reutilización de las diferentes plantillas y facilitará la corrección del diseño. En ConsMath cada plantilla es el equivalente a una función. Esto permite dividir el diseño en plantillas más sencillas y con más posibilidades de reutilización. Desde el punto de vista del diseñador la sobrecarga de la división del diseño en varias plantillas es mínima, ya que para llamar a una plantilla desde otra únicamente hay que indicar el nombre.

En las plantillas de interacción no existe el concepto de argumentos de entrada y salida como ocurre con las funciones, ya que cuando se llama a una plantilla ésta puede utilizar todas las variables y componentes de la plantilla anterior, respetando el sistema de definición en ámbitos que hemos visto en el apartado 5.2.1.3. Cuando la plantilla llamada termina, los valores finales serán accesibles desde la plantilla que la llamó, ya que los resultados quedarán insertados en el documento. Por lo tanto, para facilitar la reutilización de las plantillas, es importante documentar cómo se han de usar, cuáles son las variables de entrada, y dónde quedan registrados los

resultados finales. Esta documentación se puede hacer en un segmento oculto al usuario final o con una anotación de DFACT, asociada por ejemplo a la raíz de la plantilla. El editor de anotaciones será por lo tanto muy útil para documentar las distintas partes de la plantilla de interacción.

Los usos más avanzados de las llamadas a plantillas se dan en la recursividad y los bucles. Gracias a la recursividad podemos diseñar una plantilla para por ejemplo resolver un límite mediante la regla de L'Hôpital. Para resolver este tipo de ejercicios el profesor puede crear una plantilla para resolver un problema como el de la ecuación 5.1, donde “f(x)”, “g(x)” y “c” son expresiones de entrada a la plantilla, por ejemplo mediante las variables “f”, “g” y “c” respectivamente.

$$\lim_{x \to c} \frac{f(x)}{g(x)} \tag{5.1}$$

La plantilla para resolver este problema puede mostrar el límite, usando las variables de entrada, y proponiendo al alumno distintos métodos de resolución, incluida la resolución directa, ya que puede ser trivial calcular o simplificar el cociente, y la aplicación de la regla de L'Hôpital, implementando éste último caso con el equivalente a un bucle en programación, con un mecanismo de recursión.

Para la opción en que la resolución es directa, sólo hay que diseñar un campo de entrada para que el alumno introduzca el resultado y crear un predicado que evalúe si la respuesta es correcta. El caso que permite calcular o simplificar el cociente es similar, aunque se puede añadir un primer paso para comprobar que el cociente se ha calculado o simplificado bien, antes de resolver definitivamente el límite.

En el caso de que el alumno elija aplicar L'Hôpital, el problema se reduce al límite de f'(x)/g'(x). También será conveniente crear un predicado que evalúe si es posible aplicar esta regla, de forma que se le pueda mostrar un mensaje de ayuda en caso de error. Si se puede aplicar esta regla, el diseñador sólo tiene que crear un nuevo segmento en el documento pidiéndole al alumno la fórmula de L'Hôpital y otro para las expresiones de las derivadas del numerador y denominador, que se validarán mediante los correspondientes predicados. Los resultados de ambas derivadas se han de asociar de nuevo a las variables *f* y *g*, tomando la precaución de que en el

mismo segmento del documento en que se redefinen estas variables no sea necesario utilizar los valores antiguos de las mismas. Una vez hecho esto sólo es necesario hacer una llamada a la misma plantilla de forma recursiva.

En este ejemplo la recursividad terminará cuando el alumno llegue a un caso en que el límite se resuelva de forma inmediata. En cuanto al resultado de cada llamada recursiva, no será necesario que el diseñador haga nada especial, ya que en este ejemplo el resultado es el de la última llamada, sin ser necesario realizar ninguna transformación.

También puede ocurrir que necesitemos utilizar los resultados de las llamadas una vez se ha devuelto el control a la plantilla original. Por ejemplo, en lugar de pedir al alumno que introduzca los valores de las derivadas del numerador y denominador podemos llamar a una plantilla que asista al alumno en la resolución de dichas derivadas. En este caso será necesario crear dos restricciones que asignen los resultados de las dos llamadas a la plantilla de derivadas a sendas variables de la plantilla de resolución de límites, para poder operar posteriormente con ellas. Gracias al mecanismo de definición de variables en distintos ámbitos para cada segmento, no habrá ningún problema en tener variables con el mismo nombre en distintos puntos del documento.

Una vez que se han diseñado las distintas plantillas para modelar la resolución de cada tipo de problema o situación, es necesario crear plantillas que inicialicen los datos, y creen los problemas que han de resolver los alumnos. Esto nos lleva a que existen dos tipos de plantillas, las que crean los enunciados y las que resuelven partes de los problemas. Por lo tanto, facilitará la utilización de ConsMath organizar estas plantillas de interacción, de forma que el usuario final o el profesor tengan accesibles las destinadas a comenzar los ejercicios, dejando para el diseñador las plantillas complementarias. La propuesta en ConsMath consiste en crear una organización jerárquica de las plantillas de interacción, con una estructura similar a los directorios de un sistema de archivos, estando las plantillas destinadas al usuario en un primer nivel de la jerarquía, y las plantillas complementarias en los siguientes niveles.

5.2.3 Ejemplos y pruebas realizadas

Como hemos visto en este capítulo, ConsMath es una herramienta creada para el diseño y ejecución de pequeños cursos o ejercicios reutilizables, y que funcionan como una aplicación colaborativa basados en DFACT, y que por lo tanto pueden beneficiarse del soporte de aprendizaje colaborativo guiado.

Para las pruebas de esta herramienta hemos limitado el ámbito a ejercicios de cálculo simbólico, ya que son éstos los que mejor pueden aprovechar la potencia de cálculo que ofrece el servidor de evaluación mediante el uso de las restricciones y además pueden ser diseñados usando el conjunto de componentes de la paleta actual de ConsMath.

En primer lugar se han diseñado pequeños ejercicios para probar características específicas de ConsMath, como un conjunto de plantillas de resolución de límites que incluye la relativa a la regla de L'Hôpital que hemos descrito en el apartado 5.2.2. Este conjunto de plantillas permitió probar el paso de variables entre plantillas y la recursividad.

También se han realizado pruebas sistemáticas que han consistido en el diseño de una colección de problemas interactivos extraídos de un libro, [Simmons 1981], sobre Ecuaciones Diferenciales Ordinarias. Esta colección de problemas cubre los apartados de ecuaciones homogéneas de primer orden, ecuaciones exactas, factores integrantes y ecuaciones lineales de primer y segundo orden. Dos profesores, expertos en Matemáticas, han creado mediante demostración los tipos de problemas principales de resolución de ecuaciones de estos apartados del libro. El resultado ha sido un conjunto de quince plantillas de interacción que permiten resolver 110 ejercicios enunciados en el propio libro, y que además incluyen la posibilidad de ser utilizadas con un enunciado generado aleatoriamente, variando las condiciones iniciales de los problemas.

Las pruebas realizadas demuestran que, sin mucha dificultad, y sin conocimientos de programación, ConsMath permite crear aplicaciones para la resolución de ejercicios de Matemáticas relativamente complejos y altamente reutilizables, gracias a la abstracción y generación aleatoria de problemas.

Las pruebas también han permitido encontrar algunos problemas de usabilidad en el editor gráfico de ecuaciones utilizado, que está desarrollado por terceros. La solución ha consistido en añadir la opción de introducir las ecuaciones como texto, como alternativa al editor gráfico, lo cual en el caso de las expresiones simples es mucho más cómodo para el diseñador o los estudiantes. La solución a medio plazo consiste en crear un nuevo editor de ecuaciones gráfico más sencillo de utilizar. El editor debería permitir trabajar con un lenguaje orientado a contenido e imponer al usuario introducir expresiones Matemáticas válidas. Además el editor debe permitir crear expresiones en modo texto, sin que sea necesario utilizar la paleta gráfica para insertar los operadores más habituales.

Para la explotación de ConsMath también será necesario incluir plantillas ya diseñadas, como parte de una biblioteca estándar, especializada en las distintas materias a las que se dirija su aplicación. De esta forma el diseñador puede centrarse en los aspectos novedosos de las plantillas que desea crear, reutilizando las plantillas existentes para solucionar los problemas más comunes. Los resultados de las pruebas realizadas han mostrado que la creación de estas bibliotecas especializadas es factible, gracias a la capacidad de reutilización que permite ConsMath, y permitirá disminuir el tiempo de desarrollo de nuevas plantillas de interacción.

Esta herramienta está diseñada para la creación de pequeños cursos colaborativos. Si se pretende crear cursos más grandes, por ejemplo uno completo sobre una asignatura, surgen algunas limitaciones, como el hecho de que ConsMath no incluye un modelo del alumno que por ejemplo indique qué ejercicios ha respondido correctamente y cuáles no. Esta limitación se puede evitar integrando ConsMath con un sistema de gestión de cursos como Moodle[47].

[47] Moodle es un sistema para la gestión de cursos de código libre. Está disponible en http://www.moodle.com

Capítulo 6. Conclusiones

6.1 Principales aportaciones

El objetivo principal de esta tesis ha sido el de estudiar herramientas que faciliten la creación de material didáctico que permita el aprendizaje colaborativo y simplifique las tareas de supervisión, guiado y revisión del trabajo del alumno por parte del profesor en un entorno de educación a distancia.

Este objetivo se ha conseguido mediante cuatro aportaciones principales:

- Se ha creado un modelo, que hemos llamado aprendizaje colaborativo guiado, de los requisitos que han de poseer las aplicaciones finales.
- Se ha diseñado un framework, FACT, que simplifica la creación de aplicaciones que soporten el aprendizaje colaborativo guiado, de forma que este proceso no añada una complejidad considerable respecto a la creación una aplicación normal, no colaborativa.
- Se ha creado una técnica de programación por demostración que permite diseñar los aspectos interactivos de aplicaciones basadas en FACT. Esta técnica se ha validado mediante la implementación de una herramienta de autor, DFACT, que permite crear nuevas aplicaciones colaborativas a partir de un conjunto de aplicaciones básicas donde realizar las demostraciones.
- Se ha diseñado una herramienta de autor, ConsMath, orientada a crear cursos o problemas reutilizables con un alto contenido en Matemáticas simbólicas. Esta herramienta es una especialización de DFACT para la creación de aplicaciones educativas.

Estas aportaciones contribuyen a simplificar el proceso de creación, no sólo de aplicaciones educativas, sino en general, de aplicaciones que requieran soportar trabajo colaborativo y análisis, tanto de forma síncrona como asíncrona.

Las aportaciones a las que ha dado lugar esta tesis se han difundido mediante publicaciones en congresos y revistas especializadas, las referencias a dichas publicaciones se pueden encontrar en el apéndice.

6.2 Trabajo futuro

Las herramientas desarrolladas están orientadas a la creación de material didáctico, por parte de profesores, diseñadores o programadores. Por lo tanto las pruebas realizadas han estado enfocadas a esta tarea. Sin embargo, puesto que la mayoría de las aplicaciones que se pueden crear con estas herramientas son aplicaciones didácticas, también ayudaría a mejorar las estas aplicaciones, sobre todo en los aspectos relacionados con la usabilidad, la realización de pruebas con estudiantes. Un primer trabajo futuro será por tanto realizar pruebas con usuarios finales con las aplicaciones y cursos desarrollados.

A corto plazo se pueden mejorar algunas de las características de las aplicaciones desarrolladas para la validación de los modelos y diseños expuestos en esta tesis.

Una funcionalidad interesante que se puede añadir a FACT es la capacidad de hacer un seguimiento del trabajo en una sesión siguiendo un orden temporal. Esto puede ser útil por ejemplo para que un profesor pueda hacer un recorrido exacto de los pasos que han dado un grupo de alumnos para resolver un problema, incluyendo los errores que han cometido en el camino.

En cuanto a la resolución de ejercicios o problemas en ConsMath, los alumnos pueden trabajar de forma asíncrona o síncrona. En la resolución asíncrona los alumnos pueden trabajar en distintos momentos sobre el mismo ejercicio. Sin embargo los problemas surgen cuando los alumnos se reparten las tareas pero trabajan simultáneamente para resolver distintas partes de un mismo ejercicio. Este problema no se puede resolver directamente en FACT, ya que al unir varias ramas las acciones se deben ejecutar en un contexto distinto del original, y no hay garantía de que esto sea posible.

Para resolver este problema es necesario que las plantillas se diseñen para que puedan resolverse por partes y después se puedan integrar los resultados. La integración de los resultados se podría conseguir ofreciendo la alternativa de incorporar manualmente los resultados de los subproblemas. Un trabajo futuro sería buscar mecanismos para simplificar el diseño y uso de

este tipo de plantillas, evitando que los alumnos tengan que introducir manualmente los resultados que han conseguido por separado.

Apéndice

Artículos en revistas y capítulos de libro

- Mora, M.A., Moriyón, R., Saiz, F., 2004, Role-Based Specification of the Behaviour of an Agent for the Interactive Resolution of Mathematical Problems, Lecture Notes in Computer Science, vol. 3220, Springer Verlag,p. 187-196
- Mora, M.A., Moriyón, R., Saiz, F., 2004, Modeling Interactivity for Mathematics Learning by Demonstration, Lecture Notes in Computer Science, vol. 3119, p. 265-275
- Mora, M.A., Moriyón, R., Saiz, F., 2003, Building Mathematics Learning Applications by Means of CONSMATH, Frontiers in Education, vol. 2, p. F3F_1- F3F_6
- Mora, M.A., Moriyón, R., Saiz, F., 2003, Developing applications with a Framework for the Analysis of the Learning Process and Collaborative Tutoring, International Journal of Continuing Engineering Education and Lifelong Learning (IJCEELL), vol. 13, p. 268-279
- Mora, M.A., Moriyón, R., 2001, Collaborative Analysis and Tutoring: The FACT Framework, Advanced Learning Technology: Issues, Achievements and Challenges, IEEE Computer Society Press, p. 82-85
- Mora, M.A., Moriyón, R., 2001, Guided collaborative chess tutoring through game history analysis, Computers and Education. Towards an Interconnected Society, Kluwer, p. 243-250

Contribuciones a Congresos

- Mora, M.A., Moriyón, R., Saiz, F., 2003, Desarrollo de aplicaciones para el aprendizaje colaborativo guiado: Requisitos y soluciones, Interacción 2003
- Mora, M.A., Moriyón, R., Saiz, F., 2002, Mathematics Problem-based Learning through Spreadsheet-like Documents, Proceedings of Second International Conference on the Teaching of Mathematics
- Mora, M.A., Moriyón, R., 2001, Guided collaborative tutoring through learning history analysis: The FACT Framework, Proceedings of First European Conference on Computer-Supported Collaborative Learning

Bibliografía

[Appelt 1996] Appelt, W., Busbach, U., The BSCW system: a WWW-based application to support cooperation of distributed groups, IEEE Proceedings of WET ICE'96, Enabling Technologies: Infrastructure for Collaborative Enterprises, 1996, p. 304-309.

[Baecker 1993] Baecker, R. M., Readings in Groupware and Computer-Supported Cooperative Work, Morgan Kaufmann Publishers, Inc., 1993.

[Baker 1996] Baker, M.J.,& Lund, K., Flexibly structuring the interaction in a CSCL environment, Proceedings of the European Conf on A.I. in Edu., 1996, p. 401-407.

[Benford 1997] Benford, S., Snowdon, D., Colebourne, A., O'Brien, J. y Rodden, T., Informing the Design of Collaborative Virtual Environments, Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge, ACM Press, 1997, p. 71 - 80.

[Boyle 2002] Boyle, M., Greenberg, S., GroupLab Collabrary: A Toolkit for Multimedia Groupware, ACM CSCW 2002 Workshop on Network Services for Groupware, ACM Press, 2002, p. (4 págs.).

[Calvary 1997] Calvary, G., Coutaz, J. y Nigay, L., From Single-User Architectural Design to PAC: A Generic Software Architecture Model for CSCW, Proceedings CHI'97, 1997, p. 242-249.

[Cameron 1999] Cameron, T., Barrows, H.S., Crooks, S.M., Distributed Problem-Based Learning at Southern Illinois University School of Medicine, Proceedings of Computer Support for Collaborative Learning '99, 1999, p. 86-93.

[Castells 1999] Castells, P., Szekely, P., Presentation Models by Example, Design, Specification andVerification of Interactive Systems '99, Springer-Verlag, 1999, p. 100-116.

[Chen 1999] Chen, D. y Sun, C., A Distributed Algorithm for Graphic Objects Replication in Real-time Group Editors, Proceedings Group 99, 1999, p. 121-130.

[Chen 2003] Chen, W., Dolonen, J., Wasson, B., Supporting Collaborative Knowledge Building with Intelligent Agents, Knowledge-Based Intelligent Information and Engineering Systems: 7th International Conference, KES, Springer-Verlag GmbH, 2003, p. 238-244.

[Chung 1998] Chung G., Dewan P., Rajaram S., Generic and Composable Latecomer Accomodation Service for Centralized Shared Systems, EHCI'98, 1998, p. 129-147.

[Cobos 2002] Cobos, R, Alamán, X., Creating e-books in a distributed and collaborative way, Journal of Electronic Library on Electronic book for Education, 2002, vol. 20, núm. 4, p. 288-295.

[Cockburn 1997] Cockburn, A., Dale, T., CEVA: A Tool for Collaborative Video Analysis, Proceedings Group 97, 1997, p. 47-55.

[Cole 1998] Cole, M., Cultural psychology: A once and future discipline, Belknap Press, 1998.

[Cortés 1996] Cortés, M. y Mishra, P., DCWPL: A Programming Language For Describing Collaborative Work, Proceedings CSCW'96, 1996, p. 21-29.

[Crook 1994] Crook, Ch., Ordenadores y aprendizaje colaborativo, Ministerio de Educación y Cultura; Ediciones Morata, 1994.

[Dewan 1994] Dewan, P., Choudhary, R., Shen, H., An Editing-based Characterization of the Design Space of Collaborative Applications, Journal of Organizational Computing, 1994, vol. 4, núm. 3, p. 219-240.

[Dewan 1995] Dewan, P., Choudhary, R., Coupling the User Interfaces of a Multiuser Program, ACM Tranactions on Computer-Human Interaction, 1995, 2, núm. 1, p. 1-39.

[Dewan 1998] Dewan, P., Shen, H., Flexible Meta Access-Control for Collaborative Applications, Proceedings CSCW'98, 1998, p. 247-256.

[Díez 2003] Díez, F. Moriyón, R., Strategies for the Interactive Resolution of Calculus Problems, Computational Science- Iccs 2003, Springer-Verlag, 2003, p. 791-800.

[Dillenbourg 1995] Dillenbourg, P., Baker, M., Blaye, A.,& O'Malley, C., The evolution of research on collaborative learning, Elsevier, 1995.

[Dillenbourg 1999] Dillenbourg, P., Collaborative learning: cognitive and computational approaches, Pergamon, 1999.

[Doise 1984] Doise, W., y Mugny, G., The Social Development of the Intellect, Pergamon, 1984.

[Ellis 1989] Ellis, C.A., Gibbs, S.J., Concurrency control in groupware systems, Proceedings of the ACM SIGMOD'89 International conference on Management of data , 1989, p. 399-407.

[Ellis 1997] Ellis, C., Team automata for groupware systems, Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge, 1997, p. 415 - 424.

[Engeström 1987] Engeström, Y., Learning by expanding: An activity-theoretical approach to developmental research, Orienta-Konsultit, 1987.

[Fitzgerald 2004] Fitzgerald, R., Findlay, J., A computer-based research tool for rapid knowledge-creation: A retro-viral agent of change, Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications, 2004, p. 1979-1984.

[Garrett 2005] Garrett, J.J., Ajax: A New Approach to Web Applications, 2005, http://www.adaptivepath.com/publications/essays/archives/000385.php.

[Graham 1996] Graham, T.C.N., Morton, C.A., Urnes, T., ClockWorks: Visual Programming of Component-Based Software Architectures, Journal of Visual Languages and Computing, 1996, vol. 7, núm. 2, p. 175-196.

[Graham 1999] Graham T.C.N. y Grundy J., External Requirements of Groupware Development Tools, Engineering for Human-Computer Interaction, Kluwer Academic Publishers, 1999, p. 363-376.

[Greenberg 1991] Greenberg, S., Computer-Supported Cooperative Work and Groupware, Academinc Press, 1991.

[Greenberg 1997] Greenberg, S., The Evolution of CSCW, Past, Present and Future Developments, SIGCHI Bulletin, , vol. 29, núm. 2, p. 20-29.

[Greenberg 1999] Greenberg, S., Roseman, M., Groupware Toolkits for Synchronous Work, Computer-Supported Cooperative Work (Trends in Software 7), John Wiley & Sons Ltd, 1999, p. 135-168.

[Greif 1988] Greif, I., Computer-Supported Cooperative Work: A Book of Readings, Morgan Kaufman Publishers, Inc., 1988.

[Grinter 1998] Grinter, R. E., Recomposition: Putting It All Back Together Again, Proceedings CSCW'98, 1998, p. 393-402.

[Gutwin 1996] Gutwin, C., Roseman, M., Greenberg, S., A Usability Study of Awareness Widgets in a Shared Workspace Groupware System, Proceedings CSCW'96, 1996, p. 258-267.

[Hernández 2004] Hernández, D., Asensio, J.I., Dimitriadis, Y.A., IMS learning design support for the formalization of collaborative learning patterns, Proceedings of Advanced Learning Technologies 2004, 2004, p. 350- 354.

[Howe 1991] Howe, C., Tolmien, A., y Anderson, A., Information technology and groupwork in physics, Journal of Computer Assisted Learning, 1991, núm. 7, p. 133-143.

[Howe 1992] How, C., Tomie, A., Anderson, A., y MacKenzie, M., Conceptual knowledge in physics: The role of group interaction in computer supported teaching, Learning and Instruction, 1992, núm. 2, p. 161-183.

[Hudak 2000] Hudak, P., The Haskell School of expression learning functional programming through multimedia, Cambridge University Press, 2000.

[Ip 2003] Ip, A., Canale, R. , Supporting Collaborative Learning Activities with SCORM, Proceedings EDUCAUSE in Australasia 2003, 2003, p. 669-678.

[Jackson 1999] Jackson, L.S., Grossman, Ed, Integration of Synchronous and Asynchronous Collaboration Activities, ACM Computing Surveys, 1999, vol. 31, núm. 2es, p. No. 12.

[Kahn 2001] Ken Kahn, Generalizing by Removing Detail: How Any Program Can Be Created by Working with Examples, Your Wish Is My Command. Programming by Example, Morgan Kaufmann Publishers, 2001, p. 21-43.

[Koedinger 2004] Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B. M., Hockenberry, M., Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration, Proceedings of the Seventh Annual Conference on Intelligent Tutoring Systems (ITS), Springer Verlag, 2004, p. 162-174.

[Koschmann 1996] Koschmann, T., Kelson, A. C., Feltovich, P. J., Barrows, H. S., Computer-Supported Problem-Based Learning: A principled approach to the use of computers in collaborative learning, CSCL, theory and practice of an emerging paradigm, L. Erlbaum Associates, 1996, p. 83-124.

[Kozma 1999] Kozma, R. B., Students collaborating with computer models and physical experiments, CSCL'99 Proceedings, 1999, p. 314-322.

[Lamberty 2001] Lamberty, K.K., Mitchell, A., Owensby, J.N., Sternberg, D., Kolodner, J.L., SMILE: Promoting Transfer in a Design-Based Science Classroom, Proceedings of the DESIGN: Connect, Create, Collaborate conference, 2001, p. 74-80.

[Leontjev 1981] Leontjev, A. N., The problem of activity in psychology, The Concept of Activity in Soviet Psychology, Sharp, 1981, p. 37-71.

[Li 1998] Li, D., Muntz, R., Collaborative Objects Coordination Architecture, Proceedings CSCW 98, 1998, p. 179-188.

[Li 1999] Li, W., Wang, W., Marsic, I., Collaboration Transparency in the DISCIPLE Framework, Proceedings Group'99, 1999, p. 326-335.

[Light 1985] Light, P., y Glachan, M., Facilitation of problem solving through peer interaction, Educational Psychology, 1985, núm. 5, p. 217-225.

[Light 1994] Light, P., Littleton, K., Messer, D., y Joiner, R., Social and communicative processes in computer-based problem solving, European Journal of Psychology of Education, 1994, núm. 9(1), p. 93-109.

[Lindstaedt 1997] Lindstaedt, S. N., Bridging the Gap between Face-to-Face Communication and Long-term Collaboration, Proceedings Group'97, 1997, p. 331-340.

[Lipponen 1997] Lipponen, L., Hakkarainen, K., Developing Culture of Inquiry in Computer-Supported Collaborative Learning, CSCL'97 Proceedings, 1997, p. 164-168.

[Lopez 2003] Lopez, P.G.; Skarmeta, A.F.G., ANTS framework for cooperative work environments, IEEE Computer, 2003, vol. 36, núm. 3, p. 56-62.

[Marsic 1999] Marsic, I., Dorohonceanu, B., An application framework for synchronous collaboration using JavaBeans, Proceedings of the 32nd Annual Hawaii International Conference on System Sciences. HICSS-32 , 1999, p. (10 págs.).

[Martin 2002] Martin, B., Mitrovic, A., Automatic Problem Generation in Constraint-Based Tutors, ITS 2002, LNCS 2363, Springer Verlag, 2002, p. 388-398.

[McDaniel 1999] McDaniel, R.G., Myers, B.A., Getting more out of programming-by-demonstration, Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit, ACM Press, 1999, p. 442 - 449.

[McLaren 2004] McLaren, B. M., Koedinger, K. R., Schneider, M., Harrer, A., Bollen, L., Toward Cognitive Tutoring in a Collaborative Web-Based environment, Proceedings of the Workshop of AHCW 04, Rinton Press, 2004, p. 167-179.

[Moriyon 94] Moriyon, R., Szekely, P., Neches, R., Automatic generation of help from interface design models, Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence, 1994, p. 225 - 231.

[Munson 1996] Munson, J. y Dewan, P., A Concurrency Control Framework for Collaborative Systems, Proceedings CSCW 96, 1996, p. 278-287.

[Muukkonen 1999] Muukkonen, H., Hakkarainen, K., Lakkala, M., Collaborative technology for facilitating progressive inquiry: Future learning environment tools, Proceedings of CSCL'99, 1999, p. 406-415.

[Myers 1990] Myers, B.A., Creating user interfaces using programming by example, visual programming, and constraints, ACM Transactions on Programming Languages and Systems (TOPLAS), 1990, vol. 12, núm. 2, p. 143 - 177.

[Myers 1993] Myers, B.A., McDaniel, R.G., Kosbie, D.S., Creating Complete User Interfaces by Demonstration, Proceedings of INTERCHI'93: Human Factos in Computing Systems, 1993, p. 293-300.

[Myers 1997] Myers, B.A., McDaniel, R.G., Miller, R.C., Ferrency, A.S., Faulring, A., Kyle, B.D., Mickish, A., Klimovitski, A., Doane, P., The Amulet Environment: New Models for Effective User Interface Software Development, IEEE Transactions on Software Engineering, 1997, vol. 23, núm. 6, p. 347-365.

[Myers 2001] Myers, B.A., McDaniel, R., Demonstrational Interfaces: Sometimes You Need a Little Intelligence, Sometimes You Need a Lot, Your Wish Is My Command. Programming By Example, Morgan Kaufmann Publishers, 2001, p. 45-60.

[Orozco 2004] Orozco, P, Asensio, J.I., Garcia, P., Dimitriadis, Y.A., Pairot, C., A Decoupled Architecture for Action-Oriented Coordination and Awareness Management in CSCL/W Frameworks, Lecture Notes in Computer Science, 2004, núm. 3198, p. 246-261.

[Patterson 1995] Patterson, J.F., A taxonomy of architectures for synchronous groupware applications, ACM SIGOIS Bulletin, 1995, vol. 15, núm. 3, p. 27 - 29.

[Perret-Clermont 1980] Perret-Clermont, A.N., Social Interaction and Cognitive Development in Children, Academic Press, 1980.

[Piaget 1932] Piaget, J., The Moral Judgment of the Child, Routledge & Kegan Paul, 1932.

[Ressel 1996] Ressel, M., Nitsche-Ruhland, D., Gunzenhäuser, R., An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors, Proceedings CSCW 96, 1996, p. 288-297.

[Ressel 1999] Ressel, M., Gunzenhäuser, R., Reducing the Problems of Group Undo, Proceedings Group 99, 1999, p. 131-139.

[Roseman 1992] Roseman, M., Greenberg, S., GROUPKIT: a groupware toolkit for building real-time conferencing applications, Proceedings of the 1992 ACM conference on Computer-supported cooperative workACM Press, 1992, p. 43-50.

[Schuckmann 1999] Schuckmann, C., Schümmer, J., Seitz, P., Modeling Collaboration using Shared Objects, Proceedings ACM SIGGROUP'99, ACM Press, 1999, p. 189-198.

[Shweder 1993] Shweder, R. A. y Sullivan, W. M., Cultural psychology: Who needs it?, Annual Review of Psychology, 1993, núm. 44, p. 497-523.

[Simmons 1981] Simmons, G. F., Differential equations: with applications and historical notes, McGraw-Hill, 1981.

[Smith 1998] Smith, G. y O'Brien, J., Re-coupling Tailored User Interfaces, Proceedings of the 1998 ACM conference on Computer supported cooperative work, 1998, p. 237-246.

[Spaulding 1969] Spaulding, WB., The undergraduate medical curriculum (1969 model): McMaster university, Canadian Medical Association Journal, 1969, vol. 100, núm. 14, p. 659-664.

[Steinfield 1999] Steinfield, C., Chyng-Yang Jang, Pfaff, B., Supporting Virtual Team Collaboration: The TeamSCOPE System, Proceedings Group'99, 1999, p. 81-90.

[Steinkuehler 2002] Steinkuehler, C.A., Derry, S.J., Woods, D.K., Hmelo-Silver, C.E., The STEP Environment for Distributed Problem-Based Learning on the World Wide Web, Proceedings of CSCL'02, 2002, p. 217-226.

[Suleiman 1997] Suleiman, M., Cart, M. y Ferrié, J., Serialization of Concurrent Operations in a Distributed Collaborative Environment, Proceedings Group 97, 1997, p. 435-445.

[Sun 1996] Sun, C., Yang, Y., Zhang, Y. y Chen, D., A consistency model and supporting schemes for real-time cooperative editing systems, Proceedings of the 19th Australian Computer Science Conference, 1996, p. 582-591.

[Sun 1997] Sun, C., Jia, X, Zhang, Y. y Yang, Y., A generic Operation Transformation Scheme for Consistency Maintenance in Real-time Cooperative Editing Systems, Proceedings Group 97, 1997, p. 425-434.

[Tiessen 1999] Tiessen, E.L., Ward, D.R., Developing a Technology of Use for Collaborative Project-Based Learning, Proceedings of Computer Support for Collaborative Learning 1999, 1999, p. 631-639.

[Verdejo 2002] Verdejo, M.F., Barros, B., Read, T., Rodriguez-Artachoe, M., A System for the Specification and Development of anEnvironment for Distributed CSCL Scenarios, Intelligent Tutoring Systems : 6th International Conference, ITS 2002, 2002, p. 139-148.

[Vygotsky 1978] Vygotsky, L., Mind in Society: The Development of Higher Psychological Processes, Harvard University Press, 1978.

[Vygotsky 1993] Vygotsky, Lev S., Pensamiento y Lenguaje, Ediciones Fausto, 1993.

[Weber 1997] Weber, M., Partsch, G., Höck, S., Scheider, G., Scheller-Hony, A. y Schweitzer, J., Integrating Synchronous Multimedia Collaboration into Workflow Management, Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge, ACM Press, 1997, p. 281-290.

[Wertsch 1981] Wertsch, J. V., The Concept of Activity in Soviet Psychology, Sharpe, 1981.

[Wertsch 1988] Wertsch, J. V., Vygotsky y la formación social de la mente, Ediciones Paidós, 1988.

[Yang 2002] Yang, G.G., A Uniform Meta-Model for Modeling Integrated Cooperation, Proceedings of the 2002 ACM symposium on Applied computing, 2002, p. 322 - 328.

[Ziewer 2002] Ziewer, P., Seidl, H., Transparent teleteaching, ASCILITE 2002, UNITEC Institute of Technology, Auckland, New Zealand, 2002, p. 749-758.