Diseño de un Entorno Colaborativo y su Aplicación a Plataformas de Aprendizaje.

Departamento de la Ingeniería de la Información y las Comunicaciones.
Universidad de Murcia.

Dirigida por
Dr. Antonio F. Gómez Skarmeta

Realizada por
Mª Antonia Martínez Carreras

2005

D. Antonio Fernando Gómez Skarmeta, Profesor Titular de Universidad del Área de Telemática en el Departamento de Ingeniería de la Información y las Comunicaciones,

AUTORIZA:

La presentación de la Tesis Doctoral titulada "Diseño de un Entorno Colaborativo y su Aplicación a Plataformas de Aprendizaje", realizada por D. Mª Antonia Martínez Carreras, bajo mi inmediata dirección y supervisión, en el Departamento de Ingeniería de la información del Conocimiento, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia.

En Murcia, a 15 de Mayo de 2005

Agradecimientos

Mis agradecimientos van dirigidos a todos aquellos que creyeron en mí en la realización de este trabajo, incluyendo tanto a los que colaboraron en los desarrollos aquí presentados como a los que me animaron cuando el camino se hizo difícil.

Son muchas las personas a las cuales les agradezco de todo corazón su apoyo incondicional, tantas que necesitaría muchas hojas para escribir sus nombres. A todos los que se han cruzado en este camino, gracias por vuestra ayuda.

Índice de Contenidos.

Índice de Figuras.

Índice de Tablas.

Capítulo Primero.

Introducción.

En los últimos años está cobrando relativa importancia las aplicaciones diseñadas para la colaboración entre personas o grupo de personas. Este auge se puede apreciar en diferentes ámbitos de la sociedad, como pueden ser tanto el amplio mundo de los negocios como el ámbito educacional, pudiendo conectar a un grupo de trabajadores en el caso primero o un profesor con sus alumnos así como al grupo de estudiantes en el caso segundo.

Más concretamente el uso de las tecnologías de la información y las comunicaciones (TIC) han adquirido gran importancia en el desarrollo de estas aplicaciones. Así pues emerge el campo CSCW (Computer Supported Cooperative Work) con el objetivo de estudiar tanto el desarrollo de las aplicaciones de trabajo en grupo como el impacto que provocan en la sociedad.

Los entornos colaborativos educacionales, así como las herramientas educacionales están tomando grandes repercusiones en nuestra sociedad, llevándose a cabo en la actualidad numerosos proyectos de investigación acerca de su construcción y su uso en comunidades educacionales.

Las investigaciones llevadas a cabo durante estos años han estado basadas en la creación de entornos colaborativos y su aplicación en entornos educacionales, a través de las cuales se han realizado una serie de estudios sobre las ventajas y desventajas que proporcionan dichos entornos, tanto a nivel de arquitectura como a nivel de herramientas.

Es un hecho que dichos entornos ofrecen una serie de posibilidades al alumno enriqueciendo el proceso de aprendizaje. En el desarrollo de un entorno educacional se requiere que tanto pedagógos como informáticos aunen sus conocimientos con el objetivo de obtener un entorno que se adecue con las necesidades tanto de proferores como de alumnos.

En las investigaciones llevadas a cabo se han abarcado dos enfoques en el aprendizaje: el colaborativo y el cooperativo. A través de las teorías colaborativas hemos analizado las características de un sistema CSCL así como un campo nuevo que emerge dentro de este último, que es el aprendizaje por descubrimiento.

Para llevar a cabo la tarea colaborativa es de vital importancia encontrar una arquitectura sobre la que se base la colaboración y que sea adaptable, reutilizable y que permita incorporarle una serie de herramientas que posibiliten realizar actividades a los usuarios con el entorno. En esta búsqueda también se pretende que las herramientas desarrolladas puedan ser también reutilizables en diferentes entornos.

En la actualidad existen sistemas educacionales CSCL que requieren la integración de nuevos sistemas que permitan la colaboración síncrona, permitiendo que dicha integración no afecte a la arquitectura del entorno subyacente.

Nos enfrentamos a dos ramas de investigación enlazadas, una de ellas es la búsqueda de la arquitectura que facilite la colaboración así como el estudio de las funcionalidades que deberían mostrar las herramientas necesarias para desarrollar la tarea colaborativa y la otra es la definición de un sistema colaborativo fácilmente integrable en cualquier entorno educacional.

Así pues mediante los proyectos EDUSI, PUPITRE-NET y "Evaluación de Aplicaciones de Tele-archivos, trabajo en grupo y videoconferencia, y su impacto en la Tele-Enseñanza en el ámbito de la Educación de la región de Murcia", este último financiado por la fundación Séneca, hemos estado analizando y realizando una serie de experiencias con el entorno de enseñanza a distancia llamado JLE "Java Learning Environment", actualmente EDUSTANCE [EDU]. Dicho entorno caracterizado por ofrecer soporte al aprendizaje cooperativo, basado en contenidos, nos ha servido como punto de referencia para el estudio y la caracterización de las herramientas que se deben proveer para facilitar la tarea educacional tanto a alumnos como a profesores [Gom00].

Durante la investigación llevada a cabo en el entorno JLE, se ha estudiado la incorporación de herramientas síncronas realizando una serie de estudios para obtener las ventajas y desventajas de este sistema, las cuales servirán para el diseño del sistema colaborativo.

En la investigación previa a lo que constituye nuestro resultado se ha abarcado el estudio de la arquitectura ANTS [Gar02] resultado de la tesis doctoral de Pedro García López, de la cual se ha obtenido las bases para la propagación de eventos en la colaboración síncrona así como las nociones para la creación de una arquitectura colaborativa. A través de la plataforma ANTS se han estudiado tanto los aspectos para la inclusión de componentes así como la creación de una arquitectura genérica para la conexión de entornos heterogéneos. Estos estudios han servido de punto de referencia para el diseño de la plataforma colaborativa que constituye uno de nuestros resultados.

1. Objetivos

Más concretamente, los resultados de este trabajo de investigación han sido parcialmente financiados por los proyectos europeos ITCOLE [Lei01] y COLAB

[Col04]. Mediante el trabajo desarrollado en ITCOLE se ha desarrollado un sistema colaborativo que pude ser integrado en entornos ya existentes. Como prueba de concepto de dicho sistema, se realizó la integración en dos entornos Web, BSCL [BSCL] y FLE [FLE03].

En la integración de este nuevo sistema colaborativo dentro de cada uno de los entornos se necesita que dicho sistema cumpla los siguientes objetivos:

- Uno de los aspectos principales es la comunicación entre las herramientas síncronas y el entorno subyacente. El ciclo de vida de las herramientas síncronas conlleva la comunicación y transferencia de datos entre el entorno existente y las herramientas nuevas. Por lo tanto el sistema debe comunicar dichos estados al entorno subyacente para lo cual éste debe proporcionar los métodos apropiados.

- Además, los componentes de colaboración síncrona necesitan un canal de comunicación con altas prestaciones que permita la propagación de los cambios del estado en un ámbito distribuido. La propagación de estado impone fuertes requerimientos en el canal de comunicación con el objetivo de asegurar la comunicación uno a uno o la comunicación uno a muchos.

- Otro requerimiento esencial para la infraestructura síncrona es que sea simple, compacta, ligera y fácil de instalar. La idea principal es evitar la imposición de una sobrecarga en el entorno subyacente y por lo tanto simplificar el proceso de instalación al usuario final. Para esto, nosotros proponemos una arquitectura replicada para los componentes síncronos que se sustenten en el canal o bus de comunicación para propagar el estado entre las herramientas establecidas en la misma sesión.

- Otro de los aspectos interesantes a la hora de abordar la construcción de herramientas síncronas es proporcionar un modelo para la recogida de aquellos eventos relevantes para poder analizarlos posteriormente. Como veremos, para ello nos basaremos en el uso del canal de eventos, de tal forma que un servicio escuche los eventos de interés y los guarde en las estructuras apropiadas.

En el marco del proyecto COLAB se ha definido nuestra arquitectura colaborativa con soporte a la experimentación. A través de esta plataforma se construyó la aplicación COLAB para la experimentación en el ámbito de la física.

Con lo cual uno de los objetivos de nuestro trabajo de investigación es ofrecer una plataforma colaborativa genérica que puede ser aplicada a cualquier ámbito colaborativo incluyendo el aprendizaje por descubrimiento. Para ello dicha plataforma debe cumplir los siguientes requisitos a nivel arquitectónico:

- La arquitectura debe soportar la colaboración asíncrona, en el sentido de que debe posibilitar el guardado de la información que se maneja en cada sesión a la vez que la recuperación de dicha información en interacciones posteriores.

- La arquitectura debe proporcionar un canal de propagación de eventos para el intercambio de información entre las diferentes aplicaciones desarrolladas para

la colaboración síncrona. Más concretamente, se utilizará el modelo replicado en la programación de herramientas colaborativas de tal manera que mandarán eventos a través del canal para que el resto de colaboradores actualicen sus vistas con la nueva información.

- La información manejada debe ser tratada en sesiones independientes, de tal forma que lo que se realice en la sesión de un determinado grupo no sea visible a otros grupos.

- La arquitectura debe ser modular, tal que permita de una forma sencilla la introducción de nuevas herramientas o la inclusión de herramientas existentes. Además dicha arquitectura deberá ser construida de tal manera que permita separar la parte de interfaz de la parte de programación.

- Se debe de proveer una API clara y sencilla que permita a todos los miembros de la comunidad el manejo de esta arquitectura en el diseño e integración de herramientas sin necesidad de tener conocimiento de todas las tecnologías usadas en la elaboración de dicha plataforma.

- Esta plataforma también debe incluir soporte para almacenar toda la información necesaria del sistema así como los recursos que se vayan a usar.

2. Estructura.

Este trabajo se estructura en los siguientes capítulos. En el capítulo segundo analizaremos las tecnologías y conceptos estudiados previos a la construcción de los resultados de nuestra investigación. A través de este capítulo mostraremos las diferencias entre el *middleware síncro*no y el *middleware asíncrono*, así como veremos las arquitecturas distribuidas existentes para la construcción de aplicaciones distribuidas. En este capítulo analizaremos los principales conceptos de los entornos colaborativos, analizando algoritmos de llegada tarde así como la gestión del *Floor Control*. Dentro de los entornos colaborativos haremos un estudio de las principales tendencias de entornos de aprendizaje.

En el capítulo tercero se mostraran los estudios previos a lo que constituye nuestros resultados. En él veremos las integraciones de herramientas de colaboración síncrona llevadas tanto en el entorno JLE así como la plataforma ANTS. En el estudio de JLE también analizaremos algunos aspectos sobre las herramientas de seguimiento dentro de un entorno de educación a distancia. En este capítulo también haremos un estudio de los diseños llevados a cabo sobre la plataforma ANTS con el objetivo de soportar la colaboración a través de la experimentación.

En el capítulo cuarto mostraremos el diseño e implementación del sistema colaborativo ANTS y cómo se ha llevado a cabo su integración dentro de los entornos BSCL y FLE, proveyendo a dichos entornos la posibilidad de creación de herramientas de colaboración síncrona así como el soporte para llevar a cabo dicha colaboración. Todos estos estudios también nos sirvieron de base para la creación de una plataforma colaborativa que incluyera también las interfaces necesarias para ser el soporte de cualquier aplicación colaborativa. Más concretamente, mostraremos cómo esta

arquitectura ha sido aplicada en la construcción de un sistema de aprendizaje por descubrimiento denominado COLAB.

Finalmente en el capítulo quinto expondremos las conclusiones obtenidas de nuestro trabajo de investigación así como las vías futuras que se abren y quedan por estudiar de este trabajo.

Capítulo Segundo.

Antecedentes.

En este capítulo pretendemos mostrar todos los conceptos, fundamentos y diseños analizados que han sido estudiados para el diseño e implementación de los resultados que se obtienen de este trabajo de investigación.

Una de las bases a la hora del desarrollo de entornos cooperativos o colaborativos son los fundamentos acerca de los sistemas distribuidos. Estos constituyen el pilar de las comunicaciones llevadas a través de Internet. Más concretamente, nos centraremos en el estudio del middleware así como las posibilidades de comunicación existentes a la hora de construir una aplicación distribuida.

Este capítulo se estructura en dos bloques. En el primero de ellos se analizan las posibles alternativas que podemos encontrar en la comunicación así como en la construcción de aplicaciones distribuidas. Estudiaremos el middleware de comunicación síncrona y de comunicación asíncrona analizando las ventajas e inconvenientes de cada uno de ellos, así como estudiaremos las características de las arquitecturas de componentes. De este último tipo de arquitectura haremos un análisis entre el modelo de componentes CORBA, el estándar J2EE y la arquitectura .NET.

En el segundo bloque se muestran los conceptos y teorías fundamentales en los entornos cooperativos, mostrando las características más relevantes de las herramientas de colaboración asíncrona así como la de colaboración síncrona. En este bloque también se mostrarán diversos estudios y políticas para solucionar el problema de *late join* así como algunas políticas y mecanismos para llevar a cabo la gestión de los recursos de una sesión, *Floor Control*. Todas estas características sirven de base para el desarrollo de las aplicaciones colaborativas desarrolladas en nuestra tarea investigadora. Debido a que la aplicación de nuestro toma sus bases en el ámbito educacional, dentro de este bloque se muestra la diferencia entre lo que constituye el aprendizaje cooperativo y colaborativo. Más concretamente, abordamos las características del aprendizaje colaborativo así como dentro de este campo estudiaremos el aprendizaje por descubrimiento.

1. Sistemas distribuidos.

El crecimiento del uso de las redes de comunicación ha conllevado al incremento y la necesidad del uso de sistemas distribuidos para el diseño de sistema que puedan comunicarse entre diferentes ordenadores.

Según Bacon [Bac03] las propiedades fundamentales a tener en cuenta en un sistema distribuido son las siguientes:

- **Ejecución concurrente de sus componentes**. Los componentes de un sistema distribuido pueden estar ejecutándose en el mismo momento en el tiempo.
- **Modos de fallo independientes**. Los componentes del sistema distribuido así como la red de comunicación pueden fallar de una manera independiente.
- **Pérdida de tiempo global**. Cada proceso mantendrá diferencias en cuanto al tiempo, puesto que los relojes de cada sistema no tienen por qué mantener las mismas condiciones de tiempo.
- **Retardos en la comunicación**. El tiempo que se necesita en propagar el estado de un componente a otro.
- **Estados de inconsistencia**. La concurrencia, fallos y retardos en la comunicación pueden implicar inconsistencias en el estado mantenido por los diferentes componentes del sistema distribuido.

Todos los factores anteriores deben ser considerados a la hora de diseñar un sistema distribuido puesto que todos ellos pueden ofrecer la clave de éxito o fallo de un sistema.

Según Coulouris [Cou01] un sistema distribuido está compuesto por las siguientes capas:

Capa de aplicaciones, servicios. Esta capa se refiere al conjunto de software que se puede ejecutar desde el usuario final. Esta capa también abarca el concepto de servicios que son ofrecidos y requeridos entre procesos localizados en el mismo ordenador o en diferentes ordenadores.

Middleware. Esta capa se presenta como una capa software cuyo propósito es enmascarar la heterogeneidad y proveer un modelo de programación conveniente para las aplicaciones. El Middleware es presentado como un conjunto de procesos y objetos en diferentes ordenadores que interactúan para comunicarse y compartir recursos en sistemas distribuidos.

Plataforma. Es la capa de más bajo nivel, constituida por el hardware y software sobre el cual se fundamenta los sistemas distribuidos.

Gráficamente la relación entre estas capas es presentada en la Figura 1.

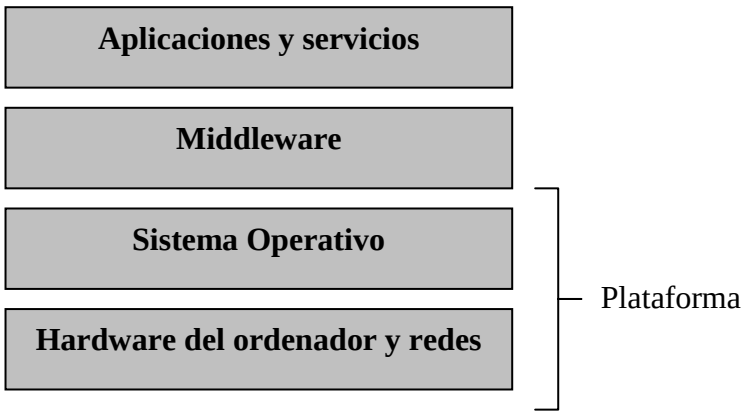

Figura 1. Capas de Software y Hardware en un sistema distribuido.

Más concretamente el estudio de nuestro trabajo está enfocado en las dos capas superiores, la capa de aplicaciones y servicios y la capa de middleware. Por lo tanto en este apartado enfocaremos nuestra atención en el middleware y los diferentes paradigmas que se ofrecen dentro de él, así como en el siguiente apartado nos centraremos en el diseño de las aplicaciones.

Una de las aproximaciones para la construcción de servicios a través de la red ha sido el uso de middleware sobre el que se facilita la construcción de nuevos sistemas. El middleware se presenta como un entorno uniforme para los componentes de un sistema distribuido. Más concretamente el middleware oculta, o abstrae cualquier heterogeneidad del sistema operativo y de los servicios de red subyacentes. Atendiendo al nivel OSI de 7 capas, el middleware abarca los conceptos de las capas de sesión, de presentación y aplicación.

En cuanto al tipo de comunicación que puede ofrecer el middleware existen principalmente los siguientes tipos:

> **Comunicación síncrona:** En este tipo de comunicación el invocador se bloquea esperando la respuesta del objeto invocado, que debe estar cargado y ejecutándose para llevar a cabo la invocación. Entre este tipo de comunicación podemos encontrar Remote Procedure Call (RPC), DCOM y CORBA.
>
> **Comunicación asíncrona:** En este tipo de comunicación el proceso que solicita una petición de servicio no se bloquea, permitiendo que continúe con la ejecución de otras tareas en paralelo hasta que reciba la respuesta a su petición. En este tipo de comunicación podemos encontrar tecnologías como son el paso de mensajes, Message Oriented Middleware (MOM) y JINI.

A continuación se detallarán las tecnologías existentes de cada uno de estos dos paradigmas. Después se presentarán las ventajas e inconvenientes de cada uno de ellos y se expondrá un modelo híbrido que contempla ambos tipos de comunicación.

1.1. Middleware de comunicación síncrona.

En la comunicación síncrona el proceso que invoca un método es bloqueado hasta que recibe la respuesta del proceso invocado, de una manera muy similar a la invocación

local de métodos. Debido al alto nivel de abstracción que presenta este tipo de comunicación este modelo resulta ser más fácil de usar que los modelos de paso de mensajes, los cuales veremos posteriormente.

Por estas razones muchas investigaciones son llevadas a cabo en el uso de **sistemas RPC** (Remote Procedure Call). Muchas de estas tecnologías llegaron a estar disponibles en los primeros años de la década 1980, como es el caso de Sun RPC. Otros como es el caso de ANSA 1989, permitieron un conjunto de estándares ejecutarse sobre una gran variedad de sistemas operativos así como servicios de comunicación (ISO). Más recientemente con la gran difusión de el paradigma orientado a objetos han aparecido nuevas tecnologías como puede ser Java Remote Method Invocation (RMI) que están convirtiéndose en tecnologías muy populares en el ámbito de los negocios y la industria.

Como se ha comentado anteriormente, podemos diferenciar dos paradigmas dentro de la comunicación síncrona. Uno de ellos es el paradigma RPC y el otro campo más reciente es el paradigma orientado a objetos, mediante el uso del modelo **Remote Method Invocation (RMI)**. A continuación detallaremos cada uno de estos paradigmas así como las tecnologías que se han desarrollado siguiendo el modelo que ambos presentan.

Un sistema RPC es un conjunto de protocolos de comunicación que se apoyan en el servicio de transporte, y poseen rutinas en lenguajes de alto nivel relacionada con el ensamblaje de los datos que son pasados a los protocolos. Además es necesario proveer un sistema de nombres para asociar los nombres de los procedimientos remotos a direcciones de red.

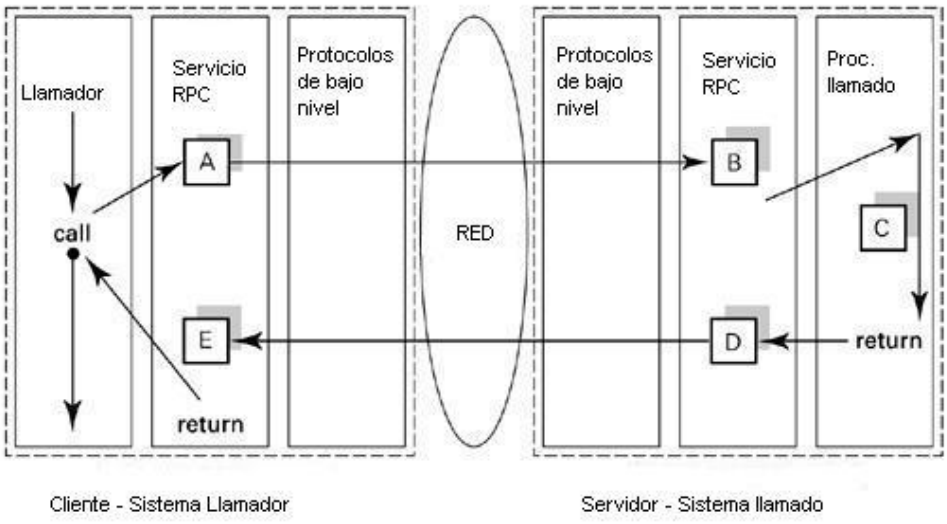

Figura 2. Diagrama de comunicación RPC.

En la Figura 2 se muestra los componentes, excepto la asociación de nombres con direcciones de red, que son invocados cuando se hace una llamada a procedimiento remoto (RPC). Normalmente un sistema RPC está basado en un modelo pregunta con acuse de recibo de la respuestas (request-reply acknowledge). Como se puede apreciar en la figura esta tecnología se basa en un proceso cliente que invoca una llamada en el proceso servidor.

En el punto A se realizan las tareas:

- Empaquetar los datos pasados en una estructura que permita la transferencia a través de la red como un mensaje o un paquete.
- Generar un identificador RPC para esta llamada.
- Establecer un timer.

Una vez realizadas estas tareas los datos son pasados a los niveles de transporte de red. Una vez recibida esta información, en el punto B se realizan las siguientes operaciones:

- Los argumentos son desempaquetados y convertidos a una forma adecuada para hacer la llamada del procedimiento local.
- El identificador RPC es anotado.

Una vez realizada la llamada del procedimiento requerido (punto C) la respuesta es devuelta en el punto D en el cual se empaquetan los datos para ser transferidos por la red y se establece un timer.

Una vez que llega la respuesta en el punto E se desempaquetan los valores retornados, se deshabilita el timer establecido en A y se devuelve el acuse-recibo de la respuesta recibida. Una vez recibido este acuse-recibo en el sistema servidor se deshabilita el timer establecido en el punto D.

Como se puede observar de la anterior explicación, ambos cliente y servidor establecen unos tiempos de espera en el servicio RPC. Dichos tiempos de espera se utilizarán por el sistema RPC para la acción a tomar en el caso de que surgiera una caída del sistema, pudiendo intentar retransmitir durante un número de veces, tras el cual si no se recibe respuesta el servicio RPC considera que la llamada remota no ha podido llevarse a cabo. Como se puede observar, es el propio sistema RPC el encargado de gestionar este número de peticiones, sin que esto influya en el nivel de aplicación.

En cuanto a las semánticas que pueden llevarse a cabo en la retransmisión del resultado, Coulouris [Cou01] establece principalmente las siguientes:

- **Maybe invocation semantics**. En este tipo de semántica el invocador no puede decir si el método remoto ha sido ejecutado una o ninguna vez. Este tipo de semántica se basa en no usar ninguna medida de tolerancia a fallos.
- **At least once invocation semantics**. En este tipo de semántica el invocador recibe o bien el resultado, en el caso de que el invocador sepa que el método fue ejecutado al menos una vez, o bien una excepción indicando que no se ha recibido ningún resultado.
- **At most once innovation semantics**. En este tipo de semántica el invocador recibe o bien el resultado, en el caso de que el invocador sepa que el método fue ejecutado al menos una vez, o bien una excepción indicando que no se ha recibido ningún resultado en el caso de que el método haya sido ejecutado uno o ninguna vez.

Más concretamente, Java RMI y CORBA utilizan la semántica “at most once invocation” mientras que Sun RPC utiliza la semántica “at least once invocation”.

Una de las grandes ventajas de los sistemas RPC es la transparencia, permitiendo que las aplicaciones realicen las llamadas al procedimiento remoto al igual que invoca las llamadas de un procedimiento local. Como se ha explicado anteriormente todo lo referente al empaquetamiento de mensajes y el tratamiento de errores es llevado a cabo por el servicio RPC sin que la aplicación tenga que verse implicada en la gestión de estas tareas.

El uso de RPC conlleva una latencia mayor debido a la retransmisión en red a la vez que un incremento en el número de fallos si se compara con las llamadas a procedimientos locales. Por esta razón el programador debe diseñar su programa intentando que el número de llamadas a procedimientos remotos sea el mínimo posible.

A la hora controlar las interacciones entre cada una de las partes de la comunicación RPC necesita proveer una serie de elementos. Además del hecho de que ambas partes se ejecutan en diferentes ordenadores implica que no se pueda realizar un acceso directo a las variables. Por lo tanto se debe proveer:

- **Interfaces de servicios.** En el modelo cliente-servidor cada servidor provee un conjunto de procedimientos que están disponibles para el uso de los clientes. Es decir, una interfaz de servicios es una especificación de los métodos ofrecidos por el servidor. En otros modelos también recibe el nombre de *Skeleton*.
- **Stub.** El rol del stub procedure es similar al de un proxy. Más concretamente se debe proveer uno por cada uno de los procedimientos en el interfaz de servicios. En la comunicación RPC se establecen dos stub, uno para el cliente y otro para el servidor. El del cliente es el encargado de empaquetar los datos y de introducir el identificador del procedimiento. El del servidor se encarga de desempaquetar los datos recibidos y de empaquetar la respuesta.
- **Dispatcher.** El dispatcher se ejecuta en el servidor y se encarga de seleccionar el stub servidor que debe de realizar la petición de la invocación remota según el identificador recibido desde el cliente.

En general tanto el stub cliente, el stub servidor y el dispatcher son generados por un compilador de interfaz a partir de la interfaz de servicios.

En el paradigma orientado a objetos la base la ofrece el uso de objetos para realizar la comunicación, a través del uso de objetos remotos se hace la invocación a métodos remotos (RMI). Aunque como se puede observar en la Figura 3 el esquema de comunicación sigue usando un paradigma muy similar al del RPC.

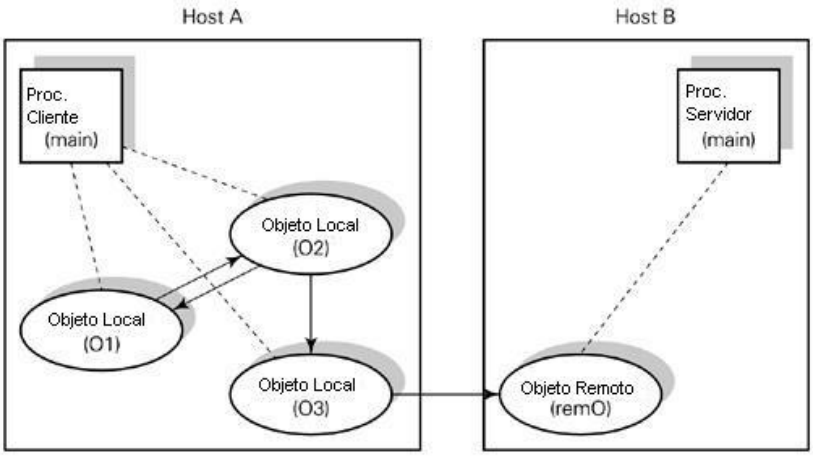

Figura 3. Modelo de comunicación de un paradigma orientado a objetos.

Al igual que RPC un sistema RMI informa de la posibilidad o la imposibilidad de realizar las llamadas, pudiendo implementar cualquiera de las semánticas expuestas anteriormente.

En cuanto a los elementos que se necesitan en este tipo de comunicación cabe destacar los siguientes:

- **Referencias a objetos remotos.** Una referencia a un objeto remoto es un identificador que puede ser usado a través del sistema distribuido para referirse a un único objeto remoto.
- **Interfaces remotas.** La clase de un objeto remoto implementa una interfaz remota. Los objetos en otros procesos sólo podrán invocar los métodos que aparecen en la interfaz remota del objeto servidor.

A continuación se explicarán las diferentes tecnologías que pueden encontrarse en ambos tipos de paradigmas.

OMG CORBA

El Object Management Group (OMG) fue creado para promover el paradigma orientado a objetos en sistemas distribuidos. De esta manera OMG CORBA (Common Object Request Broker Architecture) comenzó a definirse en el año 1989. Desde entonces, CORBA ha ganado la aceptación tanto industrial como comercial.

CORBA es un marco de aplicación que provee interoperabilidad entre objetos, el uso de diferentes lenguajes de programación y la ejecución en diferentes máquinas en sistemas distribuidos heterogéneos.

Arquitectura CORBA

En la figura 4 se muestra la arquitectura que posee un sistema CORBA. **ORB** (Object Request Broker) es el middleware que establece las relaciones cliente-servidor entre los objetos. Usando un ORB un cliente puede invocar de una manera transparente a un método del objeto servidor, que puede estar en la misma máquina o a través de la red. El

cliente no tiene por que ser consciente de la localización del objeto, su lenguaje de programación, su sistema operativo, o cualquier otro aspecto que no este relacionado con la interfaz de un objeto.

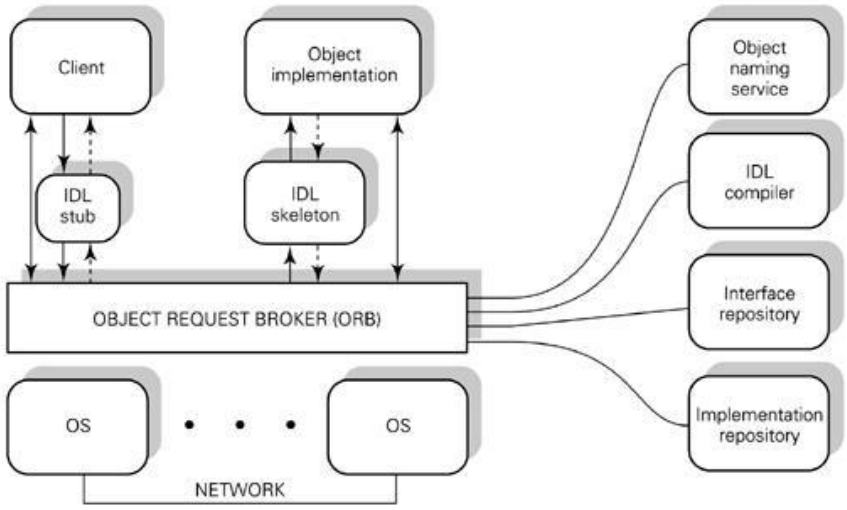

Figura 4. Arquitectura CORBA.

Los interfaces de los objetos que contienen la especificación de las operaciones y tipos que el objeto soporta deben ser especificados mediante el uso de **Interface Definition Language (IDL)**. La idea principal en la que se basa CORBA es que el IDL sea independiente del lenguaje usado, por lo que los interfaces serán definidos separadamente de la implementación del objeto. De esta manera, los objetos pueden ser definidos en diferentes lenguajes de programación y ser capaces de comunicarse gracias al uso de estos IDL.

De entre los elementos que podemos encontrar en un sistema CORBA destacaremos los siguientes:

- **Repositorio de interfaces (Interface Repository).** El repositorio de interfaces contiene las especificaciones de los objetos que se necesitan.
- **Stubs y skeleton.** Ambos son generados por el compilador IDL y son usados por el cliente y el servidor para comunicarse.

Protocolos Inter-ORB.

Para que el software que está distribuido en diferentes plataformas pueda interoperar, es necesario tener en cuenta la interoperabilidad entre las diferentes plataformas.

La interoperabilidad de la arquitectura ORB está basada en el **General Inter-ORB Protocol (GIOP)** el cual especifica la sintaxis de transferencia y un conjunto de formatos de mensajes para la interoperación ORB sobre cualquier transporte orientada a la conexión. El **Internet Inter-ORB Protocol (IIOP)** especifica como se construye GIOP sobre TCP/IP.

Las referencias de objetos pueden ser entendibles a través de plataformas interoperantes. CORBA especifica un formato estándar de referencia llamado **Interoperable Object**

Referente (IOR). Un IOR contiene la información necesaria para localizar y comunicarse con uno objeto sobre uno o más protocolos.

Además como podremos ver en el apartado 1.2, CORBA ha definido un servicio asíncrono de comunicación.

JAVA RMI

Java RMI es una extensión del modelo de objetos Java para proveer soporte para la comunicación distribuida mediante el uso de Remote Method Invocation. Es importante no confundir RMI y Java RMI, siendo el primero de ellos un modelo de comunicación y el otro la implementación que se ha realizado en Java de este modelo.

En Java RMI un objeto que realiza una llamada a un objeto remoto es consciente de que está trabajando con un objeto remoto ya que tiene que manejar la excepción *RemoteException;* y el que implementa el objeto remoto es consciente de esto puesto que debe implementar el interfaz *Remote*. Aunque este modelo de objetos es integrado en Java la semántica del paso de parámetros de uno a otro difiere puesto que el invocador y el fuente son remotos uno del otro.

El uso de Java RMI resulta sencillo puesto que se basa en el uso de un solo lenguaje, que es Java, al contrario que CORBA.

Otra de las características que destacan en este tipo de lenguaje es el uso de la serialización de objetos, la cual es usada para empaquetar los argumentos y resultados. Cualquier objeto serializable, es decir aquel que implementa la interfaz *Serializable*, pude ser pasado como argumento o resultado de Java RMI.

Otra de las características que hacen interesante el uso de Java RMI es la descarga de clases. Java está diseñado para permitir que las clases sean descargadas desde una maquina virtual Java hacia otra. Esto es particularmente interesante en sistemas distribuidos de objetos que se comunican mediante el uso de invocación remota.

A la hora de invocar un método remoto los objetos no remotos son pasados por valor mientras que los objetos remotos son pasados por referencia tanto como argumentos como resultado de la invocación RMI. Si el recipiente no posee la clase del objeto pasado por valor este es descargado automáticamente. De igual manera, si el recipiente de la referencia de un objeto remoto no posee la clase para este proxy (stub) su código es descargado automáticamente. Esto presenta las siguientes ventajas:

1. No es necesario mantener por cada usuario el mismo conjunto de clases en su entorno de trabajo.
2. Ambos, cliente y servidor pueden hacer transparente el uso de instancias de nuevas clases siempre que sean añadidas.

Como se puede apreciar en la figura 5 la comunicación entre el objeto remoto y el objeto cliente se hace, al igual que pasaba con RPC, mediante el uso de stub. El elemento RMIRegistry es el encargado de asociar las direcciones de red a las referencias de los objetos remotos. Por lo tanto para que un objeto remoto esté disponible debe registrarse en el RMIRegistry usando un nombre como identificador.

El RMIRegistry puede encontrarse o bien en la misma máquina dónde está el objeto remoto o bien en otra máquina.

Los SecurityManagers son usados en Java RMI para controlar lo que pueden hacer los objetos remotos y sus clientes.

Para que un cliente pueda acceder a un objeto remoto debe conocer la localización del RMI Registry y el nombre del objeto remoto, véase la referencia del objeto O1 en la figura 5. Normalmente para llevar esto a cabo hace uso de la clase *Naming*.

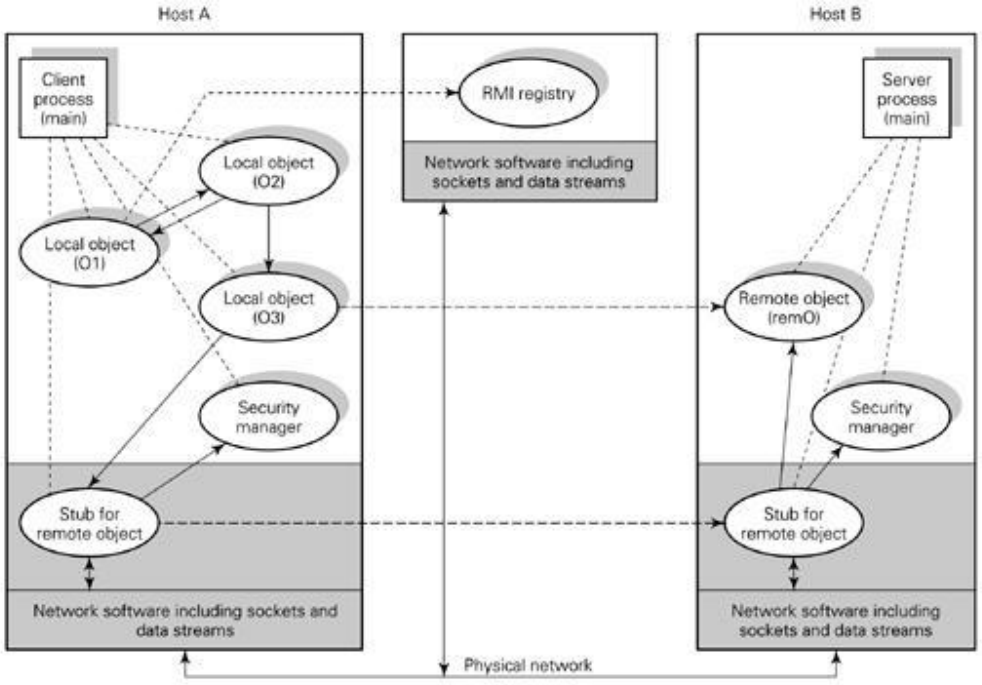

Figura 5. Modelo de comunicación en Java RMI.

En un principio Java RMI apareció como una tecnología para simplificar el diseño de sistemas distribuidos en las aplicaciones Java. Posteriormente los modelos Java RMI y CORBA se han ido acercando cada vez más hasta conseguir un alto grado de interoperabilidad. En esta línea, Java RMI puede utilizar IIOP como protocolo de transporte para comunicarse con objetos CORBA. La idea es simplificar el desarrollo de aplicaciones distribuidas Java sin perder la robustez e interoperabilidad del modelo CORBA.

Más recientemente, esta tecnología constituye una de las bases sobre las que se fundamentan las especificaciones J2EE (Java 2 Enterprise Edition) que veremos en apartados posteriores.

COM y DCOM

El modelo de componentes de Microsoft (COM) fue desarrollado para hacer más fácil a los desarrolladores de software la creación de nuevas versiones del sistema operativo Windows. Para permitir que el sistema operativo Windows funcionara en un sistema distribuido, COM fue integrado con OSF's DCE, un protocolo RPC.

El modelo conceptual es básicamente el mismo que el modelo CORBA, cuenta con su propia capa de transporte basada en DCE RPC y establece su propio lenguaje declarativo de definición de interfaces llamado MIDL. En general, DCOM constituye una extensión al modelo de componentes COM de Microsoft con el objetivo de soportar el desarrollo de aplicaciones distribuidas.

Posteriormente veremos como éstas junto con otras tecnologías han servido a Microsoft para la creación de la plataforma .NET que agrupa tanto sistemas de comunicación síncrona como comunicación asíncrona.

SOAP

SOAP (Simple Object Access Protocol) [W3C00] es un protocolo de transporte para el intercambio de información en un entorno distribuido descentralizado mediante el uso de XML. En un sentido general puede abarcar la comunicación síncrona (uso de HTTP o RPC) o bien la comunicación asíncrona (multicast, comunicación en un sentido). Lo que lo hace todavía más extensible es el número de protocolos sobre los que puede ser transportado, como pueden ser HTTP, SMTP, JMS, etc.

SOAP está orientando a aplicaciones de interoperabilidad sobre Internet en el ámbito de los denominados servicios Web donde se complementa con otros protocolos como UDDI (Universal Description, Discovery and Integration) y WSDL (Web Service Definition Language). Más concretamente, SOAP es una parte integral de la arquitectura Microsoft .NET, que podremos ver en apartados posteriores.

XML-RPC

XML-RPC es una tecnología basada en RPC en la que los flujos de comunicación están codificados mediante XML. Es una manera simple y portable para hacer llamadas a procedimientos remotos mediante el uso de HTTP. Además abarca una amplia gama de lenguajes de programación en los cuales puede ser usada, como pueden ser Perl, Java, Phyton, C, C++, PHP, Microsoft .NET, Tcl, Real Basic, Zope, Delphi, Frontier, Lisp, etc, lo cual la hace idónea para la comunicación entre sistema heterogéneos.

XMLRPC se está utilizando ampliamente como una alternativa menos pesada que SOAP para comunicaciones entre sistemas heterogéneos. De hecho, la gran cantidad de implementaciones en diferentes lenguajes de programación así como su sencillez está multiplicando los usos de XMLRPC como protocolo simple de transporte. Sin embargo SOAP es más aconsejable cuando se intentan intercambiar documentos más complejos, con la característica adicional de que permite comunicación síncrona y asíncrona y puede ser transportado con un número mucho más amplio de protocolos.

En apartados posteriores veremos como el uso de la tecnología XML-RPC nos permite la integración de sistemas heterogéneos, permitiendo una colaboración común independientemente del lenguaje en que éstos estén implementados.

1.2. Middleware de comunicación asíncrona.

Como se ha comentado anteriormente este tipo de comunicación no es bloqueante, permitiendo que el que solicita una petición pueda continuar en paralelo con el trabajo a

realizar. Fundamentalmente se basa en el servicio de mensajes, por lo cual para que sea persistente y fiable debe permitir que el receptor no tenga por que estar ejecutándose cuando el mensaje sea enviado. Una vez que el mensaje sea enviado el sistema aceptará la responsabilidad del mensaje y lo almacenará si es necesario hasta que el receptor esté listo.

Uno de los paradigmas que engloba este tipo de comunicación es el **Message Oriented Middleware (MOM).** Más concretamente, MOM es una infraestructura cliente/servidor que incrementa la interoperabilidad, portabilidad y flexibilidad de una aplicación permitiendo que la aplicación sea distribuida sobre plataformas heterogéneas. Básicamente consiste en el transporte de mensajes desde una fuente hasta un destino, siguiendo un esquema asíncrono de comunicación. Relacionado con este paradigma ha surgido el paradigma de publish/suscribe y el de notificación de eventos.

Otra de las grandes ventajas de los sistemas MOM es que unidos a protocolos de transporte multicast, ofrecen una escalabilidad de gran importancia, ya que con un mensaje pueden llegar a varios destinos.

Entre las ventajas que proporciona un sistema MOM cabe destacar las siguientes:

- **Independencia de los componentes** puesto que el emisor y el receptor no tienen que estar activos al mismo tiempo, ya que algunos sistemas MOM encolan los mensajes de los receptores que no están disponibles.
- **Ocultación de la latencia.** El interfaz de un usuario no queda bloqueado a la espera de la respuesta del servidor, sino que puede continuar llevando a cabo tareas de la aplicación. De hecho cuando recibe el evento incluso puede manejarlo en un nuevo thread o hilo, sin necesidad de parar la actividad que está llevando a cabo.
- **Independencia de la localización de los componentes.** El emisor y el receptor pueden ser migrados de un ordenador a otro en tiempo de ejecución, ya que emisores y receptores están desacoplados usando colas de mensajes.
- **Escalabilidad.** Los sistemas MOM de publicación y suscripción, que veremos en este apartado, permiten una alta escalabilidad respecto al número de receptores.
- **Sistemas conducidos por eventos.** Ya que los eventos son una forma especial de mensajes, los sistemas MOM soportan la construcción de aplicaciones distribuidas orientadas a eventos.

Como hemos comentado anteriormente otro de los mecanismos que surgen dentro de la comunicación asíncrona, es el llamado **publish/subscribe messaging middleware**. El funcionamiento de este middleware viene ilustrado en la figura 6. Este paradigma también es denominado como el sistema **productor/consumidor**.

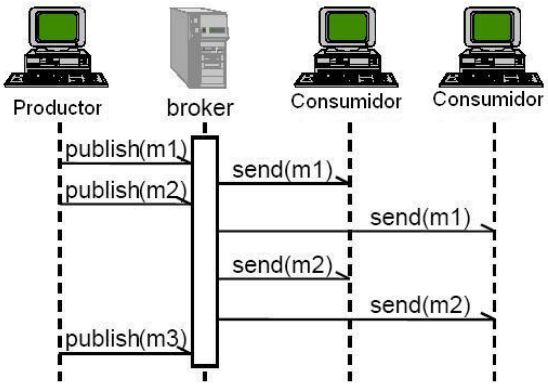

Figura 6. Middleware Orientado a Mensajes. Modelo Productor/Consumidor.

Como se puede apreciar en la figura 6, se denomina **Productor** a la entidad responsable de publicar un mensaje o tópico, y se denomina **Consumidor** a las entidades interesadas en la recepción de un tipo de mensaje. En este paradigma se trabaja con la idea de un cierto tópico al cual se suscriben los consumidores. El productor produce información y la publica bajo cierto tópico. El middleware asegura que todos los suscriptores de ese tópico reciben la información de dicho tópico. Estos sistemas llegan a ser mucho más eficientes que los sistemas RPC cuando las comunicaciones son del orden "m:m" o "1:m".

En un paradigma de **notificación de eventos**, la idea detrás del uso de eventos es que un objeto puede reaccionar al cambio de otro objeto. Los sistemas de notificaciones de eventos se basan en el paradigma publish/subscribe que se ha descrito anteriormente. Más concretamente, en el paradigma de notificación de eventos se denomina **evento** a cualquier evento que ocurre en un objeto de interés como resultado de la terminación de la ejecución de un método, y se define una **notificación** al objeto que contiene información acerca de un evento. De igual modo que en el paradigma publish/subscribe el middleware, también llamado **servidor de notificaciones**, será el encargado de mantener una base de datos con los eventos publicados y los suscriptores interesados en ciertos eventos. Otro nombre con el que se suele llamar al middleware encargado de repartir los eventos es **canal de eventos**.

Otro de los paradigmas que han surgido dentro de esta área es la **cola de mensajes**, en el cual el funcionamiento es muy similar a un buzón de correo. El productor del mensaje pone el mensaje en uno o varios buzones, y los receptores borran el mensaje del buzón cuando consideran que es hora de procesar sus mensajes.

A continuación mostraremos algunas de las tecnologías más relevantes dentro de este middleware asíncrono.

TIB/Rendezvous

Básicamente Rendezvous está basado en el paradigma publish-subscribe. Dentro de una red local cada participante ejecuta en su equipo un demonio "rendezvous" para manejar las suscripciones y publicaciones. El demonio "rendezvous" es el encargado de mantener el registro de los tópicos a los que se está suscrito, escuchar los mensajes multicast que se reciben y repartir aquellos en los que se tiene interés en escuchar. El demonio también es el primer punto de contacto a la hora de publicar mensajes.

Para implementaciones fuera del área local, cada dominio ejecuta un demonio router que se comunica con otros demonios routers usando estándares de comunicación de direccionamiento y enrutamiento.

IBM MQSeries

MQSeries [Gil96] fue desarrollado para servir CICS (Customer Information Control System), un sistema de procesamiento de transacciones ejecutándose en IBM mainframes. Entre los componentes que forman está tecnología cabe destacar los siguientes:

- **Gestor de cola**: Es un software que reside en un router de mensajes. Es responsable de una serie de colas de entrada y otras de salida. Mantiene tablas de enrutamiento las cuales usa para transferir cada mensaje desde una cola de entrada a la cola apropiada de salida, según la tabla de enrutamiento.
- **Canal de mensaje**: Es una comunicación fiable unidireccional entre el gestor de cola que envía y recibe. En un sistema basado en Internet el canal de mensajes es implementado como una conexión TCP.
- **Agente del canal de mensaje (MCA)**: Es un componente de un gestor de cola. Es responsable de chequear las colas de salida y mandar los mensajes sobre el servicio de transporte subyacente. Un MCA receptor es responsable de escuchar los paquetes de entrada y almacenar el mensaje en su cola apropiada.

Debido a que no se usan servicios de nombre ni directorio, en su lugar se usa tablas de enrutamiento. Por lo tanto con este sistema surge el problema de propagar el conocimiento de nombres globales a todos los participantes y el manejo de nombres y cambios de localización. Este problema puede ser resuelto mediante el uso de alias por parte de los clientes los cuales son traducidos a nombres globales por el software MQSeries.

Servicio de Notificación CORBA

Debido a que ciertas aplicaciones requieren el uso de comunicación asíncrona para obtener un mejor rendimiento en las notificaciones y ocurrencias de eventos, en 1993 fue introducido el servicio de eventos (CosEvent) en CORBA y en 1998 fue introducido el servicio de notificación (CosNotification).

Ambos servicios van dirigidos a un modelo desacoplado asíncrono de publicación / suscripción. CosNotification se puede considerar una evolución del servicio de eventos para constituir un verdadero sistema de notificaciones.

La especificación COSEvent recoge las principales entidades del modelo publish/subscribe. Así, el bus central o Conexión es el *EventChannel*, el Productor es llamado *Supplier*, y el Consumidor *Consumer*. Además, esta especificación recoge dos posibles modelos de interacción entre entidades y el bus de eventos: el modelo *push* (envió) y el modelo *pull* (solicitud). De acuerdo a ambos modelos, la especificación considera clases asociadas llamadas *PushConsumer*, *PullConsumer*, *PushSupplier* y *PullSupplier*.

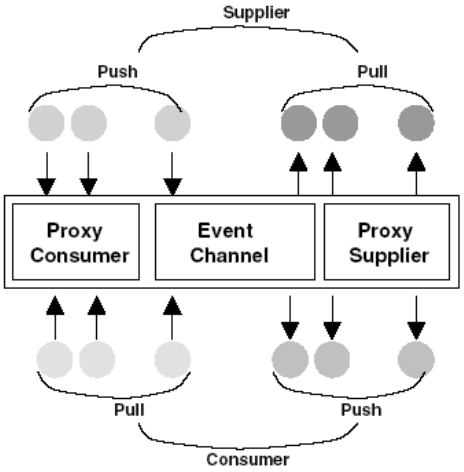

Figura 7. Servicio de notificaciones CORBA

El soporte de un modelo de envío automático de datos a los solicitantes o suscriptores (*push*) y de un modelo de solicitud de datos desde los solicitantes (*pull*) es conceptualmente muy rico y da cabida a diversos escenarios. Sin embargo, el servicio de eventos carece de funcionalidades como filtrado de eventos y envío de eventos estructurados que limitan su aplicabilidad a escenarios más complejos.

La especificación COSNotification (ver Figura 7) extiende la especificación COSEvent. Así, entre las mayores aportaciones encontramos la inclusión de un lenguaje restringido de filtrado de eventos, el soporte a mensajes estructurados, y un modelo de control de la calidad del servicio (QoS) para el bus de eventos.

Hay una serie de fabricantes comerciales que han implementado estos servicios en diferentes lenguajes como IONA OrbixTalk Event Service (C++), Orbacus Event Service (Java y C++), TAO Event Service (C++), DSTC Notification Service (Java) y Orbacus Notification Service (C++).

Los mayores problemas encontrados en los servicios de mensajería CORBA se refieren a la alta complejidad de las APIs y al bajo rendimiento de dichos servicios. Según estudios de García [Gar02] las implementaciones en C++ son considerablemente más rápidas y robustas que sus homólogas en Java, su velocidad en situaciones de alta carga puede no ser suficiente para problemas de tiempo real. Además, por muy optimizados que se diseñen estos servicios, su rendimiento viene limitado por el ORB o gestor de objetos subyacente. Al basarse en una tecnología de objetos distribuidos, es el ORB el que limita en gran parte el rendimiento y velocidad de estos sistemas.

Java Messaging Service (JMS)

JMS especifica una API para aplicaciones que usan middleware orientado a mensajes. JMS soporta dos tipos de comunicación asíncronas, la cola de mensajes y el paradigma de publicación/suscripción. En ambos modelos, un objeto es creado como el destino a través del cual los mensajes son enviados/recibidos o bien publicados/suscritos por los clientes.

En JMS una conexión se representa mediante el objeto *TopicConnection* y su acceso se regula gracias a la clase *TopicConnectionFactory*. El Productor se representa con la

clase *TopicPublisher* y dispone de métodos para enviar mensajes al canal de eventos. El Consumidor se representa con la clase *TopicSubscriber* y dispone de métodos de aviso (*callbacks*) que serán disparados en respuesta a eventos. La suscripción es denotada por la clase *Subscription* que soporta un lenguaje de suscripción basado en SQL permitiendo filtrado flexible de eventos. Por último, el Mensaje se representa por la clase *Message*, y define una jerarquía de mensajes que permiten enviar eventos en diferentes formatos (*TextMessage*, *ObjectMessage*, *MapMessage*, *StreamMessage*, *BytesMessage*).

El modelo JMS es similar al modelo OMG en el sentido que está basado en canales de eventos que son denominados *EventChannels* en OMG y *Topics* en JMS. JMS soporta suscripciones fiables como OMG COSNotification, por lo que también requiere un almacén persistente de mensajes para asegurar el envío correcto de los mismos. Por último, JMS no estandariza la gestión de la calidad del servicio (QoS) en la transmisión de eventos en el canal.

Como conclusión, podemos decir que JMS provee un API estandarizada a sistemas de notificaciones en el lenguaje Java que está ampliamente soportada entre los fabricantes existentes. La desventaja frente a OMG es que JMS está específicamente pensada para el lenguaje Java, a diferencia del modelo OMG que define IDLs genéricas multi-lenguaje.

Entre los sistemas actuales, destacamos Softwired Ibus por su modelo de calidad de servicio incluyendo canales multicast, comprimidos y encriptados. SwiftMQ también destaca por ser una solución abierta que facilita canales TCP y SSL y que permite federación de servicios. Otros vendedores como Fiorano y Progress también suministran sistemas de notificación basados en JMS de variadas prestaciones.

Jini

Jini es una especificación de eventos distribuidos que permite a un suscriptor en un maquina virtual Java suscribirse y recibir notificaciones de eventos sobre un objeto de interés de otra maquina virtual Java, normalmente en otro ordenador.

Los principales componentes en la especificación de Jini son:

- **Generador de eventos.** Es un objeto que permite a otros objetos suscribirse a sus eventos y generar notificaciones.
- **Listener de eventos remotos.** Es un objeto que recibe notificaciones.
- **Evento remoto.** Un objeto pasado por valor a un Remote Event Listener, o lo que es lo mismo una notificación.
- **Agentes.** Son agentes interpuestos entre el objeto de interés y el suscriptor.

La suscripción a eventos se hace informando al "Generador de eventos" del tipo de evento al que se quiere suscribir y especificando el "Listener" que será el receptor de las notificaciones. Java RMI es usado para la suscripción de eventos y para el envío de notificaciones desde el generador de eventos hacia el suscriptor. Como podemos observar el modelo de Jini presenta una comunicación asíncrona mediante el uso de notificaciones usando un modelo síncrono de invocación.

Una de las partes importantes en Jini son los JavaSpaces los cuales proveen facilidades para el almacenamiento y recuperación de colecciones de objetos persistentes en espacios compartidos, para transacciones o servicio de eventos. De esta manera los servicios Jini y cualquier otro servicio requerido por las aplicaciones usando Jini, pueden ser construidos usando JavaSpaces con el objetivo de almacenar y recibir objetos, manipularlos correctamente mediante el uso de transacciones y notificar a sus clientes de cualquier cambio en los cuales hayan registrado su interés.

Jini no está diseñado para grandes implementaciones distribuidas. El uso del protocolo multicast así como el uso JavaSpaces hacen que sea preferible a servicios construidos sobre implementaciones centralizadas. Sin embargo para aplicaciones ampliamente distribuidas es más adecuado el uso de sistemas como JMS, que hemos visto en apartados anteriores.

Jabber

Jabber es un conjunto de protocolos de streaming XML y de tecnologías que permiten que dos entidades en Internet puedan intercambiar mensajes, obtener características de presencia, y otra información estructurada realmente útil en colaboraciones online. Más concretamente Jabber se basa en el protocolo XMPP (Extended Message Presence Protocol) formalizado por la IETF [Jab]. Aunque la primera aplicación Jabber es un servicio de mensajería instantánea, esta tecnología puede ser la base de la construcción de cualquier aplicación colaborativa online, como puede ser un chat.

Para el modelo de envío de eventos se basa en un modelo **publish/suscribe** [JEP]**,** dónde se define el término de *Node* que será una localización virtual dónde la información pueda ser publicada y desde la cual se puedan recibir eventos. Se define *Publisher* y *Suscriber* cómo aquellas entidades que participan en la publicación o suscripción de eventos respectivamente. Con *Item* se denota a un fragmento XML que es publicado en un nodo. Debido a que su intercambio de eventos es hecho en xml, siguiendo unas etiquetas predefinidas, mediante este formato se especifica por ejemplo el tipo de operación que se esta realizando (“set”, “result” o “error”) , el publicador, el tipo de operación, etc.

```
<iq type="set"
    from="pgm@jabber.org"
    to="pubsub.jabber.org"
    id="publish1">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <publish node="generic/pgm-mp3-player">
      <item id="current">
        <tune xmlns="http://jabber.org/protocol/tune">
          <artist>Ralph Vaughan Williams</artist>
          <title>Concerto in F for Bass Tuba</title>
          <source>Golden Brass: The Collector's Edition</source>
        </tune>
      </item>
    </publish>
  </pubsub>
</iq>
```

Figura 8. Estructura para la publicación de un evento en un nodo

Más concretamente, para facilitar el manejo de estas cadenas se han creado un conjunto de librerías en diferentes lenguajes (Python, C, Java). JSO (Jabber Streaming Object) es una librería desarrollada en Java que permite manejar la información XMPP.

Otra funcionalidad que hace Jabber interesante es que permite la autenticación y guardado de información de los usuarios en Kerberos y servidores LDAP (Véase Anexo I). Además el hecho de que el protocolo sea Open source, habiendo clientes, servidores y componentes siguiendo este tipo de licencia, la hacen deseable a la hora de construir herramientas de comunicación, o cambiar cualquier información deseable en el servidores como clientes.

Debido a que el intercambio de mensajes se hace usando el estándar XML, podemos tener clientes desarrollados en diferentes lenguajes de programación, permitiendo la heterogeneidad tanto en clientes como en servidores. Los desarrollos más estables se están realizando en C++, aunque hay algunas versiones en Java éstas no están tan extendidas.

Sin embargo una de las grandes desventajas que presenta este protocolo frente a JMS o Elvin es que no garantiza un servicio de entrega fiable, lo cual la hace inadecuada para desarrollos en los cuales es necesario garantizar la entrega de la información para asegurar que la información de cada uno de los usuarios sea la misma.

Sin embargo, bajo la fundación Jabber (Jabber Foundation) se están definiendo e implementando nuevas extensiones de Jabber para favorecer la interoperabilidad con otros sistemas así como el enriquecimiento de los servicios ya ofrecidos.

DSTC Elvin

Elvin provee un servidor de notificaciones de eventos escalable y dinámico. Esta construido sobre el modelo cliente/servidor en el cual un servidor Elvin es el encargado de manejar las conexiones de los clientes, así como la transferencia de mensajes entre publicadores y suscriptores.

Elvin no incluye ningún mecanismo de persistencia, como hace JMS obteniendo así un modelo más eficaz y con mayor producción de eventos. Sin embargo, a diferencia que Jabber, si permite la entrega fiable de mensajes así como la gestión de QoS en la retransmisión de eventos del canal.

Su API es bastante sencilla y ofrece soporte a diferentes lenguajes de programación como Java, Python, Perl, C, C++, Palm y Emacs Lisp. Su Conexión se representa con la clase *Connection*, el Productor es la clase *Producer*, el Consumidor es la clase *Consumer*, la Suscripción es la clase *Subscription*, y el mensaje es la clase *Notification*. Dispone de un potente lenguaje de filtrado basado en los operadores lógicos del lenguaje C y que soporta datos textuales, numéricos y fechas.

Su modelo de conexión difiere del modelo JMS y OMG en el sentido de que no dispone de canales ni tópicos, sino un único bus de mensajes donde toda suscripción se realiza en base al contenido de los mensajes. Con lo cual el enrutamiento de los mensajes se hará comprobando la suscripción de los interesados y el contenido del mensaje. La notificación es un objeto formado por una lista de pares clave-valor. También incluye

un esquema de federación, posibilidad de establecer clustering de servidores Elvin, y soporte a encriptación del canal.

Todas estas características permiten obtener un alto rendimiento en la gestión de eventos. De ahí que este haya sido una de las soluciones sobre la que se ha trabajado a la hora de ejecutar los desarrollos que se exponen como resultado de nuestro trabajo de investigación.

1.3. Sistema híbrido.

En el desarrollo de aplicaciones distribuidas es aconsejable el uso del paradigma orientado a mensajes, puesto que ofrece una gran escalabilidad con respecto al incremento en el número de usuarios que pueden participar en el sistema distribuido, especialmente en tareas colaborativas. Esta necesidad viene recogida por la reconocida frase "Every DAD (Distributed Application Developer) needs a MOM (Message-Oriented Middleware)" [Mar99].

En un principio los diseñadores de arquitecturas software no se beneficiaban de las ventajas que puede ofrecer un sistema híbrido, es decir un sistema en que utiliza la comunicación síncrona y la comunicación asíncrona, obteniéndose en dicho sistema las ventajas que ofrecen ambos tipos de comunicaciones y disminuyendo las desventajas de las mismas.

Así mientras el modelo síncrono es especialmente útil para arquitecturas en las cuales se presta servicio a fuentes de datos, el modelo asíncrono es el idóneo para aquellos sistemas uno a muchos y muchos a muchos, permitiendo una comunicación mucho más rápida, flexible e independiente.

Más concretamente en el desarrollo de aplicaciones colaborativas en las que se requiere una aproximación muy cercana al tiempo real, se tiende a realizar el diseño basándose en la comunicación asíncrona [Mark97] y usando replicación de datos por parte del cliente. De tal forma que cada cliente mantendrá la representación local de cada uno de los datos que se han ido enviando en la colaboración y ante un nuevo evento actualizará su representación local.

Posteriormente veremos como el uso de estos modelos ofrecen grandes ventajas en desarrollo de herramientas como son pizarras, chat, mensajería instantánea, etc, haciendo la colaboración mucho más rápida.

1.4. Otros servicios distribuidos.

En las secciones anteriores hemos visto algunos de los servicios que se abarcan dentro del concepto de middleware, más concretamente hemos analizado aquellos que constituyen la base del fruto de nuestras investigaciones. En esta sección veremos algunos servicios los cuales forman parte del middleware, los cuales también son fundamentales a la hora de desarrollar una aplicación distribuida.

Entre otros servicios que podemos encontrar en los sistemas distribuidos cabe destacar los siguientes.

Servicio de nombre

Un servicio de nombre, también llamado servicio de directorio, provee información centralizada de la gestión de nombres que se facilita a los clientes a través de la red. Los nombres se refieren a ciertos objetos en la red como pueden ser ficheros, servidores, servicios, estaciones de trabajo, dispositivos y usuarios.

El estándar LDAP [Yeo95] se ha asentado en Internet como un protocolo estándar de directorio distribuido que permite búsquedas complejas y federación inter-servidor. Sun define un acceso genérico a cualquier tipo de servicios de directorio denominado JNDI (Java Naming Directory Interface) que permite acceder a servicios de todo tipo como OMG COS Naming y LDAP.

Seguridad

Una de las necesidades de los sistemas distribuidos es el manteniendo de cierto grado de seguridad. Este servicio abarca un amplio conjunto de aspectos que van desde el control de acceso a los ordenadores, redes y la información que se almacena, se procesa y se transmite. Por ello es necesario manejar mecanismos como son control de acceso y autorización, encriptación de datos y autenticación.

Más concretamente la autenticación permite la identificación de usuarios en un entorno distribuido bien mediante una autenticación básica usando login / password sobre canal inseguro o bien usando autenticación básica sobre canal seguro encriptado, usando por servidores seguros mediante el protocolo SSL. También es notable la autenticación mediante el uso sistemas de certificados e infraestructuras de clave pública (PKI). La autorización se refiere a qué está permitido hacer a cada usuario y normalmente utiliza listas de control de acceso (ACL) a los recursos y sistemas de roles.

Control de la concurrencia

El control de concurrencia es un mecanismo de bajo nivel para resolver conflictos de actualización en registros de base de datos. Algunas técnicas estándares son two-phases locking, token passing, logical timestamping, y consenso por mayoría.

Los controles de concurrencia avanzados emplean actualizaciones multinivel, bloqueos anidados, información semántica del dominio, reestructuración dinámica, y otras técnicas refinadas. En contraposición con los modelos estándares de transacción, las transacciones cooperativas en modelos avanzados intentan relajar las premisas ACID, permitiendo múltiples accesos de larga duración, eventos impredecibles, interacciones con otras actividades concurrentes y el control de usuario.

Servicio de transacción

Los conflictos estándares se reducen a “read-write” y “write-write” y son resueltos con transacciones, cuya semántica operacional garantiza la atomicidad (atomicity), consistencia (consistency), aislamiento (isolation) y durabilidad (durability), lo que es llamado las propiedades ACID. Transacciones “optimistas” son llevadas a cabo (committed) a pesar de la presencia de conflictos, si todos los de un grupo de sitios votan por aceptar la transacción y abortada en otro caso.

Más concretamente, desde Sun se proporciona la API JTA (Java Transaction API) para manejar las transacciones dentro de un ámbito Java [JTA].

Servicio de persistencia

Este servicio permite el almacenado persistente tanto de datos como de componentes en bases de datos. Más concretamente, Sun ha definido un modelo de persistencia de componentes que ataca fuentes de datos heterogéneas y facilita la persistencia de estado de los componentes de manera transparente al programador mediante el uso de DAO (Data Access Object) [DAO].

Servicio de tiempo

Una de las consecuencias del uso de sistemas distribuidos es que el manejo del tiempo en cada uno de los sistemas que forman parte de la aplicación es independiente. En muchas aplicaciones es fundamental el manejo de esta información. Para ello mediante el uso del protocolo NTP (Network Time Protocol) puede ser garantizada una sincronización en un tiempo exacto de diez milisegundos.

Servicios peer-to-peer

Debido a la diversidad de entornos que existen a través se la red, teléfonos móviles, PDAs, sistemas de navegación de coches, etc., se requieren nuevas formas de comunicación, que permitan a estos dispositivos colaborar de iguales a iguales.

En concreto, Sun ofrece la API JXTA [JXTA] cuyo objetivo es soportar aplicaciones a través de la red que sean interoperables, p2p permitiendo el encuentro de otros peers en la red con un servicio de descubrimiento dinámico a través de firewalls, compartir ficheros con cualquiera a través de la red, crear grupos de peers que puedan encontrarse unos a otros, a través de firewall, monitorizar las actividades del peer remotamente y permitir conexiones seguras entres los peers de la red.

1.5. Uso de Java middleware.

Java es un lenguaje de programación, pero su sistema de soporte circundante lo ha convertido más bien en una plataforma de computación, una base en la cual los desarrolladores de software pueden construir sus aplicaciones distribuidas [Bac03]. Los aspectos que aporta Java con respecto a las plataformas tradicionales son las siguientes:

- **Los programas Java son móviles,** o lo que es lo mismo pueden migrar a través de la red. Los datos y el código a procesar son transmitidos al mismo tiempo. Un escenario típico es el uso de applets que son mantenidos en un servidor y descargados bajo demanda.
- **Heterogeneidad es enmascarada a través de la interpretación.** Los programas son transferidos como código intermedio (bytecode). El código puede ser ejecutado siempre y cuando los programa clientes tengan una máquina virtual Java. De esta manera se hace innecesario tener que portar la aplicación a diferentes arquitecturas.
- **Interfaz browser.** Muchos navegadores contienen plataformas Java embebidas. Ejemplos de ello son Microsoft Internet Explorer en versiones anteriores a la 6.0

y Netscape en versiones 4.X. Recientemente, los navegadores tienden a usar la plataforma virtual ofrecida por el plug-in Java, de tal forma que no haya incompatibilidades ante diferentes tipos de navegadores a la hora de usar la plataforma Java.

- **Seguridad.** A pesar de lo peligroso que pueda parecer el importar código desde otra máquina el hecho que sea interpretado y que se ejecute en un a plataforma bien definida puede ser usado para definir el alcance y confinar su acceso. Más concretamente el uso de políticas y de código firmado puede restringir las operaciones a realizar por un programa transferido a través del web.

Básicamente, el hecho de que Java sea interpretado y del gran abanico de posibilidades que abre a la hora de programar servicios Web la hace realmente interesante en el desarrollo de nuevas aplicaciones.

Además el uso de Java Applets hace más eficiente la transferencia de datos, puesto que las aplicaciones gráficas son ejecutas en la máquina local de cada usuario y lo único que es necesario tranferir son los cambios y actualizaciones de la herramienta.

Más concretamente este será el lenguaje de programación sobre el cual basaremos todos los desarrollos llevados a cabo en este trabajo de investigación.

1.6. Arquitecturas distribuidas para el desarrollo de aplicaciones.

Como hemos podido observar de lo comentado anteriormente, la capa middleware conlleva una gran cantidad de servicios que se necesitan prestar a la hora de llevar a cabo las tareas requeridas en la aplicación. A lo largo del funcionamiento de una aplicación puede ser necesario cambiar ciertos servicios para atender más eficientemente las necesidades de las aplicaciones. Esta reingeniería en la aplicación puede conllevar un trabajo tedioso si el código no ha sido desarrollando siguiendo especificaciones estructuradas y la separación de cada una de los diferentes servicios no ha sido llevada de forma adecuada.

En esta sección se expondrán los paradigmas y especificaciones más relevantes en el desarrollo de plataformas software. Muchas de estas especificaciones han llegado a convertirse en la clave fundamental para el desarrollo y posterior mantenimiento de una aplicación, puesto que sus especificaciones bien detalladas y basadas en estándares implican el desarrollo de aplicaciones bien estructuradas favoreciendo la extensibilidad. De esta forma se pueden adaptar nuevos servicios y funcionalidades con cambios mínimos en la aplicación global.

Veremos primeramente el modelo de tres capas analizando las funcionalidades de este modelo y en concreto el modelo vista control. Basado en este modelo analizaremos el modelo de componentes. Mostraremos los desarrollos llevados desde el lado del cliente, Java Beans, así como los del lado del servidor, J2EE, CORBA CCM y .NET.

Uno de los modelos que se están adoptando ampliamente para la implementación de aplicaciones es el **modelo de tres capas** (Véase figura 9). Básicamente este modelo pretende organizar las diferentes partes de la programación en diferentes capas de tal manera que partes diferentes de cada una de las capas serán analizadas en sus

correspondientes niveles. Más concretamente el modelo de tres capas comprende los siguientes niveles de abstracción:

- **Capa de Presentación**. Esta capa recogerá toda la codificación del aspecto visual de la aplicación.
- **Capa de Lógica**. En esta capa es dónde se recogerá todas las acciones importantes a realizar en la aplicación, es decir todos los algoritmos o implementación de las acciones o métodos.
- **Capa de Datos**. En esta capa se tratan todas las acciones relacionadas con la búsqueda, inserción y modificación de los datos de la aplicación que permanecen guardados en base de datos, directorios o cualquier otro tipo de sistema que nos permita el guardado de información de la aplicación.

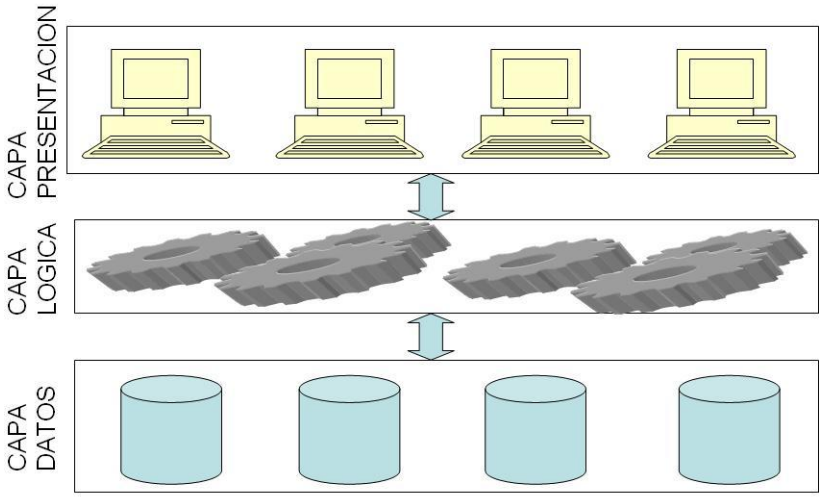

Figura 9. Modelo de tres capas

Uno de los primeros modelos que surgieron con está filosofía fue el modelo vista control MVC. MVC define una dependencia uno a muchos entre objetos, en la cual, cuando un objeto cambia de estado, todos los objetos interesados o dependientes son notificados y actualizados automáticamente. Esta clase de interacción, también conocida como publicación / suscripción, puede ser implementada en un entorno distribuido de manera muy eficiente. Cabe destacar este modelo por su gran impacto en la construcción de aplicaciones colaborativas [Gra96] [Pat96].

Pese a que este modelo de diseño simplifica y hace más claro el uso de los diferentes elementos de la aplicación, por si sólo no es suficiente para obtener una aplicación adaptable a nuevos cambios y servicios.

A continuación estudiaremos la estructura de los modelos de componentes, que básandose en este modelo ofrecen mejores características en cuanto a independencia de la fuente de datos y facilitan la reutilización de las aplicaciones.

1.7. Modelo de Componentes.

Este tipo de modelos están tomando grandes repercusiones en la actualidad puesto como podremos ver en este apartado ofrecen grandes ventajas. Más concretamente, parte de

nuestro trabajo realizado en la investigación propuesta se basa en el uso de este tipo de tecnologías.

El paradigma orientado a objetos ofrece dos ventajas importantes en la construcción de sistemas abiertos: por un lado, define la organización de un sistema como una colección de objetos que colaboran para realizar una tarea común; y en segundo lugar, consigue un alto grado de reutilización a través de los mecanismos de herencia y polimorfismo.

Sin embargo, y pese a su probada eficacia en sistemas software, este paradigma no expresa claramente la distinción entre el punto de vista computacional y el punto de vista composicional de una aplicación. Además, prevalece la visión de objeto sobre la visión de componente como entidad que debe ser compuesta con otras entidades para formar una aplicación.

La noción de orientación a componentes surge como una extensión de la orientación a objetos. Un **componente** es una unidad de software que está formado por sus propios datos y lógica, mediante conexiones bien definidas o interfaces para la comunicación. Está diseñado para su uso repetitivo en el desarrollo de aplicaciones [Szy98]. Esta concepción tiene la visión de un componente como un punto de ensamblaje de tal forma que un desarrollador puede construir aplicaciones mediante el ensamblaje de estos componentes, más que usando el desarrollo de algoritmos. De esta forma el middleware es visto como un modelo descompuesto en vez de un modelo monolítico. Véase figura 10.

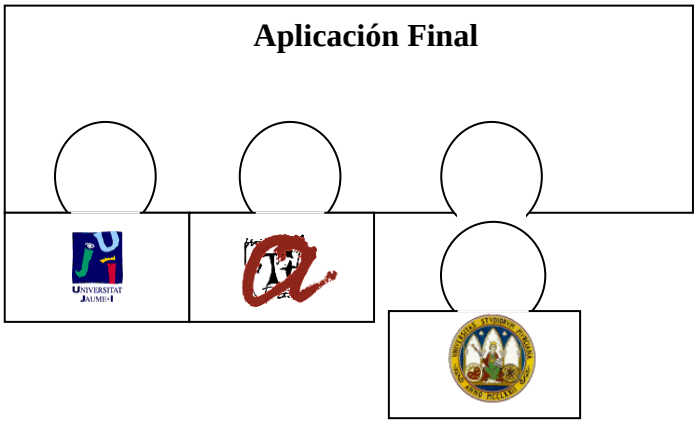

Figura 10. Aplicación construida por diferentes componentes.

El modelo de componentes abarca dos campos de actuación. Por un lado tenemos el conjunto de tecnologías creadas para el desarrollo de componentes desde el cliente, como pueden ser COM, Java Beans y los Objetos Corba. Por otro lado tenemos la perspectiva desde el servidor, con tecnologías como .NET, J2EE y CORBA CCM. A continuación se expondrán las tecnologías estudiadas en ambos campos.

Modelo de componentes en el lado del cliente.

El objetivo común de los modelos de componentes orientados al cliente es el desarrollo de componentes en entornos visuales que permitan la reutilización y empleo de los mismos en la creación de aplicaciones en el lado del cliente.

Según García [Gar02] en un entorno de programación visual, los componentes se arrastran de una paleta o repositorio de componentes hasta el formulario o contenedor visual. La herramienta de programación visual escribe código de manera automática en respuesta a las órdenes visuales e inserta el componente en el marco de trabajo del contenedor visual.

En general, cada componente tiene una serie de características y un conjunto de comportamientos. Las características pueden ser modificadas en tiempo de diseño y se suelen denominar propiedades. Estas propiedades pueden ser manipuladas en el entorno de programación visual, y su estado debe poder ser almacenado para un uso posterior (estado persistente).

En tiempo de diseño, los comportamientos de un componente visual se representan mediante eventos, que son las acciones que ocurren en el componente. Normalmente, el programador decide que hacer cuando un evento ocurre, ligando fragmentos de código a eventos concretos. Los entornos visuales interrogan a los componentes utilizando "reflexión" (introspection) para detectar sus propiedades y sus eventos y poder así visualizarlos correctamente.

Principalmente estudiaremos el modelo de Java Beans debido a su elegancia y sencillez.

Sun Java Beans

Según Graham [Gra97] "un JavaBean es un componente software que puede ser manipulado visualmente en una herramienta visual". Cada bean será diferente dependiendo de las funcionalidades que soporten, pero todos poseen las siguientes características:

- **Soporte para Introspection** lo cual permite que el constructor de una herramienta pueda analizar como trabaja un bean.
- **Soporte para Customization** lo cual permite a un usuario modificar la apariencia y comportamiento de un bean.
- **Soporte para eventos** permitiendo que los beans puedan lanzar eventos e informar a las herramientas de construcción acerca de los eventos que pueden lanzar así como los eventos que pueden manejar.
- **Soporte para propiedades** permitiendo que los beans sean manipulados por programas, así como soporte para la "customization" mencionada anteriormente.
- **Soporte a la persistencia** permitiendo que aquellos beans que han sido adecuados en una herramienta de aplicación sean guardados y restaurados.

El modelo propuesto por los JavaBeans se basa en clases Java que se adhieren a ciertas propiedades y eventos siguiendo un interfaz estándar. Las propiedades definen el estado del componente y son accedidas y reconocidas por los entornos visuales utilizando introspección o reflexión sobre la notación estándar. Los eventos son un mecanismo para propagar las notificaciones de cambio de eventos entre un objeto fuente y uno o más consumidores.

Más concretamente este modelo ha sido usado por García [Gar02] a la hora de construir una plataforma ANTS, la cual veremos en posteriores capítulos.

Modelo de componentes en el lado del servidor.

Más concretamente en esta sección nos vamos a centrar en el estudio de los modelos de componentes servidores puesto que constituyen la base del trabajo desarrollado en esta investigación. Estos servidores también son conocidos con el nombre de *servidores de aplicaciones*. Dicha tecnología introduce un cambio en el desarrollo de aplicaciones favoreciendo la portabilidad del código al igual que la extensibilidad de las aplicaciones, de tal manera que han sido ampliamente aceptadas en el mercado para el diseño de cualquier aplicación distribuida.

Primeramente analizaremos el diseño que siguen los servidores de aplicaciones y a continuación expondremos las tecnologías más importantes dentro de este campo.

Los componentes pueden generar y recibir eventos, bien publicando o emitiendo eventos o bien consumiendo eventos. Más concretamente en el modelo servidor el intercambio de eventos se diseña usando tecnologías asíncronas como pueden ser del modelo publish/subscribe y en el caso de CORBA push/pull.

Un modelo de componentes define como los componentes deben importar o exportar funcionalidades. Siguiendo este paradigma los clientes acceden a los componentes usando comunicación síncrona como puede ser RPC u otros estilos, al igual que para crear componentes o buscarlos. Concretamente para la búsqueda de componentes se hará uso de los patrones *Home*. Un **home** es un gestor de todas las instancias de un determinado tipo de componente, siendo además el responsable del ciclo de vida del componente. Por lo tanto suele gestionar las operaciones de creación, borrado y modificación, proporcionando además métodos para buscar componentes por la clave principal. En la búsqueda de los patrones home, se hace uso de los servicios de nombre.

Un **contenedor** es usado para abstraer al componente de la especificación concreta de los servicios del sistema, de tal forma que el componente accederá a ellos usando patrones de diseño. Los contenedores son responsables de realizar las siguientes tareas:

- Manejar el ciclo de vida de los componentes y notificar a los componentes de los eventos del ciclo de vida como pueden ser la activación, desactivación y progreso de la transacción.
- Provee a los componentes un acceso uniforme a servicios como son la seguridad, transacción y persistencia.
- Registra y despliega (deploy) los componentes.

En la figura 11 se muestra el diseño seguido por el modelo de componentes desde el lado del servidor. Como podemos apreciar en dicha figura el contenedor es el responsable de gestionar las peticiones de los clientes a los componentes necesarios igualmente suministra o informa a los componentes de ciertos servicios como son la seguridad, transacción, etc.

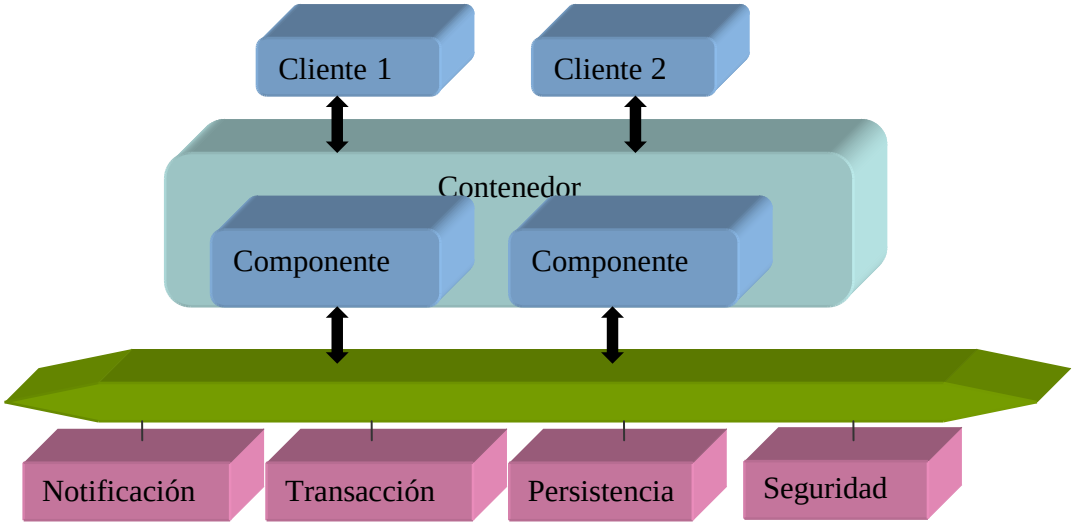

Figura 11. Modelo de componentes servidores.

Normalmente un contenedor es una rutina de entorno para la gestión de componentes que puede estar basado en una de las siguientes tecnologías; servidores web, servidores de bases de datos, servidor CORBA, un sistema operativo, etc.

Básicamente en el control de la persistencia se pueden distinguir las siguientes políticas:

- **Component Managed Persistence.** De esta manera se indica que los componentes serán los responsables de acceder a los datos persistentes siendo responsabilidad de ellos la implementación de los métodos y de originar las consultas en el lenguaje necesario para acceder a las fuentes de datos.
- **Container Managed Persistence.** De esta manera es el contenedor el responsable del uso y creación de métodos para acceder a las fuentes de datos. Normalmente la creación de la información así como la generación de los métodos acorde al lenguaje requerido por la fuente de datos será responsabilidad total del contenedor al realizar el despliegue (o deployment) de los componentes.

Cómo podremos observar posteriormente, en nuestros desarrollos nos hemos basado en el uso de la política "Container Managed Persistence" puesto que ofrece un nivel de independencia mayor en cuanto a la fuente de datos a usar y hace más rápido el desarrollo de aplicaciones.

Previo al despliegue y el registro del componente en el contenedor, éste debe ser empaquetado. El empaquetamiento de un componente se realiza en base a un fichero descriptor, normalmente en XML, que será usado por el contenedor en tiempo de ejecución y un conjunto de ficheros que implementan el componente.

Otra parte importante que se incorpora en los servidores es el componente de montaje, mediante el cual se especifican los componentes que están disponibles en el servidor, es decir, los componentes que han sido desplegados. Para ello se suele usar un fichero descriptor en XML en el cual se describe cada uno de los componentes.

Explicadas las principales características de un modelo de componentes orientados al servidor, a continuación presentaremos los posibles entornos que podemos elegir en base a la creación de aplicaciones distribuidas.

Sun Microsystems, aprovechándose de la gran cantidad de APIs existentes en el lenguaje Java, accediendo a diferentes servicios distribuidos, propuso un modelo propio aunque abierto de marco de integración de servicios. El marco propuesto se denomina J2EE (Java 2 Enterprise Edition) y ha conseguido un alto grado de aceptación entre los fabricantes de software. Así, hay una gran cantidad de servidores de aplicaciones en el mercado, incluyendo proyectos de código abierto como JBoss, que cumplen las especificaciones J2EE. Al día de hoy, el marco de integración de servicios más maduro y estable es la especificación J2EE, si bien .NET está cobrando relativa importancia en desarrollos basados en plataformas Windows.

Junto con la especificación J2EE, las alternativas existentes son la propuesta del OMG denominada CORBA Component Model y la plataforma global de Microsoft denominada .NET. Las tres alternativas definen un modelo estándar donde componentes distribuidos pueden acceder a los servicios de ciclo de vida, directorio, persistencia, transacciones, notificaciones y seguridad. El modelo OMG abarca el modelo J2EE y lo extiende para ser una plataforma multi-lenguaje, mientras que el modelo .NET se basa en las tecnologías de componentes del modelo propietario de Microsoft.

J2EE (Java 2 Enterprise Edition)

J2EE [J2EE] es una especificación del lenguaje Java para el desarrollo de aplicaciones distribuidas basada en un modelo de servidor de aplicaciones. Una de las tecnologías englobadas dentro de J2EE es EJB (Enterprise JavaBeans), la cual es una especificación que engloba la definición de componentes y sus interacciones; una tecnología de empaquetamiento para el despliegue de los componentes y un marco contenedor que engloba la seguridad, transacciones, notificación de eventos y persistencia.

EJB define dos categorías de componentes llamados *entity beans* y *session beans*. Los *entity beans* son usados para representar datos persistentes, normalmente los datos que son almacenados en una base de datos. Sin embargo los *session beans* son extensiones de las aplicaciones cliente y son responsables del manejo de ciertas tareas y normalmente su duración esta asociada a la interacción del cliente.

Con el objetivo de que los componentes, o beans, puedan intercambiar mensajes se ha incluido JMS como una parte integral de esta especificación, de tal manera que sea el soporte para el modelo publish/subscribe que requieren los servidores de aplicaciones.

En general J2EE incluye un amplio abanico de tecnologías Java que van desde el servicio de nombres (JNDI), servicio de transacciones (JTA), soporte a la creación de componentes Web (JSP, servlets,), conector a base de datos (JDBC) hasta los servicios de seguridad (JAAS, PAM, JACC). A parte de estas tecnologías incluye XML como el estándar de facto para la configuración de componentes o intercambio de datos, así como incluye una API para hacer el parser de XML (JAXP), ofreciendo soporte a SOAP. Además incluye interoperabilidad con componentes CORBA mediante el uso de RMI-IIOP así como proporciona Java IDL para invocar componentes CORBA.

Esta especificación ha sido ampliamente aceptada por los vendedores contando con numerosos productos compatibles con esta especificación.

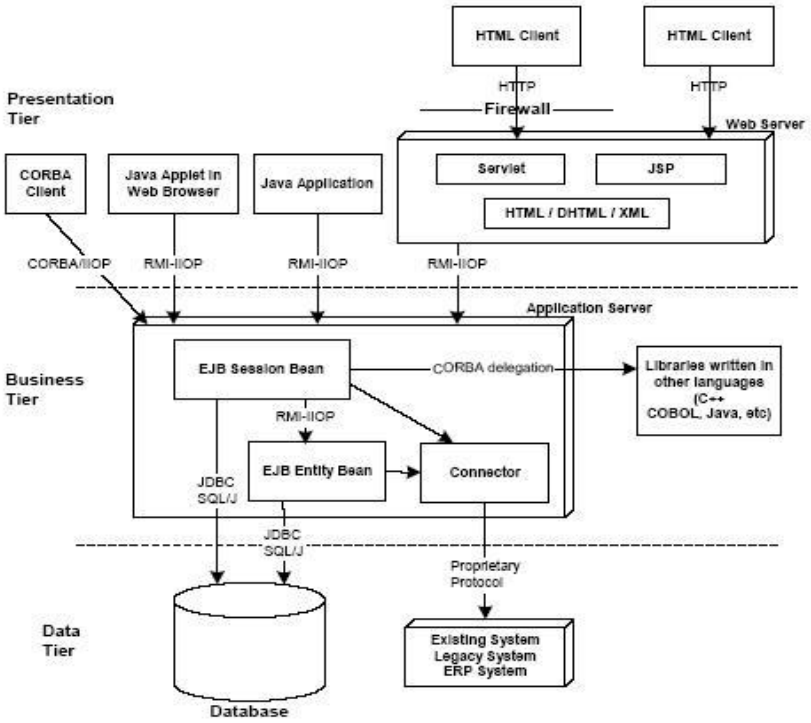

Figura 12 Arquitectura J2EE

Entre las ventajas que ofrece esta arquitectura frente a su competidora .NET [Cha01] [Hum02] cabe destacar su mayor rapidez a la hora de desarrollar código, su madurez, su mayor interoperabilidad y mayor portabilidad. Debido a estas características hemos considerado esta especificación como base para el desarrollo de una arquitectura de componentes en tareas colaborativas. Otra de las características que la hace indispensable a la hora de constituir la base del desarrollo de plataformas colaborativas es el hecho de que hay implementaciones de esta especificación que no son propietarias. Más concretamente en la investigación desarrollada trabajaremos con dos de ellos Orion y JBoss.

CORBA Componente Model (CCM)

La especificación CCM, como otras especificaciones del OMG está tardando bastante en ser estandarizada y aceptada por los fabricantes de software. Según García [Gar02] *"La complejidad de sus especificaciones y la lentitud de su evolución determinan que el mercado no reciba con más aceptación unas tecnologías tan avanzadas e innovadoras"*.

El modelo CCM ofrece la característica de ser multi-lenguaje al igual que otros conceptos que lo hacen abarcar un nivel superior al de J2EE. En el modelo CORBA las se introduce dentro del concepto de componente las *facetas* (*facets*) o la posibilidad de definir diversas interfaces remotas a un componente, los *receptáculos* que facilitan la conexión entre componentes, las *fuentes* y *pozos* de eventos para la comunicación de eventos asíncronos a componentes, y un sistema de atributos facilitando la configuración de los componentes. Además define un lenguaje de definición de

implementación de componentes (CIDL) que favorece la conexión de varios tipos de implementaciones.

Se basa en el modelo push/pull definido en CORBA para la notificación de eventos entre componentes. Al igual que J2EE abarca temas de seguridad mediante el uso de SECIOP, al igual que soporte ODBC, JDBC y otras API's de bases de datos. Usa el estándar XML para la configuración de ficheros.

Una de las grandes ventajas que presenta este modelo es la interoperabilidad que ofrece con el modelo J2EE, de tal manera que el contenedor CORBA también puede manejar componentes EJB.

Microsoft .NET

Microsoft .NET [Mic03] constituye el marco para la construcción de arquitecturas distribuidas usando tecnologías Microsoft. Microsoft .NET ha tomado las características que ofrecía Windows DNA, la cual fue la plataforma previa propuesta por Microsoft para la construcción de aplicaciones distribuidas.

Permite la construcción de aplicaciones que pueden funcionar desde el ordenador personal hasta pequeños dispositivos como pueden ser los PDA. Se basa también en el uso del estándar XML para el intercambio de información entre aplicaciones.

Con respecto a los servicios web se basa en el uso de *SOAP* (Simple Object Access Description), *WSDL* (Web Service Description Language) y *UDDI* (Universal Description, Discovery and Integration). Principalmente ofrece *ASP.NET* para la construcción de aplicaciones Web.

El acceso de datos se hace a través de *ADO.NET* el cual proporciona un marco global para el manejo de cualquier base de datos OLE así como .Net SQL server. Para el manejo de transacciones se basa en *Microsoft Transaction Server (MTS)*. Además ha creado un servicio de directorio basado en LDAP llamado *Active Directory* que centraliza los accesos a la plataforma, y dispone de un modelo de seguridad global fuertemente ligado a la plataforma Windows.

Se basa en el uso de COM y DCOM para la comunicación de componentes junto con la tecnología *Message Queue Server*.

Además soporta interoperabilidad con diferentes lenguajes de programación mediante el uso de *Microsoft Intermediate Language (MSIL)*, también llamado IL. Para que este código sea interpretado y traducido se proporciona el *Common Language Runtime (CLR)*, análogo al Java Runtime Environment de Java.

Los estudios realizados por la compañía "The Middleware Company" demuestran que bajo sistemas Windows .NET ofrece mayor rendimiento que el que pueda ofrecer cualquier sistema J2EE [Mid02], por lo cual esta arquitectura es la idónea cuando el sistema a construir es totalmente dependiente de sistemas Windows. En otro caso, debido a su falta de interoperabilidad con otros sistemas operativos y componentes como pueden ser Corba o EJB, el más adecuado es un sistema J2EE.

1.8. Conclusiones.

En este apartado hemos estudiado las diferentes tecnologías middleware para la comunicación de aplicaciones distribuidas así como las últimas tendencias en la construcción de plataformas distribuidas. De este análisis hemos concluido lo siguiente:

- Para la construcción de herramientas que necesiten una comunicación de muchos a muchos el mejor middleware es el orientado a mensajes (MOM), puesto que permite una comunicación más rápida y flexible.
- De las diferentes tecnologías estudiadas hemos podido comprobar que cada una de ellas imponen diferencias en la programación de las herramientas. Con lo cual se hace patente la necesidad de una API genérica que facilite la programación de herramientas colaborativas, sin que el cambio de tecnología middleware implique la reestructuración del sistema.
- En la construcción de una plataforma distribuida el mejor modelo a seguir es un sistema híbrido puesto que plantea las ventajas de ambos paradigmas comunicación: el síncrono es especialmente útil para arquitecturas en las cuales se presta servicio a fuentes de datos, el modelo asíncrono es el idóneo para aquellos sistemas uno a muchos y muchos a muchos.
- Relacionado con aspectos de implementación del sistema, consideramos que el lenguaje Java constituye una alternativa sólida para el desarrollo de nuestra solución. Su carácter multiplataforma, la potencia y versatilidad del lenguaje, y características como la reflexión, serialización y carga dinámica remota lo hacen especialmente interesante para el problema que queremos modelar.
- El modelo Vista Control proporciona un buen diseño para herramientas, para pequeñas aplicaciones. Sin embargo en la construcción de aplicaciones distribuidas que ofrecen un conjunto de servicios este modelo es inádecuado.
- El modelo de servidor de aplicaciones es el idóneo para la construcción de aplicaciones distribuidas, favoreciendo la independencia de la base de datos así como el manejo de ciclo de vida del componente.
- Entre todas las posibilidades tecnológicas disponibles para abordar la construcción de un servidor de aplicaciones, J2EE, CORBA CCM, y .NET hemos elegido J2EE para abordar nuestros desarrollos, puesto que resulta un estándar más maduro en el que hay implementaciones de servidores no propietarios, está sustentada sobre un gran conjunto de estándares y ofrece mayor interoperabilidad y portabilidad que sus competidoras.

2. Entornos Colaborativos (CSCW).

La gran difusión de Internet ha provocado grandes cambios en nuestra sociedad y en la forma en que los individuos interactúan. Dichos cambios son adoptados en diversos campos organizativos con el objetivo de fundar un trabajo de grupo independientemente dónde se encuentren cada uno de los miembros. Con el fin de diseñar aplicaciones que permitan el desarrollo de estos objetivos surgen los conceptos de "CSCW" y "Groupware" en relación al trabajo en grupo.

En primer lugar, podríamos definir **CSCW (Computer Supported Cooperative Work)** como el campo de investigación que examina el diseño, adopción y utilización de tecnologías informáticas cuyo objetivo primordial es facilitar los procesos de interacción y colaboración en el trabajo cooperativo. El término fue acuñado en 1984 por Irene Greif [SG85] y Paul Cashman [SC84] en un intento de agrupar a investigadores de diferentes áreas que deseaban resolver un problema común: el trabajo en grupo mediado por ordenador.

Sin embargo este campo no se refiere únicamente a la cooperación y trabajo en grupo, también tiene en cuenta las tecnologías empleadas en el desarrollo de las aplicaciones, la adaptación al medio social y la representación. En general, envuelve cualquier diseño de software interesado en el comportamiento social y organizacional, incluyendo áreas como la ingeniería de comunicación, gestión de la información, sociología y las teorías organizacionales.

Otro concepto importante es el de **GroupWare**. El término Groupware es utilizado para denotar aquellos productos o aplicaciones orientadas al soporte de trabajo en grupo. El término lo establecen Peter y Trudy Johnson-Lenz en 1981 [JJ81], vinculando directamente el trabajo en grupo (GROUP) con las herramientas software que lo soportan y facilitan (*softWARE tools*). De hecho CSCW también ha sido definido como "el estudio de las herramientas y técnicas de groupware así como sus efectos psicológicos, sociales y organizativos".

El diseño de GroupWare conlleva entender a los grupos y cómo se comportan sus miembros. También implica tener un buen conocimiento de cómo funcionan las tecnologías de red y cómo afectan éstas a la experiencia del usuario. Cuando se trata de grupos, éstos requieren una consideración especial por lo que los diseñadores deben tener conocimiento del grado de homogeneidad de los usuarios, de los posibles roles de las personas que actúan en trabajos cooperativos, de quién toma las decisiones y qué influencia tienen.

Los sistemas GroupWare ofrecen importantes ventajas sobre los sistemas monousuario. Algunas de estas ventajas se exponen a continuación:

- Facilitar la comunicación. El uso de GroupWare establece una comunicación más rápida, clara y convincente, habilitando la comunicación dónde antes no era posible.
- Habilitar la telecomunicación reduciendo costes en viajes.
- Reunir múltiples perspectivas y formalismos.

- Formar grupos con un interés común donde no era posible recoger toda esa cantidad de personas cara a cara.
- Ahorrar tiempo y coste en la coordinación del trabajo en grupo.
- Facilitar la resolución de problemas en grupo.
- Habilitar nuevas formas de comunicación, como intercambios anónimos o iteraciones estructuradas.

Otra buena razón por la que se estudia el uso y los aspectos de diseño de GroupWare es evitar el fracaso en los diseños. Típicamente, un sistema GroupWare no puede tener éxito a menos que muchos o todos los miembros implicados deseen adoptar el sistema. Por contra, un sistema monousuario puede ser útil incluso si una pequeña fracción lo adopta.

2.1. Clasificación de los sistemas GroupWare.

Varios esquemas han sido propuestos para los sistemas groupware. El más extendido en este campo de investigación es la matriz tiempo-espacio de Johansen [Joh88]. La matriz establece cuatro bloques principales de aplicaciones colaborativas en base a la coincidencia o no de los actores en tiempo y espacio. Observando la matriz quizá el bloque menos comprensible sea la coincidencia espacial pero no temporal; sin embargo se han propuesto trabajos en esta línea, de salas de reunión con paneles digitales y pantallas dónde los usuarios podrían dejar sus anotaciones o trabajos para ser posteriormente analizados o estudiados por otros usuarios en la misma habitación.

	Coincidencia temporal (síncronas)	No coincidencia temporal (asíncronas)
Coincidencia espacial (local)	Sala de conferencias electrónica (Electronic meeting room)	Sala de proyectos electrónica (Electronic project room)
No coincidencia espacial (remoto)	Pizarra compartida Videoconferencia	Foros de discusión Planificación de proyectos

Tabla 1. Clasificación de Johansen

Más concretamente las investigaciones llevadas a cabo durante nuestro trabajo abarcan casi toda la tabla exceptuando el caso de coincidencia espacial y no temporal, llevándose a cabo una comunicación a través de redes de ordenadores.

Como podemos apreciar en la tabla 1, atendiendo a la coincidencia temporal existen dos formas de enfocar la cooperación y por tanto dos grupos de herramientas groupware atendiendo a esta clasificación. Por un lado tenemos las **herramientas groupware síncronas**, que son todas las herramientas en la que es necesario que todos los colaboradores estén conectados para llevar a cabo la colaboración (coincidencia temporal). Por otro lado tenemos el conjunto de **herramientas groupware asíncronas** en las cuales no se necesita que todos los colaboradores participen al mismo tiempo, sino que pueden ir contribuyendo al trabajo en diferentes plazos de tiempo (no coincidencia temporal).

El uso de herramientas groupware síncronas permite que la colaboración sea más eficaz en el sentido que permite obtener la información y el consenso en el grupo de una manera más rápida. Este tipo de colaboración ofrece las principales ventajas de una colaboración cara a cara. Por otro lado, el uso de herramientas groupware asíncronas nos permiten el almacenamiento así como la visualización de la información para que esté disponible para todos los colaboradores independientemente de cuando se conecten. De esta forma la información de la colaboración queda recogida en las bases o repositorios del sistema.

Aunque está es la clasificación más extendida, cabe destacar que existen herramientas groupware que ofrecen ambos tipos de comunicación, pudiendo usar la colaboración asíncrona para la recuperación de la información así como el guardado de una sesión y la colaboración síncrona para el desarrollo de las tareas que se requieran realizar, en un punto determinado. De esta forma los colaboradores ven enriquecida su comunicación y la colaboración llevada a cabo mediante estas herramientas. A lo largo de este trabajo de investigación veremos las ventajas que ofrecen en el entorno educativo el poder disponer de ambos escenarios de colaboración.

A continuación mostraremos las herramientas groupware típicas de ambos tipos de colaboración.

Aplicaciones GroupWare asíncronas

- **Sistemas de workflow.** Permiten que los documentos sean pasados a través de la organización mediante una serie de procesos fijados. Los sistemas de workflow pueden proveer características como el encaminamiento, desarrollo de formularios y soportar diferentes roles y privilegios. La difusión de estas herramientas en el mundo de los negocios ha llegado a ser tan importante que se ha creado el WorkFlow Management Council con el objetivo de especificar los estándares y funciones básicas de un sistema de workflow [WMFC] así como un conjunto de interfaces para la interoperabilidad.

- **E-mail.** Es una de las más antiguas y común de las aplicaciones GroupWare. Mientras que la tecnología básica está diseñada para pasar mensajes entre dos personas, actualmente los sistemas básicos de e-mail incluyen características interesantes para reenvío de mensajes, archivar mensajes, crear grupos de correo e incluir ficheros en un mensaje. Otras características que han sido exploradas incluyen ordenaciones automáticas y procesamiento de mensajes, encaminamiento automático, y una comunicación estructurada (mensajes requiriendo cierta información).

- **Newsgroups y mailing lists.** Son similares a los sistemas de e-mail, excepto que son elegidos para mensajes entre grupos grandes en lugar de la comunicación uno a uno. En la práctica la principal diferencia entre newsgroups y mailing lists es que newsgroups sólo muestra los mensajes a un usuario cuando son explícitamente requeridos (servicio bajo demanda), mientras que los mailing lists entregan los mensajes cuando están disponibles.

- **Hypertext.** Es un sistema para enlazar diversos documentos de texto entre sí, siendo el Web un ejemplo obvio. Algunos sistemas de hipertexto incluyen la

posibilidad de saber qué usuarios han visitado una determinada página o enlace, o al menos conocer con qué frecuencia se realizan las visitas y tener así una idea aproximada de lo que el resto de los colaboradores está haciendo en el sistema.

- **Group calendars.** Permite planificación, gestión de proyectos y coordinación entre grupos de personas. Algunas de las opciones que presentan es detectar cuándo existen conflictos entre personas en una determinada planificación o buscar si existe tiempo disponible para realizar una reunión.

- **Sistemas de edición colaborativos.** Permite la realización de documentos entre varios colaboradores los cuales realizan anotaciones sobre el documento estableciendo turnos para ello y de forma que cada colaborador sea capaz de ver las anotaciones realizadas por los demás.

- **Gestores de documentos (Document Management System).** Los gestores de documentos compartidos nos permiten dejar documentos (material de estudio) en una zona accesible por varios usuarios. Muchos de ellos incluyen soporte al control de versiones, bloqueo de documentos, sistemas de búsqueda de los documentos así como la definición de diferentes roles y privilegios en el uso de las carpetas o zonas compartidas dónde se organizan los documentos.

Aplicaciones GroupWare síncronas

- **Sistemas de video conferencia.** Permite la transmisión de audio y/o video entre varias personas, ofreciendo la posibilidad de que dichas comunicaciones sean del tipo 1-1, 1-N o N-N.

- **Pizarras compartidas.** Permite a dos o más personas visualizar una zona de dibujo común así como realizar anotaciones o dibujos sobre ella. Las tareas permitidas por este tipo de pizarras van desde las más simples a las más sofisticadas como puede ser la realización de presentaciones y permitir la colaboración en proyectos de ingeniería o diseño gráfico.

- **Chat.** Permite a varias personas escribir mensajes en tiempo real en un espacio público de forma que cada uno de los participantes en el chat tenga constancia de quién envía cada mensaje.

- **Sistemas de ayuda a la toma de decisiones (Decision Support Systems) .** Ayuda en la toma de decisiones a grupos de personas destinados para ello.

- **Sistemas de Mensajería Instantánea.** Permiten que los usuarios mantengan una lista de contactos a los cuales les pueden enviar mensajes o ficheros. Es similar a un sistema de e-mail sólo que se puede establecer una comunicación más rápida si ambos usuarios están conectados al mismo tiempo. Uno de los aspectos más relevantes de estas herramientas es la visualización de características de presencia, pudiendo de esta manera saber qué usuarios están conectados y cuales no. Muchos de ellos incluyen una cola de mensajes que permiten mantener dichos mensajes y enviarlos en el momento que el destinatario se conecte al sistema.

2.2. Principios del diseño GroupWare.

Los investigadores en CSCW han identificado una serie de principios que deberían contemplarse en el desarrollo de una aplicación groupware con el fin de obtener ventajas similares a las ofrecidas en colaboraciones cara a cara. En esta sección veremos algunos principios y conceptos que influyen en el diseño de una herramienta groupware.

Task Coupling (Grado de cooperación)

Con este concepto se refleja el grado de acoplamiento en el que los colaboradores trabajan, en contra de la independencia. Dewan y Choudhary [Dew98] argumentan que un sistema cooperativo debe soportar una cooperación flexible. Sin embargo el grado de cooperación (cómo de estrecha es la relación entre los colaboradores) va a depender del uso de la aplicación, dependiendo por tanto de los grupos que la vayan a usar.

What You See Is What I See.

Una de los primeros conceptos establecidos a la hora de manejar el área compartida de la aplicación es el denominado **What You See Is What I See (WYSIWIS)** [Ste87], mediante el cual se pretende mantener un contexto consistente de la información compartida en todos los participantes. Más concretamente, este concepto define un modo de colaboración síncrona dónde todos los participantes ven exactamente lo mismo y aproximadamente al mismo tiempo.

Sin embargo WYSIWIS es demasiado estricto, con lo cual seguir este principio de forma tan rigurosa será beneficioso y útil dependiendo para qué tipo de entorno se esté desarrollando la aplicación. Debido a que para muchas aplicaciones colaborativas WYSIWIS no es útil, se han propuesto una serie de relajaciones que abarcan las siguientes dimensiones.

- **Espacio.-** Las localizaciones de todos los elementos compartidos son las mismas siguiendo WYSIWIS. Sin embargo a veces es útil permitir a los colaboradores tener vistas independientes de los elementos compartidos.
- **Tiempo.-** Bajo WYSIWIS de forma estricta, los cambios en la información compartida son vistos por todos los colaboradores casi simultáneamente. Relajando esta restricción podemos permitir que el sistema envíe modificaciones en intervalos más óptimos, quizás diferentes espacios de tiempo para cada colaborador. Si esta característica se relaja totalmente se obtiene un sistema asíncrono.
- **Población.-** Siguiendo WYSIWIS todo el grupo ve la misma información. A veces puede ser útil permitir que los subgrupos se comuniquen entre ellos sin necesidad de que el grupo entero vea toda esta información.
- **Compatibilidad**.- Bajo WYSIWIS todo el mundo ve la misma representación de los datos compartidos. Relajando esta restricción podemos proveer a cada colaborador con diferentes representaciones, quizás dependiendo del rol en la colaboración.

Sin embargo la gran ventaja que se obtiene al seguir WYSIWIS de forma estricta es que todos los colaboradores tienen la garantía de que están visualizando lo mismo (véase figura 13). Por lo que se plantea el problema de cómo se pueden relajar las restricciones de WYSIWIS de manera que los colaboradores no pierdan la noción de que están trabajando juntos. A continuación se discutirá las condiciones sobre la relajación de WYSIWIS y el tipo de información que debería ser provista para recuperar la ventaja obtenida por WYSIWIS de forma estricta.

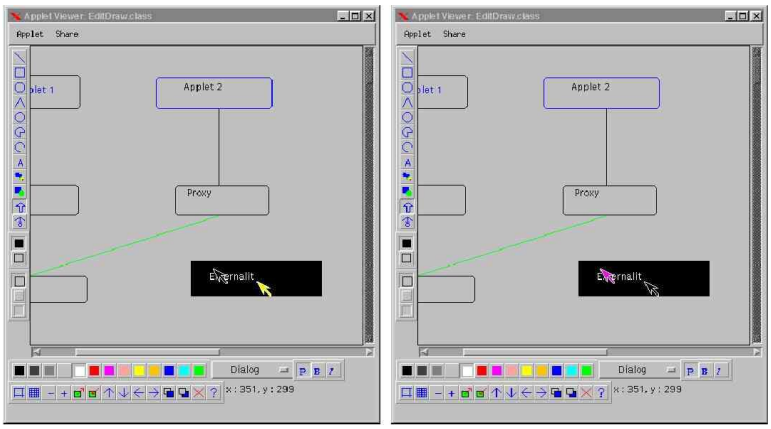

Figura 13. Esquema de la relación entre WYSIWIS

What You See Is What I Think You See (WYSIWITYS)

En interacciones cara a cara es fácil saber lo que realizan cada uno de los componentes del grupo. Sin embargo, cuando los colaboradores están distribuidos mucha de esta información se pierde. Sin esta información, los grupos tienen dificultad para coordinar el control de acceso, la comunicación y el trabajo.

WYSIWIS de forma estricta es útil ya que todos los colaboradores saben lo que está viendo el resto. Sin embargo en el mundo real hay colaboraciones sin seguir WYSIWIS, como puede ser una reunión, en las que se puede conseguir una percepción de lo que cada uno esta viendo. Para contemplar este tipo de colaboración, Randall Smith introdujo el concepto de **What You See Is What I Think You See (WYSIWITYS)** [Smi92]**.**

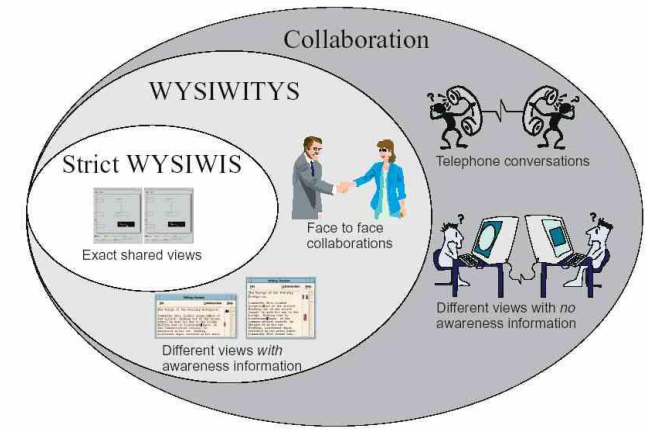

Figura 14. Esquema de la relación entre WYSIWITYS

El principio WYSIWITYS requiere que cada colaborador sea capaz de construir un modelo de la experiencia de otro colaborador. WYSIWIS de forma estricta sigue este principio y es por tanto parte de WYSIWITYS. WISIWITYS provee una directiva útil de cómo relajar WYSIWIS y mantener la percepción de colaboración. Mediante el suministro de información que logra WYSIWITYS, los colaboradores tienen garantizado que los datos siguen manteniendo el principio WYSIWIS.

Group Awareness (Percepción del grupo)

Greenberg, Gutwin, Roseman et al de la Universidad de Calgary [Gut96] describen los siguientes aspectos que deben proporcionarse a la hora de mantener una percepción fiel de la actividad que se está desarrollando en el grupo por cada colaborador:

- **Presence awareness (Percepción de asistencia).** Se refiere al conocimiento general de los usuarios del grupo que están conectados y la localización de los mismos.
- **Social awareness (Percepción del interés).-** Es el conocimiento social de los miembros del grupo, incluyendo aspectos como el interés que muestran los otros colaboradores, la atención prestada y el estado emocional de éstos.
- **Structural awareness (Percepción de jerarquía).-** Es el conocimiento de los roles de los colaboradores, posición en sus tareas, tareas y progreso dentro del grupo.
- **WorkSpace awareness (Percepción del área de trabajo).-** Es definido como el conocimiento instantáneo de las interacciones y/o modificaciones de otros participantes dentro del área de trabajo.

En un entorno colaborativo uno de los aspectos fundamentales a tener en cuenta es el llamado **"workspace awareness"**. Debido a que el individuo interacciona con el ordenador y puede que no tenga contacto cara a cara con las personas con las que colabora, es necesario proporcionar información para mostrarle al individuo que no se encuentra solo en la tarea a desarrollar y dónde y cómo puede interaccionar con el resto de los colaboradores.

La tabla 2, muestra los elementos más relevantes y las preguntas que podrían realizarse los colaboradores con respecto de cada uno de los elementos, dentro del workspace awareness.

Elementos	Cuestiones relacionadas
Identidad	¿Quién está participando en la actividad?
Localización	¿Dónde están?
Nivel de Actividad	¿Están los colaboradores en el workspace? ¿Con qué rapidez trabajan?
Acciones	¿Qué están haciendo? ¿Cuales son sus actuales tareas y actividades?
Intenciones	¿Qué es lo que van a hacer los colaboradores? ¿Dónde quieren estar?
Cambios	¿Qué cambios están haciendo los colaboradores? ¿Dónde se han hecho los cambios?
Objetos	¿Qué objetos están usando?

Alcance	¿Qué es lo que pueden ver?
Habilidades	¿Qué pueden hacer?
Esfera de influencia	¿Dónde pueden tener efecto?
Expectativa	¿Qué es lo próximo que necesitan de mí?

Tabla 2. Elementos de workspace awareness.

Otro aspecto de la percepción del área de trabajo es el llamado *actividad social*, descrito por Ackerman y Starr [Ack96] como la acumulación de trabajo individual. La información de la actividad social es usada por los grupos de varias formas:

- Primeramente, esto ayuda a los individuos a enfocar su propia actividad.
- Segundo, el saber si otros están trabajando o no puede tener un efecto profundo en la motivación individual.
- Finalmente, el éxito de sistemas groupware depende de mantener un cierto umbral de actividad social.

Proceso de Diseño en GroupWare.

A la hora de abordar el diseño de un sistema GroupWare es interesante comenzar teniendo una perspectiva de los usuarios potenciales, qué objetivos quieren lograr, y cómo realizarán su trabajo. Para aplicaciones groupware como pueden ser e-mail o videoconferencias, tener una perspectiva de los usuarios puede resumirse en saber cómo se va a llevar a cabo la comunicación. Sin embargo, en otras tareas colaborativas como puede ser un sistema educacional, es interesante tener en cuenta que número de usuarios van a participar en esta colaboración para poder hacer el diseño más adecuado para este grupo.

Sin embargo, obtener una perspectiva de los usuarios que van a colaborar en el sistema multiusuario resulta más difícil que en el caso de sistema mono usuario puesto que:

- Organizar y programar grupos es más difícil que hacerlo con individuos.
- Es difícil obtener la forma de interacción del grupo de antemano, mientras que es posible determinar un estudio previo de las características individuales.
- El establecimiento de grupos depende de la forma de interacción, y el tiempo que ha estado constituido el grupo afecta a los patrones de comunicación.
- Los grupos cambian rápidamente en el proceso de formación.
- Los grupos son dinámicos produciéndose cambios de roles.
- Modificar prototipos puede ser técnicamente difícil debido a la complejidad añadida por los grupos.
- En software para grandes empresas, testear nuevos prototipos puede ser difícil o imposible debido a la alteración causada por la introducción de nuevas versiones en la organización.

A la hora de diseñar aplicaciones groupware es mejor empezar con estudios con el objetivo de obtener una perspectiva de un tipo particular de grupo u organización que puede ser usado en el sistema groupware. Varios tipos de estudios pueden ser realizados: entrevistas, encuestas, análisis de los aparatos usados en el proceso de trabajo, examinar los procesos, etc. En todos los casos, el objetivo es identificar las

tareas de los usuarios y sus metas, entender cómo se comunica el grupo y qué tipo de estructuras y roles tienen lugar.

2.3. Modelo de distribución de la información.

Una de las partes principales a analizar en el desarrollo de una aplicación es elegir el tipo de middleware más apropiado para la construcción de la herramienta colaborativa y qué modelo de distribución de datos se va a seguir.

En cuanto al modelo de distribución de datos según Rapaport [Rap91] se pueden distinguir los tipos que se muestran a continuación.

Modelo centralizado. En este modelo la información asociada con el proceso del grupo es almacenada y gestionada en una localización central. Este modelo no requiere información privada de la información almacenada. Todos los miembros leen la misma información de la misma localización, con lo cual no hay diferentes copias. Sin embargo este modelo tiene como desventaja el cuello de botella que supone que todos los usuarios consulten en la misma localización para obtener los datos. Otra desventaja de este modelo es que si cae el sitio central dónde la información se está guardando, la colaboración no se puede llevar a cabo.

Modelo distribuido no replicado. En este modelo se basa en usar diferentes localizaciones para almacenar diferentes procesos de grupo. De tal forma que existen diferentes objetos con información única en cada una de estas localizaciones, que juntos proveen la información global de la colaboración. Cada cambio o modificación se hará sobre el objeto responsable de cierta tarea. Este modelo tiene la ventaja de que si cae un nodo, se pierde parte de la información con la que se esta colaborando pero no toda, como pasaba en el modelo centralizado. Otra de las ventajas de este modelo es que se distribuye la carga, de tal forma que todas las modificaciones y consultas no se hacen sobre el mismo objeto.

Modelo replicado. En este modelo cada proceso cliente tiene una réplica de los datos que se están procesando en el grupo. De tal manera que cualquier cambio es enviado a todos los miembros del grupo para que puedan mantener una vista consistente de los datos. Principalmente este modelo tiene dos ventajas, todos los clientes tienen una copia de los datos, por lo que ante la caída de uno de ellos los demás usuarios pueden colaborar y la otra ventaja es que los tiempos de respuesta son más cortos.

El modelo de distribución de datos dependerá bastante de la aplicación a construir. Con lo que no se puede establecer un modelo idóneo para la construcción de una aplicación. Si el modelo de comunicación es mucho a muchos, como hemos visto anteriormente el mejor modelo de comunicación es el de propagación de eventos y por tanto el mejor modelo para la distribución de datos es el modelo replicado.

A pesar de que el modelo replicado ofrece muchas ventajas en cuanto a la rapidez de comunicación de cambios así como la consistencia ante fallos, también implica el estudio y el tratamiento de nuevos problemas, como mantener la consistencia de los

datos, establecer mecanismo de *Floor Control* (control de turnos) y analizar el problema de la llegada tarde de usuarios en la sesión.

Más concretamente, esté será el modelo elegido a la hora de desarrollar las herramientas colaborativas síncronas en el marco de aplicación de nuestro trabajo de investigación, puesto que ofrece grandes ventajas ante este tipo de aplicaciones que requieren bastante rapidez en la propagación de los cambios. En cuanto al middleware apropiado para la construcción de este tipo de herramientas que requieren la comunicación muchos a muchos, ya hemos visto en el apartado anterior que el más adecuado es el de comunicación asíncrona. En el desarrollo de aplicaciones distribuidas, como indica Munindar R. Singh [Sin99] *" las interaciones en la manera que son conducidas deben tener la expresividad de sincronía, mientras que el diseño de la interacción deben tener la reusabilidad de la asincronía"*.

2.4. Elementos de los entornos colaborativos: Floor Control.

Dommel y Garcia-Luna-Aceves [Dom97] definen un entorno colaborativo C como una cuádrupla:

$$C = (S, U, R, F)$$

de un conjunto de sesiones $S = U_i S_i$, un conjunto de usuarios $U = U_j U_j$, un conjunto de recursos $R = U_k R_k$ y un conjunto de floors, $F = U_l F_l$, dónde i, j, k y l son números naturales. Un floor esta asociado con un recurso y es garantizado a usuarios que trabajan con el recurso. Las sesiones pueden estar agregadas en supersesiones y de una manera recursiva forman sub-sesiones dependiendo de un grupo dinámico de sesiones de comienzo. Esta reformación de sesiones online puede ser permanente o temporal, y los atributos de las sesiones son heredadas desde las supersesiones a sus hijos, pero pueden ser sobrescritos. Este concepto de sesiones jerárquicas refleja la organización natural de reuniones cara a cara.

Muy similar a la noción dinámica de sesiones, los usuarios también forman grupos estáticos con estructuras que pueden servir como formularios para la pertenencia a sesiones. De una manera adicional, invitados temporales o futuros miembros pueden unirse a sesiones abiertas, mientras que en sesiones cerradas sólo podrían unirse mediante una invitación. Sesiones con intereses comunes o diferentes puede o bien unirse o bien separarse. En sesiones específicas, cada usuario puede tener una configuración inicial en las herramientas y restricciones o posibilidades de manejar una herramienta de acuerdo con el propósito de la sesión y el régimen de *Floor Control* que se quiera imponer. Los usuarios que son miembros de diferentes grupos pueden también participar en múltiples sesiones, estando activo en una de ellas y en la demás ocioso.

Las **sesiones** están caracterizadas por su tamaño (en términos de número de personas), su estructura (plana contra jerárquica), miembros dinámicos (crear, invitar, unir, pausar, intercambiar y abandonar), propósito (conferencia, reunión, panel, evaluación, entrevista, practica, etc) duración, agenda, organización (informal, formal), procedimiento de decisión (consenso, votación, guiada) y el ámbito de distribución (local, área amplia, global). Cada una de las combinaciones de estas características crea problemas relacionados con la conversación y la colaboración.

Los **usuarios** están caracterizados por sus agregaciones (individuos, grupos), rol social (coordinador, hablador, auditor, tomador de notas, etc), identidad (anónimo, conocido), autoridad (plena, restringida y privilegiada), su tipo de entrada (solo, consenso del grupo, grabado), los recursos o medios en uso, y finalmente el enlace usado por la colaboración (solo enviar, solo recibir, o bi-direccional). Los roles sociales de los usuarios de una sesión reflejan la orientación del grupo y dirigen el propósito de la sesión, imponiendo reglas de interacción formal o informal en sus participantes. Los roles pueden se definidos antes del comienzo de la sesión o diseñado en tiempo de ejecución.

Los **recursos** son objetos con diferentes clases de aplicaciones. El tipo de recurso caracteriza si es de tipo texto, gráfico, o con necesidades de tiempo real e identifica el propósito para el cual sirve. Los recursos pueden ser virtuales, o pueden representar dispositivos remotos. Además los recursos no necesitan ser propietarios de la sesión dónde son accedidos, pero podrían ser hospedados desde una máquina fuera de la sesión. Un atributo de seguridad indicará si son públicos, privados o supervisados.

Floor Control

La manipulación concurrente de la información compartida es una parte integral de todos los procesos de grupo. Así pues, las herramientas de trabajo colaborativo necesitan algún tipo de control de concurrencia con el objetivo de mantener la consistencia de la información.

En el contexto de las bases de datos ya hemos explorado en que consiste el control de la concurrencia y el mantenimiento de las premisas ACID. Sin embargo, en el contexto de las herramientas de trabajo colaborativo este control de concurrencia deber ser redefinido. Más concretamente se conoce con el nombre de *Floor Control*, y en esta sección analizaremos en que consiste, los diferentes mecanismos que presenta y las diferentes políticas que pueden establecerse.

El *Floor Control* es el mecanismo que permite la compartimiento de recursos y su posterior uso sin que se produzcan conflictos en los accesos. Se aplica sobre todo en el caso de actividades colaborativas síncronas e interactivas. Según Dommel y Garcia-Luna-Aceves [Dom97], el uso de *Floor Control* permite a los usuarios de herramientas multimedia utilizar y compartir recursos sin conflictos en los accesos.

Los sistemas colaborativos llegan a ser más polimórficos con respecto a la mezcla de recursos que pueden ser usados al mismo tiempo. La adaptación de protocolos de *Floor Control* con respecto a los tipos de recursos necesita ser evaluada según medidas de rendimiento, las cuales tienen en cuenta restricciones de QoS.

Un ejemplo de escenario en un entorno colaborativo podría ser el que se muestra en la figura 15.

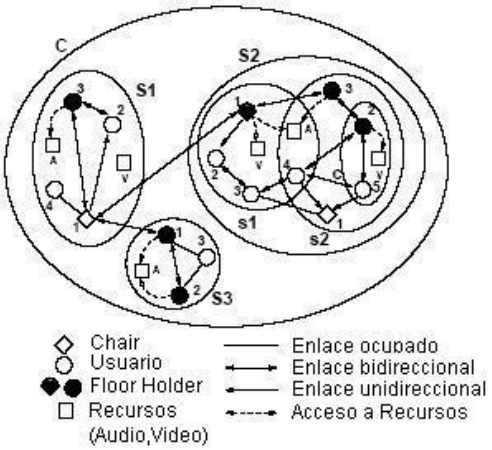

Figura 15. Ejemplo de colaboración entre diferentes sesiones.

Como se puede observar, todas las sesiones excepto la S3 están guiadas por un chair. Dichos roles pueden ser establecidos a priori o de manera online con el consenso de todos los usuarios que participan. La intersección de S1 y S2 muestra la compartimiento de recursos a través de los límites de las sesiones de tal manera que los usuarios pueden formar parte de ambas sesiones al mismo tiempo.

Una taxonomía de atributos para las sesiones, usuarios, floors y recursos ayuda en la selección de una estrategia de control apropiada para facilitar la colaboración.

Más concretamente, los floors pueden ser caracterizados por su estado (held (poseído), idle (ocioso), requested (solicitado), free (libre)), su proceso de garantizado y liberado (explícito por un coordinador, implícito por límites de tiempo, etc), su funcionamiento (operacional, realimentación), su política de asignación– (encolado, no encolado), la granularidad (aplicado a la aplicación entera, a ventanas o a aspectos gráficos, etc) y permisos (leer, escribir, ejecución, transmitir, anotar, mover, etc).

Las sesiones comprenden una gran variedad de usuarios compartiendo (posiblemente en sesiones especificas) recursos e información multimedia, todo controlado mediante floor en acceso y uso. Un floor está asociado con un recurso o información (que es compartido/a por varios usuarios) y se concede a los usuarios que trabajan con dicho recurso o con dicha información.

Entre los conceptos que se pueden manejar referentes al floor cabe destacar los siguientes:

Floor owner. Es quien crea el floor y proporciona la información o el recurso compartido con el que está asociado.

Floor holder. Es quien posee el floor en un momento dado.

Floor arbiter. Es el usuario que se encarga de controlar los floors para un determinado recurso, como un super-usuario del floor.

Chair. Es el usuario que hace posible la sesión.

Entre las tareas que se realizan desde el *Floor Control* podemos destacar las siguientes:

- Proporcionar exclusión mutua sobre objetos compartidos en accesos concurrentes.
- Conceder o denegar determinadas peticiones de acuerdo a las políticas establecidas.
- Hacer llegar el estado de todos los recursos compartidos (floors) a todos los sitios conectados.
- Transmitir peticiones entre diferentes sitios.
- Manejar derechos de acceso a datos temporales.
- Autorizar o denegar el uso de acuerdo a la información de la sesión.
- Enviar los cambios de estado del floor a los miembros colaboradores.

El diseño del *Floor Control* conlleva tener en cuenta características del sistema y del interfaz del usuario. En cuanto a las características del sistema esto conlleva consideraciones arquitectónicas, estructura del paquete de control, difusión y mantenimiento de la información de control, disciplinas y manejo de excepciones. En cuanto al diseño de la interfaz de usuario conlleva representación, "triggering", servicio y manipulación del control con respecto a la progresión en el tiempo en la tarea colaborativa. Mientras que las consideraciones del sistema están más relacionadas con la implementación persistente de los "mecanismos de *floor control*", las perspectivas del usuario están más relacionadas por las "políticas de *Floor Control*".

Mecanismos de Floor Control

Mecanismos de Floor Control: "regulan uniformemente el flujo de control y los eventos de sincronización de los distintos sitios". Se distinguen los siguientes mecanismos de *Floor Control*:

Negociation. Se construye sobre la política "free for all", en principio todos los recursos están disponibles para todos los usuarios y luego habrá que establecer algún tipo de negociación a la hora de dar servicio a las peticiones. Realmente es un esquema "no floor".

Token passing. El floor va pasando de un miembro a otro de acuerdo a alguna topología establecida.

Token asking. Los usuarios pueden solicitar el floor según lo vayan requiriendo, en lugar de que éste vaya pasando de un usuario a otro como en el caso del mecanismo anterior.

Time stamping. Es un mecanismo que permite ordenar los eventos en base a un mecanismo global de reloj sincronizado o un número de secuencia. Se suele utilizar junto con otros mecanismos para una mejor exactitud.

Two-phases locking. Permite establecer bloqueos sobre documentos compartidos en una primera fase y liberarlos cuando se complete la actividad para la que fueron bloqueados en una segunda fase.

Blocking. Los bloqueos se llevan a cabo a través de semáforos que gestionan las secciones críticas estableciendo un orden FCFS (first-come first-served) para atender las peticiones.

Activity sensing. Las actividades recibidas en el canal dan lugar a rechazar las peticiones siguientes hasta que la actividad haya terminado. Después de esto, las nuevas peticiones deben ser reeditadas.

Reservation. Consiste en localizar determinados recursos en predeterminadas secuencias, duraciones o ranuras de tiempo.

Dependency detection. Trata de reordenar las peticiones y actividades para evitar los conflictos.

En definitiva podríamos clasificar dichos mecanismo dependiendo de la necesidad de mantener una cola o no, de requerir el floor por adelantado o no y dependiente o no de la tarea.

En definitiva el mecanismo de *Floor Control* seleccionado dependerá de los requerimientos de las aplicaciones dónde se vaya a aplicar. Para interacciones entre usuarios, grandes retardos creados por la espera en la cola o por tiempos grandes en el paso del floor deben de ser evitados ya que el flujo de trabajo podría verse parado.

Políticas de floor control

Políticas de Floor Control "determinan como pueden ser solicitados y concedidos los floors, permitiendo nuevas solicitudes pre-emption, y cambios en los servicios subscritos. Estos autores distinguen entre las siguientes políticas de control:

Free for all. Todos los usuarios pueden acceder a los recursos compartidos en cualquier momento.

Chair-guidance. Uno de los usuarios se elige como árbitro del uso de un floor específico. Para cada floor existe la posibilidad de que los roles de chair y de árbitro sean desempeñados por la misma persona o por dos personas distintas.

Agenda-orientation. La tarea de asignación del floor corresponde a un determinado cronograma (o agenda) definido antes de que la sesión empiece o creado "en tiempo de ejecución", dependiendo de la finalidad de la sesión.

Time-orientation. Las peticiones al floor tienen timeouts definidos a través de eventos o condiciones del sistema. Cada titular del floor puede tener restricciones de time-out individuales.

Predefined ordering. Los floor son ofrecidos o solicitados siguiendo una secuencia establecida de acuerdo a la topología de los sitios colaborativos que participan (p.e. paso del token utilizando round-robin, FCFS, ...)

Ad hoc reordering. Almacena las peticiones para cada recurso en una cola y las va atendiendo de acuerdo a criterios de puntualidad, prioridad y QoS.

Election. Permite que lo usuarios voten cual será el próximo titular del floor o la aceptabilidad de una operación en un determinado sitio. Si hay consenso se acepta y si no se realiza un nuevo proceso de elección.

Lottery scheduling. Se usan tickets de lotería, como en un mecanismo de probabilidad.

De una manera más general podríamos clasificar las políticas según la siguiente clasificación:

- **Monarquía**. Hay un chair (coordinador) que controla la sesión, el cual puede estar definido a priori o bien puede ser elegido por votación consensuada de todos los miembros.

- **Democracia**. Hay un conjunto de leaders privilegiados.

- **Anarquía**. No hay ningún tipo de control.

Otra clasificación de *Floor Control* más general puede ser la siguiente [Bor00]:

- **Optimista.** Es aquel en que todos pueden modificar los datos en cualquier momento. Este tipo de política no garantiza que los datos sean consistentes en todo momento.
- **Pesimista.** Intenta evitar que se produzcan conflictos. Muchas aplicaciones necesitan un control riguroso acerca del acceso a los datos.

El tipo de política y mecanismo a elegir en el *Floor Control* dependerá de la rigurosidad con qué deba tratar la aplicación con el acceso a recursos.

Consideraciones en la Implementación.

Para indicar que tipo de *Floor Control* se puede hacer de dos maneras: declarativamente, a través de reglas declarativas, o mediante programación, implementando nuestro propio gestor de *Floor Control*.

Li and Muntz [Li98] se declina por la primera opción es su articulo COCA, mientras que Dommel y Garcia-Luna-Aceves usan el tipo segundo [Dom97].

Floor Control es un concepto a nivel de aplicación que se hace tangible mediante el interfaz de usuario. El uso de ciertos elementos gráficos como iconos y colores hacen más fácil entender el estado en el que se encuentra el floor de determinado recurso.

Las interacciones suceden a través de diferentes estados a través del tiempo que afectan a la generación, reserva y paso del floor.

Iniciación: Los floors son creados en el inicio de la sesión o al unirse a la ejecución de una sesión. Para sesiones persistentes, los floors están siempre activos. Para sesiones temporales los floors se van creando conforme se usa el recurso.

Conducción: Durante la sesión, el estado del floor es mantenido en una tabla para todo los movimientos locales y remotos que afectan al proceso de trabajo y a la consistencia de los datos. Las peticiones e información de un floor son enviadas a todos los sitios que usan el recurso.

Terminación y excepciones: Dos casos diferentes de pérdida de floor son aquellos ocasionados por retiradas voluntarias o involuntarias. La primera de ellas puede ser causada por la expulsión del usuario o bien por el abandono de éste o bien por la terminación de la sesión. La pérdida involuntaria es debida a fallos en los enlaces o de las entidades que colaboran. El estado del floor debe ser mantenido durante una sesión por si un miembro abandona temporalmente o una sesión dañada se recupera. Si una sesión está coordinada por un chair y este cae, otro miembro debe ser elegido como el árbitro de la sesión. De igual manera, si el chair deja la sesión, se deben asegurar que todas las actividades inacabadas supervisadas por este árbitro se concluyen. Si el floor owner cae, los datos originados desde este sitio también desaparecen de la sesión y dicha situación debe ser notificada. Si el *floor holder* cae, el floor debe ser reasignado a un sucesor.

2.5. Llegada tarde.

En la colaboración síncrona uno de los problemas que es necesario solucionar es el problema de la "Llegada tarde", en el cual se le de soporte a aquellos usuarios que llegan tarde a la sesión y que desean participar en ella. Un usuario que llega tarde necesita ser inicializado con el estado compartido actual antes de que el usuario sea capaz de participar en la sesión. Uno de los problemas a resolver en la llegada tarde es que hacer cuando el estado de la sesión es grande y complejo.

Los algoritmos de llegada tarde a nivel de aplicación pueden ser divididos en dos tipos centralizados o distribuidos.

Los algoritmos de llegada tarde centralizados requieren de una aplicación que sea capaz de actuar como un servidor de llegada tarde para todo estado que está compartido. Una instancia de aplicación que llega tarde contactaría con el servidor de estado el cual le devolverá la información relevante del estado.

La ventaja principal de un enfoque centralizado es la simplicidad con respecto al manejo de la consistencia. Al mismo tiempo un servidor centralizado conlleva las principales desventajas de un sistema centralizado, puesto que la caída de este servidor indicaría la imposibilidad de continuar manejando el problema de la llegada tarde.

En el enfoque distribuido el rol de servidor de estado o parte de él es asignado a un subconjunto de las aplicaciones en la sesión. De esta forma se reduce el problema del cuello de botella de mantener un solo servidor.

Según Jürgen Vogel, Martin Maule et al [Vog03] para que un algoritmo de llegada tarde sea eficiente debe usar el conocimiento de la aplicación. La replicación de todos los paquetes es una solución no aceptable según ellos. Más concretamente, exponen una

solución totalmente distribuida para el problema de llegada tarde la cual se expondrá a continuación.

Su propuesta se basa en un conjunto de servidores de estado, algunos con todo el estado y otros con parte del estado de la sesión. Proponen un algoritmo basado en el algoritmo "exponential feedback raise" para la selección del responsable de enviar el evento del estado: cada miembro de la sesión que esté interesado en enviar el estado de la sesión cogerá un numero aleatorio x $\in$ [0;1). Si $x < 1/N$, dónde N es el tamaño del grupo, la instancia de la aplicación es seleccionada y actúa inmediatamente. Si $x \geq 1/N$ un timer es establecido con el valor $t = T(1+ \log_N x)$, dónde T es la latencia máxima deseada hasta que al menos un miembro sea elegido. Si el timer del usuario expira, habrá sido seleccionado y transmitirá el estado o la petición de estado al grupo. Todas las aplicaciones que reciban esta información pararán sus timers.

Existen muchas aproximaciones para resolver la elección del servidor de estado así como la transferencia de los datos desde este servidor de estado al nuevo usuario. Una propuesta anterior [Gey00] indica que una vez elegido el servidor de *late join* mediante consenso, la transferencia de datos se realizará entre el cliente que la solicitó y el servidor de estado, sin que ningún otro colaborador pueda escuchar esta información a través del canal.

En nuestro trabajo de investigación hemos resuelto este problema de muchas formas, todas ellas basadas en mecanismo de coordinación para la selección del servidor de estado. Posteriormente veremos como las hemos llevado a cabo.

En cuanto a las políticas del envío del estado cuando un usuario llega tarde Jürgen, Martin [Vog03] proponen las siguientes:

> **No late join:** Esta política es elegida por la aplicación para indicar que no está interesada en ciertos componentes. Mediante la selección de esta política se reduce considerablemente la cantidad de información requerida en la inicialización del usuario que llega tarde.
>
> **Immediate late join:** Esta política es elegida cuando se requiere cierta información de la sesión para participar en la tarea colaborativa, es decir cuando se requiere el estado de algunos componentes para colaborar. El estado de estos componentes será enviado inmediatamente cuando sean requeridos.
>
> **Event-triggered late join:** Para algunas aplicaciones podría ser razonable sólo requerir componentes cuando estos sean el propósito de un evento. Junto al aplazamiento de la petición del estado hasta que el realmente necesitado, está política también incrementa significantemente la probabilidad de que múltiples usuarios que llegan tarde a la sesión se beneficien de una transmisión de estado (en el caso de que se use multicast para las transmisión).
>
> **Network-capacity-oriented late join:** Una aplicación puede elegir esta política cuando los estados de los subcomponentes no sean requeridos inmediatamente. Con esta política, el servicio de llegada tarde monitoriza la entrada y salida del tráfico de red de la instancia. Sólo si el trafico de red esta por debajo de un cierto umbral establecido por la aplicación se pedirá el estado del subcomponentes.

Esta política intenta retrasar la transmisión del estado hasta que la carga de la red sea baja.

Muchas son las políticas que se podrían tener en cuenta a la hora de implementar el algoritmo de llegada tarde. Sin embargo hay que tener en cuenta que los retardos en la transmisión del estado incrementan la probabilidad de que el último participante que posea los datos abandone la sesión. Por lo cual, en la elección de la política de llegada tarde habrán dos factores principales necesarios a tener en cuenta, uno es el número de usuarios de la tarea colaborativa y el otro es la importancia de recibir cuanto antes el estado de la herramienta para que la colaboración se pueda llevar a cabo.

Como podemos observar esta política a la hora de tratar el *late join* presenta ventajas como la eficiencia en la elección del encargado de enviar el estado así como la tolerancia a fallos.

2.6. Interfaz gráfica.

Uno de los aspectos importantes en el desarrollo de herramientas colaborativas mediante el uso del ordenador es la inclusión de elementos gráficos. Más concretamente el estudio del interfaz de usuario es una parte de la investigación dentro del campo de HCI (Human Computer Interaction) la cual es una disciplina que abarca el diseño, evaluación e implementación de la computación interactiva de sistemas para el uso humano así como el estudio de los sucesos que ocurren a su alrededor.

En el desarrollo del conjunto de aplicaciones hay que tener en cuenta que la interfaz es la única parte del sistema que es visible a los usuarios finales [Kli01] con lo cual es necesario asegurar que la interfaz transmite los aspectos más relevantes para la perfecta comunicación del grupo, así como una distribución ordenada de los datos como las características que hemos visto anteriormente en la percepción del grupo, así como constituyen una parte esencial en la implementación de *Floor Control* y el entendimiento de la información que se comunica en el sistema.

El estudio de la interfaz gráfica conlleva cómo se presenta la información al usuario final (por ejemplo los layout) así como los iconos que se usan para representar ciertas características.

Una de las formas para mostrar gráficamente los usuarios que están conectados así como poder apreciar la tarea que están realizando en un determinado momento es el uso de telepunteros [Beg98]. Un telepuntero indica la participación de un colaborador remoto dentro del área de trabajo (Véase figura 16). Normalmente cada usuario tiene un color. Diferentes formas y nombres pueden ser usados para identificar a los usuarios.

Figura 16. Telepunteros para los usuario Laura y Bo .

Entornos avanzados gráficamente hacen uso de las tecnologías 3D y el concepto de mundos virtuales. MUD (Multi-User Dimensions) y MOO (Mud Object Oriented) son tecnologías que permiten a múltiples usuarios acceder a la base de datos compartida

simultáneamente. La base de datos MOO representa mediante objetos "Habitación" (*Room*) conectados entre si por objetos "Puerta"(*Portal*). Los usuarios de suelen denominar "Avatars" y en las Habitaciones podemos encontrar objetos "Cosa"(*Thing*).

2.7. Uso del Groupware a través de la Web.

A la hora de diseñar una aplicación es fundamental tener en cuenta si va a distribuirse en forma de aplicación o como una aplicación a través del Web.

El uso de aplicaciones cliente separadas conlleva la instalación del programa en la máquina. Cualquier cambio de la interfaz o de la versión de la aplicación implica la instalación del nuevo sistema. Además la posibilidad de que diferentes colaboradores tengan distintas versiones de la aplicación puede implicar problemas a la hora de la colaboración, por ejemplo compatibilidad de versiones en los objetos a distribuir etc.

Otro de los inconvenientes que se presentan en el uso de aplicaciones es que se debe saber de antemano a que servidor debe de conectarse para llevar a cabo la tarea colaborativa. El cambio de dichos servidores implica el cambio en la configuración de las herramientas cliente.

Sin embargo todos estos problemas pueden ser solventados mediante el uso de sistemas a través del Web. En los desarrollos de los primeros sistemas a través del Web surgían una serie de limitaciones en cuanto a la funcionalidad y a la lentitud de ejecución. Sin embargo el uso de Java ayuda a la eficiencia de ejecución de las aplicaciones puesto que el código es transmitido desde el servidor hasta el cliente, dónde se ejecuta. El uso de las ultimas tecnologías como JavaWebStart, permite mantener en caché la aplicación y hacer un control de versiones, de tal forma que si el cliente ya tiene descargada la última versión disponible no es necesario realizar la transferencia de la aplicación.

Aunque el uso a través del Web ofrece un conjunto de ventajas considerable, dependiendo del uso que se vaya a hacer de la aplicación puede que convenga distribuirla en forma de aplicación más que a través del Web.

2.8. APIs para el desarrollo de herramientas GroupWare.

Existen una gran variedad de Apis que facilitan el desarrollo de aplicaciones colaborativas. En esta sección analizaremos brevemente algunas de ellas.

MMConf

En 1990, Crowley, Bolt, Beranek y Newman [Cro90] escribieron acerca del desarrollo de una herramienta basada en X-Windows y escrita en C llamada MMConf. Esta herramienta usa una arquitectura replicada, permite telepunteros que pueden ser usados por un usuario cada vez y también permite concurrencia usando *Floor Control*. Otra de las características que provee esta herramienta es que puede compartir ficheros.

Suite

Dewan y Choudhary de la Universidad de Purdue [Dew91] desarrollaron la herramienta Suite. Suite usa una arquitectura replicada y define una serie de primitivas en C para

modificar las replicas.Siguiendo una programación con Suite el desarrollador no tiene que preocuparse si la aplicación será usada posteriormente por una persona o por varias.

Rendezvous

Rendezvous es un herramienta groupware y lenguaje desarrollado por Hill et al. Bellcore [Hil94]. Rendezvous corre bajo plataformas X y está implementado en Common Lisp. Rendezvous es la única herramienta en la que se usa una arquitectura semi-replicada, pero todos los procesos replicados permanecen en el mismo hosts. Por lo que las aplicaciones en Rendezvous son conceptualmente replicadas pero físicamente centralizada.

Una de las contribuciones de Rendezvous es la facilidad de uso de la abstracción de comunicación de su paradigma de programación, llamado Abstraction-Link-View (ALV), el cual separa los datos de una aplicación de su interfaz. Una instancia de una abstracción contiene los datos que serán compartidos entre todos los colaboradores. Vistas replicadas visualizan los datos compartidos. Cada vista está asociada con una abstracción mediante un enlace, el cual asegura que los cambios realizados en la abstracción son reflejados en las vistas correspondientes a los enlaces. A la inversa, cuando un usuario cambia una vista, el enlace unido a esta vista actualiza la abstracción. Así pues los desarrolladores que usan Rendezvous no necesitan propagar explícitamente los cambios.

GroupKit

GroupKit es una herramienta desarrollada por Greenberg, Roseman et al en la Universidad de Calgary [Gro].

GroupKit y las aplicaciones desarrolladas con él usan Tcl/Tk, el cual puede ejecutarse en múltiples plataformas incluyendo Microsoft Windows, X y Macintosh. Así pues, las aplicaciones desarrolladas mediante GroupKit tienen la característica de multi-plataforma al igual que las aplicaciones desarrolladas con los toolkit basados en Java.

Las aplicaciones en GroupKit son enteramente replicadas. Solamente el registro (“registar”), el cual mantiene la lista de sesiones activas, es centralizado. El registro provee habilidades para la gestión de sesiones. GroupKit ofrece tres abstracciones de programación para la comunicación entre replicas. Uno, llamado multicast remote procedure call, permite a una réplica invocar a procedimientos en replicas remotas. Otra abstracción de comunicación es provista a través del uso de entornos compartidos. Todas las replicas ven cualquier cambio hecho en el entorno compartido. Finalmente, GroupKit proveen mecanismos para multicast y recepción de mensajes.

GroupKit ha sido pionero en el uso de interfaces multiusuario. Así pues, GroupKit incluye muchos componentes de interfaz de multiusuario, como son telepointers o multiuser radar.

Java Shared Data Toolkit

La herramienta Java Shared Data Toolkit [Bur99] ha sido desarrollada por Rich Burridge, que fue publicada por JavaSoft en el año 1998, aunque ya estaba disponible en el año 1997.

JSDT ha sido definido para soportar aplicaciones colaborativas y altamente interactivas. Provee una API sencilla para la construcción de herramientas colaborativas basadas en el manejo de sesiones y canales.

Se basa en un modelo Productor/Consumidor para el manejo de los eventos. Para el envío de datos e información se ofrece en la clase *Channel* un conjunto de métodos *sendXXX* a través de los cuales se envian datos al canal o a un usuario específico. Mediante la clase *ChannelConsumer* se ofrece el método dataReceived a través del cual las aplicaciones pueden recibir los eventos de datos que se producen dentro del canal. Con el fin de escuchar una serie de eventos se ofrecen una serie de listeners, bien sean en la sesión *SessionListener* o bien sea en el canal *ChannelListener*.

Además, esta API provee soporte eficiente de comunicaciones con mensajes multicast, la habilidad de asegurar el reparto de mensajes secuenciados uniformemente y un mecanismo de sincronización distribuida basado en *Tokens*. Además provee la habilidad de compartir arrays de bytes entre los miembros de una sesión.

Uno de los objetivos de JSDT es proporcionar una interfaz común para comunicaciones generales entre varias partes, escondiendo la implementación y protocolos subyacentes. JSDT provee una API genérica, que puede trabajar con sockets, RMI, HTTP o multicast usando LRMP (Lightweight Reliable Multicast Protocol) de forma transparente para el usuario.

Figura 17. Arquitectura ofrecida por JSDT v1.5.

El uso de esta API no garantiza la entrega fiable de los mensajes que se distribuyen entre las aplicaciones. Pese a eso, esta API ha sido usada en la investigación previa al desarrollo de nuestro trabajo, puesto que ofrece una serie de cualidades bastante interesantes a la hora de desarrollar herramientas colaborativas.

Habanero.

La herramienta Habanero fue desarrollada por Annie Chabert, Ed Grossman Larry Jackson y Stephen Pietrovicz de NCSA [Cha99]. Habanero provee sincronización de eventos y estados para múltiples copias de una aplicación software. Las aplicaciones software pueden soportar cualquier plataforma, ya que soporta Java. Habanero trabaja mediante la replicación de aplicaciones entre los clientes y el compartimiento de los cambios de estado. Cuando un cliente se une a la sesión, se le envía información de qué aplicaciones se están ejecutando en esa sesión. Cada aplicación envía la suficiente información para que el nuevo cliente complete la réplica. Habanero también asegura que todos los clientes ven los mismos datos.

Habanero también provee un objeto para realizar *Floor Control*, llamado arbitrador, estos controles junto con sus eventos pueden ser transformados en cualquier momento. La clase más importante de arbitrador es la que provee bloqueos. Los bloqueos en Habanero son objetos para restringir el acceso a recursos. Además, Habanero provee un entorno colaborativo que soporta gestión de sesiones, permitiendo a cada participante crear, unirse, dejar y ojear sesiones.

2.9. Sistemas CSCL.

Dentro del área de CSCW ha surgido un nuevo campo en el cual se enfocan una serie de estudios y teorías para marcar una serie de directrices en el desarrollo de un sistema educacional mediado por el uso del ordenador.

Principalmente en el área de la educación y del aprendizaje surge una clara diferenciación entre los términos cooperativo y colaborativo emergiendo dos paradigmas diferentes en el planteamiento del enfoque del aprendizaje.

Más concretamente, la solución propuesta en este trabajo de investigación queda enmarcada dentro del Aprendizaje Colaborativo, en el cual el aprendizaje se basa en la experiencia compartida de los estudiantes.

Dentro del Aprendizaje Colaborativo surge un nuevo tipo de aprendizaje en el ámbito de las ciencias, a través del cual los alumnos colaboran, razonan y argumentan sobre los resultados de experimentos. Este nuevo aprendizaje recibe el nombre de Aprendizaje por Descubrimiento.

En este apartado analizaremos las bases fundamentales sobre las que se sustenta el aprendizaje colaborativo al igual que mostraremos las características que debe proporcionar una arquitectura que pretenda dar soporte al aprendizaje por descubrimiento. Todos estos fundamentos servirán de base en la construcción de nuestros resultados, tanto para el sistema colaborativo síncrono así como para la plataforma para el aprendizaje por descubrimiento.

En el campo educacional surgen básicamente dos modelos de aprendizaje de la relación entre estudiantes y profesores. Ambos modelos presentan un enfoque diferente en la construcción del conocimiento. Más concretamente Panitz [Pan97] define las diferencias de estos dos enfoques en la definición de lo que es Aprendizaje Cooperativo y Aprendizaje Colaborativo.

La premisa subyacente entre la diferencia de ambos paradigmas se basa en el enfoque constructivista, en el cual el conocimiento es descubierto por los estudiantes y transformado en conceptos que el estudiante puede relacionar. Dichos conceptos pueden ser reconstruidos y expandidos a partir de nuevas experiencias de aprendizaje. Por lo tanto, el *aprendizaje colaborativo* consiste en la participación activa de cada uno de los estudiantes en contraposición de la aceptación pasiva de información de un profesor. Con lo cual el aprendizaje se construye mediante un conjunto de transacciones e interacciones entre estudiantes y profesores, en un establecimiento social, como base en la manera que ellos construyen el conocimiento.

Desde el punto de vista de Ken Bruffee [Bru95] la educación es un proceso de culturización a través de conversaciones constructivistas. Principalmente identifica dos tipos de conocimiento. Por una lado reconoce el "*Foundational Knowledge*" o el conocimiento basado en fundamentos, que es aquel conocimiento básico representado por las creencias justificadas socialmente, como puede ser las reglas de gramática, procedimientos matemáticos, hechos históricos, etc, el cual es el conocimiento apropiado para un aprendizaje cooperativo.

Por otro lado denomina "*Nonfoundational Knowledge*" o el conocimiento no basado en fundamentos, como aquel que se deriva a través del razonamiento y de las preguntas contra el hecho de memorizar los nuevos conceptos. Sobre este tipo de conocimiento se basa el aprendizaje colaborativo.

Bruffee considera estas dos aproximaciones linealmente, puesto que el aprendizaje colaborativo está diseñado para recoger aquello que el aprendizaje cooperativo no abarca. De esta manera, los estudiantes aprenden los fundamentos en los primeros años de su educación y luego pueden ampliar sus capacidades de crítica, razonamiento y entendimiento de las interacciones sociales, involucrándose más activamente en su proceso de aprendizaje del cual pueden tomar cierto control a través de tareas colaborativas.

Según la definición de Panitz el *aprendizaje cooperativo* es un conjunto de procesos los cuales ayudan a las personas implicadas a interaccionar juntos con el objetivo de conseguir un fin específico. Además este aprendizaje es controlado y gobernado por la figura del profesor y se basa en unos contenidos, actividades y análisis para llevar a cabo la tarea educativa.

Muchos de los elementos que forman parte de aprendizaje cooperativo también pueden utilizarse en el aprendizaje colaborativo. Un ejemplo claro pueden ser un entorno dónde el profesor deja uno materiales de estudio, los cuales a su vez les pueden servir en el razonamiento y posterior discusión.

Aunque nuestro trabajo de investigación se ha centrado mayoritariamente en el desarrollo de infraestructuras y herramientas para dar soporte al aprendizaje colaborativo, también se ha trabajado con entornos en el cual la guía del aprendizaje la imponía el profesor. Un ejemplo de este tipo de entornos cooperativos es JLE (Java Learning Environment) en el cual la Universidad de Murcia ha participado activamente, como podremos mostrar en el siguiente capítulo.

El área de aprendizaje colaborativo (CSCL) [Ros96] es una área multi-disciplinar que surge de las influencias del área de trabajo colaborativo (CSCW) y teorías del aprendizaje colaborativo que apuestan por un aprendizaje centrado en el grupo y en la socialización del proceso de aprendizaje basado en la colaboración.

Dicho aprendizaje se basa en el intercambio entre iguales para el desarrollo compartido de modelos del conocimiento conseguidos a través del establecimiento de objetivos comunes en un grupo. Así, la creación conjunta de conocimiento, se fundamenta en escenarios de aprendizaje basado en problemas (PBL- Problem Based Learning) en los que el conocimiento individual se obtiene de la interacción y socialización con los miembros del grupo [Jon01][Sta00].

Los sistemas CSCL se han definido como aquellos entornos que facilitan y mejoran la interacción colaborativa entre los miembros de un grupo y disponen de mecanismos de distribución y compartimiento del conocimiento y experiencia generada por los miembros de la comunidad.

La diferencia más destacable entre los términos CSCW y CSCL es que CSCW se centra más en las técnicas de comunicación mientras que CSCL se basa más en lo que está siendo comunicado. De ahí que este término sea más usado en el ámbito educacional [Kli01].

La colaboración tiene una contribución bastante influyente en el aprendizaje. Un ejemplo de esto es mostrado en un estudio llevado a cabo por Springer, Stanne y Donovan [Spr99] a partir del cual se mostró que el aprendizaje conjunto de pequeños grupos incrementa el rendimiento de estudiantes en áreas como son las Ciencias, Matemáticas, Ingeniería y Tecnología. A través de su estudio mostraron que los grupos de aprendizaje conllevan a mejores rendimientos académicos y un incremento en la persistencia de la tarea de aprendizaje.

Según Wouter R. van Joolingen [Joo00] en un proceso orientado a la discusión en el aprendizaje colaborativo hay ciertos factores los cuales pueden hacer la colaboración más efectiva:

- **Mantenimiento de una base común.** Los participantes deben ser conscientes del objetivo de la tarea y mantener un enfoque común.
- **Responsabilidad compartida, igualdad y mutualidad.** Los participantes necesitan tener roles significantes y ser responsables de la tarea de aprendizaje como un todo.
- **Soporte mutuo y crítica.** Parece ser el vehículo central en el aprendizaje colaborativo, haciendo que dos o más aprendices alcancen mayores metas que las que pueden alcanzar individualmente.
- **Verbalización y construcción.** La discusión y la argumentación estimula la formulación explícita del conocimiento, con lo que esto ayuda al estudiante con el rendimiento del proceso cognitivo.
- **Elaboración.** Los estudiantes aprenden de la ayuda elaborada provistas por otros colaboradores.
- **Afinación cognitivamente y socialmente.** Los aprendices están a mayor nivel de entendimiento que el de un profesor y un alumno. Por lo tanto se cree que la comunicación dentro del grupo de aprendices puede ser más efectiva que el aprendizaje guiado por un profesor.

El soporte de colaboración a través de herramientas mediante el uso de ordenadores impulsa uno o más de estos factores. Las herramientas que habilitan la comunicación ayudan a los aprendices a expresar sus conocimientos de una manera explícita o permitir que se negocie o se trate una tarea común. Más concretamente, el marco de nuestro trabajo se basa en la construcción de los soportes así como de las herramientas que permitan habilitar la comunicación entre los estudiantes, con el objetivo de favorecer la tarea del aprendizaje.

A continuación veremos diferentes enfoques en el aprendizaje así como la diferencia fundamental con los sistemas CSCL.

Sistemas de Gestión de cursos online vs Sistemas CSCL.

Los sistemas **CSCL** (Computer Support for Collaborative Learning) son definidos como aquellos sistemas los cuales realzan la interacción entre los miembros de un grupo y facilitan el compartimiento y distribución del conocimiento así como la experiencia entre todos los miembros del grupo. De esta forma, la definición incluye una gran variedad de sistemas que van desde los de gestión de cursos con herramientas colaborativas, a los sistemas CSCW y entornos de construcción de conocimiento. En la especificación de un sistema CSCL tenemos que hacer clara la distinción entre dos tipos de sistemas:

1. Sistemas de gestión de cursos online, publicaciones de material de aprendizaje y de entrega con herramientas colaborativas.

2. Aplicaciones Groupware (de trabajo en grupo) soportando la construcción de conocimiento a través de grupos de estudio.

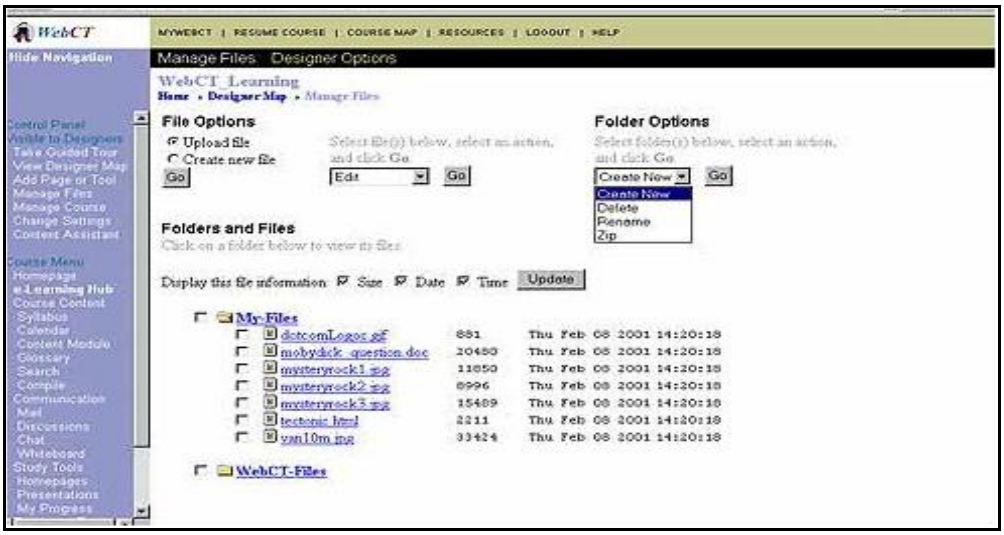

Figura 18. Gestión de ficheros en WebCT.

El primer grupo (como son WEBCT, TopClass, y JLE-Edustance) se trata de sistemas que son diseñados para reproducir material de aprendizaje fácilmente accesible por múltiples usuarios. El contenido del curso es la parte central del sistema y la posibilidad de comenzar la discusión entre el grupo de aprendizaje es provista como una característica suplementaria mediante herramientas colaborativas. Estos sistemas están basados en los tradicionales de Computer Based Training (CBT).

Sin embargo, muchos autores en el campo CSCL (Gerry Stahl, Teemu Leionen), piensan que en los sistemas de gestión de cursos las herramientas de comunicación quedan marginadas o apartadas de lo que forma parte de la tarea de aprendizaje. La atención se enfoca en el aprendizaje de los recursos más que en la experiencia compartida.

Por contrario, el segundo caso, se trata de sistemas que promueven a los estudiantes a construir su propio conocimiento durante el proceso de aprendizaje, confiando en la experiencia compartida antes que en la información provista por un profesor. Estos sistemas se basan en el uso de herramientas colaborativas que permiten a los estudiantes (usuarios) la construcción de su propio conocimiento y el compartimiento de este con otros, con el objetivo de alcanzar un entendimiento común en el grupo. Además este tipo de sistema, también añade la funcionalidad de tener un área compartida dónde se localizan el material de estudio. Estos sistemas son los que realmente nos interesan en el campo CSCL, puesto que la base central es la colaboración y como resultado de esta colaboración se obtiene el material de estudio.

Aprendizaje a distancia vs Aprendizaje Local.

Con respecto al aprendizaje a través de la red existen estos dos tipos de sistemas:

Sistemas de aprendizaje a distancia.

Sistemas que soportan grupos de aprendizaje ya existentes en los que los colaboradores tienen encuentros cara a cara.

Parece que el primer tipo de aplicaciones de aprendizaje está en vigencia en el mundo comercial: hay una gran cantidad de sistemas enfocados en el aprendizaje a distancia. La motivación básica para este tipo de sistemas es el coste/eficiencia. Los estudiantes no necesitan un lugar determinado para llevar a cabo sus actividades de aprendizaje y los profesores son capaces de manejar una gran cantidad de estudiantes. Todavía existen notables inconvenientes en los entornos de aprendizaje a distancia debido a la ausencia de contacto físico entre los estudiantes.

El aprendizaje llevado a cabo en grupos requiere un gran nivel de confianza y motivación por parte de los estudiantes, lo cual es difícil de lograr en comunidades de personas las cuales podrían no encontrarse nunca cara a cara. Las aplicaciones de aprendizaje a distancia tienen que tener en cuenta tanto la complejidad de ofrecer ayuda a los estudiantes así como la dificultad de transferir las jerarquías existentes en el mundo real al mundo virtual.

En la segunda clase de entornos no se tiene el propósito de reemplazar los encuentros cara a cara de los colaboradores. Se ve el aprendizaje como un proceso continuo embebido en varios espacios sociales e incluido en diferentes tipos de prácticas, es decir, usaremos este tipo de sistema como herramienta adicional al aprendizaje tradicional enriqueciendo el aprendizaje mediante el uso de las Tecnologías de la Información y la Comunicación (TIC).

Hay que tener en cuenta que la educación a distancia propone un cambio radical frente a la educación tradicional, y la incorporación de este tipo de técnicas debe ser introducida de una forma gradual, sobre todo en colegios o centros que no son de grado superior, en los que los alumnos no han alcanzado la suficiente madurez como para ser conscientes de su implicación y su participación en la tarea de aprendizaje. En el aprendizaje a distancia se requiere un compromiso del alumno en cuanto a que la responsabilidad de llevar a cabo la tarea de aprendizaje va a ser suya, por lo cual este compromiso sólo puede llevarse a cabo en personas lo suficientemente maduras y responsables.

Herramientas colaborativas CSCL.

Como veremos posteriormente, nuestros estudios se han enfocado en la construcción tanto de soportes como de herramientas para el apoyo del aprendizaje colaborativo. Por eso, un paso fundamental en este estudio es analizar las distintas herramientas que se pueden proveer en un entorno educativo.

De acuerdo con los propósitos de los sistemas CSCL es necesario contar y realizar un estudio sobre las herramientas colaborativas síncronas y asíncronas que van a ser los medios para la construcción del conocimiento. Algunas aplicaciones pueden ser usadas tanto en modo síncrono como asíncrono y algunas de ellas podrían tener módulos diferentes según el tipo de uso que se vaya a hacer de ellas.

Colaboración asíncrona.

Una característica importante de los sistemas de colaboración asíncrona es que hace los datos persistentes y accesibles a otros. También asegura que todo lo que se haga puede ser atendido por todos los usuarios/estudiantes, a diferencia de algunas herramientas síncronas. Debido a estas características las herramientas asíncronas son consideradas de gran importancia en los sistemas CSCL.

Existen muchas funcionalidades de herramientas CSCL ofrecidas para la comunicación asíncrona entre ellas vamos a destacar las siguientes:

- **Espacios para compartir documentos**. Es una característica muy importante en los sistemas CSCL pero como ya hemos dicho anteriormente no es la parte central. Los espacios de documentos compartidos nos permiten dejar documentos (material de estudio) en una zona accesible por varios usuarios.

- **Espacios para discusión**. Muchos de los sistemas CSCL consideran la discusión asíncrona como una característica que forma parte central del sistema CSCL. Hay numerosos entornos los cuales ofrecen funcionalidades básicas de un foro: los estudiantes son capaces de enviar sus notas, replicas a otras previas y visualizar las notas de forma que se vean todas y las replicas que han generado cada una de las notas de discusión.

  Sin embargo hay que tener cuidado en como se muestra esta información. Un sistema CSCL debería mostrar esta información de una forma estructurada de forma que permita a los alumnos permanecer alerta acerca de sus objetivos como aprendices. Los investigadores y profesores han informado de la posibilidad de asignar ciertas categorías de búsqueda para que la información que se muestre a los alumnos se ciña a lo que buscan.

  También hay que señalar que el uso de etiquetas para las notas asignando iconos y colores es muy importante para darle significado al proceso de

aprendizaje, de tal forma que los alumnos puedan asignarle una etiqueta a su nota dependiendo de a qué categoría pertenezca.

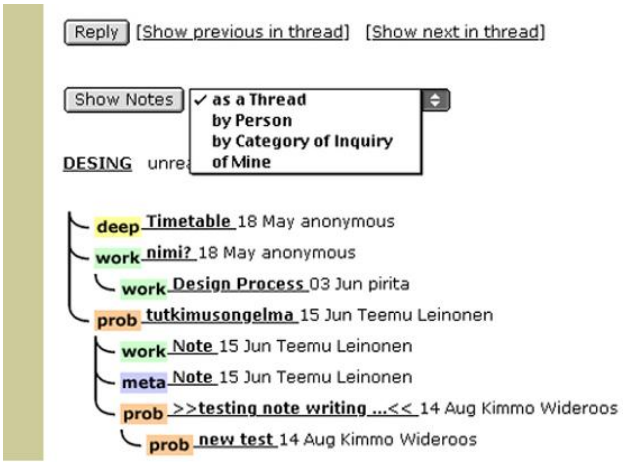

Figura 19. Interfaz de construcción del conocimiento de Fle2

- **Pizarras**. Un componente importante en muchos sistemas CSCL es la pizarra, la cual puede ser usada tanto de forma asíncrona como síncrona. La diferencia importante es que cualquier cosa hecha en la pizarra es visible y accesible para todos los participantes del grupo. Los objetos y otro tipo de datos creados tienen un nivel de persistencia: la posibilidad de cambiar los de otros varía según los sistemas. Hay sistemas que permiten bloquear los objetos de tal forma que sólo puede modificarlos el propietario. Otros, sin embargo, permiten modificar cualquier objeto en el área compartida.

  Esta herramienta permite a los usuarios representar mediante diferentes formas su conocimiento, pudiendo hacer uso de mapas conceptuales para enlazar diferentes elementos o para clasificar cualquier concepto.

Colaboración síncrona.

Como hemos mencionado anteriormente la comunicación síncrona es esencial en la educación a distancia, dónde los estudiantes carecen de otro tipo de comunicación en tiempo real. Las herramientas síncronas soportan la percepción de la telepresencia y por lo tanto muchas veces son usadas para reemplazar los encuentros cara a cara.

La comunicación síncrona tiene ciertas ventajas con respecto a la comunicación asíncrona como son: interacciones directas e inmediatas entre los usuarios, que permite a los usuarios regular sus interacciones de acuerdo al contexto y situación; tutorización y gestión dinámica y fluida en la gestión de un proceso de aprendizaje; colaboración realzada mediante la creación de representaciones espaciales del contexto de aprendizaje y construcción del conocimiento en los mundos de avatars.

Sin embargo en entornos CSCL la comunicación síncrona puede traer algunos aspectos indeseables en una discusión significativa y en el proceso de aprendizaje. Además una discusión tiende alejarse de la comunicación síncrona si alguno de los miembros no tiene la posibilidad de participar. El diseño de herramientas síncronas en entornos CSCL

debe de realizarse con especial atención teniendo en cuenta las sugerencias de los tutores y las formas en que se va a realizar la comunicación para facilitar el proceso de aprendizaje.

En cuanto a las herramientas síncronas en entornos CSCL podemos tener en cuenta las siguientes:

- Pizarras y presentadores.
- Chat basados en texto.
- Chat basado en texto en mundos de avatares.
- Audio chat y video-conferencia.
- Mensajería instantánea.

Presence Awareness.

Una de las necesidades importantes de los sistemas CSCL es cómo representar a los estudiantes y mostrar su presencia, de tal forma que los alumnos al conectarse al entorno tengan una representación de que hay otros usuarios, de quién está dirigiendo la sesión, etc.

Hay sistemas en los que se representan a los alumnos mediante su fotografía, mediante imágenes de tal forma que cuando estén online hagan algún tipo de mueca (como abrir o cerrar los ojos), mediante una lista mostrando sólo los usuarios que están online, etc.

Mientras menos contacto exista entre los alumnos será necesaria una representación más detallada de estos. Todo esto tiene que ser tomando en cuenta para que el alumno no se sienta solo y perdido ante la tarea educativa. Muchas veces el fracaso de estos sistemas viene asociado a la soledad con la que se encuentra el alumno. Por ello este es un aspecto muy importante a tener en cuenta en el desarrollo de cualquier sistema CSCL, para que el alumno tenga conciencia de que no está solo en el entorno.

Herramientas para el análisis de la actividad.

En sistemas basados en contenidos, normalmente la evaluación del aprendizaje se realiza a través de una serie de exámenes. Sin embargo, en los sistemas de aprendizaje colaborativo es necesario contar con otros mecanismos de evaluación ya que la base del aprendizaje se construye en la colaboración entre usuarios.

Desde el punto de vista pedagógico [DeB03] una de las dimensiones más ricas para evaluar las actividades del aprendizaje colaborativo consiste en el análisis del nivel de participación, de los intercambios de conocimiento ocurridos durante una actividad o durante el curso; en definitiva, de la dinámica de trabajo del grupo y de cada uno de los miembros. Está puede analizarse desde diferentes perspectivas, como puede ser la evaluación de la participación y la valoración manifestada por los colaboradores.

Para realizar un análisis de la participación es necesario recoger información acerca de los mensajes enviados, objetos dibujados, notas escritas, notas que responden a otra anterior, etc. El uso de la comunicación por ordenador puede favorecer la recogida de dicha información en soportes de bases de datos pudiendo así ofrecer un conjunto de

herramientas para posteriormente poder presentarla al evaluador de una manera resumida y mediante el uso de herramientas gráficas. Más concretamente la recogida de estos datos ofrece una medida cuantitativa de la comunicación realizada.

En muchas ocasiones, no sólo se requiere medir cuantitativamente el trabajo realizado por la colaboración de los alumnos sino que se requiere medir la calidad de dicha colaboración. Para ello es necesario mostrar información más detallada mostrando por ejemplo el contenido de los mensajes, la visualización de los objetos dibujados, etc. De igual manera el uso de las nuevas tecnologías puede proporcionar un conjunto de herramientas que permitan la visualización del contenido del intercambio entre usuarios.

En la tabla 3, se muestra las dimensiones útiles para la realización de la evaluación [DeB03].

Nivel de participación	Número total de mensajes enviados.
	Mensajes enviados por persona.
	Mensajes por temática.
	Mensajes en relación con el momento de trabajo (inicio, final, …)
	Destinatario del mensaje.
	Objetos dibujados en una pizarra.
	Mensajes respondidos a un mensaje propuesto.
Naturaleza de la participación.	Aporta recursos, información nueva, referencias, pruebas de la experiencia.
	Sintetiza, resume el nivel de acuerdo y desacuerdo, planeando nuevas vías para avanzar en el proceso de trabajo.
	Integra los nuevos acuerdos.
	Propone problemas, interrogantes ante el trabajo realizado.

Tabla 3. Dimensiones de la observación de la participación.

Aprendizaje por Descubrimiento (Discovery Learning)

Dentro del campo del aprendizaje colaborativo emerge el aprendizaje por descubrimiento (Discovery Learning) [Jon98], en el cual los aprendices colaboran en el dominio de la experimentación a través de inferencias e hipótesis que formulan a través de su experiencia con las herramientas de simulación. La idea básica del aprendizaje por descubrimiento es que los estudiantes construyen su conocimiento a través de la experimentación usando un conocimiento base del cual los alumnos reúnen cierta información desde el entorno.

Los entornos por descubrimiento suelen basarse en simulaciones por ordenador puesto que ofrecen un modo seguro y flexible de representar el dominio de una manera que el aprendiz puede jugar y experimentar con él. En este tipo de aprendizaje, los aprendices deben generar hipótesis, diseñar experimentos, predecir la salida, interpretar los resultados y reconsiderar las hipótesis [Joo97] a la hora de construir el conocimiento en esa área.

Sin embargo, los aprendices pueden fallar en las hipótesis probadas, en el diseño de experimentos, o en la interpretación de los resultados. Con el objetivo de que los aprendices puedan realizar su aprendizaje por descubrimiento con éxito, deben de tener soporte para ello desde el entorno de aprendizaje, el cual puede contener herramientas cognitivas las cuales les permitan el soporte de uno o más procesos de aprendizaje. Los desarrollos en el aprendizaje por descubrimiento pretenden encontrar nuevas formas para el soporte y creación de un diálogo de aprendizaje entre el entorno y los aprendices, estableciendo también las condiciones bajo la cuales se puede llevar un proceso de aprendizaje beneficioso.

Diseño de arquitectura para el aprendizaje por descubrimiento

Con el objetivo de proveer las bases para la creación de una arquitectura que pudiera facilitar el aprendizaje por descubrimiento Wouter van Joolingen [Joo00] expone un diseño de lo que una arquitectura de este tipo debería incluir.

Concretamente según Joolingen dicha arquitectura conformaría una arquitectura basada en componentes para el aprendizaje de las Ciencias a través del uso de la experimentación. Según Wouter, los ingredientes básicos que conforman dicha arquitectura serían los siguientes:

> **Marco de Referencia.** Con esto se indica que se debe mantener un marco común de referencia, de tal forma que independientemente de la visualización que se ofrezca de la información (tabla, gráfica, herramienta de simulación) las diferentes herramientas deben ser capaz de interpretarlas. El uso de estándares como XML favorece la creación de un marco de referencia común.
>
> **Herramientas colaborativas.** Estas herramientas deben actuar como componentes que manejan las interacciones entre aprendices y dan soporte para el mantenimiento de la comunicación entre diferentes usuarios. Por otro lado, dichas herramientas deben también ofrecer un soporte para la comunicación con otros componentes dentro de la arquitectura, como puede ser la del cliente con un experimento que se ejecuta remotamente en otro equipo.
>
> **Espacio de experimentación**. Representa el dominio de la tarea sobre la cual se lleva la tarea de aprendizaje. Es el generador de los elementos de los cuales se razona, se reconstruye nuevos modelos, etc. De hecho el espacio de experimentación provee la información para el marco de referencia. Se puede decir que constituye el área de colaboración dónde los usuarios llevarán a cabo la fase de experimentación.
>
> **Escenarios colaborativos.** En la colaboración síncrona se debe proporcionar las mismas vistas del espacio de experimentación a todos los colaboradores. Una vez en el espacio de experimentación los aprendices deben tomar turnos a la hora de colaborar con ciertas herramientas. Herramientas de comunicación general como son el chat, whiteboard, no requieren un control de turnos para colaborar con ellas.

Sin embargo en la colaboración asíncrona, los aprendices trabajan por su cuenta y lo único que necesitan es guardar su trabajo en el sistema y poder recuperar la versión exacta del trabajo que han realizado cuando se conectan de nuevo en el entorno. De igual manera, cada vez que abandonan el entorno la información debe ser guardada en el sistema.

En la figura 20 se muestra el esquema según Wouter para la colaboración síncrona (imagen de la izquierda) y para la colaboración asíncrona (imagen de la derecha).

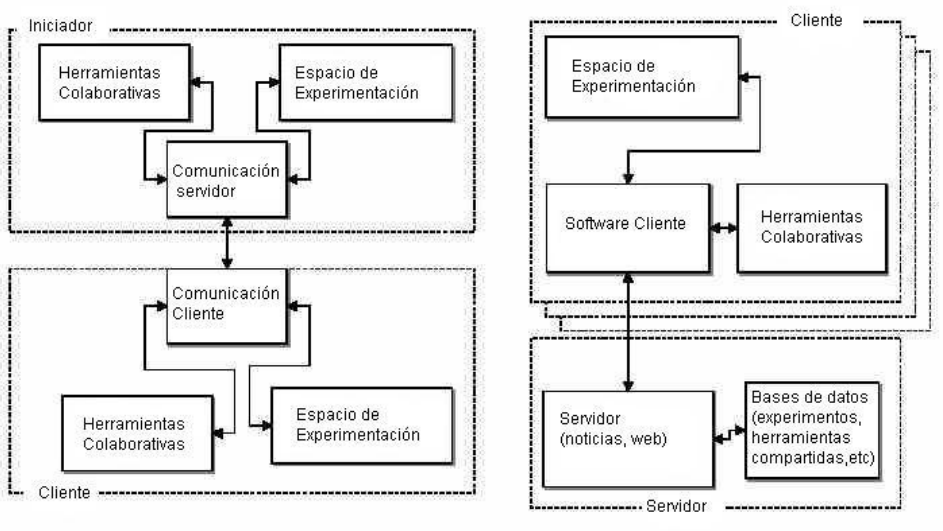

Figura 20. Arquitectura para la colaboración síncrona y asíncrona dentro de una plataforma para el aprendizaje por descubrimiento.

2.10. Entornos Colaborativos.

Entornos CSCW.

En esta sección trataremos los sistemas CSCW más destacados de las últimas décadas.

Sistema Lambda MOO

LambdaMOO [Cur92] [Cur93] es un sistema interactivo, multi-usuario y programable inicialmente diseñado para la construcción de juegos de aventuras con interfaces textuales. Sin embargo, su versatilidad y abstracción avanzada le ha permitido ser utilizado como base de sistemas colaborativos en diferentes ámbitos.

Sus siglas vienen de MUD *Object Oriented*, donde MUD deriva de *Multi User Dungeon* y se trataba de un juego de aventuras orientado a objetos. Lo importante de este sistema es su extensibilidad y sus facilidades de programación que han hecho que su uso se extienda a un gran número de comunidades virtuales.

Su modelo conceptual es sin duda su aportación más interesante. Un MOO es una base de datos que almacena objetos con sus propiedades y sus métodos. La topología del espacio se representa mediante objetos "Habitación" (*Room*) conectados entre si por objetos "Puerta"(*Portal*). Los usuarios de suelen denominar "Avatars" y en las Habitaciones podemos encontrar objetos "Cosa"(*Thing*). El sistema es extensible y programable con lo que está específicamente diseñado para incorporar nuevos objetos.

Por último, cabría decir que incorpora un modelo de seguridad que garantiza el acceso de los usuarios a los objetos del entorno.

El modelo arquitectónico es en cambio bastante pobre. Si bien es comprensible dadas las tecnologías existentes en su momento de creación. El lenguaje de desarrollo es el lenguaje C, y los clientes se conectan al servidor LambdaMOO utilizando *telnet* u otro cliente textual especializado. El protocolo utilizado para la transmisión de cambios es TCP-IP. Dada su edad, la mayoría de los servicios distribuidos tuvieron que ser implementados totalmente por los creadores del sistema. Así, su modelo de persistencia es una base de datos que se carga íntegramente en memoria cuando el sistema está en ejecución. Su modelo de concurrencia también ha debido ser programado junto a la base de datos.

Su aportación más relevante es sin duda su modelo conceptual orientado a objetos que destaca por su gran extensibilidad. Dicho sistemas sirvió de base para la construcción de la Plataforma ANTS [Gar02], la cual es fruto de una tesis desarrollada dentro de la Universidad de Murcia. Al final de esta sección veremos las características más importantes de dicho sistema.

BSCW (Basic Support for Cooperative Work).

Esta plataforma surge como un desarrollo del GMD, ahora llamada FIT, dentro del proyecto POLIKom. BSCW [BSCW] soporta áreas compartidas a través del web mediante las cuales los usuarios pueden compartir sus documentos y organizarlos de una manera estructurada mediante el empleo de carpetas o "folder". Además dicho sistema incorpora el sistema NESSIE el cual es un sistema de conciencia y monitorización de eventos, que también forma parte del desarrollo de GMD-FIT. Mediante este sistema de monitorización se ofrecen los eventos más importantes relacionados con el uso de los elementos más importantes.

Esta plataforma ha estado en uso desde la mitad de 1995 y todavía sigue recibiendo cambios y manteniendo versiones más actualizadas. En las últimas versiones provee un cambio de interfaz bastante mejorado, incluyendo herramientas como los calendarios de grupo que permiten la planificación de todos los eventos a llevar a cabo por los miembros del grupo así como el tablón de discusiones, en el cual los usuarios pueden debatir acerca de cualquier tema de interés.

La colaboración que ofrece es básicamente asíncrona y su modelo de comunicación se basa en el uso de HTTP para realizar cualquier operación. Está desarrollado en Python, lo cual previo a su instalación es necesaria la instalación de dicho interprete. En este sistema, la gestión de usuarios se realiza mediante el uso de direcciones electrónicas, a través de las cuales se pueden recibir notificaciones de los eventos nuevos en el sistema, evitando de esta manera tener que entrar en el sistema para ver si ha habido algún cambio.

Ya en sus últimas versiones provee una API basada en XML-RPC llamada XBSCW para implementar herramientas que se puedan integrar en el BSCW.

Como podremos observar posteriormente, dicho sistema será la base del modelo asíncrono de una de las integraciones de nuestro sistema colaborativo síncrono.

Plataforma ANTS.

La plataforma ANTS es el fruto de las investigaciones llevadas a cabo por Pedro García [Gar02], desde la cual hemos colaborado activamente tanto en el desarrollo de aplicaciones sobre esa arquitectura como en la integración de un modelo de sincronización.

A través de su trabajo de investigación García desarrolló una arquitectura genérica tanto para entornos cooperativos como para entornos colaborativos. En el diseño de esta arquitectura se abordan los siguientes niveles: el nivel de aplicación, el nivel de servicios colaborativos, el nivel de adaptación y el nivel de servicios distribuidos.

La capa de aplicación está destinada a los desarrolladores que quieran extender el marco para crear nuevos componentes o funcionalidades y es donde se engloba el conjunto de herramientas de la arquitectura. Más adelante veremos como una de las integraciones de una pizarra electrónica ha sido llevada a cabo en este nivel. En esta capa es posible insertar dos tipos de componentes: los componentes colaborativos del entorno de colaboración y los actuadores del sistema de monitorización.

La capa de servicios colaborativos se divide a su vez en dos sistemas fundamentales: el módulo de colaboración (ANTS.CORE) y el módulo de monitorización (ANTS.AWS). El módulo de colaboración comprende los servicios que cubren los modelos de interacción, coordinación y seguridad. Parte de nuestra investigación se centro en la introducción de políticas de coordinación así como elementos de sincronización (Token). El módulo de monitorización comprende los servicios que soportan el modelo de monitorización también definido en el modelo conceptual.

La capa de adaptación se encarga de definir fachadas software a una serie de servicios distribuidos para simplificar el acceso a los mismos. Así, esta capa define fachadas software a una amplia gama de sistemas de notificaciones avanzados y a los servicios distribuidos ofrecidos por los servidores de aplicaciones. En este punto también se facilita un marco o punto de acceso uniforme y simplificado a sistemas de calidad de servicio (QoS) para el canal de comunicaciones.

Por último, la capa de servicios distribuidos comprende los sistemas de red utilizados para sustentar la arquitectura propuesta. El punto principal de esta capa son los marcos de integración de servicios distribuidos o también llamados servidores de aplicaciones. Destacamos en este punto los sistemas de publicación /suscripción por ser una pieza clave de la arquitectura propuesta.

El hecho de que esta estructura sea modular dará lugar a muchas posibles integraciones de herramientas de dicha herramienta y hará que sea reutilizable en gran parte en otras muchas.

Como veremos posteriormente esta arquitectura ha sido la base para el desarrollo de la nueva versión de JLE, así como de un conjunto de integraciones de herramientas llevadas a cabo en este trabajo de investigación. Más concretamente, su modelo de propagación de eventos servirá de base para los resultados obtenidos en este trabajo de investigación.

En este apartado analizaremos tanto los diversos entornos que existen en esta área CSCL así como un conjunto de herramientas colaborativas que ayudan en la tarea de aprendizaje del alumno.

Entornos CSCL

FLE (Future Learning Environment)

FLE [Lei02] [FLE03] (Future Learning Environment) es un entorno web de aprendizaje colaborativo construido sobre Zope incorporando funcionalidades y conceptos específicos de CSCL. Básicamente FLE contiene tres aplicaciones para el aprendizaje.

El área compartida es denominada *WebTop* (véase figura 21) y su principal aportación reside en artefactos específicos de construcción conjunta de conocimiento. Así, dispone de una herramienta de construcción de conocimiento (*Knowledge Building*) que es en realidad una herramienta de discusión por hilos modificada para que sea posible categorizar cada mensaje en una fase del proceso de aprendizaje (problema, teoría en progreso, profundización de conceptos). Se ha estudiado que esta categorización contribuye a que los alumnos aumenten su comprensión y conciencia del proceso educativo. La tercera herramienta es la denominada *Jamming* que consiste en un espacio colaborativo compartido para la construcción de artefactos digitales (imágenes, video, audio, texto), permitiendo además el manejo de versiones.

Figura 21. Interfaz del entorno FLE3.

Como veremos posteriormente este entorno al igual que el BSCW, ha constituido de base para la integración de un sistema colaborativo.

TeamWave

TeamWave [Ros01] es un entorno comercial avanzado que dispone de espacios de colaboración avanzados orientados al proceso de aprendizaje. La abstracción fundamental de TeamWave es la habitación (*room*) donde alumnos y profesores disponen de artefactos de colaboración como pizarras compartidas, compartimiento de documentos, herramientas de votación y sistemas de anotación colaborativa. Los espacios se pueden conectar por puertas (*Doorway*) que crean un entorno más

situacional para los usuarios. En TeamWave se distingue entre instructor y alumnos y es posible programar actividades en el tiempo orientadas a mejorar el proceso de construcción conjunta de conocimiento. La flexibilidad del entorno permite colaboración síncrona y asíncrona ya que los resultados de interacciones síncronas puedan ser almacenados en modo persistente. Quizás el mayor defecto de TeamWave es su carencia de un marco potente de monitorización y control del progreso de los usuarios en el proceso de aprendizaje.

Limu

Limu [LIMU] presenta principalmente un conjunto de herramientas que favorecen la difusión de clases online. Posee artefactos de comunicación síncrona como son una pizarra así como un chat para la comunicación de los usuarios ofreciendo un conjunto de características gráficas a nivel de presencia bastante avanzadas, como son el uso de caras con ojos parpadeantes para indicar la presencia del usuario. Otra de las características de este sistema es el uso de petición de turnos previo a la colaboración dentro del área compartida de la pizarra. También ofrece herramientas asíncronas como son los foros.

Los sistemas MOO se han utilizado ampliamente en ámbitos de aprendizaje colaborativo e incluso para soportar Campus Virtuales como el de la Diversity University [DU01]. Un caso destacado es el de Tappedin que ha creado interfaces Web apropiadas a LambdaMOO con el objetivo de crear una comunidad educativa online. Dicho proyecto ha sido especialmente útil en proyectos de aprendizaje colaborativo debido a la riqueza conceptual y herramientas de colaboración disponibles en el entorno. En esta línea, los espacios compartidos de estos sistemas son Habitaciones (*Room*) donde se encuentran objetos (*Thing*) o componentes de colaboración. Estos conceptos situacionales favorecen la inmersión en el entorno y son conceptualmente superiores a las carpetas (*Folder*) de un sistema como BSCW. Los artefactos ofertados comprenden chats, navegación colaborativa, anotación, y compartimiento de documentos entre otros.

Aplicaciones para el aprendizaje.

SenseMaker [Bel97] o KnowledgeForum [KF301] también crean artefactos compartidos de creación conjunta de conocimiento donde se etiquetan los conceptos del problema en base a su fase de aprendizaje.

Belvedere [Pao+95] es otra herramienta clásica en CSCL que ofrece a los aprendices un estructura para la construcción de argumentos científicos. Con esta herramienta los aprendices pueden colaborar de manera síncrona en la construcción de hipótesis, evidencias y la unión entre ellas. Además Belvedere posee una utilidad llamada *Advisor* la cual analiza la estructura del argumento y sugiere directrices para una extensión más amplia del argumento que se está diseñando. Esta herramienta sirve de base para la estructuración o diseño de problemas relacionados en el ámbito de las ciencias.

SimQuest [Sim] es una herramienta especialmente usada para la creación de entornos para el aprendizaje por descubrimiento, posibilitando el diseño de recursos multimedia. Mediante esta herramienta se pueden construir modelos, interfaces para el aprendiz, actividades y explicaciones.

2.11. Conclusiones.

En este apartado hemos vistos las principales características y principios de diseño de las herramientas de trabajo en grupo. Como hemos comentado con anterioridad, la aplicación de nuestra tarea de investigación ha sido enmarcada en la creación de sistemas y plataformas educacionales. Más concretamente, nuestro campo de investigación abarca el estudio CSCL así como la colaboración síncrona dentro de este campo. A través del estudio realizado en este capítulo podemos concluir lo siguiente:

- En el desarrollo de aplicaciones distribuidas es importante que la información que manejan los usuarios sean las mismas en todas las localizaciones. Si bien el principio WYSIWIS parece el más idóneo para la colaboración síncrona, es demasiado estricto, con lo cual en nuestras investigaciones nos basaremos en el principio WYSIWITYS.
- Puesto que nuestros desarrollos e investigaciones serán llevadas a cabo en aplicaciones y plataformas dónde la colaboración será de muchos a muchos, es decir colaboración síncrona, el modelo de distribución de datos que usaremos será el modelo replicado.
- Debido al uso del modelo replicado de datos y al uso de colaboración síncrona, debemos integrar e incluso adoptar mecanismos de control de turnos, *Floor Control* y algoritmos de llegada tarde, *Late join*.
- Otro de los aspectos que debemos tener en cuenta en la construcción de cualquier aplicación es el uso de una buena interfaz gráfica, puesto que es la manera que tienen los usuarios de interactuar en el sistema. Más concretamente, en nuestro trabajo de investigación este punto llega a ser fundamental puesto que los resultados obtenidos van dirigidos al uso en colegios.
- Una arquitectura que servirá de base para el desarrollo de nuestro modelo de propagación de eventos es el resultado de la tesis del doctor García, la plataforma ANTS, cuya arquitectura modular permite la reutilización de algunos de los módulos. Más concretamente nosotros haremos uso del módulo de propagación de eventos, puesto que aporta un modelo genérico que permite el establecimiento de conexiones para el intercambio de eventos independientemente de la tecnología subyacente usada.
- Con el fin de que los colaboradores no se sientan solos, en cualquier aplicación colaborativa se hace necesario incluir algunas de las características de Awareness. Más concretamente en la construcción de herramientas para la colaboración síncrona es fundamental incluir características como son el *Presence Awareness* y *WorkSpace Awareness*, pudiendo de esta forma saber qué usuarios están conectados en la sesión y quién ha trabajo en el área compartida.
- Dentro del campo CSCL emerge el concepto de Aprendizaje por Descubrimiento, que es aquel basado en la experimentación compartida. Para que una plataforma pueda soportar el Aprendizaje por Descubrimiento es necesario establecer unos escenarios de colaboración, conjunto de herramientas colaborativas, un espacio para la experimentación así como un marco de referencia común de tal forma que usando diferentes herramientas se pueda visualizar la misma información.
- En cualquier entorno colaborativo se hace necesario la integración de herramientas de análisis que nos permitan analizar el nivel de participación asó como la naturaleza de la participación (si aporta ideas nuevas, recursos, referencias…).

Capítulo Tercero.

Estudio y Definición de Escenarios para el Aprendizaje Colaborativo.

Previo a los resultados obtenidos en nuestro trabajo de investigación, se han realizado diversos estudios e integraciones tanto de herramientas colaborativas así como de herramientas de seguimiento en diferentes tipos de entornos colaborativos.

En este estudio realizado se ha trabajado con las primeras versiones de un entorno educacional a distancia llamado JLE (Java Learning Environment) así como la plataforma ANTS resultado del trabajo de investigación llevado a cabo por el doctor García [Gar02].

De las investigaciones llevadas a cabo en el entorno educativo JLE se obtuvo no sólo la integración de una herramienta colaborativa así como propuestas de herramientas de seguimiento, sino también una serie de estudios llevados a cabo a través de los proyectos Pupitre-net y EDUSI, de los que se estableció un marco de referencia para la caracterización de entornos educativos a distancia [Gom00].

Como se ha expuesto anteriormente en el capítulo de antecedentes la plataforma ANTS es una plataforma basada en la arquitectura J2EE que propone un marco general tanto para la construcción de plataformas cooperativas así como colaborativas. El trabajo llevado a cabo en la plataforma ANTS abarcó principalmente los siguientes campos: el estudio de un mecanismo de sincronización, la integración de un componente de pizarra en dicha plataforma y la creación de una aplicación sobre la plataforma ANTS la cual accedía a todos los recursos configurados en la plataforma.

En este capítulo primeramente se analizarán los trabajos llevados a cabo dentro del JLE, tanto mostrando los resultados obtenidos de la integración de una herramienta colaborativa síncrona así como el estudio de herramientas de seguimiento del alumno.

En el segundo bloque de este capítulo veremos la estructura de la arquitectura ANTS, así como los pasos realizados en la integración de un componente pizarra dentro de dicha plataforma. Además, en nuestro estudio analizaremos los cambios necesarios en dicha arquitectura para que sirva como soporte para la construcción de aplicaciones enfocadas en la experimentación colaborativa, es decir el Aprendizaje por Descubrimiento.

1. JLE.

Uno de los primeros entornos desde los cuales se comenzó nuestra tarea investigadora, fue el entorno educacional Java Learning Environment (JLE). JLE fue un sistema tecnológico que reproducía un entorno virtual de formación basado en la definición de "Formación Presencial Virtual y a Distancia utilizando Aplicaciones Telemáticas" [Ege00]. Esta herramienta fue desarrollada por la empresa ESSI y actualmente es distribuida con el nombre de EDUSTANCE [EDU].

Los usuarios del entorno de formación JLE son el profesor, el alumno y el administrador del sistema. Este sistema es un sistema cooperativo de aprendizaje en el cual la formación se basa a través del aprendizaje de una serie de contenidos (CBT).

JLE estaba basada en el Web, lo que le confiere facilidad en el manejo, minimización de los requerimientos técnicos por parte del usuario e independencia de los puestos; el usuario, desde cualquier ordenador, podía acceder al entorno simplemente conectándose al servidor residente e identificándose como usuario del sistema.

JLE reproducía las directrices técnicas que establece IMS (Instructional Management System)[IMS], con el propósito de generar una herramienta formativa que cumpla los estándares de dicha organización, en su afán por optimizar el uso de Internet en contextos formativos.

Más concretamente, la Universidad de Murcia participó en este proyecto activamente tanto en la integración de herramientas para la colaboración así como el estudio de dicha plataforma a través los proyectos PUPITRE-NET y EDUSI para la difusión de estas nuevas tecnologías. En estos estudios se llevaron a cabo una serie de experiencias en colegios e institutos de la región de Murcia.

PupitreNet fue un proyecto de investigación financiado por el Programa Nacional de Aplicaciones y Servicios Telemáticos de la Comisión Interministerial de Ciencia y Tecnología (CICYT) en el que participaron investigadores de ocho universidades españolas (Universidad de Murcia, Universitat Rovira i Virgili, Universitat de Valencia, Universitat Jaume I, Universitat de les Illes Balears, Universidad de Sevilla, Universitat Autónoma de Barcelona y Universidad Autónoma de Madrid). Su objetivo fue la definición e implementación de una arquitectura tecnológica y conceptual de una plataforma virtual para la formación a distancia en Internet, adaptada a diversos entornos educativos y dotada de los medios necesarios para el intercambio remoto de información en tiempo real.

Fruto de esta investigación fue el entorno de tele-formación Java Learning Environment (JLE). Más concretamente nuestro trabajo consistió en la gestión del servidor de JLE situado en la Universidad de Murcia que, junto con el servidor situado en la Universidad Rovira i Virgili, debía dar servicio a profesores y alumnos de las ocho universidades anteriormente citadas.

En el grupo EDUSI se recogió a un colectivo de profesores y alumnos de colegios e institutos de la Región de Murcia los cuales experimentaron con nuevas tecnologías. Una de estas tecnologías fue el entorno de tele-enseñanza JLE. Concretamente la

experiencia consistió la incorporación de unidades didácticas realizadas por diferentes profesores en el entorno JLE para que posteriormente sus alumnos pudieran acceder a ella.

Atendiendo a los resultados de las pruebas así como a las diferentes necesidades de profesores y alumnos, realizamos un estudio acerca de la categorización de entornos educacionales a distancia el cual viene reflejado en el Anexo II, en un artículo presentada en RedIris [Gom00] y en el capítulo de un libro [Gom03].

Además de estos estudios, este entorno constituyó la base para la integración de herramientas colaborativas síncronas así como la caracterización de herramientas de seguimiento en un sistema de gestión de cursos, CMS (Course Management System).

1.1. Integración de una pizarra en el entorno JLE.

Previo a la integración de nuestras herramientas en este sistema fue necesario el estudio del funcionamiento de dicha plataforma. Básicamente la arquitectura que presentaba JLE v1.0 estaba basada en el uso de JavaWebServer (JWS) para el manejo del servidor web así como la gestión de sesiones. En estas primeras versiones del producto JLE, la interfaz gráfica se manejaba mediante uso de JHTML. Esta plataforma también daba soporte a la internacionalización mostrando los menus dependiendo del idioma que se eligiera en la entrada del usuario (*LoginJsIdiom)*.

Cada sesión web que inicia el usuario se correspondía con un gestor de RMI (*RMIGestor)* el cual era el responsable del acceso a la base de datos y de devolver al objeto sesión la información correspondiente a cualquier entidad de la plataforma (usuario, grupo en el que trabaja, cursos en los que está matriculado, etc). Todas las aplicaciones dentro del sistema (chat, correo, editor de exámenes, editor de cursos, etc) se construyeron mediante el uso de Applet y se cargaban mediante el empaquetamiento de jars.

Para la integración de la pizarra colaborativa se hizo uso de la API JSDT la cual ya ha sido comentada anteriormente en el capítulo de Antecedentes. El uso de JSDT nos proporcionó un sistema de comunicación basado en el paradigma *publish/suscribe* que nos permitía mantener una colaboración síncrona de una manera rápida y eficiente. Además, mediante su mecanismo de tokens nos permitió la sincronización de los usuarios en el envío del estado de la sesión a los usuarios que llegaban tarde así como en el manejo de los recursos que integraban la herramienta.

La integración de los componentes en el sistema se realizó a través de la inclusión de una serie de paquetes distribuidos en ficheros jars, los cuales serían invocados por JHTML para cargar la aplicación.

El modelo de comunicación diseñado en esta integración como se muestra en la siguiente figura, dónde se muestra las nuevas integraciones junto con la arquitectura subyacente del sistema JLE.

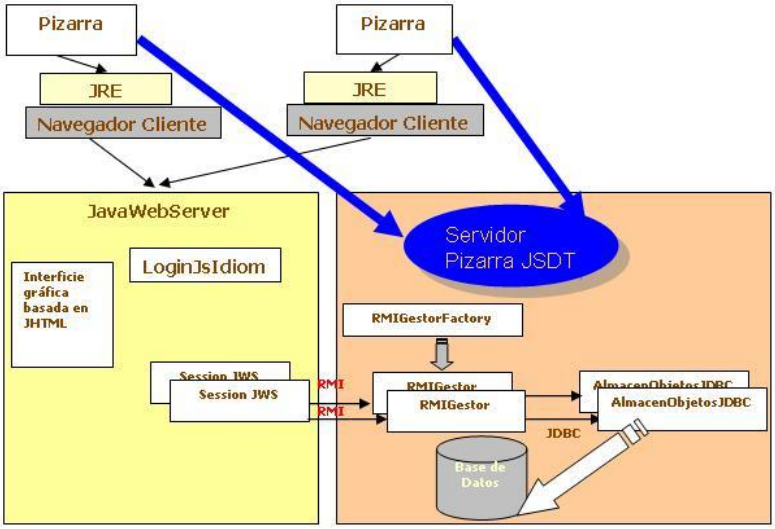

Figura 22. Integración de la pizarra colaborativa en el sistema JLE

Una vez arrancado el applet, los clientes pizarra se comunican mediante los eventos a través de los canales creados mediante el uso de JSDT.

En esta integración, esta herramienta simplemente ofrecía una colaboración síncrona sin la posibilidad de guardar información en el sistema. Básicamente estaba formada por un chat, una herramienta de presentación colaborativa y el uso de un puntero en la marcación de las diapositivas mostradas. Estas dos últimas opciones sólo eran visualizadas por aquellos usuarios que tenían el rol profesor. Gráficamente la apariencia de dicha aplicación viene mostrada en la figura 23.

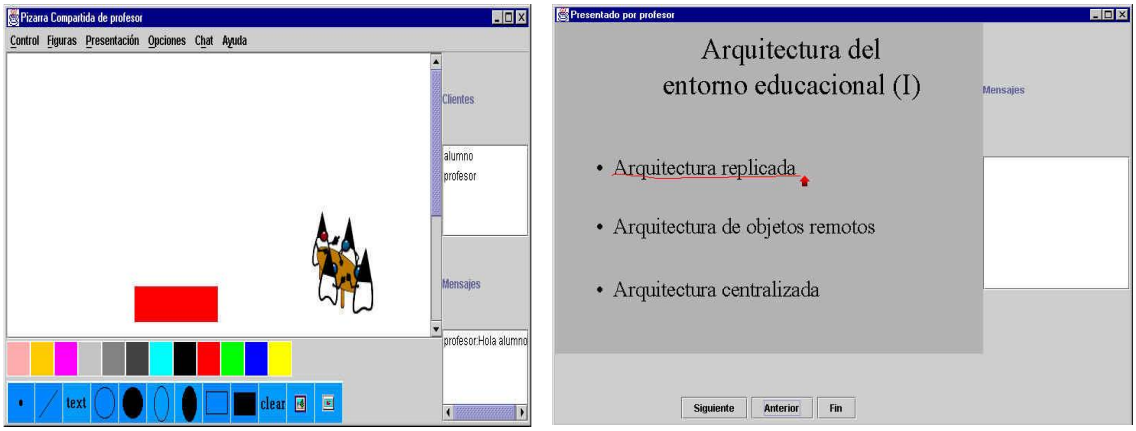

Figura 23. Herramienta de pizarra junto con herramienta de presentación

La comunicación de JSDT se basa en el uso de envío de eventos que son consumidos por los clientes interesados. El diseño de las herramientas colaborativas se ha basado en un modelo replicado sobre un servicio de notificación de eventos, de tal manera que los clientes mantienen una representación local de los datos y ante la recepción de nuevos eventos el cliente los recoge y los almacena en su representación local. Más concretamente, en el caso de la aplicación pizarra se almacenan el conjunto de figuras dibujadas en un objeto *Vector*.

En la figura 24 se muestra el diagrama UML de la aplicación *WhiteBoardUser*. Para el envío de eventos, se hará uso de los métodos *send* proporcionados en la clase *Channel* de JSDT. Para poder escuchar los eventos del canal, la aplicación usa una clase que implementa del *ChannelConsumer*, la cual proporciona un método *dataReceived* que servirá para tratar el evento recibido. Si bien existe la interfaz *ChannelListener*, la API también ofrece una clase abstracta *ChannelAdaptor* la cual implementa la anterior y cualquier clase que extienda de ella puede recibir los eventos de join, leave e invited entre otros.

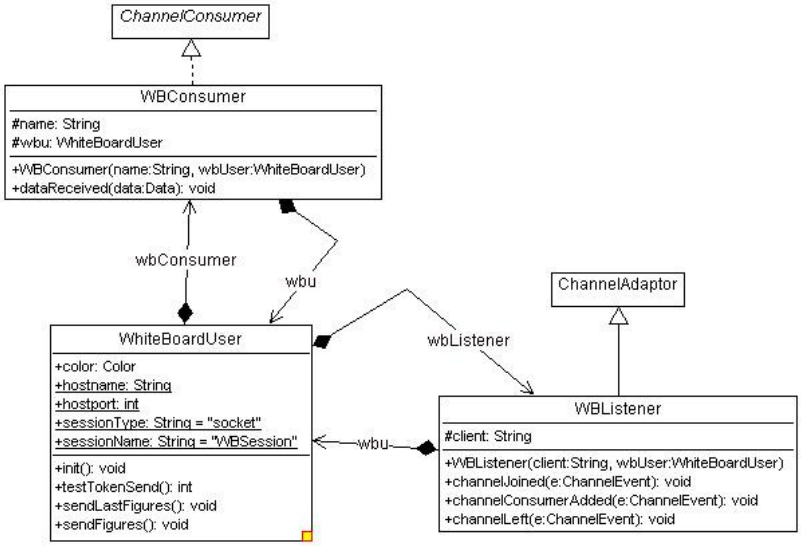

Figura 24. Diagrama UML de la aplicación pizarra.

En este tipo de integración la herramienta desarrollada es consciente de que la conexión se realiza usando jsdt ya que la herramienta estaba programada usando la API JSDT para la gestión de la colaboración.

A continuación expondremos cómo fue abordada algunas de las características colaborativas tanto en la gestión de la zona compartida así como los aspectos relevantes del diseño de la colaboración.

Presence Awareness.- Esta característica ha sido llevada a cabo basándonos en la API JSDT ya que ofrece un método para recoger los usuarios que están conectados llamado *listConsumerNames*. Con el objetivo de mantener dicha lista actualizada sin la necesidad de estar invocando al método cada cierto tiempo, mediante el listener *WBListener* se reciben los eventos de unión de usuarios con *channelJoined* y mediante el método *channelLeft* se trata la salida de los usuarios. Gráficamente, dicha lista de usuarios conectados es mostrado en un cuadro del panel.

Workspace Awareness.- Cómo ya hemos comentado anteriormente es importante saber que usuarios participan o han participado en la zona compartida. Para ello en el desarrollo gráfico de esta herramienta se incluyó la posibilidad de saber que usuario ha realizado cierta figura. De esta manera seleccionando el objeto deseado se puede obtener la identidad del creador de la figura. De esta manera se contempla de una manera sencilla el workspace awareness.

Llegada Tarde. En cuanto a la implementación del algoritmo de *late join* o llegada tarde se optó por un sistema distribuido para el mantenimiento del estado del área compartida. Debido a que no hay una gran cantidad de colaboradores dentro de un curso, se optó por que todos ellos fueran servidores del estado, de esta manera todos ellos deben recibir el estado de la aplicación lo antes posible. Mediante el uso del *WBListener* y el método *channelJoined()* se puede saber cuando un usuario llega tarde.

Para asegurarnos de que sólo un usuario accede a enviar el estado de la aplicación se hace uso de un token, el cual es grabado por el primer usuario que entra en la sesión. De esta manera, todos lo clientes ante la llegada de un nuevo cliente intenta acceder a una zona de código a la cual solo podrá acceder aquel que tenga el token. Una vez que éste accede al token envía al usuario correspondiente el estado actual de la pizarra.

Cuando se detecta que un usuario ha llegado nuevo, se comprueba el estado del token, si éste está libre se hace el grab. En el momento que el usuario nuevo añade un consumidor, el usuario dueño de este token envía al nuevo colaborador el estado de la sesión colaborativa. Si el usuario dueño de este token abandona la sesión previo a dejar todo, libera el token.

En cuanto a la política seleccionada es un *inmediate late join* en el cual se requiere todo el estado para la entrada en la colaboración. Si tras ciertos intentos el estado no puede obtenerse, la herramienta entra en la colaboración de todas formas.

Floor Control.- En cuanto al mecanismo para manejar el *Floor Control* en la aplicación desarrollada en esta integración, nos basamos en el uso del token asking, es decir el manejo de tokens proporcionado por JSDT. Las políticas a seguir en la coordinación del área compartida dependen de la tarea a realizar. Básicamente se controlo el envío de datos al usuario que llega tarde, el manejo de la presentación y otra de ellas es la zona de dibujo.

En cuanto a la primera de ellas, se hará uso de una política FCFS, puesto que el primero que solicita el token es el que lo recibe. Sin embargo no se mantiene una lista de las solicitudes, puesto que la lista de los usuarios cambia muy rápidamente y por tanto sólo se dará el token al primero que lo solicite.

En cuanto a la presentación el mecanismo de control ha sido realizado mediante bloqueo, de tal manera que aquellos usuarios que son alumnos (los receptores de una presentación) no pueden manejar los botones de control del manejo de las transparencias. Y en cuanto a la política de *Floor Control*, es mediante el uso de chair-guidance, ya que es uno de los usuarios, el profesor, es el que va manejando el control de la sesión.

En cuanto al área de dibujo se ha sincronizado el acceso al área mediante el uso del mecanismo token asking, de tal forma que aquel que sea el poseedor del token será el que pueda dibujar. Este token es solicitado tan pronto el usuario intenta dibujar cualquier objeto en la pizarra y se libera cuando el dibujo ha sido

terminado. Esta manera coordinada nos ayuda en la generación de identificadores en las figuras, puesto que se puede hacer de manera secuencial.

Otra forma de sincronización para que otros usuarios no borren lo que uno de ellos ha realizado es el bloqueo a nivel de objeto. Para realizar esto se ha introducido un valor de bloqueo en la figura. Si al dibujar una figura ésta intersecta con otra bloqueada se impedirá que se dibuje dicha figura.

En cuanto a la herramienta de chat se presenta con un control "free for all" de tal manera que todos los usuarios pueden escribir mensajes sin la necesidad de pedir turno.

El uso de JSDT proporciona una manera rápida y sencilla de crear herramientas colaborativas. Sin embargo esta API no garantiza una entrega fiable de los datos como hace JMS o Elvin. En cualquier tarea colaborativa en la que los usuarios deben percibir la misma información es de vital importancia que los sistemas usados nos garanticen una calidad de servicio en la distribución de mensajes. Concretamente, esta necesidad se acentúa en entornos educativos, en los cuales los alumnos deben percibir la misma información con el fin de poder adquirir las bases comunes del aprendizaje.

Por este motivo es de vital importancia usar un middleware de comunicación asíncrona que nos ofrezca un alto grado de calidad de servicio. De ahí que en nuestra propuesta se haya hecho uso de otras tecnologías más avanzadas.

El uso de diferentes tecnologías implica una diferente reestructuración en la forma en que las aplicaciones son programadas y en la forma en que reparten sus eventos. Por ello se hace necesaria la creación de una API de comunicación que permita crear herramientas o artefactos de colaboración de una manera sencilla y que a su vez ofrezca a las herramientas desarrolladas sobre ella, independencia de las tecnologías subyacentes para el envío de eventos.

Otra de las funcionalidades que se hacen necesarias en las herramientas síncronas es la de poder trabajar sobre una sesión previamente guardada, de tal manera que cualquier usuario pueda revisar o trabajar sobre los conceptos que se han explicado. Sin embargo, esta solución no es posible realizarse si el entorno dónde se realiza la integración no incorpora una API para la comunicación de ambos sistemas.

1.2. Integración de herramientas de seguimiento en JLE.

Entre las herramientas de principal interés en cualquier entorno educacional caben destacar aquellas que nos muestran la información acerca del uso de las aplicaciones por parte de los alumnos de la plataforma, mostrando información de cuanto tiempo han estado conectados, del acceso a los recursos y en definitiva la participación del alumno.

Otra de las investigaciones llevadas a cabo dentro de la plataforma JLE fue el estudio de su arquitectura para el soporte del sistema de logs, así como de los cambios necesarios a realizar en dicha estructura para que pudiera recoger la información adecuada para su posterior muestra en herramientas gráficas. Previo a este estudio, la información que se obtenía del JLE en una herramienta de seguimiento viene ofrecida en la figura 25.

Seguimiento visualización recursos

Hora Visita		Curso	Recurso	
Día	Hora	Nombre Curso	Tipo	Nombre
alupepa [Pepa Ayuso Pérez]				
06/09/00	10:31	Ciencias Sociales	text/html	PRINCIPAL.htm
06/09/00	10:32	Ciencias Sociales	text/html	UNIV.htm
06/09/00	10:32	Ciencias Sociales	text/html	Programaci%C3%B3n.htm
06/09/00	10:32	Ciencias Sociales	text/html	M-E.htm
06/09/00	10:32	Ciencias Sociales	text/html	EGIPTOYMESOPOTAMIA.htm
06/09/00	10:32	Ciencias Sociales	text/html	INTRODUCCI%C3%93N.htm
06/09/00	10:32	Ciencias Sociales	text/html	ACTIVIDADESI.htm
06/09/00	10:34	Proyecto EDUSI	text/html	index.htm
06/09/00	10:34	Proyecto EDUSI	text/html	ingles.htm
06/09/00	10:35	Proyecto EDUSI	text/html	eso101.htm
06/09/00	10:35	Proyecto EDUSI	text/html	voc1v3.htm
06/09/00	10:35	Proyecto EDUSI	text/html	voc1v3w.htm
06/09/00	10:35	Proyecto EDUSI	text/html	voc1v3c.htm
06/09/00	10:38	Proyecto EDUSI	text/html	ingles.htm
06/09/00	10:38	Proyecto EDUSI	text/html	prueba3m.htm
06/09/00	10:42	Proyecto EDUSI	text/html	prueba3m.htm
06/09/00	10:42	Tecnología. Unidad IV	text/html	index.html
06/09/00	10:43	Tecnología. Unidad IV	text/html	coche1.html
06/09/00	10:43	Tecnología. Unidad IV	text/html	poleas1.html
06/09/00	10:43	Tecnología. Unidad IV	text/html	vistas1.html
06/09/00	10:43	Tecnología. Unidad IV	text/html	eval.html

Figura 25. Herramienta de seguimiento en JLE v1.0

Como hemos mencionado anteriormente en el capítulo de antecedentes, uno de los puntos fundamentales en la interacción humano-computador (HCI) es el uso de un interfaz adecuado, mediante el uso de gráficos adecuados.

Si bien la herramienta de seguimiento ofrecía información acerca de los recursos que se accedían, no mostraba ni una interfaz adecuada ni toda la información relevante para poder inferir el comportamiento de los usuarios o del grupo grupos en relación a los recursos accedidos.

Uno de los parámetros interesantes a evaluar dentro de los sistemas CMS es la información cuantitativa del tiempo que ha empleado el alumno trabajando con los contenidos. Una de las posibles soluciones que se indicó fue estructurar los cursos por temas, de tal manera que se pudiera analizar el tiempo que el alumno tardaba en cada tema. Para recoger está estimación de tiempo, también fue necesario incluir en la B.D. la hora de entrada así como la hora de salida de la lectura del tema. Está información muestra el comportamiento de un individuo ante el sistema. Con estas indicaciones y mediante el uso de las librerías gráficas apropiadas se podría obtener unas herramientas como se muestran en las figuras 26

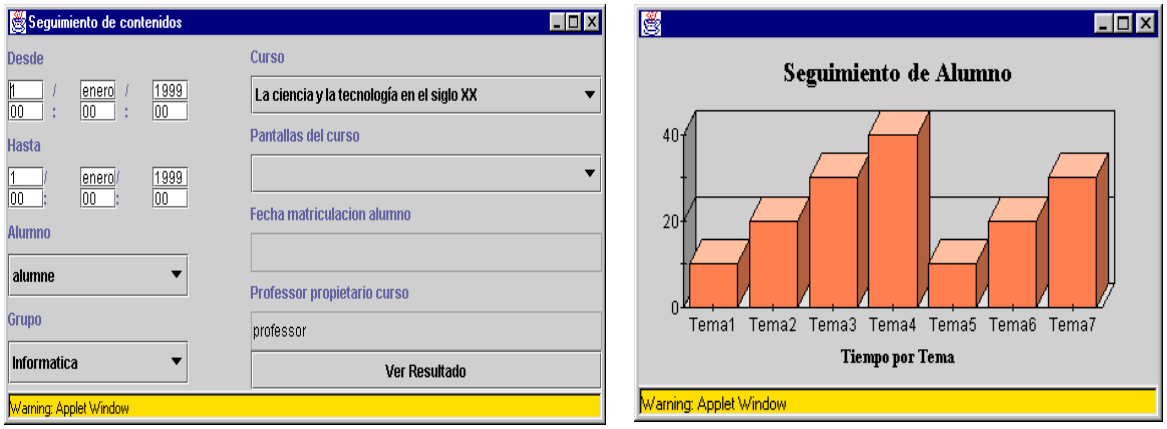

Figura 26. Esquema básico de un sistema de log

El profesor podría estar interesado en realizar el seguimiento de un determinado grupo de alumnos. En este caso y siguiendo con la necesidad de estructurar los cursos, el profesor podría conocer cual es el alumno que ha dedicado mayor y menor tiempo al estudio de cada tema, cual es el tema que más tiempo ha llevado al grupo en general, etc.

Otra de las formas de englobar y analizar a los usuarios es la introducción de una serie de gráficos modelo en el cual el profesor indique el tiempo que él estima que un alumno debería dedicar a cada uno de los temas del curso para poder así realizar comparaciones entre lo estimado y la realidad.

Dichas investigaciones fueron la base para un proyecto fin de carrera [Ayu01] en el cual se trabajó sobre la arquitectura ANTS puesto que es la actual arquitectura que sigue la nueva versión de JLE que ahora se llama EDUSTANCE.

1.3. Conclusiones.

En la integración de una pizarra colaborativa dentro del entorno educacional JLE se estudió la arquitectura de dicho sistema para la mejor integración de esta herramienta. Puesto que el sistema subyacente no ofrece ninguna API para llevar a cabo la colaboración, se hizo uso de la API JSDT sobre la cual se desarrolló la pizarra colaborativa.

En el desarrollo de esta aplicación tuvimos en cuenta a su vez ciertas características en el diseño de la colaboración como son Presence Awareness, *Floor Control* y un algoritmo de Late Coming.

Como ya se ha observado en capítulos previos el diseño de las herramientas colaborativas de comunicación síncrona viene dado por sistemas MOM puesto que resultan ser los más eficientes para comunicación entre varios colaboradores. Si bien dichos sistemas no garantizan una respuesta de la realización de una acción, a diferencia que los sistemas de comunicación síncrona, se requiere el uso de sistemas que tengan un alto grado de fiabilidad en el reparto de eventos, pudiendo así ofrecer ciertas garantías en la recepción de la información. Por ello tecnologías como JSDT resultan inadecuadas para el desarrollo de aplicaciones.

Además hemos podido comprobar que el desarrollo de una aplicación en una determinada tecnología, implica la modificación de la aplicación en el caso de cambio de la tecnología subyacente. Las tecnologías cambian constantemente, ofreciendo nuevas posibilidades así como mejoras en el reparto de información de unas a otras. Por lo tanto se hace necesario la creación de una API genérica que permita el desarrollo de aplicaciones colaborativas independientemente de la tecnología subyacente para el reparto y suscripción de eventos. Este punto constituye una de las principales líneas abarcadas en nuestro trabajo de investigación.

Otro de los puntos interesantes a abordar es el uso de herramientas que permitan tanto la colaboración síncrona como la colaboración asíncrona. Si bien la colaboración síncrona es llevada a cabo mediante el uso de tecnologías MOM, para la colaboración asíncrona se hace necesario el uso de una API desde el entorno educacional que permita la comunicación de ciertas operaciones desde la herramienta de colaboración hacia el

entorno. Para el desarrollo de esta API se hace patente el uso de middleware de comunicación síncrona.

Cómo hemos mostrado, en el estudio llevado a cabo dentro de JLE no sólo se ha mostrado la integración de herramientas colaborativas sino que se han ofrecido una serie de mejoras en el sistema de logs con el objetivo de hacer uso de ciertas herramientas gráficas para facilitar las tareas de análisis dentro del sistema de logs. Toda la interacción ordenador usuario se ve ampliamente influenciada por el uso de interfaces gráficas apropiadas.

Dicho estudio fue conducido a través de un proyecto final de carrera en el cual se analizó una mejora del sistema de logs existente y se integró dicho estudio en la plataforma de colaboración ANTS.

2. Plataforma ANTS.

Otra de las plataformas en las cuales hemos investigando y en la que hemos participado activamente, es la plataforma ANTS la cual constituye el resultado de la tesis del doctor García [Gar02]. En esta arquitectura se estudió la integración de mecanismos de *Floor Control*, la introducción de un componte pizarra dentro de dicha arquitectura así como la construcción de una aplicación la cual integrará la conexión con diferentes aplicaciones mediante el uso de estándares de comunicación. También se colaboró en el estudio de un sistema Awareness [Garb02] y de un sistema de logs para la arquitectura de EDUSTANCE.

A lo largo de este apartado podremos apreciar como se llevaron a cabo las dos implementaciones, tanto la del componente a instalar dentro de la plataforma así como una aplicación que usa la plataforma para acceder a los componentes configurados en cada una de las sesiones.

Esta plataforma constituye la base arquitectónica de la herramienta educacional EDUSTANCE, así como de otros proyectos de investigación.

En apartados anteriores hemos mostrado una descripción general de la arquitectura de la plataforma ANTS. En este apartado veremos aquellos aspectos relevantes de dicha arquitectura a la hora de incluir herramientas, así como de hacer uso de ellas. Veremos como la plataforma ANTS ofrece una serie de módulos que permiten el desarrollo de herramientas colaborativas, sin el conocimiento del tipo de Middleware asíncrono utilizado. Una de las características de la platforma ANTS es que para el desarrollo de aplicaciones distribuidas está basada en el modelo de programación de JavaBeans.

2.1. Integración de un componente pizarra.

Como hemos podido observar en apartados anteriores la arquitectura ANTS se estructura en cuatro capas, véase figura 27. Una de ellas es la de red, la cual queda implementada por el uso de J2EE, otra es la de adaptación la cual constituye la base para la comunicación a través del Bus de eventos. Más concretamente esta API pretende abstraer del uso del servicio de notificaciones empleado a las capas superiores ANTS CORE y ANTS AWS.

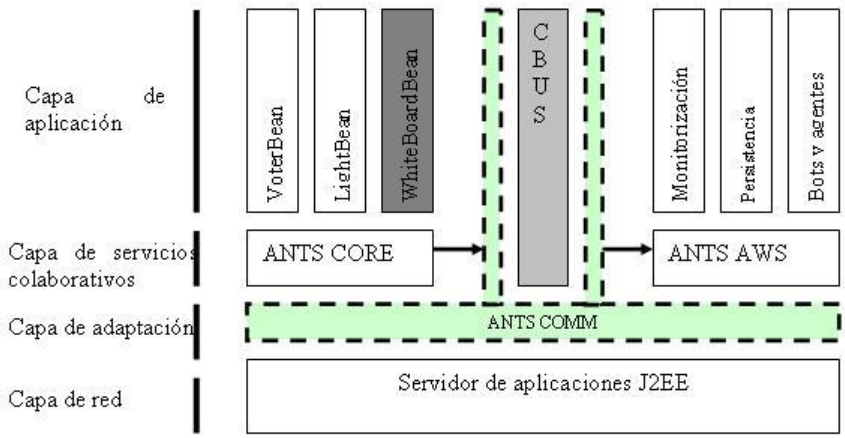

Figura 27. Arquitectura modular de la plataforma ANTS.

Así pues ANTS CORE, la cual constituye la API de desarrollo para las aplicaciones colaborativas, proporciona el uso de los servicios ANTS.COMM sin que el programador de las aplicaciones tenga que decidir sobre que servicios se realizará el transporte de los mensajes. Esta característica hace de ANTS.COMM una de las bases sobre la cual fundamentar cualquier canal de eventos en aplicaciones distribuidas que requieran la propagación de su estado entre las demás aplicaciones.

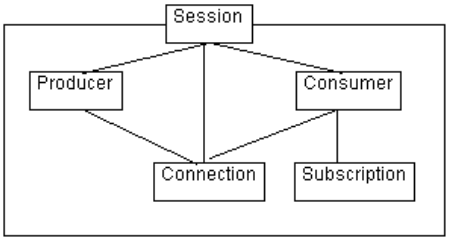

Figura 28. Fachada de ANTS.Comm

Como se puede apreciar en la figura 28, el diseño de ANTS.COMM se basa en el concepto publish/suscribe y provee una abstracción del manejo de sesiones abstrayendo la información de la tecnología MOM subyacente. Para ello proporciona la clase *Session* a través de la cual se envian las notificaciones mediante los métodos *send* y la interfaz *RemoteEventListener* que a través de su implementación se puede diseñar el tratamiento de las notificaciones recibidas.

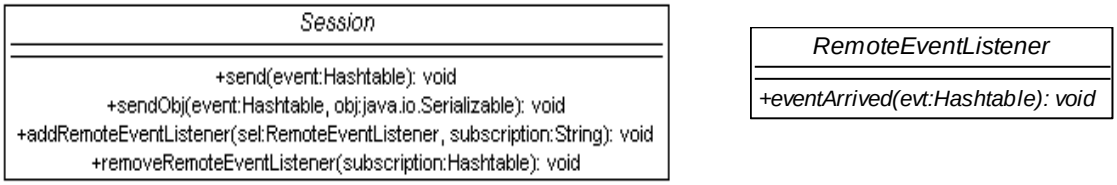

Figura 29. Descripción UML de la clase Session de ANTS.COMM.

Como se puede observar del diagrama UML representado en la figura 29, la notificación así como la suscripción son representadas mediante el uso de Hashtables. Un Hashtable es un diccionario de pares clave-valor que resulta muy apropiado para la representación de eventos. Así pues las suscripciones dentro de ANTS.COMM son representadas por los Hashtables, implicando la conjunción de todas las claves y valores (key1=v1 & key2=v2) a la hora de realizar el lenguaje de filtro del sistema de notificación subyacente.

Otra de las características de esta arquitectura es su modelo de programación, inspirado en las especificaciones de JavaBeans, la cual comprende mecanismos de persistencia utilizando serialización de objetos, introspección utilizando reflexión y descriptores de componentes, y empaquetado mediante ficheros compactados *jar* (java archive file).

Esta solución propone que el estado de los componentes se mantenga con un sistema de propiedades remoto localizado en el servidor, y que los eventos se propaguen de manera automática por el bus distribuido de forma transparente a los clientes. La persistencia de los componentes será gestionada por el sistema de propiedades, la introspección se

realizará utilizando reflexión y descriptores XML, y el empaquetado también utilizará los ficheros empaquetados *jar*. Por lo tanto el sistema ANTS ha mapeado los conceptos y elementos diseño de JavaBeans a un escenario distribuido. A continuación estudiaremos el modelo de componentes y su mapeo al contenedor ANTS.

Propiedades (properties) definen el estado del componente y las características. Son accesibles mediante los constructores de las herramientas visuales usando ficheros XML (introspección). El modelo JavaBeans distingue entre propiedades normales e indexadas y define convenciones estándares de nombrado para definirlas. Además se definen dos categorías: propiedades vinculadas ("bound properties") y propiedades restringidas ("constrained properties"). Las primeras proveen notificación de los cambios de propiedades para suscriptores registrados y las restringidas permiten que los objetos subscriptores aprueben o veten cambios en esas propiedades.

Eventos (events) definen el comportamiento del componente y también son obtenidas mediante el uso de introspección en la construcción de los componentes. El modelo Java de delegación de eventos está basado en patrones de observación y permite la suscripción y notificación de eventos. De igual manera que las propiedades, la especificación JavaBeans define una convención en el uso de los nombres para la suscripción y desuscripción de eventos sobre un componentes.

Más concretamente esta arquitectura provee mecanismos basados en objetos distribuidos EJB para almacenar en base de datos el estado de las propiedades y emular el servicio distribuido de eventos java usando un sistema de notificaciones.

Por último, para instalar un componente en la plataforma necesitaremos empaquetarlo en ficheros *jar* incluyendo un fichero descriptor en el lenguaje de definición de componentes.

Al igual que en los sistemas MOO en la arquitectura ANTS todos los objetos heredan de *Thing* y pueden así acceder al sistema persistente de propiedades. El diagrama de esta clase viene mostrado en la figura 30. El sistema de gestión de sesiones se basa en tres clases fundamentales: *World*, *Place* y *Link*. El objeto *World* no es más que una entidad de agrupación de sesiones donde podemos aplicar políticas de coordinación de manera conjunta. La sesión compartida es representada por el objeto *Place*. El *Place* es una clase esencial en la arquitectura propuesta pues define el contexto común donde interactuarán usuarios con componentes o artefactos y donde se aplicarán políticas de coordinación. Un *Link* tiene una sesión de origen y una sesión de destino. Los *Links* o puertas sirven como conexión entre sesiones y son la base a sistemas situacionales avanzados como entornos de realidad virtual.

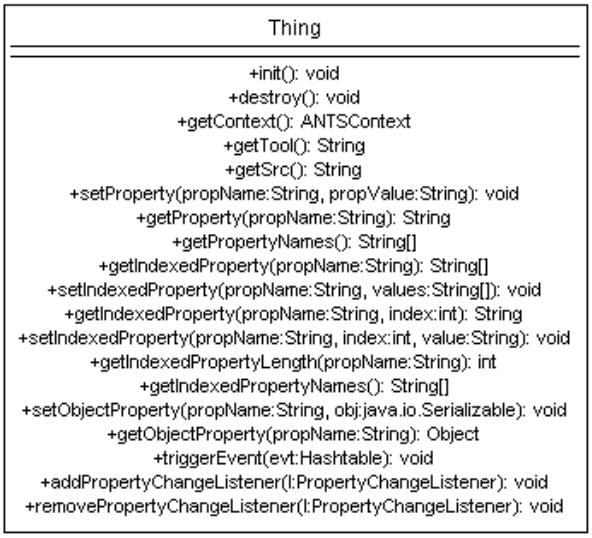

Figura 30. Representación UML de la clase Thing.

Más concretamente, la abstracción de Place será la que nos permita realizar la comunicación de las herramientas colaborativas (véase figura 31). A partir de su API podemos observar grandes similitudes con la que ofrece JSDT. Básicamente esta última constituyó la base principal para la construcción de la API ANTS.CORE debido a su sencillez y su manejo a la hora de programar nuevas aplicaciones.

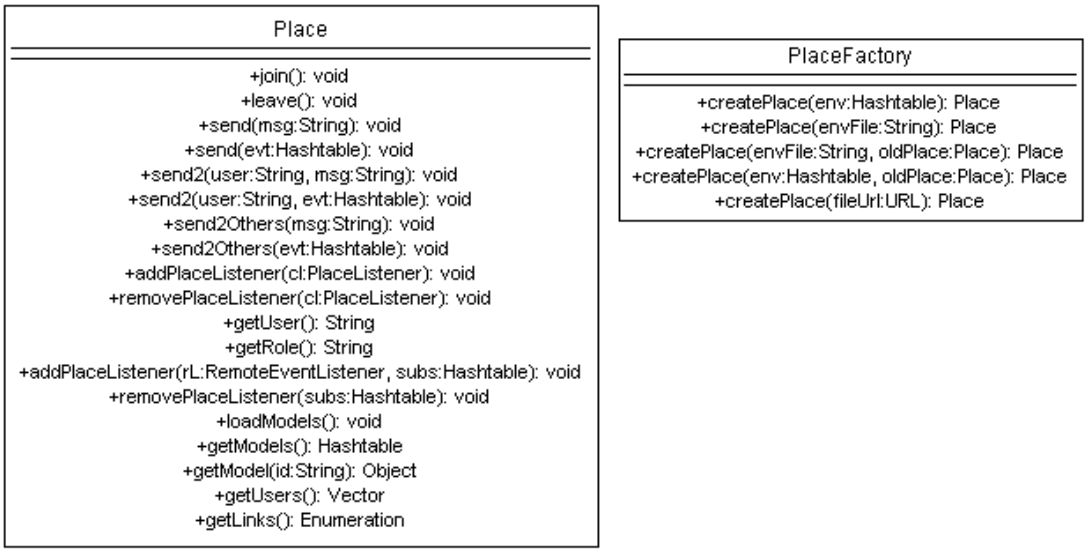

Figura 31. Representación UML de las clases relacionadas con la abstracción de Place.

Otra de las características de este sistema es el empleo de modelos, de tal forma que la aplicación trabajará con un modelo de componente sobre el cual realizará las operaciones más importantes. En la figura 32 se pone de manifiesto el ciclo de vida de los componentes, una vez que se establece la conexión en el *Place*, sesión de colaboración. Si bien estos modelos están disponibles dependiendo del *Place* en el que se desee colaborar, también es necesario que desde la aplicación se haga uso de las herramientas visuales para poder manejar estos modelos.

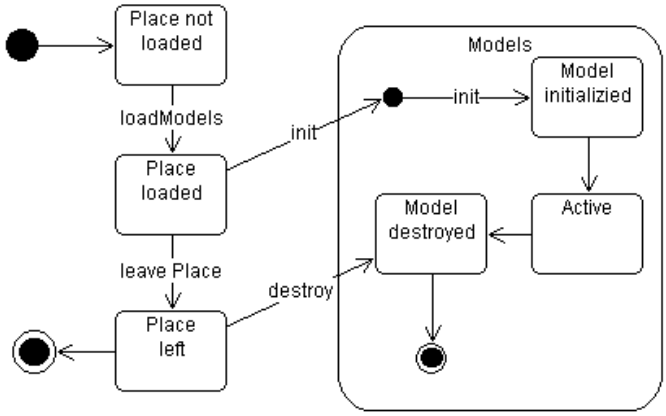

Figura 32. Ciclo de vida de un componente

Una vez explicados los conceptos generales de la plataforma ANTS pasamos a describir los elementos que se han incluido para la introducción de un nuevo componente el cual consiste en una herramienta de pizarra.

Familiarizados con la plataforma ANTS, la inclusión de nuevos componentes se plantea de la siguiente manera. Siguiendo el modelo de programación de componentes impuesto en esta arquitectura, previa a la construcción de la aplicación pizarra (*WhiteBoardBean*) ha sido necesario la creación de un modelo, más concretamente la clase *WhiteBoardModel* mostrado en la figura 33. Dicho modelo proporciona los métodos básicos para la propagación de las propiedades y eventos necesarios por la aplicación pizarra. Además gestiona la suscripción y desuscripción de los listeners interesados en recibir las notificaciones referentes al modelo, los cuales necesariamente deberán implementar *WhiteBoardListener* figura 33, mediante el cual se comunican ciertos eventos, más concretamente el *drawObject* y *drawColor*, para poder tratar el objeto nuevo dibujado así como los cambios de colores que ocurran en la aplicación pizarra respectivamente.

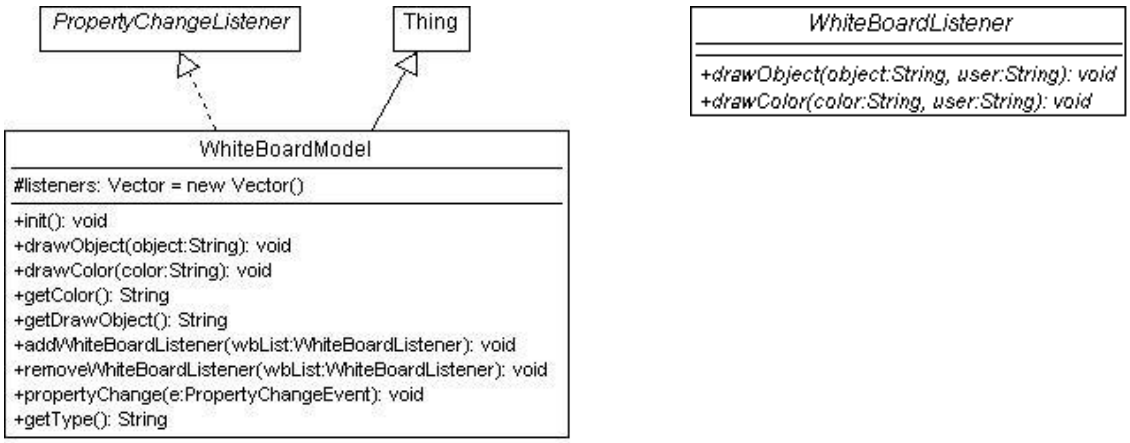

Figura 33. Representación UML de las clases que constituyen el componente whiteboard.

Como cualquier componente, es necesario proporcionar un descriptor que defina al componente. Este fichero, que será definido usando el formato XML, será empaquetado en el jar junto con las clases del componente.

En la figura 34 se muestra el diseño global de la aplicación pizarra *WhiteBoardBean* la cual mediante el uso del modelo *WhiteBoardModel* propaga sus eventos, drawObject y drawColor, los cuales son recibidos y tratados por otros objetos en su implementación del *WhiteBoardListener*, más concretamente en nuestro caso lo trata el *WBModelListener*. Sin embargo a través de este modelo no se envía el objeto, puesto que como podemos apreciar *setProperty* de la clase *Thing* sólo admite objetos de la clase String.

Una posible solución sería convertir el objeto de dibujo en xml. Pero surge el problema con los objetos imágenes ya que estos vienen representados por un array de bytes más una serie de parámetros que indican el valor del objeto. Por tanto, en la realización de dicha aplicación optamos por usar el método *onMessage* que nos proporciona el *PlaceLisener* y desde este método analizar el tipo de objeto recibido. Teniendo en cuenta lo anterior y con el objetivo de poder escuchar cuando un usuario se une o deja el *Place* se ha incluido un listener, *WBListener*, el cual nos servirá para asegurar la característica de Presence Awareness como podremos comprobar posteriormente y tratar los mensajes que lleguen en *onMessage*.

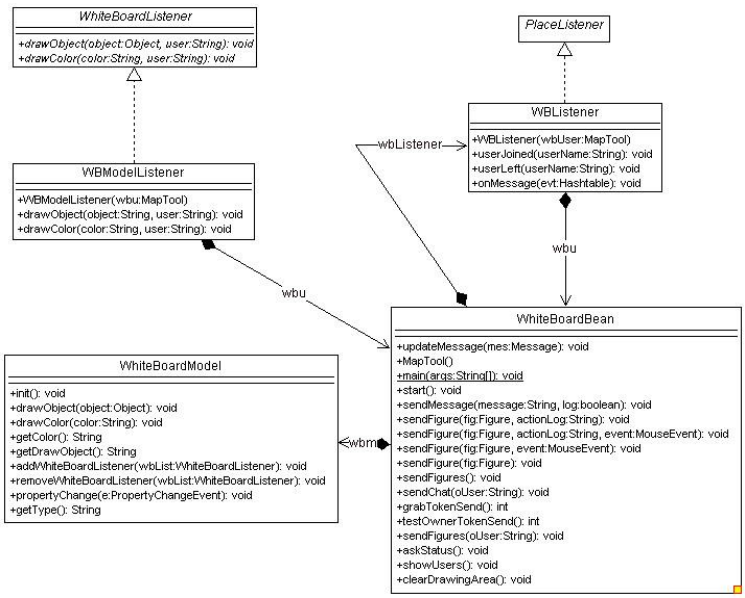

Figura 34. Diagrama UML del bean whiteboard.

Siguiendo el modelo de la arquitectura ANTS el programador se abstrae del tipo de servidor de notificaciones usado. En cuanto a los parámetros que necesita la aplicación para establecer la sesión son mostrados en la figura 35.

```
env.put(ants.core.Context.ROLE,"sysadmin");
env.put(ants.core.Context.PLACE,"wbplace");
env.put(ants.core.Context.WORLD,"wbworld");
env.put(ants.core.Context.BASE,"http://127.0.0.1:80/ants/");
env.put(ants.core.Context.ANTS_SERVER,"ormi\\://127.0.0.1/ants");
```

Figura 35. Esquema de conexión del bean whiteboard en la arquitectura ANTS.

Más concretamente el tipo de comunicación asíncrona que se emplea dentro del place vendrá definida por la entidad World dónde se encuentre dicho place.

En cuanto a las características colaborativas de la aplicación se estructuró de manera similar a la integración realizada en JLE explicada anteriormente. En esta integración no se cuenta con la introducción de una herramienta de presentación ya que la plataforma ANTS ya incluía una.

Presence Awareness. Al igual que se explicaba en la integración de JLE, se obtiene la lista de usuarios invocando sobre el place el método *getUsers*. Esta lista va cambiando conforme se unen nuevos usuarios o dejan el place o sesión. Este cambio de información es gestionado mediante el *WBListener*.

WorkSpaceAwareness. Esto es manejado mediante el uso de un puntero en el cual se muestra el usuario que está realizando cierta tarea.

Llegada tarde. Se mantiene un esquema distribuido de servidores de estado de la pizarra. Para sincronizar el acceso al envío del estado se hace uso de un elemento token. Y la forma de participación es basada en una política *inmediate late join*, en la cual el usuario intenta obtener todos los datos de la sesión actual. Esto constituye un punto importante ya que todos los colaboradores son servidores de estado.

Floor Control. De igual manera que se ha comentado anteriormente, se aplicaran politicas de *floor control* para la coordinación de quién será el responsable de suministrar la información de estado a aquellos usuarios que lleguen tarde a la sesión al igual que el acceso del área compartida.

En cuanto a la primera de ellas, se hará uso de una política FCFS, puesto que el primero que solicita el token es el que lo recibe. Sin embargo no se mantiene una lista de las solicitudes, puesto que la lista de los usuarios es muy cambiante y por tanto sólo se dará el token al primero que lo solicite.

En cuanto al área de dibujo se ha sincronizado el acceso al área mediante el uso del mecanismo token asking, de tal forma que aquel que sea el poseedor del token será el que pueda dibujar. Este token es solicitado tan pronto el usuario intenta dibujar cualquier objeto en la pizarra y se libera cuando el dibujo ha sido terminado.

2.2. Arquitectura genérica para soportar el Aprendizaje por Descubrimiento basada en la plataforma ANTS.

Una de las ramas de nuestra investigación está enfocada en el desarrollo de una plataforma para el Aprendizaje por Descubrimiento. Uno de los primeros diseños e implementaciones que abordamos se fundamentaba en la plataforma ANTS. Debido a que esta plataforma aborda la colaboración síncrona como asíncrona favorece los dos **Escenarios de Colaboración** posibles de colaboración dentro del Aprendizaje Colaborativo.

Otra de las características que la hacían interesante es el uso de XML para la configuración de los componentes. Con el uso de un modelo adecuado se puede favorecer un **Marco de Referencia Común**, haciendo que diferentes herramientas puedan mostrar la misma información de diferentes formas.

Como ya hemos podido comprobar esta plataforma permite la creación de sistemas colaborativos basándonos en el lenguaje Java. Sin embargo, las exigencias de algunos proyectos o plataformas implican la utilización de diferentes lenguajes de programación porque las aplicaciones ya están construidas y reprogramarlas conlleva un nuevo coste. Más concretamente, en el desarrollo del proyecto Colab [Col04] se planteó el uso de ciertos laboratorios previamente desarrollados por el instituto AMSTEL los cuales estaban programados en otros lenguajes, como es C y Pascal.

Por lo tanto, nos tuvimos que plantear el diseño de una arquitectura heterogénea basándonos en ANTS para englobar el conjunto de herramientas colaborativas y la comunicación entre ellas. En definitiva lo que se pretendió fue como abordar el diseño del **Espacio de Experimentación**, de tal forma que los resultados obtenidos en un experimento pudieran ser visualizados en cada una de las aplicaciones de los colaboradores conectados.

El aspecto principal en el desarrollo de esta arquitectura genérica lo constituyó la existencia de múltiples herramientas desarrolladas en distintos lenguajes de programación que necesitaban ser integradas en el sistema como en una caja de herramientas, de tal forma que para el usuario final esta diferencia no quedará visible y no le implicase instalaciones adicionales. Esto implicó la elección de una capa intermedia que permitiera 'pegar' todas estas herramientas como un puzzle.

En el diseño de esta arquitectura nos encontramos con múltiples soluciones, pero nos enfrentábamos al desafío de buscar una arquitectura que provocase el menor impacto sobre las herramientas ya desarrolladas y nos permitiese desarrollar las nuevas sobre la plataforma ANTS. A continuación exponemos el modelo de programación en el que nos basamos así como dos posibles soluciones que tuvimos en cuenta para llevar a cabo la programación de esta plataforma genérica.

Modelo de programación.

El modelo básico de programación que propusimos consistía en una simplificación del Reference Model for Open Distributed Processing RM-ODP [ODP]. En el modelo que plateamos una operación puede ser o bien una operación pregunta-respuesta o una notificación. El primer caso corresponde a una operación realizada con el paradigma síncrono siguiendo un modelo RPC mientras que el segundo corresponde al paradigma asíncrono siguiendo un modelo de comunicación de servidor de notificaciones.

XML-RPC fue la elección para llevar a cabo la filosofía RPC, puesto que es una manera de comunicación simple y portable para hacer llamadas a procedimientos remotos sobre HTTP. Básicamente esta tecnología nos servirá como punto de comunicación con las herramientas ya diseñadas y el nuevo sistema. Para esta integración se requería que se construyese en Java una interfaz para la comunicación entre nuestro sistema y la aplicación ya existente a través de XML-RPC con la aplicación.

Puesto que el resto de herramientas serán desarrolladas bajo la plataforma ANTS, la tecnología para la comunicación asíncrona se baso en el uso de la API ANTS.CORE, haciendo uso de su servicio de comunicación asíncrona ANTS.COMM.

Basándonos en los conceptos anteriores diseñamos dos arquitecturas posibles para llevar a cabo la integración, siendo el primer modelo que presentamos el más adecuado y el elegido. A continuación exponemos los dos diseños planteados.

Diseño de la arquitectura

En este primer diseño se plantea el uso del paradigma Vista/Control para la inclusión de las herramientas ya existentes. De tal forma que se realice la codificación en Java de la parte gráfica de las aplicaciones ya existentes haciendo uso de XML-RPC para llevar a cabo la ejecución de la parte lógica de estas aplicaciones.

La API ANTS.CORE será el bus de eventos para transmitir anuncios, especialmente para controlar los eventos de interfaz. El esquema de la figura 36 muestra el primer diseño realizado de esta arquitectura.

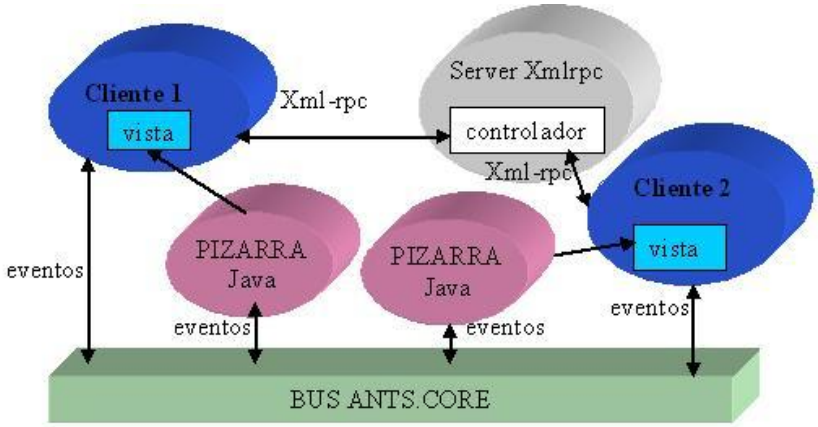

Figura 36. Arquitectura Génerica siguiendo el modelo vista/control y vistas en Java.

Sin embargo, el esquema previamente diseñado no es el más adecuado para servir a herramientas de colaboración síncrona que se conectan con simuladores, ya que el acceso de las diferentes herramientas al simulador producen diferentes valores en sus llamadas XML-RPC, puesto que éstas se ejecutan en distintos instantes y por tanto la simulación da diferentes resultados. Así pues, siguiendo este esquema las herramientas colaborativas ofrecen diferentes valores en sus visualizaciones rompiendo uno de los principios básicos del área compartida (WISIWITYS) y por tanto ofreciendo diferente información a los colaboradores.

Un ejemplo claro de esta desincronización era la visualización de la ejecución de un resorte. Puesto que cada cliente preguntaba por la posición del resorte en distintos instantes, se obtenía diferentes valores y por tanto la visualización estaba desincronizada.

Para solucionar este problema se planteo otro diseño, mostrado en la figura 37, en el cual sólo uno de los clientes, el que arranca la herramienta, establecía la conexión con el servidor mediante el uso de XML-RPC. Conforme esta herramienta obtenía el resultado del experimento (p.e. la posición del resorte), lo distribuía a través del Bus de eventos, de tal forma que los demás clientes escuchaban estos eventos y por tanto actualizaban los valores de su visualización. De esta manera, ambos clientes mostraban la misma información ya que trabajaban con los mismos valores del experimento, cumpliendo el principio WISIWITYS.

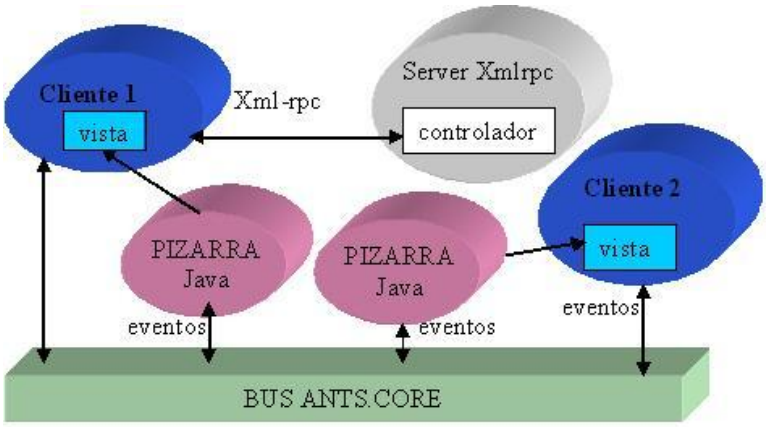

Figura 37. Arquitectura Genérica siguiendo el modelo vista/control y vistas en Java.

Si bien el anterior diseño es el más adecuado puesto que ofrece una solución de comunicación más eficiente en el entorno heterógeneo, existen ciertas herramientas en las que no se puede diseñar las interfaces visuales java.

Con el objetivo de que estas herramientas puedan comunicarse con el entorno y con el resto de herramientas que incorpora el sistema, el diseño se amplió mediante el uso de proxys de tal forma que el intercambio de eventos entre las herramientas y el bus ANTS.CORE se realice mediante estos proxys. Cada uno de ellos dispone de dos interfaces: una para la comunicación con ANTS.CORE y la otra con la herramienta. Esta segunda interfaz podía ser o bien una conexión TCP/IP, la cual es la mejor solución para obtener eficiencia en el flujo de eventos, o también podía ser una interfaz XML-RPC, seguramente más lenta pero más fácil de usar.

En esta solución, debido a que las herramientas no estarían totalmente integradas en la aplicación principal, para que se pudiera realizar la colaboración y todos los usuarios pudieran tener las mismas vistas, todas estas herramientas deberían estar instaladas en cada uno de los ordenadores de los colaboradores, y deberían mandar eventos entre ellas para sincronizarse para poder escuchar y reaccionar ante los mismos eventos.

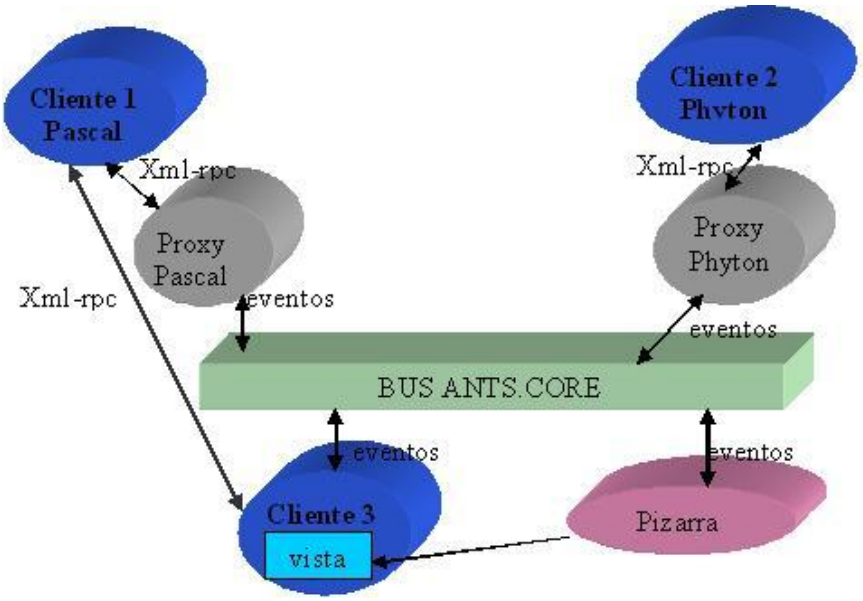

Figura 38. Arquitectura Génerica siguiendo el modelo de proxy.

Como podemos apreciar en la figura 38, en este tipo de diseño no se obtiene una vista totalmente integrada de las herramientas. Por lo cual se recomendaba codificar el mayor número de herramientas siguiendo el esquema visto previamente, usando este esquema sólo en aquellas herramientas que por su naturaleza no permita la recodificación.

Aplicación Final

A continuación expondremos la implementación de una aplicación que desarrollamos basándonos en el primer diseño expuesto para esta arquitectura. A través de esta implementación analizaremos cuales fueron las causas que nos llevaron a diseñar otra arquitectura para llevar a cabo la experimentación colaborativa.

La aplicación desarrollada consiste en una pizarra colaborativa que permite abrir o bien otras herramientas configuradas en el place dónde se realiza la conexión o aquellas herramientas visuales que están configuradas para establecer una comunicación XML-RPC con el simulador del experimento.

Siguiendo el modelo de comunicación expuesto por la arquitectura ANTS, la conexión se llevaba cabo mediante el uso del concepto de *World* y *Place*. La herramienta final pretende obtener la información relacionada con las herramientas que están disponibles en un place y poder lanzarlas al comenzar la sesión. En la figura 39 se muestra un ejemplo de la visualización de la aplicación final.

Esta herramienta hará uso tanto de colaboración síncrona, como de la colaboración asíncrona, está ultima vendrá dada para la recuperación de información de sesiones anteriores.

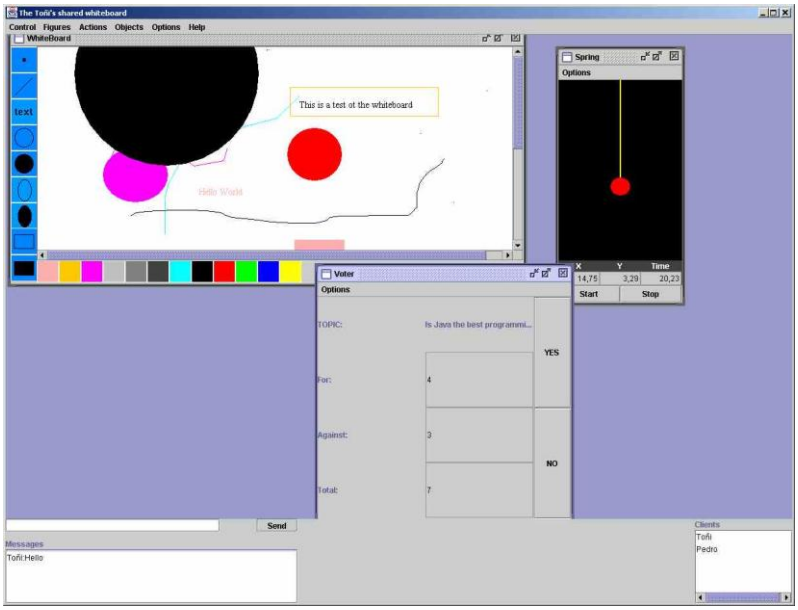

Figura 39. Aplicación pizarra COLAB sobre la plataforma ANTS.

Entre las herramientas que se incluyen en dicha aplicación se ofrece **capturador colaborativo**, mediante el cual se captura cualquier imagen del ordenador del cliente actual y se envía a los demás colaboradores. Puesto que las herramientas de esta

aplicación deben guardar el estado, la forma de guardar el estado de dicha aplicación será haciendo uso de un programa servidor que cree la copia de la imagen en un directorio público del Web (*/world/place*) el cual guarde dicha imagen. Con el objetivo de que dichas imágenes sean visibles en los demás clientes, una vez que dicha información ha sido guardada en el servidor, se les envía a los demás la url de dicha imagen.

Por lo tanto el servidor estará a la escucha de los eventos que pueda generar el capturador para la creación del fichero en el *world* y *place* correspondientes a la sesión dónde esta herramienta ha sido lanzada. Como ocurre con los objetos figura de la pizarra, esta herramienta hará uso del place también para el envío de los bytes asociados a la imagen. Puesto que los bytes de la imagen pueden no enviarse en un solo evento, se mandan porciones hasta que se complete. De igual forma el servidor irá guardando las porciones que se reciban en el fichero correspondiente a la sesión hasta que se finalice la transmisión. Dicho esquema es mostrado en la figura 40.

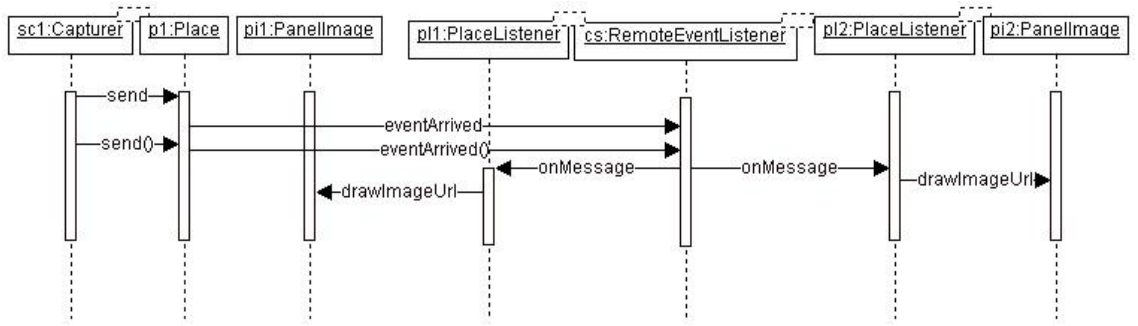

Figura 40. Diagrama de secuencia del capturador de imágenes colaborativo.

En cuanto a la aplicación pizarra otra de las cuestiones a abordar es el guardado de la información dentro de la sesión. Si bien herramientas como el *Voter* pueden guardar su estado a través del uso de modelos impuestos en ANTS, la pizarra posee ciertos elementos, como son las imágenes, cuyas propiedades no pueden ser convertidos o tratados como objetos del tipo String. Teniendo esto en cuenta, ha sido necesario incorporar un servicio adicional el cual realice el guardado de la información en ciertos directorios del servidor, indicando el *world* y *place* dónde se lleva a cabo la sesión.

El ciclo de vida de la aplicación sigue el esquema mostrado en la figura 41, mostrando todos los pasos en la fase de conexión y los pasos cuando se requiere la desconexión de la aplicación.

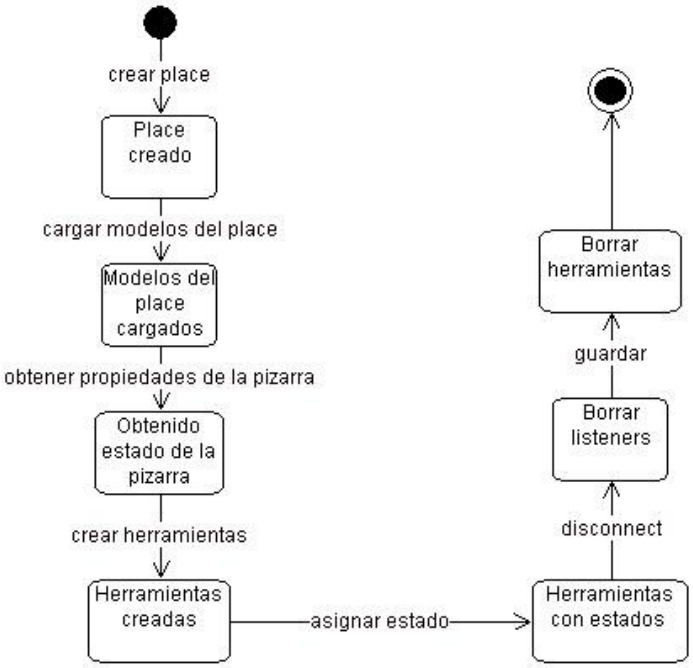

Figura 41. Ciclo de vida de la pizarra COLAB.

El desarrollo de esta aplicación sigue el diseño expuesto en el modelo Figura 39, incluyendo una herramienta resorte que es externa a la arquitectura ANTS y de la cual se ha incluido una interfaz gráfica basada en Java para comunicarse con la herramienta externa.

Como hemos visto anteriormente la solución más adecuada desde el punto de vista de la colaboración síncrona es que la conexión XML-RPC la realice sólo el usuario que arranca la vista de la herramienta, en nuestro caso el resorte. Una vez que este cliente recibe los eventos del simulador, es el encargado de enviarlo a los demás usuarios del grupo mediante el uso de ANTS.CORE. De esta manera, todo el mundo visualizaba la misma información y casi al mismo tiempo, solucionando el problema de la de-sincronización en las vistas. Sin embargo este modelo implicaba el conocimiento de la arquitectura ANTS en cuanto al desarrollo de nuevos componentes, lo cual resultaba una desventaja bastante importante para aquellos programadores provenientes de otros ámbitos como pueden ser la física. El ejemplo del resorte es ilustrado en la figura 42.

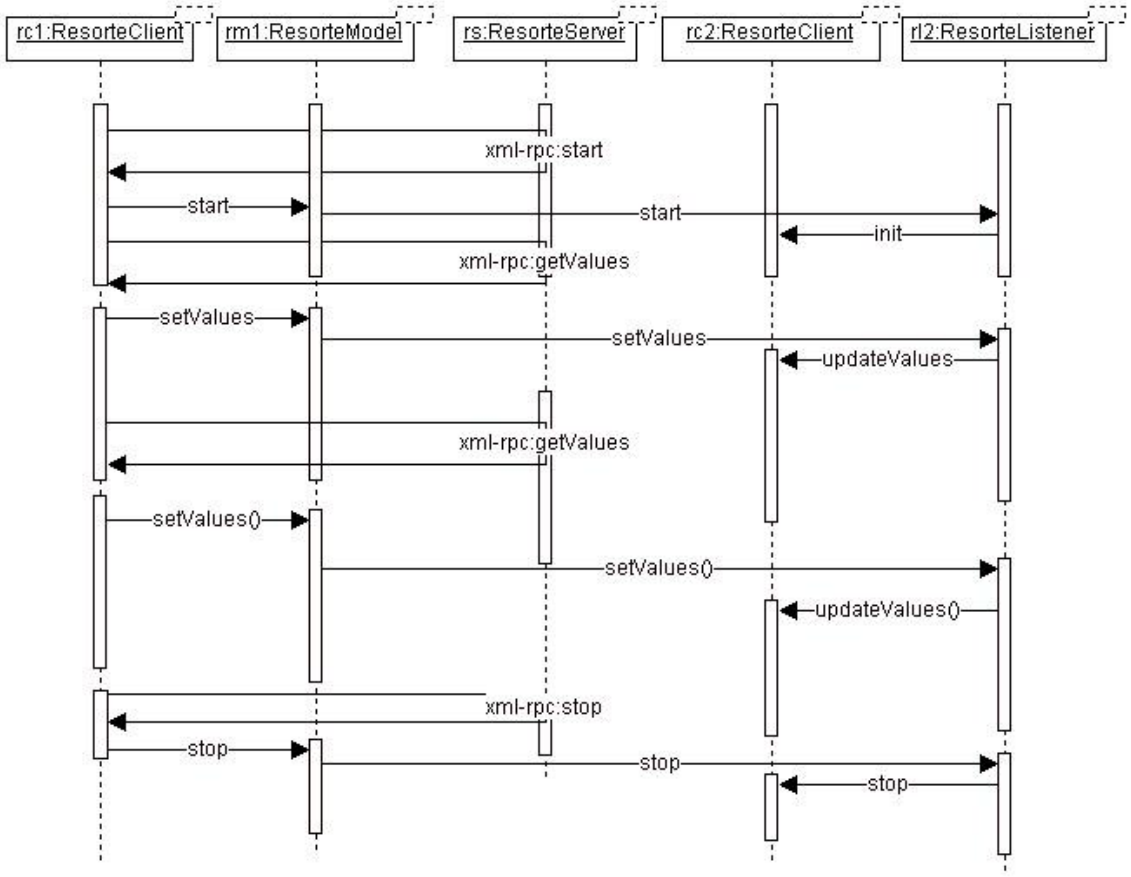

Figura 42. Diagrama de secuencia de la aplicación resorte.

La integración de nuevos componentes o herramientas en la arquitectura ANTS implicaba la recompilación de ANTS.CORE y su posterior despliegue. Sin embargo, lo ideal podría ser incluir una nueva aplicación incluyendo el nuevo jar en el servidor de aplicaciones, cambiando ciertos ficheros de configuración y poder dar de alta la nueva herramienta en la aplicación final sin la necesidad de tener que recompilar todo el sistema.

Pese a su modelo altamente estructurado y modular, la complejidad de la programación de nuevas herramientas dentro de la plataforma ANTS puede verse incrementada fuertemente por el hecho de no estar familiarizados con las tecnologías JavaBeans y el manejo de la colaboración.

Además su modelo de propiedades basadas en parámetros String, resulta no ser el más adecuado para aquellas aplicaciones dónde la mayoría de los datos a tratar dependen de valores binarios, como puede ser los valores resultados de una experimentación.

El hecho de que los artefactos o herramientas no sigan el esquema impuesto de los modelos implica que desde el place y su operación *loadModels* ya no se pueda cargar los ficheros de todos estos objetos, de manera similar que ocurría con la pizarra y el guardado de su estado, perdiendo de esta manera una forma integrada de introducir nuevas herramientas.

Si bien este entorno puede ser útil para diseñar otro tipo de entornos colaborativos, debido a su incapacidad de contemplar tipos binarios en los modelos, su complejidad

para programadores no familiarizados con estas tecnologías y los requerimientos impuestos en el proyecto Colab tuvimos que plantear otro diseño de un entorno colaborativo que sirva de base para la construcción de una plataforma para el aprendizaje por descubrimiento, la cual analizaremos en el capítulo cuarto.

2.3. Conclusiones.

En este apartado hemos mostrado el estudio realizado en la plataforma ANTS tanto para la introducción de componentes (Beans) así como la creación de una arquitectura genérica para la experimentación colaborativa.

En un principio pensamos en usar la arquitectura ANTS como la base para la creación de una arquitectura para el Aprendizaje por Descubrimiento puesto que esta arquitectura permite la colaboración síncrona y asíncrona así como el uso de XML para guardar información. Estas dos características eran fundamentales para los diferentes **Espacios de colaboración** y un **Marco Común de Referencia.**

En este apartado hemos mostrado como realizar una integración de las herramientas desarrolladas sobre ANTS con experimentos ya creados en otros lenguajes de programación (C, Pascal). De esta forma se pretendía crear el **Espacio de Experimentación**, a través del cual los usuarios con la interfaces gráficas en Java podían invocar y obtener respuesta de los métodos en los experimentos mediante el uso de la comunicación síncrona, usando XML-RPC. Una vez que la aplicación obtiene el resultado del servidor, distribuye estos valores a través del bus de comunicación mediante el uso de la API ANTS.CORE.

Como hemos podido apreciar, ANTS impone un modelo de programación muy sofisticado mediante el uso de modelos, sin embargo estos son inadecuados en una arquitectura basada en el envío de datos binarios, lo cual implica al programador analizar cada uno de los eventos obtenidos, conllevando además el conocimiento de la parte tecnológica del sistema. Esto conlleva la dificultad en la integración de nuevos componentes puesto que se requiere el conocimiento de mucha parte tecnológica.

Todos estos problemas junto con el requerimiento de la parte pedagógica de la creación de conjunto de APIS que facilite la creación e integración de nuevas herramientas, nos obligó al diseño de otra plataforma.
Del estudio de ANTS hemos apreciado como su sistema modular en el cual se ofrece un conjunto de APIs para la creación de herramientas colaborativas sin la necesidad de conocer las tecnologías subyacentes.

Más concretamente el uso de la API ANTS.CORE y lo que es más, el uso de ANTS.COMM permite el uso de diferentes tecnologías MOM sin que las aplicaciones se vean afectadas por esto. Así pues, estas APIs constituirán las bases para la creación de nuestro marco de aplicación en el desarrollo de este trabajo de investigación.

Capítulo Cuarto.

Diseño de un Entorno Colaborativo y su Aplicación a Plataformas de Aprendizaje.

En el capítulo anterior se han analizado diferentes entornos colaborativos al igual que se han llevado a cabo una serie de estudios e integraciones con el fin de encontrar las ventajas e inconvenientes de dichos entornos. Estas investigaciones llevadas a cabo tanto en la plataforma JLE así como la plataforma ANTS han servido de base para la creación entornos colaborativos aplicados al aprendizaje, los cuales constituyen el resultado de este trabajo de investigación. Uno es un sistema colaborativo para ser integrado en otros sistemas ya existentes y el otro constituye una plataforma colaborativa que permite tanto la colaboración asíncrona como la síncrona.

Como veremos a lo largo de este capítulo, ambos sistemas se han construido basándose en el bus de eventos presentado en la plataforma ANTS, obteniendo de esta manera independencia sobre el tipo de tecnología subyacente. El Bus de eventos constituye uno de los puntos clave en el diseño y construcción de ambos sistemas, puesto que toda la información de la colaboración síncrona fluirá sobre él, y el éxito o fracaso de esta colaboración dependerá enormemente de factores como son la eficacia y fiabilidad que nos proporciona el BUS.

Como veíamos en el capítulo tercero, existen muchos entornos educacionales basados en Web que carecen de un sistema colaborativo síncrono, así como de herramientas, que permita la colaboración casi en tiempo real de cada uno de sus miembros. Uno de los resultados que hemos propuesto para solucionar este aspecto es la creación del sistema colaborativo ANTS. En este capítulo abordaremos tanto el diseño y desarrollo de dicho sistema así como su integración en dos sistemas CSCL, llamados BSCL y FLE. Más concretamente, BSCL es un entorno basado en la arquitectura BSCW con ciertas

funcionalidades colaborativas para constituir un sistema CSCL. Las investigaciones llevadas a cabo en estos dos sistemas así como la inclusión de las nuevas herramientas fueron financiadas por el proyecto europeo ITCOLE "Innovative Technology for Collaborative Learning and Knowledge Building" (IST 2000-26249) [Lei01].

En el capítulo de antecedentes veíamos como Wouter Van Joolingeng exponía los requerimientos que se necesitaban en la creación de un entorno para el aprendizaje por descubrimiento. En el capítulo tercero veíamos la plataforma ANTS para la creación de sistemas cooperativos y colaborativos. Sin embargo, esta plataforma carece de la infraestructura adecuada para el desarrollo de experimentos igual que impone un alto grado de conocimiento en la programación de componentes. Por lo tanto, nuestro objetivo se basa en construir una plataforma colaborativa que soporte todos los ámbitos de la colaboración (entre ellos la experimentación), proveyendo las interfaces necesarias para el desarrollo de cualquier aplicación colaborativa. En el desarrollo de estas interfaces otro de los objetivos a cumplir es que en el desarrollo de nuevas herramientas el programador no tenga que conocer todos los aspectos técnicos de las tecnologías subyacentes de la plataforma.

Esta plataforma ha constituido el marco de aplicación sobre el que se ha desarrollado la aplicación COLAB que es el resultado del proyecto europeo "Collaborative Laboratories for Europe" (IST-2000-25035) [Col04].

En el primer apartado de este trabajo analizaremos tanto el diseño e implementación del sistema colaborativo ANTS y su integración en los sistemas BSCL y FLE. A continuación, en el segundo bloque de este capítulo veremos el diseño de una plataforma colaborativa y su aplicación en el aprendizaje por descubrimiento.

1. Diseño del Sistema Colaborativo ANTS.

Como ya hemos comentado anteriormente, existe una gran variedad de entornos educacionales, normalmente asíncronos los cuales requieren de un sistema síncrono para potenciar la colaboración entre los usuarios y hacerla más parecida a la comunicación "face to face" (cara a cara). Dicho requerimiento se hace más fuerte cuando dichos entornos pretenden constituir sistemas educacionales CSCL, ya que la integración de estas herramientas constituye una base fundamental para la comunicación entre los estudiantes y permite la construcción del conocimiento de una manera más rápida.

Además se necesita no sólo que las herramientas síncronas manden información entre ellas, sino que también se comuniquen ciertos eventos al entorno que las contiene para así tener una visión totalmente integrada dentro de dicho entorno.

Existen herramientas colaborativas que requieren tanto de una colaboración síncrona como de una colaboración asíncrona de tal forma que puedan guardar su estado una vez que se acaba la sesión y recuperar la última versión cuando se vuelve a iniciar la colaboración. Por ello es fundamental que exista alguna forma de comunicación entre el entorno y las herramientas colaborativas.

Además el uso de la colaboración síncrona y asíncrona enriquece la colaboración entre los diferentes usuarios permitiendo mediante la colaboración síncrona que se comuniquen de una manera más rápida y directa y habilitando mediante la colaboración asíncrona la posibilidad de participación y compartimiento del conocimiento a aquellos usuarios que por algún motivo no pudieron conectarse en el período en que fue llevada a cabo.

En este apartado mostramos una de las soluciones obtenidas dentro de nuestra tarea de investigación, proponiendo un sistema colaborativo para facilitar la incorporación de herramientas de colaboración síncrona en entornos Web. De esta forma se pretende enriquecer la integración de herramientas colaborativas síncronas en entornos CSCL fortaleciendo de esta manera la construcción del conocimiento mediante la tarea compartida. En dicho diseño es primordial la integración de un agente que sirva como medio integrador entre el nuevo sistema colaborativo y el entorno subyacente dónde se realiza dicha integración.

Como veremos posteriormente en este apartado, dicho sistema ha sido integrado con éxito en dos plataformas educacionales. Una de ellas la constituye el sistema BSCL y la otra el sistema FLE.

La integración de este nuevo sistema dentro del entorno existente debe cumplir los siguientes requerimientos:

- Uno de los aspectos principales es la comunicación entre las herramientas síncronas y el entorno subyacente. El ciclo de vida de las herramientas síncronas conlleva la comunicación y transferencia de datos entre el entorno existente y las herramientas nuevas. Por lo tanto el sistema debe comunicar dichos estados al entorno subyacente para lo cual éste debe proporcionar los métodos apropiados.

- Además, los componentes de colaboración síncrona necesitan un canal de comunicación con altas prestaciones que permita la propagación de los cambios del estado en un ámbito distribuido. La propagación de estado impone fuertes requerimientos en el canal de comunicación con el objetivo de asegurar la comunicación uno a uno o la comunicación uno a muchos.

- Otro requerimiento esencial para la infraestructura síncrona es que sea simple, compacta, ligera y fácil de instalar. La idea principal es evitar la imposición de una sobrecarga en el entorno subyacente y por lo tanto simplificar el proceso de instalación al usuario final. Para esto, nosotros proponemos una arquitectura replicada para los componentes síncronos que se sustenten en el canal o bus de comunicación para propagar el estado entre las herramientas establecidas en la misma sesión.

- Otro de los aspectos interesantes a la hora de abordar la construcción de herramientas síncronas es proporcionar un modelo para la recogida de aquellos eventos relevantes para poder analizarlos posteriormente. Como veremos, para ello nos basaremos en el uso del canal de eventos, de tal forma que un servicio escuche los eventos de interés y los guarde en las estructuras apropiadas.

El proyecto ITCOLE "Innovative Tecnolgy for Collaborative Learning and Knowledge Building" (IST 2000-26249)[Lei01] tuvo como objetivos la construcción de un sistema CSCL usando como arquitectura base BSCW al cual se le introdujo el sistema colaborativo ANTS así como cambios en su interfaz dando lugar al sistema actual BSCL (Basic Support for Collaborative Learning)[BSCL]. Otro de los objetivos dentro de este proyecto fue incorporar el sistema colaborativo ANTS en el entorno FLE, el cual hemos visto en anteriores capítulos.

Para llevar a cabo este proyecto participaron pedagogos, psicólogos, informáticos y diseñadores gráficos. Desde la parte pedagógica cabe destacar la participación de la Universidad de Helsinki, Universidad de Amsterdand, Universidad de Salerno, Universidad de Rome La Sapienza, Universidad de Atenas y la Universidad de Utrech. En cuanto a la parte tecnológica cabe destacar la participación de la empresa Fraunhofer Institute for Applied Information Technology (FIT) y la Universidad de Murcia. En cuanto al diseño gráfico ha sido llevado a cabo por la Universidad de Arte y Diseño de Helsinki.

En ambos entornos integramos una pizarra destinada al diseño de mapas conceptuales los cuales ayudan a los estudiantes en el aprendizaje de nuevos conceptos así como sus posibles relaciones con otros conceptos [Hon00]. En el caso del BSCL también realizamos la integración de una herramienta de mensajería instantánea para el intercambio de mensajes entre los usuarios conectados a dicha herramienta sin necesidad de hacer dicha información pública a todo el grupo. Más concretamente la pizarra es llamada *"Maptool"* y el cliente de mensajería es llamado *"Instant Messenger"*.

También realizamos herramientas de análisis de información para los logs generados tanto del uso del BSCL así como del uso de la herramienta de maptool. En este apartado

analizaremos tanto el formato de los eventos de logs así como las posibles herramientas y gráficas para mostrar dicha información.

A lo largo de este apartado veremos el diseño del sistema colaborativo ANTS así como las integraciones llevadas a cabo en BSCL y FLE.

1.1. Diseño de colaboración.

Una de las claves fundamentales del sistema colaborativo ANTS es el Bus de eventos sobre el cual fluirá toda la información de la colaboración. En la plataforma ANTS hemos podido comprobar como mediante el uso de ANTS.COMM se pueden manejar las operaciones de sesiones y envío de datos del bus de eventos independientemente de la tecnología subyacente.

Basándonos en las ventajas fundamentales que ofrece ANTS.CORE hemos adaptado su API de comunicación para trabajar fuera de un entorno J2EE, aprovechando así la capacidad de construcción de herramientas de manera independiente del servicio de notificaciones. Esto se obtiene mediante el uso de las librerías ANTS.COMM.

Debido a las altas prestaciones ofrecidas por DSTC Elvin frente a otros servidores de notificaciones, dicho servidor constituyó la tecnología subyacente en la que se basó ANTS.COMM para las integraciones llevadas a cabo en los los entornos BSCL y FLE.

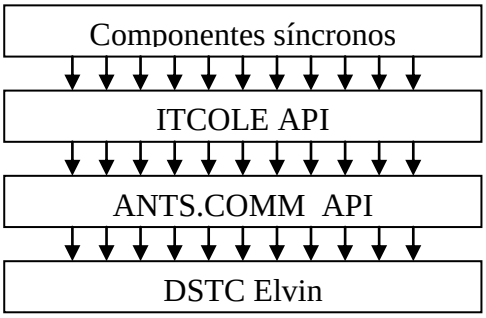

Figura 43. Capas de Adaptación Software.

En la figura 43 se puede apreciar la estructura de capas que se sigue en el modelo de comunicación de los componentes o herramientas colaborativas síncronas dentro del sistema colaborativo ANTS. Estas herramientas son desarrolladas usando la API ITCOLE, la cual ofrece al programador un conjunto de operaciones para manejar la transferencia de información que se recibe en una determinada sesión. A su vez está API se basa en el uso de ANTS.COMM para la creación de sesiones y el envío de datos. El uso de dicha librería es transparente al programador, puesto que para el desarrollo de aplicaciones solo se manejarán los métodos ofrecidos en las clases de la librería ITCOLE.

Para la creación de esta arquitectura de capas, en la librería ANTS.COMM no se ha realizado ningún cambio, por lo cual su modo de funcionamiento es idéntico al que ofrece la arquitectura ANTS.

En cuanto a la API ITCOLE es el resultado de suprimir en ANTS.CORE todas las llamadas a objetos persistentes ya que esta información debe ser proporcionada por el entorno educacional dónde se integren las nuevas aplicaciones (véase la figura 44).

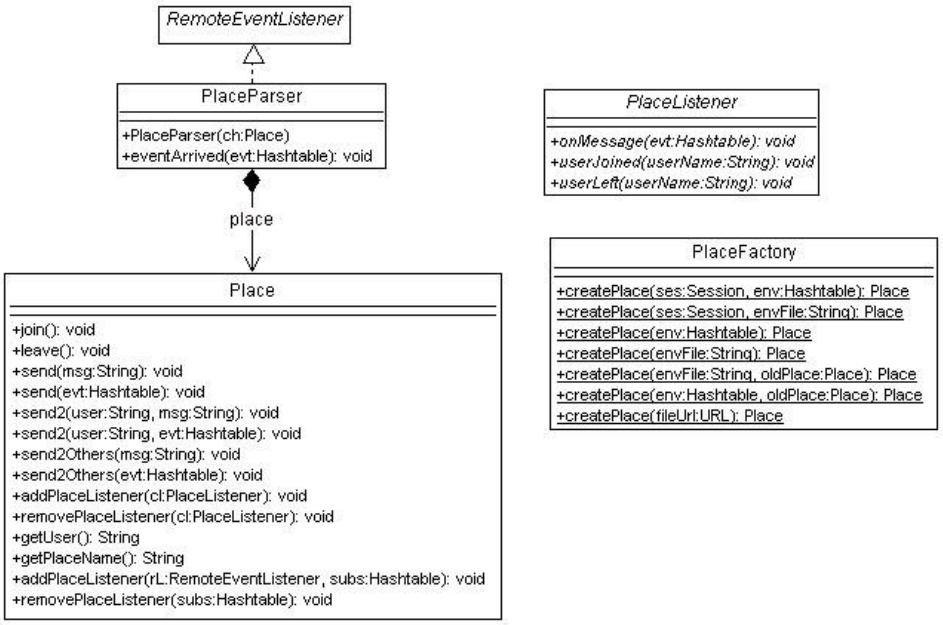

Figura 44. Api ITCOLE para el desarrollo de herramientas en el sistema colaborativo ANTS.

En cuanto a su modelo de propagación de eventos se basa en el bus definido por ANTS.COMM de tal forma que cualquier cambio en el servidor de notificaciones subyacente no implica remodificación en las aplicaciones desarrolladas sobre la API ITCOLE.

1.2. Diseño de Integración con entornos asíncronos.

El apartado anterior muestra la obtención de una API para el desarrollo de herramientas síncronas. Como hemos comentado anteriormente, en cualquier entorno y más especialmente aquellos destinados a la educación, el uso de herramientas que permitan tanto de la colaboración síncrona como la asíncrona enriquecen la comunicación y la difusión del conocimiento.

Además existen ciertas caracteríticas como puede ser el mantenimiento de la lista de usuarios online, que no pueden ser ofrecidas por el entorno educacional, con lo cual se requiere un servicio que permita suministrar la información necesaria del entorno a las herramientas de colaboración que se integran.

Con el objetivo de establecer la comunicación entre las herramientas síncronas y la plataforma educacional se ha diseñado un servidor o agente intermedio. Como ya hemos comentado anteriormente el eje central de la colaboración la constituye el Bus de eventos puesto que a través de él se propaga la información relevante. Por eso el agente crea una sesión mediante el uso de la API ANTS.COMM para de esta forma poder escuchar todos los eventos que puedan surgir de los diferentes places o sesiones dónde las herramientas colaborativas han sido lanzadas.

Cada uno de los servicios que se suministran desde el agente, se suscribirá mediante un conjunto de valores-clave que determinarán el tipo de evento a escuchar y serán tratados

mediante un Listener que implementa *RemoteEventListener* dedicado para cada una de las suscripciones necesarias.

Cuando este servidor arranca, recoge los valores de conexión y del tipo de servidor de notificaciones de un fichero de inicialización manteniendo un conjunto de parámetros de timeout, de debug, path, etc para cada uno de los servicios que se arrancan en el servidor.

Entre los servicios que se proporcionan en este servidor intermedio cabe destacar los siguientes:

- Gestión de los usuarios on-line en el sistema, al cual llamaremos *UserManagement*.
- Obtención del entorno educacional de la información de las sesiones previas, y comunicación de esta información a la herramienta colaborativa que lo requiere. Este es el servicio denotado como *ImportSession.*
- Generación de las sesiones a guardar y comunicación de dicha información al entorno educacional subyacente, el cual denotaremos como *UploadService.*
- Servicio para la gestión de logs de las herramientas colaborativas, permitiendo así guardar una serie de información que posteriormente pueda ser analizada, denotado por *LogService*.

En la figura 45 se muestra el esquema general de los servicios a integrar para la comunicación de ciertas operaciones con el entorno.

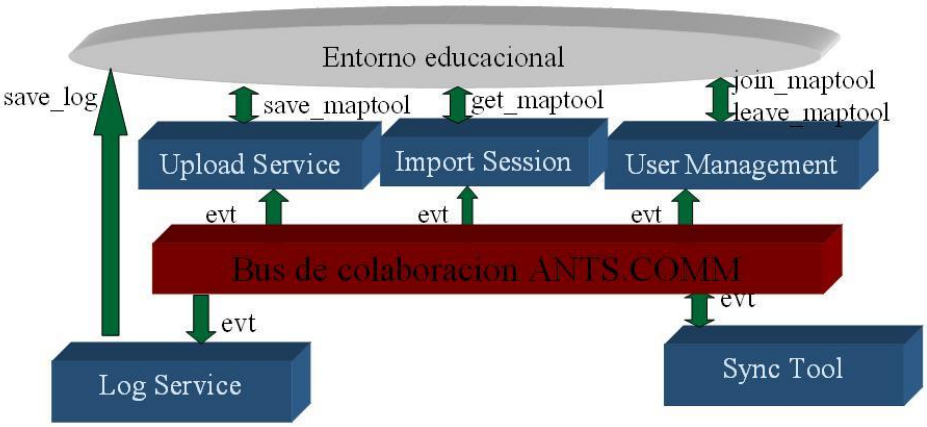

Figura 45. Servicios requeridos para la integración de herramientas colaborativas síncronas con un entorno asíncrono.

Más concretamente, estos servicios serán los encargados de establecer la comunicación con el entorno subyacente mediante una API que dicho entorno debe ofrecer. Entre las tecnologías y posibilidades para el desarrollo de dicha API pueden considerarse el uso de tecnologías RPC como pueden ser XML-RPC, SOAP, http, etc, las cuales nos permiten tanto realizar la comunicación de ciertas operaciones al entorno subyacente como la obtención de información de dicho sistema. De la figura 45 también se deduce las operaciones que deben ser suministradas por la API del entorno asíncrono, como son *save_log*, *save_session*, *get_session*, *join_sesión*, y *leave_session*.

Si bien existen servicios que pueden ser implementados de manera general independientemente del entorno educacional elegido, algunos o parte de ellos se verán influenciados por el método de comunicación y los servicios prestados por parte del entorno. Más concretamente en la integración llevada a cabo en BSCL y FLE el agente recibe el nombre de *MaptoolUtils*.

En el siguiente apartado detallaremos las operaciones básicas que se definieron para la comunicación entre las herramientas colaborativas diseñadas en los entornos BSCL y FLE.

1.3. Operaciones básicas en el entorno asíncrono.

Entre las operaciones que la herramienta de maptool ejecuta en el entorno educativo se indicaron las siguientes:

- ***join_maptool*** con esta operación se indica que un cliente se ha unido a una sesión colaborativa de maptool.
- ***leave_maptool*** con esta operación indica que un cliente ha abandonado una determinada sesión colaborativa.
- ***save_maptool*** con esta operación se guarda una serie de ficheros que contienen el estado del maptool.
- ***get_maptool*** con esta operación se solicita al sistema educativo la información correspondiente al fichero que le pasa como parámetro.
- ***get_maptoolSessions*** con esta operación el maptool pide al sistema educativo que se le suministre información acerca de un objeto.

Para llevar a cabo dichas operaciones tanto el entorno BSCL como FLE suministraron una interfaz desde la cual se pudiera hacer las llamadas desde el sistema colaborativo hacia el entorno.

Algunas de estas operaciones no pudieron ser ejecutadas por la herramienta colaborativa, ya que surgieron los siguientes problemas: las implementaciones de java de los distintos navegadores producían diferentes acciones ante las llamadas HTTP y el tiempo en resolver la operación *get_maptoolSessions* implicaba la imposibilidad de trabajar con la herramienta colaborativa puesto que parecía que la herramienta quedaba inestable.

Por lo tanto, para solucionar dichos problemas en el sistema educativo integramos dichas operaciones dentro del agente o servicio intermedio (*MaptoolUtils)* que forma parte de nuestro sistema colaborativo ANTS.

Más concretamente para la comunicación de nuestro sistema con BSCL se eligió el protocolo http puesto que era el protocolo de comunicación ya existente en BSCW y re-implementarlo suponía un alto coste en recursos y tiempo. Aunque actualmente dicho entorno cuenta con una API XML-RPC [X-BSCW], en el momento en que estas integraciones se llevaron a cabo la única posibilidad de comunicación era el uso de http. En el caso de FLE se optó por el uso de las librerías JPE (Java Python Extension)

que ofrecen la ventaja de poder ejecutar tanto objetos Java como objetos Python en el mismo proceso. De esta forma el agente *MaptoolUtils* y el servidor FLE no necesitan un protocolo externo para hablar uno con otro.

En cuanto al paso de los parámetros necesarios para arrancar las herramientas se hará a través de una página HTML la cual será usada para la llamada de la correspondiente herramienta. A continuación se expondrá los aspectos mas relevantes de la comunicación entre en nuestro sistema colaborativo y los entornos BSCL y FLE [Wol02].

Interfaz de Comunicación BSCL

En la figura 46 podemos apreciar el esquema general que se proporciona desde el sistema BSCL. Pudiendo usar la aplicación síncrona así como el agente MaptoolUtils llamadas HTTP al sistema para recuperar información.

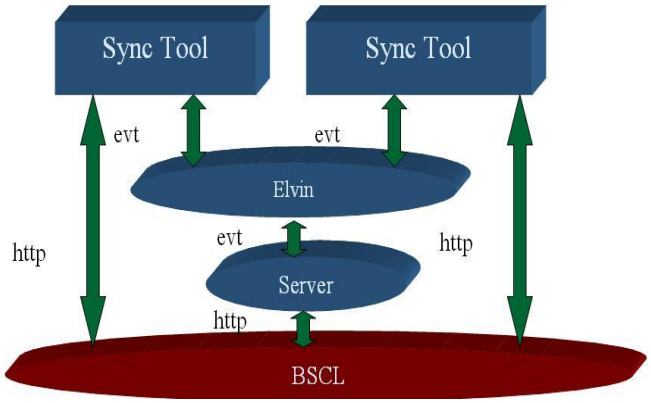

Figura 46. Comunicación de BSCL con el sistema colaborativo ANTS.

Los parámetros que se pasan para arrancar la herramienta maptool son mostrados en la figura 47.

```
<param name="elvinserver"      value="bscl.gmd.de">
<param name="elvinport"        value="2917">
<param name="bsclserver"       value="bscl.gmd.de">
<param name="bsclport"         value="80">
<param name="bsclpath"         value="/bsclpub/bscl.cgi">
<param name="userid"           value="denter">
<param name="username"         value="John Denter">
<param name="token"            value="upkgdi2yTDMTr3ZswB2Y1WIQCMUW5c5B">
<param name="language"         value="en">
<param name="flags"            value="1">
<param name="objectid"         value="153">
<param name="objectname"       value="Group: boys">
<param name="folderid"         value="57">
<param name="foldername"       value="Course: Math">
<param name="groupid"          value="57">
<param name="groupname"        value="Course: Math">
```

Figura 47. Parámetros de información para el arranque de maptool.

El parámetro **userid** denota un usuario perteneciente a BSCL. El parámetro **flags** informa de los estados del BSCL. Si es 1 indica que el usuario tiene permisos para

grabar la sesión de maptool, si es 2 indica que el servidor BSCL está ejecutándose bajo un servidor seguro. El parámetro **token** es la autorización necesaria para que un usuario pueda realizar ciertas operaciones en la sesión actual. Los parámetros **elvinserver y elvinport** servirán de conexión al servidor Elvin instalado en la misma máquina que el BSCL.

El parámetro **language** nos indica el idioma que tiene dicho usuario asignado en el sistema. Este parámetro es de especial interés puesto que ayuda en la internacionalización de las herramientas construidas. Los parámetros **objectid, folderid, groupid** son necesarios tanto para las invocaciones HTTP así como para la creación de sesiones dónde se establecer la colaboración a través de las herramientas colaborativas.

Estos parámetros serán de ayuda en la creación de las diferentes sesiones que se puedan crear de maptool. Más concretamente, el objectid es usado para establecer las diferentes sesiones, puesto que da un identificador único de la localización dónde se encuentra la herramienta colaborativa.

De igual manera se definieron los parámetros para el arranque de la herramienta InstantMessages (véase figura 48).

```
<param name="elvinserver"   value="bscl.gmd.de">
<param name="elvinport"     value="2917">
<param name="bsclserver"    value="bscl.gmd.de">
<param name="bsclport"      value="80">
<param name="bsclpath"      value="/pub2/bscl.cgi">
<param name="username"      value="Mª Antonia Martinez Carreras">
<param name="userid"        value="amart">
<param name="useridn"       value="5600">
<param name="language"      value="es">
<param name="groupid"       value="4983">
```

Figura 48. Parámetros de información para el arranque del InstantMessages.

Operaciones de join y leave.

Las siguientes dos operaciones son usadas para comunicar a BSCL que el usuario se ha unido o abandonado la sesión. Si la operación se ha realizado con éxito se devuelve un código 200, en otro caso se devuelve un código 403.

```
http://<bsclserver>:<bsclport><bsclpath>/0/<objectid>?
                    op=join_maptool&user=<userid>&auth=<token>
http://<bsclserver>:<bsclport><bsclpath>/0/<objectid>?
                    op=leave_maptool&user=<userid>&auth=<token>
```

Operaciones de recuperación y guardado de datos.

La operación para obtener el fichero dónde se guardan los objetos de maptool para su posterior recuperación en la nueva sesión se realiza mediante la siguiente operación:

```
http://<bsclserver>:<bsclport><bsclpath>/0/<objectid>?
                    op=get_maptool&user=<userid>&auth=<token>
```

En el caso de que se pueda recuperar sin problema junto con el fichero se devuelve un código 200. En otro caso se devuelve un código de error 404.

Para el guardado de la información de maptool se encapsula la información en un nuevo tipo MIME, el cual será manejado por BSCL como un objeto dentro del sistema. En este objeto se apunta a tres ficheros que son los siguientes:

- Un fichero txt dónde se guarda la información generada en el chat de la herramienta.
- Un fichero jpeg dónde se guarda una imagen de la información generada de la última sesión.
- Un fichero map que guarda los objetos serializados creados en la última sesión de maptool.

Para realizar la operación de guardado se invoca la siguiente llamada:

```
http://<bsclserver>:<bsclport><bsclpath>/0/<objectid>?
                    op=_save_maptool
```

dónde

```
Content-Type : multipart/form-data; boundary="<boundary>"
```

y el contenido de la petición sigue este patrón:

```
This is a multi-part message in MIME format.

--<boundary>
Content-Disposition: form-data; name="op"

save_maptool
--<boundary>
Content-Disposition: form-data; name="user"

<userid>
--<boundary>
Content-Disposition: form-data; name="auth"

<token>
--<boundary>
Content-Disposition: attachment; filename="file.txt";
name="file.txt"
Content-Type: text/plain
Content-Length: <size_of_chat_contents>

<chat_contents>
--<boundary>
Content-Disposition: attachment; filename="file.jpg";
name="file.jpg"
Content-Type: image/jpeg
Content-Length: <size_of_whiteboard_contents>

<whiteboard_contents>
--<boundary>
Content-Disposition: attachment; filename="file.map";
name="file.map"
Content-Type: application/octet-stream
Content-Length: <size_of_maptool_data>
```

```
<maptool_data>
--<boundary>--
```

Una vez invocada esta operación es responsabilidad del BSCL decidir dónde guardar dichos ficheros.

Con el objetivo de poder acceder a cada uno de estos ficheros que conforman el tipo MIME del nuevo objeto BSCL llamado "MaptoolSession" se incorporaron las siguientes operaciones *mget*

1. Para obtener el fichero de los objetos serializados de maptool.

```
http://<bsclserver>:<bsclport><bsclpath>/0/<objectid>?op=mg
        et&user=<userid>&auth=<token>&type=data
```

2. Para obtener el fichero jpeg generado del maptool.

```
http://<bsclserver>:<bsclport><bsclpath>/0/<objectid>?op=mg
        et&user=<userid>&auth=<token>&type=board
```

3. Para obtener el contenido del chat.

```
http://<bsclserver>:<bsclport><bsclpath>/0/<objectid>?op=mg
        et&user=<userid>&auth=<token>&type=chat
```

Operación de descriptores XML de los objetos BSCL.

El BSCL no cuenta con una operación para obtener los objetos *MaptoolSessions* que existan en el entorno a los que tiene acceso el usuario. Sin embargo, en su interfaz provee una operación para la obtención de un fichero descriptor XML del objeto (carpeta, curso, documento) en el que se está llevando a cabo la colaboración. De esta manera mediante analizando dicho fichero podemos obtener mediante exploración recursiva en carpetas, cuales son los objetos MaptoolSessions a los que tiene acceso el usuario e ir construyendo así la lista de dichos objetos. Como veremos posteriormente, esta operación ha sido llevada a cabo en el agente *MaptoolUtils*.

La operación que proporciona BSCL para la obtención del fichero descriptor del objeto actual es la siguiente:

```
http://<bsclserver>:<bsclport>/<bsclpath>/0/<objectid>?op=mget&user
=<userid>&auth=<token>&mode=xml
```

Interfaz de Comunicación FLE

Como ya hemos comentado anteriormente en la integración con FLE hemos hecho uso de las librerías JPE las cuales nos permiten realizar operaciones en el entorno asíncrono de una manera más rápida. Esta comunicación viene representada en el esquema de la figura 49.

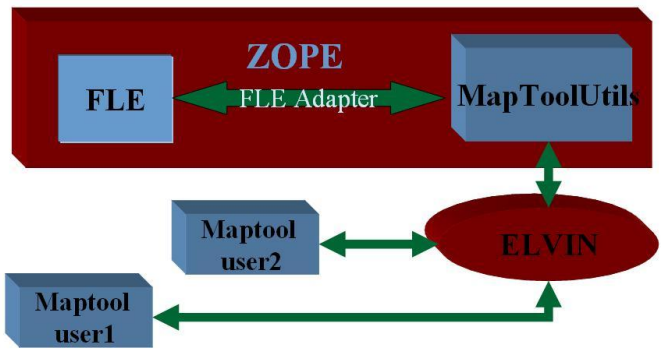

Figura 49. Esquema de integración del sistema colaborativo ANTS en FLE.

FLE [FLE3] es un producto construido sobre el entorno Zope, tomando muchas de las funciones y métodos que incorpora dicho entorno. Como se muestra en la figura 49 se usa una API llamada FLEAdapter entre el entorno FLE y el agente MapToolUtils, la cual permite ejecutar en Python una serie de métodos que proporciona FLE para la comunicación síncrona y así devolver los resultados al servidor *MapToolUtils*.

Para que dicha comunicación se lleve a cabo fue necesario proporcionar una instancia de FLE al FLEAdpater de tal forma que pueda invocar a los métodos que proporciona FLE, y a su vez, fue necesario que el *MaptoolUtils* recogiera como parámetro el objeto FLEAdapter al ejecutarse. Así cuando el Maptool requiera la ejecución algún método el MaptoolUtils ejecutará el método correspondiente del FLEAdapter.

Desde el entorno FLE se implementaron los siguientes métodos en Python, cuya funcionalidad es realizar las operaciones necesarias desde la herramienta de maptool:

- maptool_join(arg[])
- maptool_leave(arg[])
- maptool_get(arg[])
- maptool_list(arg[])
- maptool_save(arg[])

La interfaz FLEAdpater proporciona los métodos que están mostrados en la figura 50.

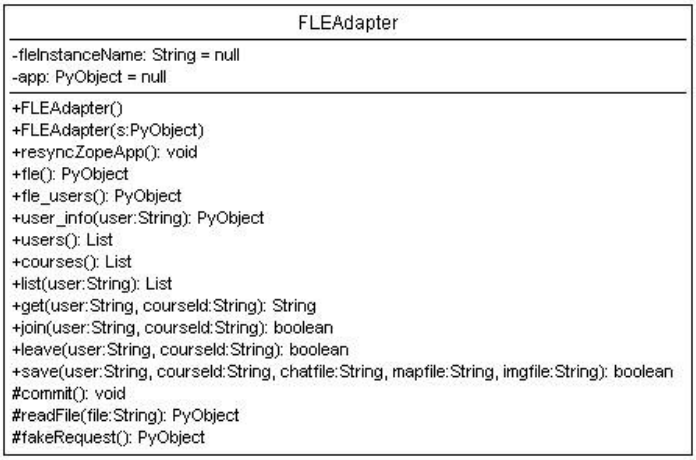

Figura 50. Clase FLEAdapter para la comunicación entre MaptooUtils y FLE

En este caso todas las invocaciones a los métodos proporcionados por FLE serán llevados a cabo por el agente MapToolUtils.

El hecho de que la comunicación entre el MaptoolUtils y el FLE se haga dentro del propio entorno hace que dicha comunicación sea más rápida. Una de las mediciones que se llevo a cabo fue la comparación de la operación save del maptool en ambas plataformas, BSCL y FLE. Dichas pruebas se realizaron bajo servidores en las mismas condiciones tanto de prestaciones como de carga y el tiempo fue medido desde que el *MapToolUtils* invocaba la operación hasta que obtenía la respuesta de si se había realizado con éxito o no. En la gráfica que se puede observar en la figura 51 se muestran los resultados de estas mediciones.

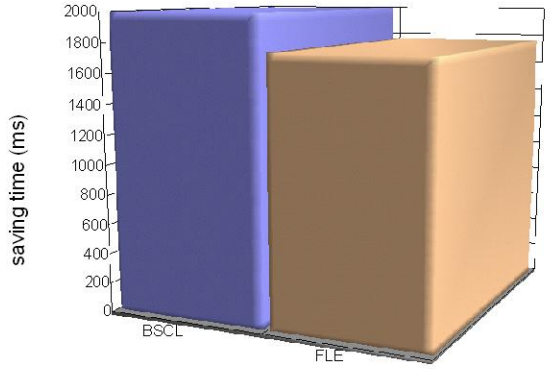

Figura 51. Medidas de rendimiento de la operación save en FLE y en BSCL

Los parámetros que se pasan desde el FLE a la herramienta Maptool son mostrados en la figura 52.

```
<param name="elvinserver"      value="sauron.dif.um.es">
<param name="elvinport"        value="2917">
<param name="userid"           value="denter">
<param name="username"         value="John Denter">
<param name="language"         value="en">
<param name="coursename"       value="Course: Math">
<param name="courseid"         value="57">
```

Figura 52. Parámetros pasados desde el FLE a la aplicación Maptool.

En cuanto al establecimiento de la sesión desde el FLE se contempla que se haga a través del parámetro courseid que identifica unívocamente a cada curso dentro del área.

1.4. MaptoolUtils.

El MaptoolUtils constituye un servidor intermedio entre las aplicaciones y el entorno. El fin de dicho servidor es comunicar operaciones de las herramientas de colaboración síncrona al entorno de colaboración asíncrono. Con el fin de separar cada una de las diferentes tareas llevadas a cabo por este servidor, se gestionan diferentes servicios dentro de él. A continuación explicaremos cada uno de estos servicios que se proporcionan.

UserManagement.

Una de las características fundamentales de los sistemas CSCL es el Presence Awareness o lo que es lo mismo el conocimiento de los usuarios que están conectados en el sistema. Una de las dificultades que se presentan en los entornos asíncronos es el mantenimiento de la lista de usuarios online, ya que no pueden detectar cuando un usuario se ha quedado sin conexión en las herramientas síncronas. Por ello hemos tenido que diseñar y proporcionar un servicio desde el agente que se encargue de mantener esta lista de usuarios para mostrar de forma fiable aquellos usuarios conectados en el sistema.

Cuando el servicio *UserManagement* recibe un evento join de un usuario dentro de una herramienta colaborativa, primeramente analiza si dicho usuario tiene alguna sesión activa mediante el uso de la clase *UserManagerCheckState* (véase figura 53). Si este es el caso, ejecuta un leave en el entorno educacional.

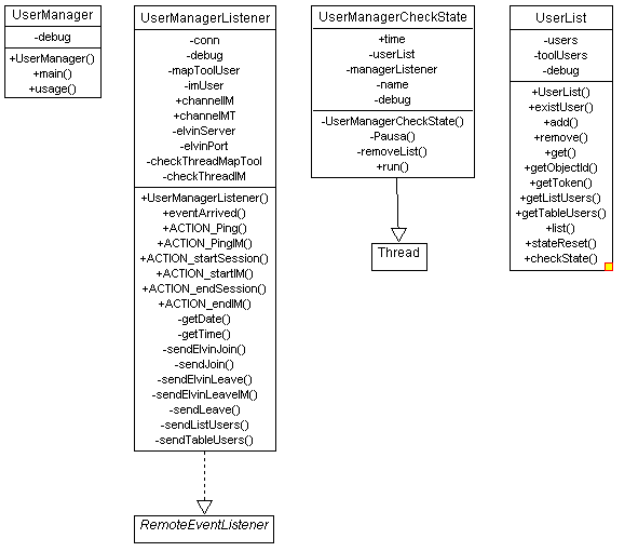

Figura 53. Esquema UML del servicio UserManager.

Las recomendaciones pedagógicas indican que en cualquier entorno colaborativo dedicado al aprendizaje no es aconsejable que los usuarios mantengan más de una sesión abierta en una herramienta colaborativa en un instante dado. El mantenimiento de varias sesiones influiría negativamente en la tarea de aprendizaje de cada una de las sesiones que mantuviera al mismo tiempo. Por lo cual, si el usuario está realizando un join implica que la sesión en la que estuviese anteriormente debe ser abandonada.

Si el usuario no estaba registrado, se guardará la información relevante de este usuario y se ejecutará una operación join en el sistema educacional. Esta información se mantiene en una lista por sesión recogida en la estructura *UserList.* Puesto que un usuario si puede tener sesiones activas de diferentes herramientas a la vez, se llevará una lista por cada una de las herramientas que se construyan dentro del sistema.

Para saber que la sesión de un usuario en una herramienta colaborativa está activa, ésta debe enviar un evento de Ping de tal forma que el servidor pueda comprobar si el usuario sigue en actividad. Si tras un cierto tiempo el servidor percibe que la aplicación no manda dicho evento, realizará el abandono de dicho usuario eliminándolo de la lista

de usuarios activos e invocando a la operación *leave* del sistema. A continuación se detalla el algoritmo de presence awareness llevado a cabo para el mantenimiento de una lista online.

Algoritmo de Presence Awareness.

Por cada una de las herramientas existentes, se arranca un hilo o thread *UserManagerCheck* que es el encargado de chequear la lista de los usuarios conectados tratando de averiguar si cada uno de estos usuarios posee un estado inconsistente. Por cada usuario en cada una de las listas se mantiene un campo ***state*** el cual indica el número de veces que el usuario ha realizado un evento de ping. La variable **state_down** indica el umbral bajo el cual se supone que el usuario no está activo en la sesión de la herramienta correspondiente. El proceso que lleva a cabo dicho thread es mostrado en el diagrama de la figura 54.

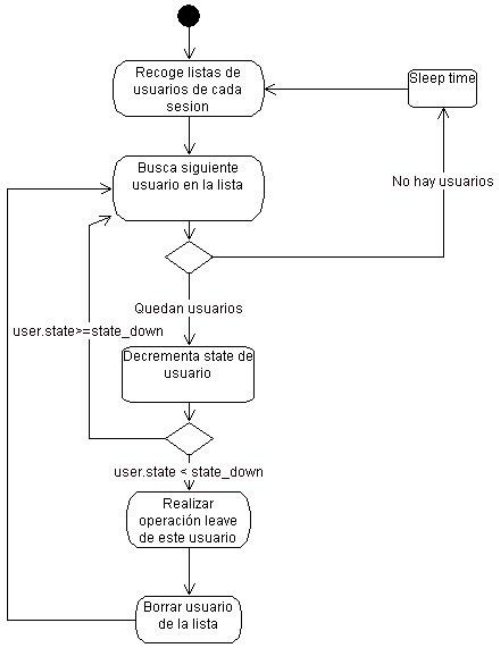

Figura 54. Diagrama de actividad del UserManagerCheck

Con el objetivo de escuchar los eventos de *join* y *leave* de los usuarios, y así poder manejar la lista de usuarios, se hace uso del *UserManagerListener* el cual se suscribe con los valores mostrados a continuación.

```
subs.put("SERVER", "SERVER");
subs.put("USERMANAGER", "ANTSCLIENT");
```

Para que dicho servicio pueda escuchar estos eventos, las herramientas colaborativas deberán notificar su estado de actividad mediante el envío de la anterior información, pares clave-valor, en la notificación.

Anteriormente se ha mostrado el esquema general de este servicio. Gran parte de este funcionamiento se ve inalterado en la integración de los diferentes entornos educacionales, solo cambia los mecanismos usados para hacer el join y el leave de los

usuarios dentro de cada uno de los entornos. Más concretamente, el diagrama de inicio de la herramienta Maptool viene mostrado en la figura 55.

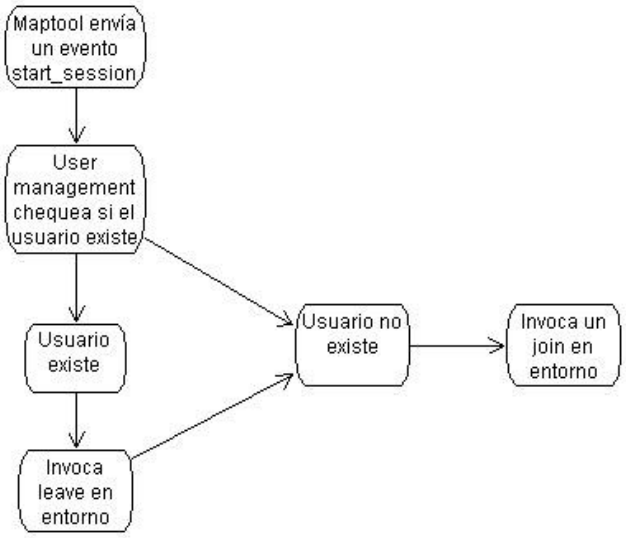

Figura 55. Diagrama de estados del algoritmo de invocación de una sesión.

UploadService.

Los parámetros de suscripción de dicho evento vienen dado por los valores mostrados a continuación:

```
subs.put("SERVER", "SERVER");
subs.put("OBJECT_TYPE"   , "SAVE");
```

Más concretamente la herramienta colaborativa responsable debe enviar el estado de dicha herramienta debe incluir estos pares de valores para que se pueda realizar la operación de guardado. Cuando el servicio Upload recibe este evento se encarga de transformar la información en una representación establecida y depositar dicha información en el entorno subyacente. De esta manera se asegura un guardado de la información que posteriormente puede ser requerida. Tras volcar esta información, dicho servicio informa a la herramienta cliente si esta operación pudo realizarse con éxito o no.

En cuanto a la información a guardar procedente de la herramienta Maptool, siguiendo las recomendaciones pedagógicas se ha decidido mantener los siguientes ficheros:

- Un fichero .txt que guarde los mensajes de chat que se generan en una sesión, guardando la fecha, usuario que manda el mensaje y el mensaje enviado.

- Fichero .jpeg que guarde una imagen con toda la información que hay en el área de dibujo.

- Fichero .map que mantenga la información de los objetos serializados para la posterior recuperación del estado del maptool en siguientes sesiones.

Como ya hemos comentado antes, para realizar la operación de guardado el servidor debe escuchar un evento *SAVE* que contenga el estado actual de la sesión. Dicho evento

se genera o bien mediante el uso del botón “save” de la aplicación usuario o bien porque el último usuario de una sesión de maptool abandona la aplicación.

Una vez escuchado este evento el servidor procede a la creación de dichos ficheros e incorpora estos en el sistema educacional subyacente. En el caso de BSCL los encapsula en un tipo MIME, “MaptoolSession”, y pasa estos valores a través de la llamada HTTP dispuesta para ello en la interfaz del BSCL. El objeto MaptoolSession se reflejará en el curso como un objeto de BSCL. En el BSCL la visualización de los ficheros .jpeg y .txt se realiza accediendo a dicho objeto dentro del curso dónde se estuvo colaborando (véase figura 56).

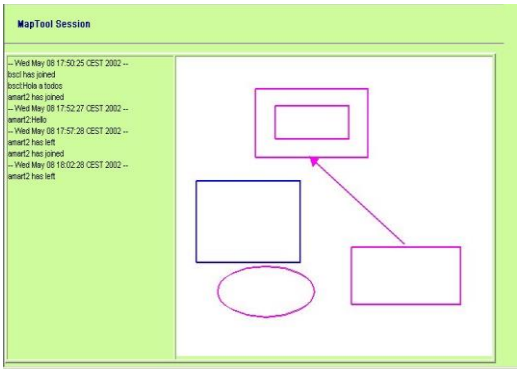

Figura 56. Visualización en BSCL de los ficheros del maptool a través del objeto MaptoolSession.

Figura 57. Visualización en FLE de los ficheros del maptool.

En el caso del FLE los ficheros son también encapsulados en un nuevo tipo Mime pero son mostrados como ficheros separados que se pueden visualizar accediendo a cada uno de los mismos, según se muestra en figura 57.

ImportSession.

Con el objetivo de facilitar la inclusión de la información de otras de sesiones de maptool del sistema que han sido previamente guardadas, se ha incorporado una operación de importado de sesiones en el maptool. Esta operación es ejecutada en el MaptoolUtils y una vez que dispone de los resultados se los comunica a la herramienta de maptool que los solicitó mediante el envío de un evento a través del Bus de comunicación. Para recibir dichos eventos el servicio se suscribe usando los valores que se muestran a continuación:

```
subs.put("SERVER", "SERVER");
subs.put("IMPORTMANAGER", "ANTSCLIENT");
```

Además dicho servicio requiere una invocación a uno de los métodos que suministre el entorno subyacente para recuperar la información requerida y de esta forma enviar al cliente que solicitó esta petición la información necesaria.

En el caso de BSCL esta búsqueda de sesiones es realizada mediante análisis de ficheros XML que define un determinado objeto (carpeta o documento) de cada estructura de carpetas que devuelve el sistema BSCL. Para ello se hace uso de la operación mget indicando el usuario y carpeta dónde se encuentra, según el formato especificado en la sección 1.2. De esta forma se va explorando recursivamente en las carpetas obteniendo así la lista de todas las sesiones de maptool a las que puede acceder el usuario. Una vez acabada la búsqueda se devuelve la lista al cliente que lo solicitó.

Más concretamente la descripción XML de cada objeto es un conjunto de elementos ARTIFACT. En concreto los objetos carpeta ("folder") en su tipo tienen un nombre que siempre acaba con la palabra "folder". Si además dicha carpeta contiene en su campo ACTION un valor 'jget' indica que dicho ARTIFACT puede ser accedido por el usuario que realiza la operación mget. Esta descripción viene mostrada en la figura 58.

```
<ARTIFACT id="5028" name="Group 2" type="GroupFolder" info="Group"
icon="/bscl2resources/icons/group_folder_s.gif" shared="no" container="yes">
<ACTIONS><ACTION name="jget" type="entry">Get</ACTION>
</ACTIONS>
</ARTIFACT>
```

Figura 58. Descripcion XML de una carpeta.

En la descripción de un objeto MaptoolSession el campo tipo contiene la subcadena "MapToolData". Los permisos de acceso a dicho elemento por parte de este usuario viene dado por el campo ACTION name indicando con valor 'get' que se tiene acceso a dicha información. Esta descripción viene mostrada en la figura 59.

```
<ARTIFACT id="5214" name="MapTool Session" type="MapToolData"
info="MapTool Data" icon="/bscl2resources/icons/s_maptool.gif" shared="no"
container="no">
<ACTIONS><ACTION name="get" type="entry">Ver</ACTION>
</ACTIONS>
</ARTIFACT>
```

Figura 59. Descripción XML del objeto MaptoolSession.

En el caso de FLE esta operación se realizará mediante la invocación del método list de la clase FLEAdapter, la cual invoca al método *maptool_list* ofrecido por la interfaz del FLE.

ChatControl

Este servicio construye el fichero de chat de una sesión de Maptool, de tal forma que cada cierto tiempo establece una marca de tiempo. De esa manera posteriormente se

podrá analizar a que periodo de tiempo corresponde cada parte de la conversación guardada.

LogService

El servicio de log se encarga de recoger una serie de eventos que se producen por el uso del maptool y los almacena en un fichero de log o bien en base de datos. Dicha información constituye las bases para la construcción de herramientas de seguimiento que muestran la actividad de los alumnos con el uso de las herramientas.

El progreso de un alumno en el desarrollo de sus tareas, así como la participación de este usuario en la colaboración para la construcción del conocimiento son características importantes en cualquier sistema CSCL.

Si bien el servicio de logs dependerá en gran parte de los eventos que se quieran recoger de cada una de las herramientas, su funcionamiento vendrá dado por una suscripción hacia un Listener con los valores mostrados a continuación.

```
subs.put("SERVER"  , "SERVER");
subs.put("FILE_TYPE", "LOG");
```

Con respecto al envío de eventos de logs, las herramientas desarrolladas deben ser responsables de enviar el par de valores que se han mostrado anteriormente junto con el evento generado por alguna acción. De esta forma, el sistema de logs podrá escuchar dichos eventos y realizar las acciones según los requerimientos indicados (véase tabla 4).

En cuanto al sistema de logs desarrollado en BSCL en un principio el guardado se hizo a través de un fichero en el cual la herramienta de maptool genera los siguientes eventos para que el servicio de logs lo pudiera guardar.

Tool	Acción	Descripción
Maptool	create_chat_note	A new chat line
	create_draw_object	Create new draw object
	delete_draw_object	Delete draw object
	End_session	End of the Maptool session
	First_move_object	Move draw object (only the first Maptool event)
	First_resize_object	Resize draw object
	Save	Save session
	start_session	Start session

Tabla 4. Tabla de los eventos recogidos de maptool log.

En cuanto al formato usado en el fichero se recoge la información mostrada en la figura 60. A parte de las acciones realizadas se recoge la fecha, hora, usuario,

context_objetct_id y target_object_id las cuales nos ayudarán en la ubicación del curso en el cual tiene lugar el evento.

{date} /t {time} /t {username} /t {tool} /t {action} /t {context_object_id} /t {contex_object_name} /t {target_object_id} /t {target_object_name}

Figura 60. File Log Format

Sin embargo la recogida de estos eventos sólo proporciona información acerca de los objetos que se han creado en una sesión así como quién los ha creado.

Si bien esta información resulta bastante interesante para obtener información acerca de la percepción del área de trabajo, así como la participación de cada uno de los usuarios, es insuficiente para poder establecer herramientas que quieran mostrar un análisis más exhaustivo del trabajo realizado. Posteriormente analizaremos los cambios necesarios en la estructura de logs para poder obtener más información acerca de las interacciones de los usuarios.

1.5. Integración de herramientas colaborativas en el entorno..

Mediante el uso de Java se proporciona una manera estructurada de crear aplicaciones haciendo uso de empaquetamiento en ficheros jars. De esta forma se definen diferentes paquetes por cada herramienta creada e incorporada en el entorno.

Tanto BSCW como FLE son entornos los cuales proporcionan el acceso de la información y la colaboración a través de la Web. Por ello optamos por el uso de Applets para la integración de herramientas dentro de ambos entornos.

Como ya hemos dicho, la construcción de herramientas síncronas en ambos entornos se llevará a cabo mediante la integración del sistema colaborativo ANTS. Por tanto dichas herramientas han sido construidas medidante el uso de la API ITCOLE.

Las herramientas incluidas seguirán un modelo replicado de programación de tal forma que mantendrán una copia local del estado actual de la sesión y propagarán en el Bus de eventos los cambios que se produzcan en el área compartida. Así pues las herramientas lanzadas en la misma sesión podrán escuchar estos eventos y actualizar sus vistas.

Para establecer las sesiones las herramientas hacen uso del objeto Place de tal forma que crean el place (sesión colaborativa) mediante la clase *PlaceFactory.createPlace()*, proporcionando un hastable con los pares clave-valor para realizar la creación de la sesión. Véase figura 61.

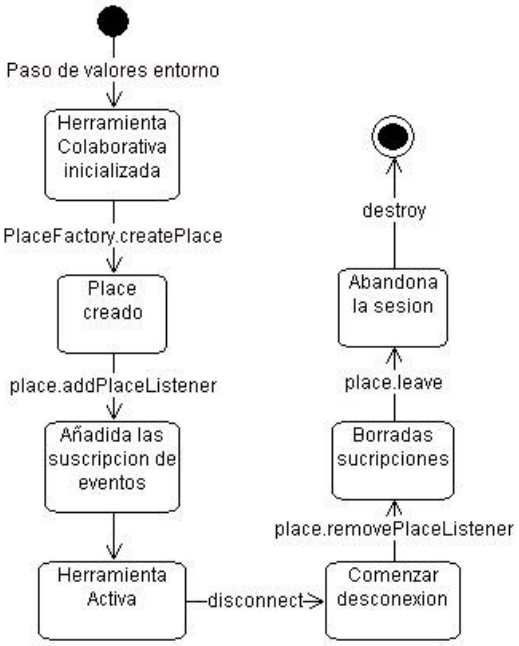

Figura 61. Ciclo de vida de las herramientas desarrolladas con la API ITCOLE

Más concretamente el entorno subyacente (bien sea FLE o BSCL) proporciona una serie de información para establecer las sesiones colaborativas indicando usuario que arranca la herramienta, el curso, carpeta o zona dónde se va a establecer la sesión, máquina y puerto en la que se gestiona el bus de colaboracion, y más información dependiendo de lo que se quiera hacer o permitir en dicha sesion. Estos son los parámetros anteriormente especificados.

Además para poder recibir las notificaciones, las nuevas herramientas deben implementar la interfaz *PlaceListener* a la cual deben suscribir su interés añadiendo el listener al place creado con *addPlaceListener()* de la clase Place, indicando el valor de la suscripción así como el listener que implementa las acciones. De igual forma, previo a la destrucción de las herramientas estas deben de borrar su suscripción mediante el metodo *removePlaceListener* proporcionado en la clase *Place*.

En cuanto al modelo de comunicación seguido por las herramientas se muestra en la figura 62. Como se puede apreciar cualquier aplicación suscrita a una determinada sesión propaga los nuevos eventos mediante el uso de cualquiera de los metodos send proporcionados por la clase Place, la cual a través de la sesión manejada de ANTS.COMM enviará el evento al Bus de comunicación. Esta notificación llegará a los demás colaboradores o suscriptores a través del listener creado en la aplicación que implementa *PlaceListener.* Este evento será manejado en el método *eventArrived* el cual realizará las acciones apropiadas para actualizar las vistas de la aplicación colaboradora.

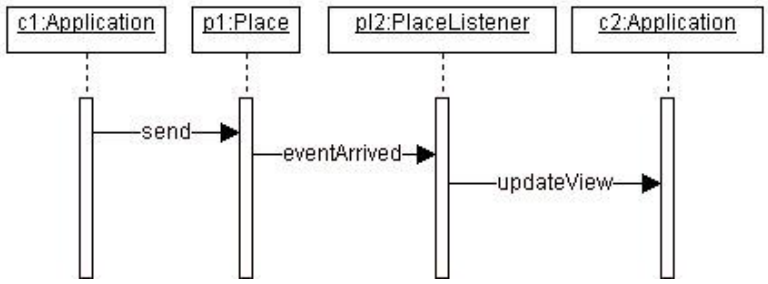

Figura 62. Modelo de comunicación de las herramientas en el sistema colaborativo ANTS.

Siguiendo estas pautas se han desarrollado las dos herramientas colaborativas síncronas las cuales fueron integradas en los entornos BSCL y FLE.

Además, todas las herramientas de colaboración síncrona deben incluir un Thread que se encargue de enviar un evento de ping cada cierto tiempo. En dicho thread se debe construir una notificación que incluya los pares de valores indicados a continuación:

```
evt.put("USERMANAGER","ANTSCLIENT");
evt.put("ACTION","PING");
```

1.6. Herramientas Colaborativas.

Como hemos comentado anteriormente, las herramientas colaborativas han sido diseñadas mediante el uso de la API ITCOLE, de tal forma que la herramienta no tiene que saber el servidor de notificaciones específico que se usa para realizar la conexión. De hecho desde el entorno asíncrono se pasará la información necesaria para poder realizar la conexión con el servidor de notificaciones apropiado.

Este paso de información se realiza a través de los parámetros pasados a los applets que se lanzan desde el entorno educacional, bien sea BSCL o bien sea FLE [Mart02, Mart03]. Básicamente las herramientas desarrolladas fueron dos, la herramienta Maptool y la herramienta InstantMessages.

Maptool

La herramienta Maptool consiste en una herramienta gráfica colaborativa para el desarrollo de mapas conceptuales. Posee un área de dibujo dónde se realizan los diagramas conceptuales y un área de chat dónde los usuarios pueden colaborar mediante el uso de mensajes.

A continuación mostraremos los aspectos más relevantes que se han tenido en cuenta en el desarrollo de dicha herramienta. Más concretamente la figura 63 muestra el diagrama UML de dicha herramienta.

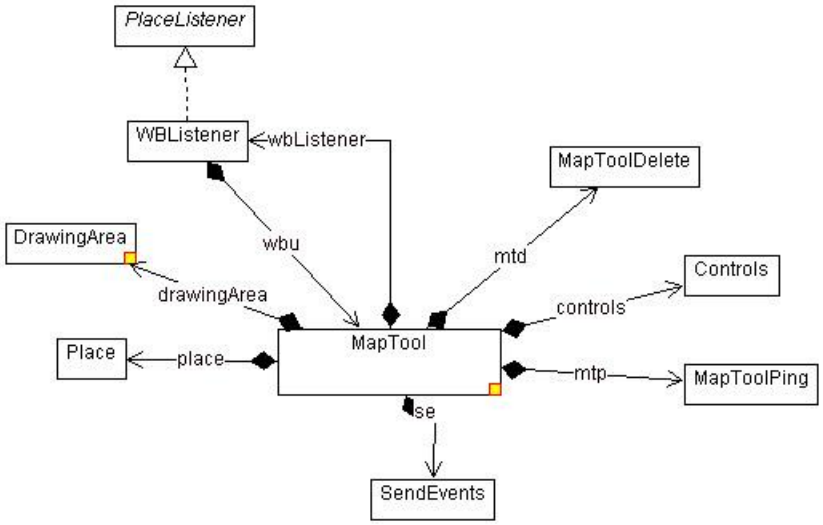

Figura 63. Diagrama UML de la herramienta Maptool

Gestión de envío de eventos

Una de las características a tener en cuenta en el diseño de una herramienta colaborativa síncrona es cuando propagar el evento de una nueva actualización. En herramientas de tipo chat o de comunicación de mensajes cuando el mensaje está terminado se envía a los demás.

Sin embargo en herramientas, como puede ser una pizarra colaborativa, en las cuales el usuario necesita realizar una serie de pasos intermedios previos a la obtención del objeto final, es necesario realizar un análisis de los eventos a enviar no comprometiendo el principio WISIWITYS y favoreciendo la eficiencia en el envío de eventos, evitando de esta forma el bloqueo o congestión en el receptor.

Más concretamente, para el envío de eventos de creación de figuras, así como de modificaciones de figuras existentes que requieran el uso del evento de arrastre del ratón, se activa un hilo o Thread *SendEvents* encargado de cambiar una variable de "false" a "true" indicando con su estado a true que se puede enviar el evento generado. Dicho valor se establece a "false" cada vez que se envía un evento. Este hilo se activa una vez que se activa el evento *"dragged"* que es el evento de inicio de la figura y finaliza cuando termina el evento de arrastre del ratón.

Tras una serie de pruebas se estableció un segundo como periodo de envío de estados intermedios, comprobando que el usuario final no llega a percibir cambios bruscos en los pasos intermedios para la obtención del nuevo estado en las figuras realizadas.

Presence Awareness

Una de las características necesarias en la visualización de las herramientas colaborativas es la lista de usuarios conectados en la sesión. Como hemos podido apreciar anteriormente, nuestro sistema incorpora un algoritmo de Presence Awareness que nos permite mantener la lista de usuarios conectados en cada una de las sesiones. Si bien dicha operación es realizada desde el servidor, es necesaria cierta información por parte de las herramientas indicando su actividad. Para ello el Maptool dispone de un

thread llamado *MapToolPing* (véase figura 64), que cada cierto tiempo envía un evento de ping al servidor MaptoolUtils indicando que está vivo.

Una vez conectado el cliente en la aplicación solicita la lista de usuarios conectados al MaptoolUtils, de esta manera puede mostrar desde su interfaz los usuarios conectados en la sesión.

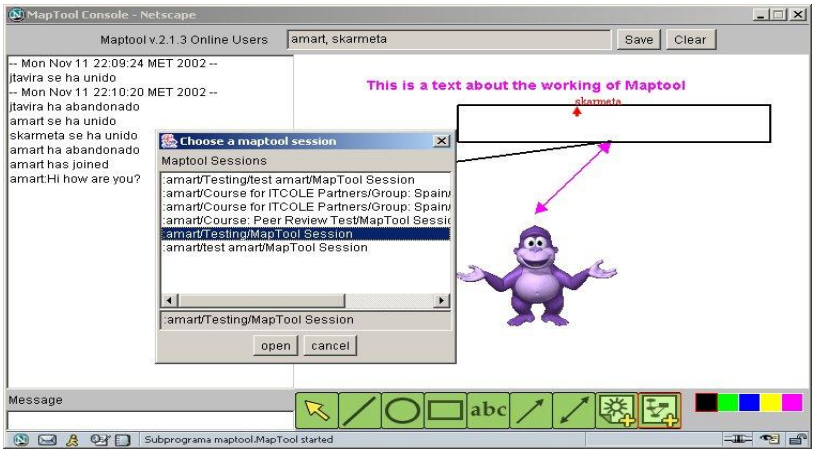

Figura 64. Interfaz de la herramienta maptool.

Structural Awareness

Para mostrar información acerca de la participación de los usuarios en el área de trabajo se ha incluido un puntero junto el nombre del colaborador. Puesto que en cada evento se indica el responsable de éste, la aplicación local representa el nombre de dicho usuario junto con la imagen de un puntero.

Llegada Tarde

Una de las características necesarias a implementar en cualquier herramienta colaborativa síncrona es la de llegada tarde o *late join*. En el diseño de este algoritmo en esta aplicación todos los colaboradores mantienen el estado completo de la sesión, convirtiéndose todos ellos en servidores de estado. Esta decisión queda justificada por el hecho de que en sesiones de grupos de aprendizaje el número de usuarios suele ser bastante reducido, raramente superando cinco usuarios por sesión.

A través de la API de los entornos subyacentes dónde se llevo a cabo la integración de la herramienta Maptool, no se proporciona ningún mecanismo de sincronización persistente. Este algoritmo fue implementado de tal forma que el usuario que llegaba tarde preguntaba al primer usuario de la lista de usuarios conectados el estado de la sesión. De esta forma, el nuevo usuario recibe el Vector con todas las figuras disponibles y otro con todos los mensajes de chat que se han enviado durante la sesión y así los puede actualizar en su vista.

En cuanto a la política llevada a cabo hemos hecho uso de un *Inmediate late join* recibiendo todo el estado actual. En el caso de que no se reciba respuesta, la aplicación vuelve a preguntar por la lista de usuarios online y vuelve a enviar el evento de petición

de estado. Si tras un número de reintentos dicho estado no ha sido recibido, el cliente entra a la colaboración sin representar el estado.

En la solución propuesta nos basamos en un algoritmo selectivo, en el cual se selecciona a uno de los usuarios conectados del grupo. De tal forma que la comunicación se establece directamente con uno de los servidores de estado (es decir uno de los colaboradores de la sesión). En nuestro caso particular la elección del servidor la realiza el propio late-comer. Sin embargo existen otras muchas posibilidades como enviar un mensaje en la sesión y aplicar un algoritmo de consenso entre los colaboradores para elegir a un representante.

Otra de las variantes totalmente válida para este entornos sería el algoritmo que proponen Jürgen Vogel y Martín Mauve [Vog03], sin embargo el diseño del algoritmo de esta aplicación fue llevado a cabo en el año 2001 mientras que la solución propuesta por estos autores apareció en el año 2003.

Si bien nuestro algoritmo ofrece mayor rapidez en la elección del servidor de estado puesto que es tomado directamente por el usuario que llega tarde, tiene una gran dependencia con el algoritmo de Presence Awareness implementado. De tal manera que si la lista de usuarios conectados no es fiable o tarde tiempo en darse cuenta que un usuario se ha quedado colgado en el uso de la aplicación, el algoritmo de *late join* fallaría.

Internacionalización

La visualización de las etiquetas se hace en base a unos ficheros que contienen una estructura de pares clave valor, los cuales son guardados en directorios con el nombre correspondiente a cada uno de los lenguajes incorporados. Más concretamente el formato sigue el esquema que se muestra en la figura 65.

```
saveLabel = Speichern
onlineUserLabel = Online Benutzer
clearLabel = Löschen
messageLabel = Meldung
userJoinLabel = ist beigetreten
userLeftLabel = hat verlassen
noSaveMaptoolLabel = MapTool konnte nicht gesichert werden durch
saveMaptoolLabel = Maptool wurde gesichert durch
introduceImgLabel = Bild einfügen
chooseMapLabel = MapTool Session auswählen
mapSessionLabel = MapTool Sessions
openLabel = Öffnen
cancelLabel = Abbrechen
loadingLabel = Loading
```

Figura 65. Etiquetas alemán para la visualización en el maptool

Floor Control

Siguiendo los requerimientos de la parte pedagógica del proyecto se implementó una política “free for all”. Si bien esta política es flexible a la hora del manejo de los usuarios, en la implementación es necesario tener en cuenta un mayor número de detalles.

Puesto que no se sigue una coordinación en quién dibuja, es necesario tener en cuenta que identificador asignar a cada una de los dibujos que se realizan en el área. Ya no será válido un esquema secuencial puesto que dos usuarios pueden asignar desde su representación local el mismo número a una figura, y por tanto el envío de dos figuras al mismo tiempo implicará que una de ellas no sea tenida en cuenta.

Como este número debe ser irrepetible en cada uno de los usuarios, ha sido creado usando el userid y el tiempo actual en el que ha sido creado la figura.

Seguridad

Uno de los aspectos tenidos en cuenta en el desarrollo de esta aplicación fue la *seguridad*. Puesto que la aplicación es un applet y se requiere realizar operaciones de búsqueda en los directorios locales de los clientes, así como guardar temporalmente imágenes, se requirió el uso de elementos seguros.

Básicamente hay dos posibilidades para abarcar la seguridad desde un applet, una de ellas es configurando un fichero llamado PolicyTools de la máquina local del cliente y la otra es el uso de certificados y código firmado.

La configuración de las PolicyTools requiere la instalación de JRE o JDK en cada una de las máquinas clientes. Uno de los requisitos que se exigía en la elaboración de dichas herramientas es que su funcionamiento no debería implicar la instalación previa de ningún paquete pudiendo así funcionar bajo navegadores como son el Netscape 4.X así como el Explorer versión 5.0. Por lo tanto en la elección del método tuvimos que optar por la firma de código.

Pese a que Netscape 4.X y Explorer v5.0 proveen una plataforma Java compatible con JDK1.1.5, la forma de implementar la seguridad en ambos navegadores es diferente y por lo tanto conllevó a distintas formas de firmar el código. De igual manera sucedió con los navegadores que tenían instalado el plugin de Java, con lo cual fue necesario firmar el código para dicha plataforma.

Puesto que a priori no es posible conocer que tipo de plataforma Java existe en el navegador del cliente, se creo un JavaScript que se inicializa cuando se accede al maptool en la plataforma. Este JavaScript es responsable de ejecutar el código de la clase "JavaVendor" para obtener la plataforma Java en la que se va a ejecutar la aplicación y de esta manera construir la página de inicialización del applet de la aplicación, cargando la versión del código firmado adecuado. Dicho esquema es mostrado en la figura 66.

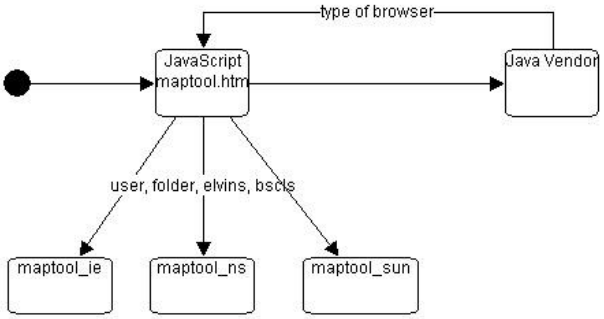

Figura 66. Esquema previo a la descarga del fichero correspondiente previa a la ejecución de la herramienta maptool.

Medidas de rendimiento de maptool

Uno de los objetivos que se pretendían cubrir en el desarrollo de nuevas herramientas así como la integración del *sistema colaborativo ANTS*, era la facilidad de utilización, independencia del navegador y la eficiencia en el uso de recursos. Debido a esto último se llevaron a cabo una serie de medidas de rendimiento respecto a las descargas y a la operación de guardado. Estas medidas de rendimiento son mostradas mediante las figuras 67 y 68.

Más concretamente la figura 67 nos muestra las medidas de rendimiento asociadas a las descargas de la aplicación por parte del cliente con respecto a diferentes ordenadores.

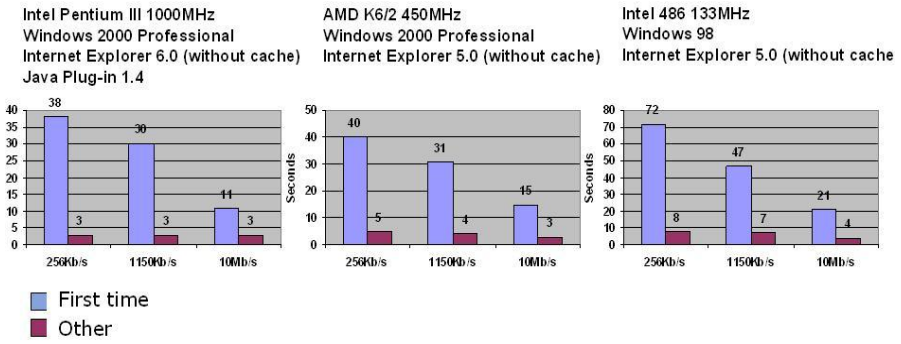

Figura 67. Medidas de rendimiento de la descarga del paquete apropiado firmado de Maptool.

En cuanto a las operaciones de guardado en la herramienta de maptool se hizo las mediciones mostradas en la figura 68 usando una conexión de 10Mbps.

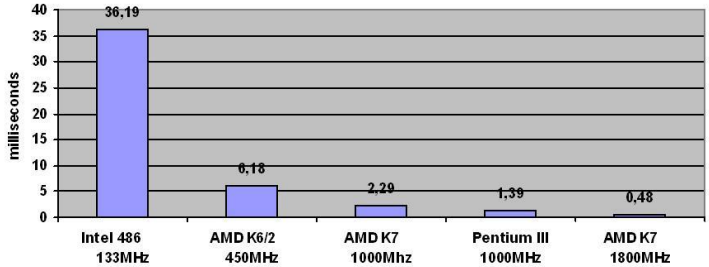

Figura 68. Medidas de rendimiento en la realización de la operación save con diferentes procesadores.

InstantMessages.

El InstantMessages es una herramienta de mensajería instantánea usada para el intercambio de mensajes entre usuarios de una lista. Al igual que el Maptool es una herramienta creada mediante el uso de Applets Java, de tal forma que para su inicialización se pasan un conjunto de parámetros que determinarán la sesión, usuario y toda la información necesaria para trabajar en dicha herramienta.

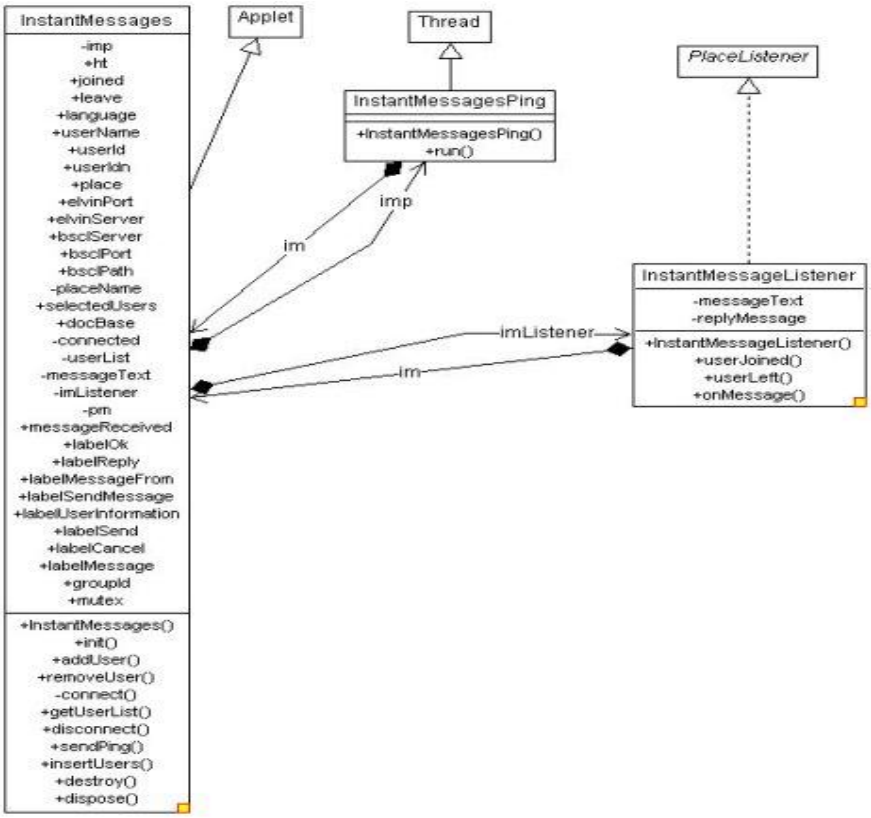

Figura 69. Diagrama UML de la herramienta InstantMessages.

Como se puede apreciar en la figura 69, dicha herramienta también incorpora un mecanismo de ping *InstantMessagesPing* para que desde el servidor se pueda llevar correctamente la lista de usuarios conectados a dicha herramienta.

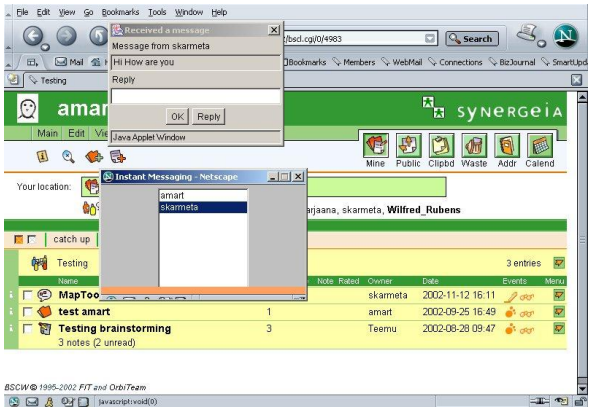

Figura 70. Imagen de la herramienta de InstantMessages.

Esta herramienta proporciona la lista de usuarios conectados en el momento en que está activa, ofreciendo posibilidades para enviar mensajes o bien obtener información del

usuario seleccionado. Para está última operación es necesaria la comunicación con el entorno subyacente, en este caso el BSCL. Más concretamente, en la figura 70 se muestra el aspecto visual de la integración de dicha herramienta en el entorno BSCL.

Debido a que en esta herramienta no se representa un estado común entre varios colaboradores, no se implementó algoritmos de llegada tarde ni se tuvo en cuenta mecanismos de *Floor Control*.

1.7. *Herramientas de seguimiento.*

Entre las herramientas necesarias en un entorno educacional cabe destacar aquellas que facilitan el análisis de lo que ha sucedido en cada una de las sesiones que se han llevado a cabo.

Partiendo de las especificaciones de la parte pedagógica del proyecto, se requirió que la información del BSCL así como de la herramienta colaborativa Maptool fueran guardadas en ficheros. En secciones previas de este apartado hemos visto qué eventos eran de interés en el uso de maptool. Basándonos en la información recogida se puede mostrar la información relativa a un curso, o a un usuario. La información que se puede obtener viene dada por los elementos que se han escrito en el Chat junto con los elementos que se han dibujado. En la figura 71 se muestra la aplicación construida para la obtención de información de la interacción de los estudiantes con el uso del maptool. Esta información se representa bien en gráficas o bien mediante tablas excel.

	A	B	C	D
1				
2		Course 10		
3		CHAT	DRAW	
4	amart	199	102	
5	amart2	124	12	
6	bscl	0	2	
7	jose	6	3	
8	jtavira	17	25	
9				
10				
11				

Figura 71. Herramienta MaptoolLog para la visualización de los eventos producidos.

Puesto que el uso de ficheros hace bastante lenta la consulta, en nuestro diseño final propusimos el uso de una base de datos en la cual se guardaran los eventos necesarios.

En uno de los primeros diseños se planteaba la creación de una interfaz a través de la cual el sistema BSCL pudiese volcar los eventos de logs al bus de eventos y de esta forma el agente o servidor intermedio los pudiese almacenar en base de datos como muestra en la figura 72. Sin embargo al final se optó por el mantenimiento en un fichero.

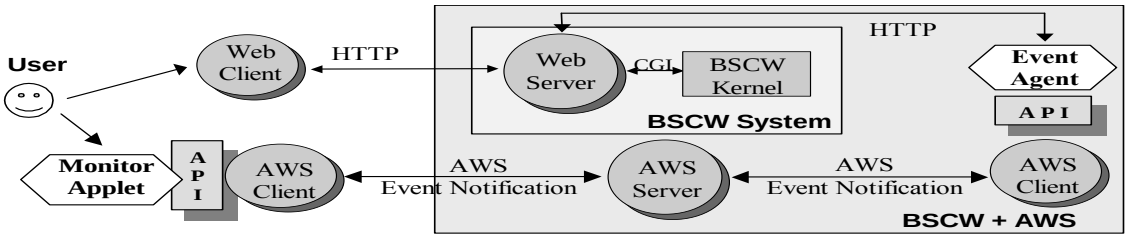

Figura 72. Modelo de logs en el que se recogen tanto los obtenidos de BSCL como los de las herramientas colaborativas.

Para agilizar las consultas de dicho fichero, se realizó un programa que analizaba cada patrón de entrada del fichero de logs guardando información acerca de los cursos, usuarios, información de participación en tablones de discusión (knowledge building), etc en tablas de la base de datos. A su vez también se tuvo que incluir tablas para las relaciones existentes entre cada una de las entidades.

De esta forma se elaboró otra herramienta web que ofrecía información acerca del número de notas leídas y escritas por cada alumno, número de respuestas escritas, tipo de notas usadas por los estudiantes, todas las notas que han usado y eventos de BSCL. Algunas de las pantallas que se generan por el uso de los diferentes opciones en la herramienta son ilustradas en la figura 73.

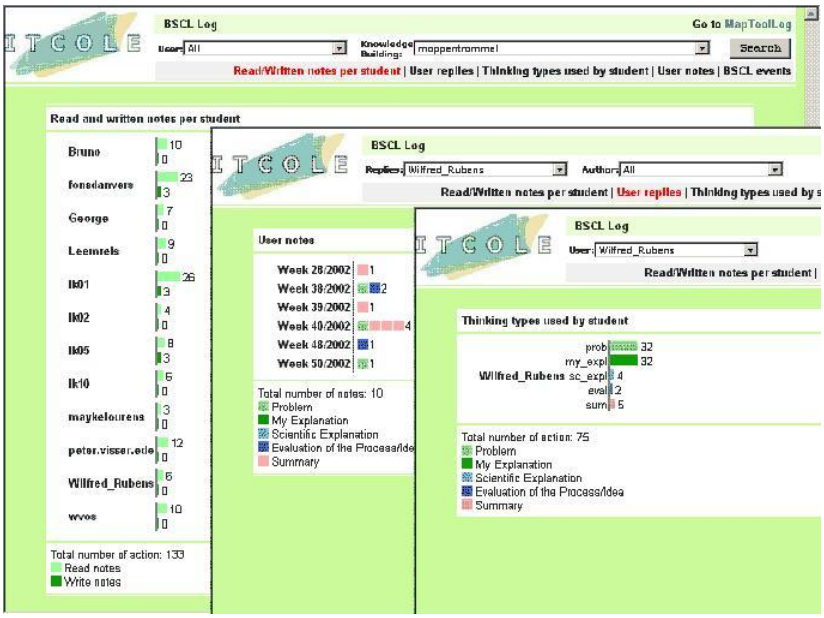

Figura 73. Diferentes muestras de información ante las diferente opciones elegidas en la herramienta BSCL Log.

Si bien este conjunto de herramientas ofrecen información general acerca de la actividad que han llevado a cabo los individuos dentro del área, a veces es preciso contar con una serie de herramientas que nos muestren el proceso llevado a cabo en determinada sesión.

Como hemos visto anteriormente, la herramienta *Maptool* dispone de características asíncronas que permiten a un usuario volver a reanudar una sesión con la información guardada de la última sesión, así como visualizar dicha información mediante el uso de una imagen.

Sin embargo, para aquellos usuarios que no pudieron atender a dicha sesión colaborativa, bien sean profesores bien sean alumnos, resulta más constructivo poder reconstruir la información de manera muy similar a como se ha realizado en la sesión colaborativa. De esta manera un usuario que no pudo participar en la sesión síncrona, podría participar de la experiencia compartida mediante el uso de una herramienta que le permitiera recuperar dicha información y poder visualizarla paso a paso.

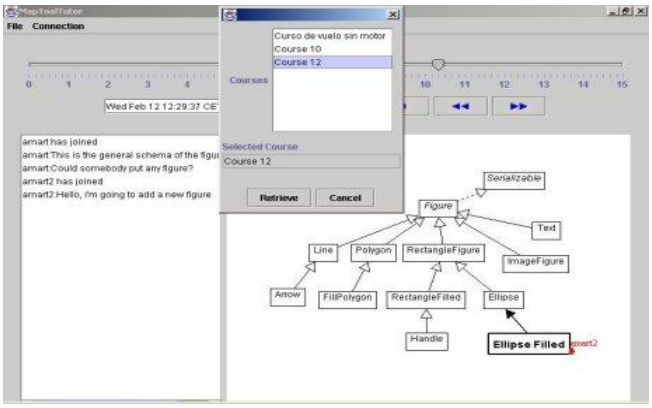

Figura 74. MaptoolTutor que permite la reproducción de una sesión mediante selección del curso y rangos de fechas.

En la figura 74 se muestra la aplicación *MaptoolTutor*. Podemos apreciar que sus mandos principales consisten en los mismo que pueden observarse de cualquier herramienta reproductora, botón de play, botón para realizar avanzar un paso, botón para retroceder un paso, botón de pausa y botón de parada. De igual manera que ocurre con la herramienta *Maptool* se muestra un panel para el área de dibujo en la cual se van dibujando las figuras y un panel para el área de chat dónde se reproducen los mensajes.

Con el objetivo de ofrecer la información de una sesión de Maptool y mostrarla desde que se inicializó por primera vez hasta la ultima modificación, se han introducido nuevos campos en los registros de logs. En cuanto a la información necesaria a guardar para poder construir una herramienta de estas características destacamos los siguientes campos que deben ser enviados por la herramienta *Maptool*:

- currentMills.- Tiempo en milisegundos.
- responsible.- Usuario responsable de dicha acción.
- obj.- El objeto que se ha transformado, creado o borrado o un mensaje.
- action.- Tipo de acción realizada. Los posibles valores son create_draw_object, delete_draw_object, create_chat_note, end_move_object, end_resize_object, clear_drawings.
- position.- Posición del puntero.

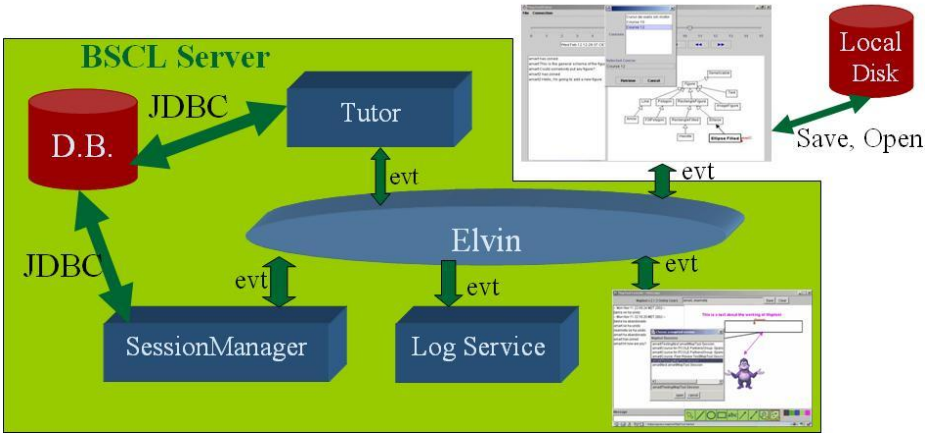

Figura 75. Esquema de comunicación en el almacenamiento y recuperación de los eventos de logs.

Para no interferir en el sistema de logs existente se ha creado nuevas tablas dónde se recoge la información requerida para la nueva extensión en el sistema de logs. El esquema del proceso llevado a cabo viene mostrado en la figura 75. Se puede apreciar la integración de dos nuevos servicios incorporados en el agente servidor.

El servicio *SessionManager* es el encargado de recoger los eventos de interés que envía la herramienta de Maptool en una sesión colaborativa y guardarlos en la correspondiente tabla de eventos. Para el funcionamiento de dicha herramienta se requiere la conexión con el servicio de notificaciones empleado, enviando la petición requerida por el usuario al servicio *Tutor* el cual está activo en el agente de *MaptoolUtils.* Este servicio se encarga de realizar las consultas pertinentes en la base de datos y de ofrecer los datos solicitados por la herramienta *MaptoolTutor*. Una vez recibidos estos, la herramienta calcula la franja de tiempo a visualizar y procede a la reproducción de la sesión colaborativa.

El hecho de que la herramienta de tutorización se conecte con el bus de eventos favorece la independencia en la base de datos elegida, puesto que será responsabilidad del SessionManager de obtener los datos y enviárselos a través del bus de eventos. Otro de las funcionalidades que podrían haberse obtenido de este modelo de comunicación es poder escuchar los eventos que están sucediendo en un momento determinado desde la herramienta colaborativa sin la necesidad de conectarse al sistema.

1.8. *Evaluación del sistema BSCL.*

Más concretamente, en la Figura 76 vemos la evaluación de cada una de las herramientas evaluada por los profesores dónde (1) implica muy desfavorable, y (6) muy favorable.

Functionality	Primary School			Secondary School			Total		
	Mean	N	S.D.	Mean	N	S.D.	Mean	N	S.D.
Groups	5,25	24	0,85	4,81	16	1,56	5,07	40	1,19
Uploading of documents, URL's etc.	4,71	24	0,86	4,53	17	1,07	4,63	41	0,94
MapTool	3,68	19	1,64	3,08	12	1,31	3,45	31	1,52
Instant Messaging	3,88	16	1,96	3,29	7	1,25	3,70	23	1,77
Knowledge Building Area	5,00	24	0,93	4,94	17	1,03	4,98	41	0,96
Thinking types	4,64	22	1,29	4,07	14	1,00	4,42	36	1,20
Address Book	3,94	18	2,73	5,56	18	3,03	4,75	36	2,96
Calendar	3,72	18	2,70	5,83	18	2,85	4,78	36	2,94

Figura 76. Evaluación del impacto pedagógico de las herramientas del BSCL en cada nivel escolar.

Los profesores manifestaron que "*Hay una necesidad significativa y sentido en herramientas como Maptool, sobre todo porque el Maptool parece ser adecuado para el soporte de diseño colaborativo y la construcción del conocimiento colaborativa, tormenta de ideas y colaboración en general*" [Ces03].

En la Figura 77 se muestra la evaluación del entorno BSCL por parte de los alumnos los cuales evaluaron tanto los valores educacionales como el uso del software de este entorno. En la barra vertical estos valores son evaluados desde el 1 al 5 siendo 1 la puntuación más desfavorable y 5 la más favorable. Cada uno de los elementos que formaron parte de la evaluación son detallados a continuación:

1. Social Awareness. Se calcula mediante las respuestas a la pregunta: "Fue fácil ver lo que otros estudiantes estaban realizando".
2. Collaboration. Se calcula mediante las respuestas a las preguntas: "Fue fácil colaborar con otros estudiantes". "El profesor los animó a colaborar". "Les hubiese gustado más colaborar con más usuarios durante el proyecto".
3. Usability and support. Se calcula mediante las respuestas a las preguntas: "Fue fácil colaborar dentro de BSCL". "Estuvieron suficientemente guiados por el profesor durante el proyecto". "Se les dio las suficientes instrucciones de cómo estudiar dentro del entorno". "Se perdieron en el uso del entorno". "No supieron que hacer dentro del entorno".
4. Externalising idea for understanding. Implica preguntas como: "Expliqué mis ideas a otros estudiantes usando BSCL". "Me ayudó ver la ideas/notas creadas por mi". "Fue útil ver las ideas/notas de otros estudiantes". "Fue fácil usar nuevas conexiones entre ideas".
5. Structuring the process of inquiry. Implica preguntas como: "Mientras trabajaba con el entorno entendí como funcionaba el proceso de preguntas". "Fue fácil estructurar el proceso de preguntas dentro del entorno".

Atendiendo a las preguntas de cada uno de los cinco anteriores parámetros se obtuvo los datos mostrados en la Figura 77.

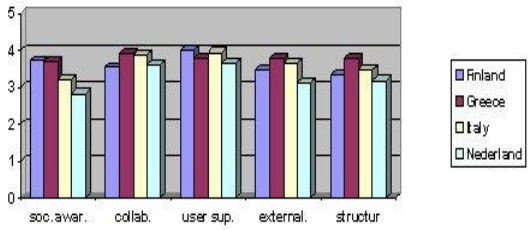

Figura 77. Evaluación de los alumnos de diferentes aspectos tras el uso del entorno BSCL.

En la figura 77 se muestra la evaluación de los alumnos del BSCL, la cual fue un éxito. Más concretamente, acerca del uso de Maptool, los estudiantes que la usaron la puntuaron como una herramienta fácil de usar, con la cual podían comunicarse fácilmente con otros usuarios y desarrollar diagramas conceptuales a través de ella. Sin embargo hubo un gran número de usuarios que no la usaron por falta de una buena infraestructura técnica dentro de los colegios dónde se llevaron a cabo los experimentos.

El éxito del uso de estas tecnologías está fuertemente relacionado con las infraestructuras de red y las inversiones tecnológicas realizadas en los colegios, así como de la disponibilidad de contar con un servidor instalado en la propia Intranet de los colegios.

1.9. Conclusión.

Con el desarrollo de este esquema de comunicación hemos obtenido un sistema colaborativo que permite la integración de un conjunto de herramientas síncronas en un entorno ya existente. Al igual que ocurría con la plataforma ANTS se ofrece una API para el desarrollo de herramientas colaborativas independiente del modelo elegido para llevar a cabo el paradigma MOM o de comunicación asíncrona.

La pieza fundamental del éxito o fracaso de la colaboración dentro del sistema la constituye el bus de eventos puesto que toda la información del sistema fluye a través de él. No sólo sirve de apoyo en la colaboración síncrona entre herramientas sino que también constituye la base central sobre la que hemos basado la comunicación entre las herramientas y el sistema educacional subyacente. Además consituye el motor para gestionar los eventos del log tanto para guardarlos como para su recuperación.

Como hemos podido apreciar el agente servidor entre las herramientas colaborativas y el entorno educacional ofrece un conjunto de servicios extensibles que pueden enriquecer dicha integración proporcionando comunicación asíncrona siempre y cuando el entorno contenedor proporcione una API para llevar a cabo dichos servicios.

Hemos mostrado el diseño de un sistema colaborativo síncrono así como la integración de este sistema en dos entornos web llamados BSCL y FLE mostrando la facilidad de integración de este sistema en ambos entornos.

En el diseño de este sistema ofrecemos un conjunto de servicios básicos que pretenden dar soporte al guardado y obtención de información, así como un sistema de logs que sirva para el almacenamiento de los eventos relevantes para posteriormente poder analizarlos. Si bien este sistema está planteado para la colaboración síncrona, la comunicación de ciertas operaciones al entorno subyacente a través de estos servicios y la API del entorno, favorecen la colaboración asíncrona.

Dicho sistema resuelve problemas como son el mantenimiento de la lista de los usuarios online llevando un control de aquellas herramientas cliente que siguen trabajando y de las que han quedado en estado de inactividad. Para ello, se ha diseñado un algoritmo de Presence Awareness el cual puede ser aplicado en cualquier entorno educacional. Los únicos cambios a realizar en dicho esquema son las llamadas a los procedimientos join y leave implementadas en los entornos, siempre y cuando dichos sistemas necesiten saber los usuarios que hay conectados en el sistema.

Además hemos mostrado como dicho sistema ha sido integrado en dos plataformas educacionales siguiendo diferentes modelos de comunicación con el entorno subyacente. De esta manera se demuestra la reutilización del sistema y su fácil adaptación en cualquier entorno Web.

Otro de los resultados obtenidos es el diseño de un sistema de logs para recoger los eventos que generan las herramientas colaborativas. Dicho sistema de eventos constituye un elemento fundamental para recoger aquellos eventos que se envian en la colaboración y que nos sirvan para la obtención de información relevante acerca del trabajo realizado por los estudiantes. En este diseño el bus de eventos vuelve a ser la pieza principal puesto que mediante la escucha de los eventos que se transfieren a través de él se puede recoger aquellos que son necesarios para el guardado de logs.

Las medidas de rendimiento llevadas a cabo han mostrado como la incorporación del nuevo sistema no sobrecarga al entorno educacional, haciendo posible la comunicación desde cualquier punto sin necesidad de grandes requerimientos en el equipo que se conecta. Como hemos visto al principio de esta sección este era uno de los objetivos que se pretendian obtener, puesto que la inclusión de un sistema que enlenteciera al entorno existente implicaría el rechazo al uso de las nuevas herramientas.

Con este resultado hemos obtenido un sistema colaborativo que ofrece flexibilidad, adaptabilidad y reutilización en la integración en diferentes entornos web existentes.

Con los resultados obtenidos de la evaluación de los estudiantes podemos concluir que la introducción de herramientas colaborativas dentro de sistemas educativos juega un papel fundamental en la discusión y entendimiento del grupo. Además, el uso de las la colaboración síncrona junto con la asíncrona ayudan a difundir el conocimiento obtenido en la colaboración y plasmarlo en documentos dentro de los entornos.

El uso de las herramientas colaborativas síncronas constituye un enclave fundamental para obtener el conocimiento tácito de cada uno de los colaboradores, pudiendo dejar dicho conocimiento plasmado en documentos para su posterior revisión.

2. Diseño de una Plataforma Colaborativa aplicada al Aprendizaje por Descubrimiento.

En el capítulo de antecedentes se ha mostrado una serie de características de los sistemas CSCL. Además se ha profundizado en uno tipo de aprendizaje que surge dentro del campo de CSCL que es el llamado Aprendizaje por Descubrimiento.

Hemos mostrado como Wouter establece una serie de necesidades en una plataforma que sirva de soporte para el aprendizaje por descubrimiento, ofreciendo a su vez diferentes escenarios para la colaboración síncrona y para la colaboración asíncrona.

De nuestras experiencias previas podemos indicar que la unión de ambos escenarios enriquece la colaboración ofreciendo una base sólida para hacerla más rápida y eficaz ofreciendo a su vez la flexibilidad en el tiempo. Además la combinación de ambas facilita la obtención del conocimiento tácito de los colaboradores reflejados en documentos y ficheros que nos sirven para su posterior visualización.

Por lo que hemos podido apreciar en el capítulo tercero el modelo de programación que impone la plataforma ANTS resulta ser demasiado complejo para la creación de aplicaciones por parte de programadores no pertenecientes al ámbito de la informática. Como hemos analizado anteriormente la plataforma ANTS no ofrece un modelo de datos adecuado para que las herramientas que dependan de datos binarios puedan guardar su estado siguiendo la programación en modelos.

El segundo resultado obtenido en nuestro trabajo de investigación tiene como fin proveer una plataforma colaborativa que pueda ser aplicable a todos los ámbitos de colaboración incluyendo el aprendizaje por descubrimiento y que en dicha plataforma sea relativamente sencilla la integración de nuevas herramientas por toda clase de programadores.

Una plataforma de Aprendizaje por Descubrimiento necesita una serie de requisitos especiales para llevar a cabo la colaboración a través de la experimentación. Si bien se trata de un sistema colaborativo, también exige ciertas características como son *Marco de Referencia*, *Herramientas colaborativas*, *Espacio para la experimentación* y *Escenarios de colaboración* que han sido comentadas con anterioridad.

El diseño de nuestra plataforma colaborativa ha tenido su aplicación en un entorno de Aprendizaje por Descubrimiento, más concretamente aplicado a la experimentación dentro de la Física. Para incluir este campo dentro de los ámbitos de colaboración de nuestra plataforma colaborativa, es necesario diseñar la colaboración tanto entre los mismos tipos de herramientas colaborativas así como entre los experimentos y las herramientas colaborativas destinados a interpretarlos.

Además, con el desarrollo de esta plataforma se pretende proporcionar un conjunto de interfaces las cuales faciliten al programador el diseño de herramientas colaborativas sin

tener el total conocimiento de las tecnologías subyacentes empleadas en la construcción de dicha arquitectura.

Una de las aplicaciones desarrolladas sobre esta plataforma ha sido enmarcada dentro del proyecto de investigación COLAB en el cual han colaborado desde la parte pedagógica las universidades University of Twente (Amsterdand) y University of Kiel (Germany). En la implementación de herramientas colaborativas así como de experimentos cabe destacar la colaboración de Amstel Institute (Netherlands), Studio Teos (Italy), el grupo COLOS de la Física de la Universidad de Murcia y nuestro grupo de investigación ANTS.

2.1. Diseño de la plataforma.

Con este diseño pretendemos ofrecer una plataforma colaborativa genérica que puede ser aplicada a cualquier ámbito colaborativo incluyendo el aprendizaje por descubrimiento. Para ello dicha plataforma debe cumplir los siguientes requisitos a nivel arquitectónico:

1. La arquitectura debe soportar la colaboración asíncrona, en el sentido de que debe posibilitar el guardado de la información que se maneja en cada sesión a la vez que la recuperación de dicha información en interacciones posteriores.

2. La arquitectura debe proporcionar un canal de propagación de eventos para el intercambio de información entre las diferentes aplicaciones desarrolladas para la colaboración síncrona. Más concretamente, se utilizará el modelo replicado en la programación de herramientas colaborativas de tal manera que mandarán eventos a través del canal para que el resto de colaboradores actualicen sus vistas con la nueva información.

3. La información manejada debe ser tratada en sesiones independientes, de tal forma que lo que se realice en la sesión de un determinado grupo no sea visible a otros grupos.

4. La arquitectura debe ser modular, tal que permita de una forma sencilla la introducción de nuevas herramientas o la inclusión de herramientas existentes. Además dicha arquitectura deberá ser construida de tal manera que permita separar la parte de interfaz de la parte de programación.

5. Se debe de proveer una API clara y sencilla que permita a todos los miembros de la comunidad el manejo de esta arquitectura en el diseño e integración de herramientas sin necesidad de tener conocimiento de todas las tecnologías usadas en la elaboración de dicha plataforma.

6. Esta plataforma también debe incluir soporte para almacenar toda la información necesaria del sistema así como los recursos que se vayan a usar.

7. Con el fin de mantener un sistema de logs la plataforma debe incluir un mecanismo para la recogida de aquellos eventos de interés proporcionados por la interacción del usuario. Para llevar a cabo esta recogida y envío de eventos, nos basaremos en el

uso de un canal de eventos a través del cual el sistema de logs se suscriba a ciertos valores y las herramientas propaguen los eventos de interés.

Cómo podemos observar en el diseño de esta plataforma el bus de eventos vuelve a ser la pieza fundamental sobre la cual se lleva a cabo la colaboración, teniendo que transferir tanto la información de los resultados de los experimentos así como la información intercambiada entre las herramientas de la plataforma. De igual manera que ocurría con el sistema colaborativo ANTS, el bus de eventos constituye el eje central de la propagación de los eventos de logs, recogiéndose de él toda la información de interés para su posterior guardado.

Previa a la estructuración de las interfaces y del sistema en general, la parte pedagógica definió el conjunto de conceptos e información en la que se debía estructurar un sistema de aprendizaje. Esta definición de conceptos está basada en los mundos virtuales, es decir la clase de objetos MOO. De esta manera se define las entidades que forman parte del sistema y las relaciones entre ellas siguiendo el siguiente esquema:

1. Todos los usuarios se agruparán mediante el uso de Grupos.

2. Cada grupo podrá acceder a *Building* (Edificio), que es el término empleado para agrupar a los usuarios atendiendo a los diferentes objetivos a tratar en la tarea colaborativa. Más concretamente en la aplicación de esta plataforma al aprendizaje por descubrimiento los edificios indican los diferentes grupos de experimentos que se pueden tratar, como por ejemplo, edificios relacionados con la mecánica, la gestión de agua, la electricidad, etc.

3. Cada *Building* esta formado por un conjunto de *Floor* (Plantas). Cada uno de los *Floor* indicará un nivel de dificultad dentro del concepto de edificio, es decir, de la tarea colaborativa que se esté llevando a cabo. En nuestro caso de aplicación el uso de diferentes *Floor* nos ayuda a establecer diferentes niveles de dificultad atendiendo a la clase de experimentos que se recogen en cada planta.

4. En cada *Floor* existen *Room* (Habitaciones) cada una de las cuales ofrecerá un conjunto de herramientas atendiendo a las tareas específicas de dicha área o habitación. Más concretamente, en la aplicación COLAB las habitaciones existentes son:

    - *Hall*, es un área de encuentro de los diferentes miembros del grupo una vez que entran en la planta.
    - *Meeting* consiste en la habitación dónde se realizarán las discusiones, razonamiento y planificación de las tareas a realizar del resultado obtenido.
    - *Theory* es la habitación en la cual se experimenta acerca de las teorías relacionadas con el experimento en cuestión.
    - *Laboratory* es la habitación en la cual se experimenta a través del uso de simuladores que se encuentran en máquinas remotas, ofreciendo los resultados de estos experimentos a través de herramientas visuales destinadas a ello.

5. La abstracción de *Phenomena* (Experimentos) se refieren a los experimentos y simuladores disponibles. En la plataforma se debe proporcionar los mecanismos

necesarios para soportar tanto simuladores software así como laboratorios remotos. Más concretamente, los *simuladores* serán programas ejecutándose en un servidor permitiendo a diferentes grupos colaborar con diferentes instancias del mismo tipo de simulador. Por otro lado, los *laboratorios remotos* comprenden entidades hardware ejecutándose en localizaciones específicas. Debido a las limitaciones hardware, solamente un grupo será habilitado para el uso de cada uno de los laboratorios remotos.

6. El concepto de *VisualTool* (Herramienta Visual) recoge aquellas herramientas que permiten visualizar en la representación del cliente toda la información relativa a la colaboración de los usuarios. En el caso de que hayan experimentos (*Phenomena*) definidos, también habrán *VisualTool* responsables de tratar los eventos recibidos desde los experimentos y mostrarlos a los colaboradores.

De los estudios realizados dentro de la plataforma ANTS hemos podido observar que el uso de las especificaciones J2EE constituye una base fundamental en la construcción de nuevas aplicaciones, abstrayendo al programador del uso de transacciones, del lenguaje de programación usado y ofreciendo independencia en la base de datos usada. De igual manera su amplio conjunto de estándares facilita la creación de código portable. Más concretamente, en nuestros desarrollos nos hemos basado en el uso de la política *"Container Managed Persistence"* ya que ofrece un nivel de independencia mayor en cuanto a la fuente de datos a usar y hace más rápido el desarrollo de aplicaciones.

Con el objetivo de obtener una arquitectura que nos permita almacenar la información que se maneje en el entorno, tanto de configuraciones iniciales como de sesiones, hemos basado la parte asíncrona de este sistema en tecnologías J2EE (Java 2 Enterprise Edition) [J2EE] usando el servidor JBOSS [JBOSS] en nuestros desarrollos. Más concretamente nuestra decisión por JBOSS se ve influida por su carácter de código abierto con lo cual no se necesita ninguna licencia para su instalación. Este es un aspecto fundamental sobre todo cuando se trabaja en entornos educativos, los cuales uno de los principales receptores de estas tecnologías suelen ser los colegios tanto de primaria como secundaria.

Mediante el uso de *entity beans* podemos gestionar cada una de las entidades necesarias en el desarrollo de esta arquitectura, es decir todas aquellas estructuras que son gestionadas por parte del servidor como tablas de una base de datos.

De la arquitectura ANTS hemos visto como el uso de un bus de eventos basado en la librería ANTS.COMM proporciona una fachada ante los sistemas de notificación usados, confiriendo de esta manera la característica de extensibilidad a cualquier arquitectura construida sobre ella.

Por lo tanto, para la realización del canal de eventos de esta nueva arquitectura haremos uso de esta API. Sobre este canal de eventos se llevaran a cabo el intercambio de información de las diferentes sesiones que establecen las herramientas colaborativas así como la recogida de eventos del sistema de logs que se va a proporcionar. Debido a que en esta plataforma debe incluir soporte a la colaboración síncrona, el bus de eventos será uno de los pilares fundamentales sobre el cual se basará la colaboración.

En cuanto a las sesiones, básicamente se organizan a nivel de planta (*Floor)* y a nivel de habitación (*Room).* De esta forma, especificando el grupo, *Building y Floor* estableceremos una comunicación a nivel de *Floor*. Si además se indica el *Room* dónde se encuentran los usuarios, se establecerá una sesión a nivel de *Room.* Las herramientas y/o experimentos que envíen sus eventos en una sesión de *Floor* independiente de la habitación que estén los colaboradores mostrarán la misma información. Sin embargo, las herramientas que colaboran a nivel de *Room* mostrarán diferente información en las distintas habitaciones (*Room*) dónde se lleve la colaboración.

Con el objetivo de abstraer al programador del uso directo de estas tecnologías y la dificultad que ellas implican, en el diseño de esta arquitectura se han incluido dos elementos clave que son el *Broker* y el *Environment*.

Básicamente el *Broker* permite obtener información de todos los *entity beans* de la plataforma J2EE de una manera más sencilla. El *Environment* pretende ser la interfaz usada por las herramientas de colaboración desarrolladas en la plataforma para realizar las conexiones al *Broker* al igual que el envío de eventos a través del bus ANTS.COMM.

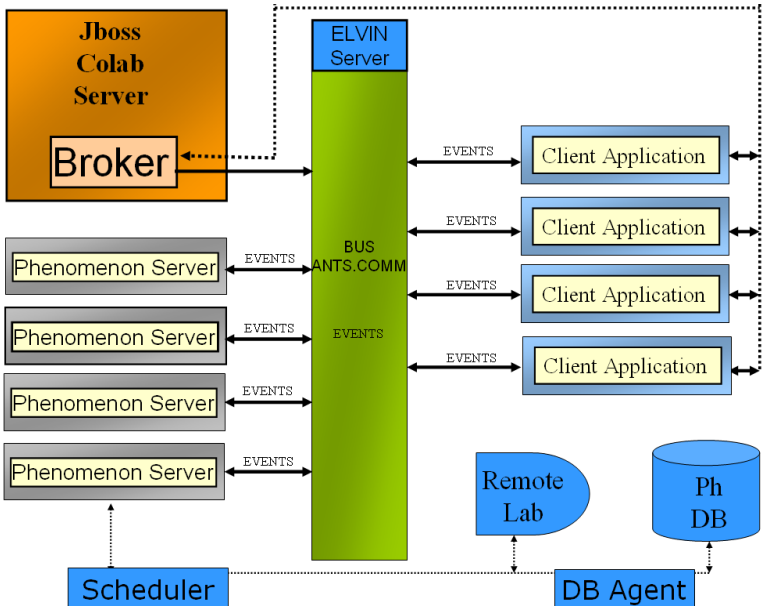

Figura 78. Diseño de la plataforma Colab

En definitiva, se proporcionará una Plataforma Colaborativa en la que se debe tener en cuenta los usuarios que tienen acceso a la aplicación (U), la gestión de sesiones (S), los recursos a manejar (R) así como un conjunto de políticas de *Floor Control* (F), abarcando de esta manera todos los elementos de un sistema colaborativo como hemos visto en el capítulo segundo.

2.2. Broker

Con el fin de facilitar el acceso a cada uno de los *entity beans* se ha implementado un *SessionBean* denominado *Broker* el cual permite mediante una API crear las instancias de cada uno de los *entity beans* que constituye la aplicación, así como modificarlas o destruirlas encapsulando su uso a los usuarios finales.

Siguiendo las especificaciones que impone J2EE respecto a los *SessionBean* cada cliente establece a través del *Environment* una sesión con el *Broker*. De tal forma que se manejará una instancia diferente de dicho *SessionBean*.

De esta manera las operaciones de lectura podrán ser ejecutadas paralelamente aunque no ocurrirá lo mismo con las de escritura. Por el hecho de usar la propiedad "Container Manager Persistence", los problemas de concurrencia son solucionados por el servidor J2EE liberando al programador de la implementación de un control de gestión de recursos.

Este esquema es ilustrado en la figura 79, de la que se puede observar como el *SessionBean Broker* hace referencia a todos los *entity bean*s del sistema. Si bien muchos de ellos harán uso de la base de datos para guardar información, otros como en el caso de las sesiones, harán uso de ficheros a través de los cuales irán guardando la información necesaria de la configuración tanto de edificios (*Building*), plantas (*Floors*), habitaciones (*Room*s), herramientas visuales (*VisualTools*) y experimentos (*Phenomenon*).

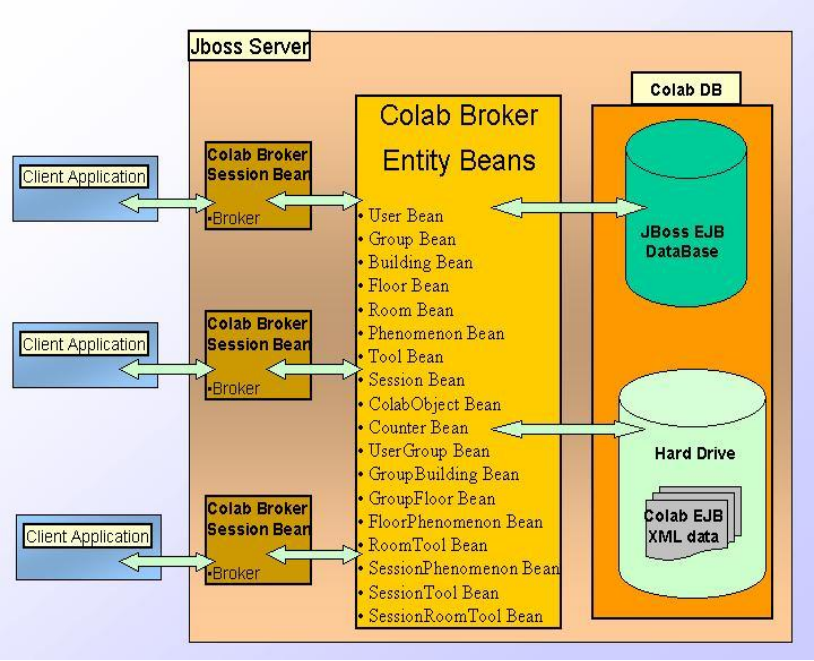

Figura 79. Esquema del Broker

Cualquier operación relacionada con cualquier *entity bean* del sistema así como las relaciones existentes entre ellos serán manejadas mediante el uso del interfaz proporcionado por el *Broker*. En la tabla 5 se muestran cada uno de los *entity bean*s y su funcionalidad dentro del sistema.

NOMBRE	DESCRIPCION
UserBean	Gestiona la información de los usuarios.
GroupBean	Gestiona la información de los grupos.
UserGroupBean	Gestiona la información referente a la asociación de usuarios a grupos.
BuildingBean	Gestiona la información de los *Buildings*.
GroupBuildingBean	Gestiona la información de la asociación de grupos y *Buildings*.
FloorBean	Gestiona la información de los *Floors.*
PhenomenonBean	Gestiona la información de los *Phenomenon*.
FloorPhenomenonBean	Gestiona la información de la asociación de *Phenomenon* en *Floor*.
RoomBean	Gestiona la información de *Rooms*
ToolBean	Gestiona la información de *Tools* o herramientas de la aplicación.
RoomToolBean	Gestiona la información de la asociación de *Tools* dentro de *Room*.
SessionBean	Gestiona la información de una sesión establecida por un grupo.
SessionToolBean	Gestiona la información de cada una de las Tool que están asociadas a una sesión establecida en un Floor.
SessionRoomToolBean	Gestiona la información de las Tools de una determinada que están asociadas a una sesión establecida en una *Room*.
SessionPhenomenonBean	Gestiona la información referente a los *Phenomenon* que están asociados a una sesión establecida en un *Floor*.
CounterBean	Gestiona los identificadores asociados a cada entidad.
ColabObjectBean	Gestiona la información de los objetos que se crean por el uso de herramientas, o bien porque se incorporan al sistema.
TokenFloorBean	Gestiona los mecanismos de sincronización.

Tabla 5. Descripción de los entity beans que forman parte de la arquitectura.

Más concretamente, la figura 80 muestra el esquema de las relaciones de cada uno de los *entity beans* que forman la arquitectura indicando además la cardinalidad correspondiente a las asociaciones entre ellos. El *EntityBean* TokenFloor no queda reflejado en dicho esquema puesto que no tiene ninguna dependencia con respecto de los restantes *entity beans*.

Como vemos, el acceso a la plataforma se hará mediante el uso de grupos, en los cuales se recoge a un grupo de usuarios (U). De esta forma se aborda qué usuarios tienen acceso a qué recursos dependiendo de cómo se establezcan las relaciones entre cada uno de los elementos que forman la plataforma.

Mediante el uso de *ColabObject* se gestionará el conjunto de objetos que pueden ser manejados desde la aplicación, pudiendo así guardar los valores de una simulación *"dataset"* o de cualquier modelo *"model"*. El uso de dichos objetos viene dado por la necesidad de imponer una base común de tal forma que múltiples herramientas podrán

recoger dichos valores e interpretarlos de acuerdo con la representación impuesta en la definición de la herramienta o aplicación.

En este objeto, los datos son representados mediante el uso del estándar XML y atendiendo a un modelo de datos común para todas las herramientas, que podrá ser interpretado mediante el uso de los correspondientes parsers. De igual manera, la configuración de todos los recursos del sistema, como son las *VisualTools* y *Phenomenon* hacen uso de la misma especificación de modelo. De esta manera, los diferentes recursos pueden interpretar la misma información favoreciendo la característica de ***Marco de Referencia*** la cual es considerada por Wouter en su especificación de un entorno de aprendizaje por descubrimiento**.**

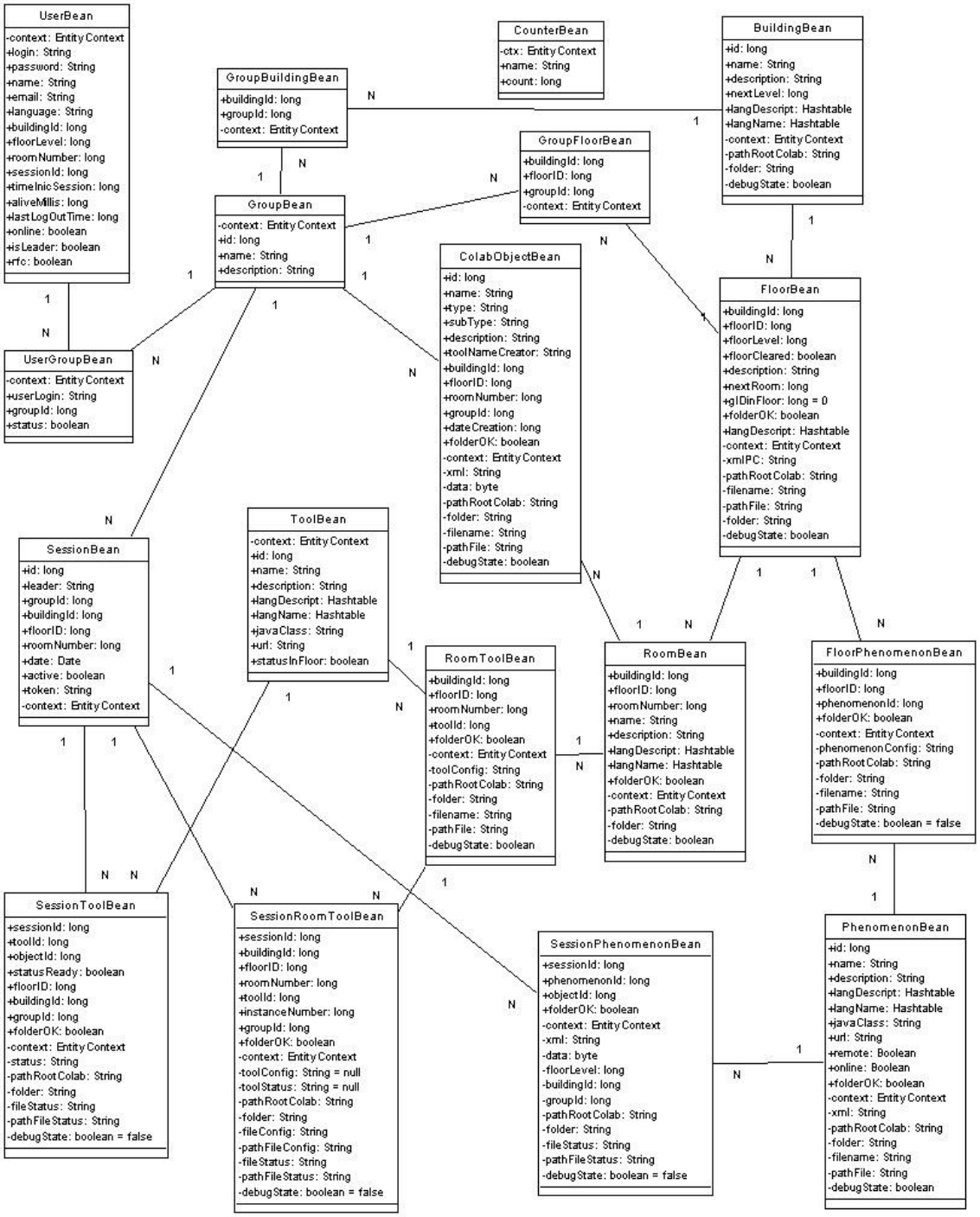

Figura 80. Diagrama de asociación de los entity beans.

La interfaz *Broker* proporciona el conjunto de operaciones que pueden realizarse en cada uno de los beans así como la asociación de éstos ofreciendo métodos para la creación, destrucción, lectura de datos y modificación de cada uno de ellos. La creación de los *entity beans* relacionados con las sesiones sólo se llevará a cabo por el uso de las siguientes operaciones:

- *public void getIntoBuilding(String userLogin, long groupID, long buildingID)*
- *public void getOutOfBuilding(UserDetails userD, long groupID)*
- *public SessionDetails getIntoFloor(String userLogin, long gID, FloorDetails floorD)*
- *public void getOutOfFloor(UserDetails userD, long gID)*
- *public UserDetails getIntoRoom(String userLogin, SessionDetails session, RoomDetails room)*
- *public UserDetails getOutOfRoom(UserDetails user, SessionDetails session)*

De esta manera se abarca el conjunto de sesiones (S) existentes en la plataforma, clasificadas en dos niveles: el de *Floor* que es más general y el de *Room* que es más restringido y desde el cual las herramientas se pueden suscribir a eventos que se propagan a través del *Floor*.

Para la introducción de datos así como las asociaciones de cada una de las entidades dentro de la base de datos del entorno, se ha creado una aplicación que gestiona dichos datos a través del *Broker* y permite la creación, modificación, asociación y borrado de éstos.

Esta herramienta permite la entrada de datos del sistema y la asignación de ficheros XML a cada una de las acciones que lo requieren. No sólo sirve para asociar información, sino también para recoger los ficheros de configuración de una herramienta en una determinada sesión o también para la copia de sesiones de un grupo a otros de la planta.

A su vez permite hacer copias y recuperaciones de toda la información almacenada en un determinado servidor. Más concretamente, en la figura 81 se muestra el esquema de dicha aplicación.

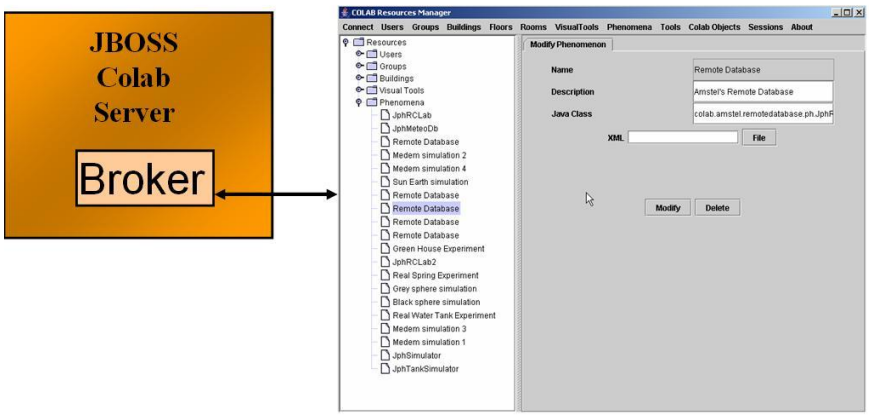

Figura 81. Gestor de recursos.

2.3. Environment.

Como ya hemos comentado anteriormente, el *Environment* constituye el interfaz usado por las aplicaciones clientes, así como las implementaciones de los experimentos o *Phenomenon.* Su principal objetivo es poder recoger la información del repositorio de datos a través del *Broker* así como del envío y recogida de eventos del Bus de comunicación. De esta forma las *VisualTools* o herramientas visuales de la aplicación cliente serán capaces de recoger la información guardad en el sistema así como colaborar entre ellas enviando y recibiendo información a través del Bus usando el *Environment.*

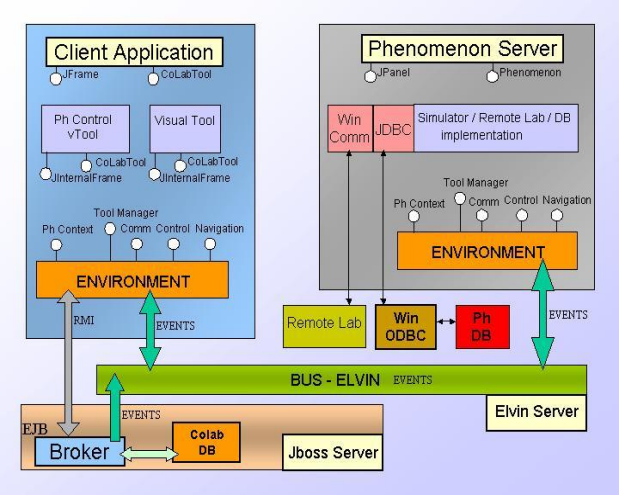

Figura 82. Esquema de relaciones del Environment con el resto de herramientas del sistema.

Como se puede observar de la figura 82 el *Environment* es el encargado de realizar la transmisión de eventos a través del bus de comunicación, manteniendo una sesión con el servidor de notificaciones elegido mediante el uso de su clase *Session*. De esta forma, el bus de comunicación será implementado de manera independiente a las tecnologías subyacentes para hacer el reparto de eventos. En cuanto a la información de dicho servidor es mantenida en el *Broker* mediante el uso de un *entity bean* denominado *MomConfiguration*.

Como se puede apreciar en la figura 82, el *Environment* realiza mediante el modelo de comunicación síncrono llamadas al *Broker* para obtener la información almacenada en las fuentes de datos, mientras que usa el modelo de comunicación asíncrona para propagar eventos entre las herramientas distribuidas.

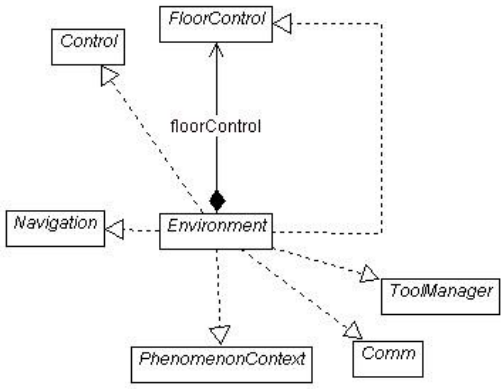

Figura 83. Descripción UML de la clase Environment.

Del esquema UML ofrecido en la figura 83, se puede observar que la clase *Environment* implementa un conjunto de interfaces, cada uno dispuesto para definir un conjunto de operaciones diferentes. Estas interfaces son *Comm* para el intercambio de eventos entre las herramientas colaborativas, *PhenomenonContext* para el tratamiento de la información referida a los experimentos, *Control* para manejar eventos tales como el cambio de leader, cambio de habitación, recogida de estado, *Floor Control* que maneja métodos relacionados con la gestión de *Floor Control*, *Navigation* para llevar a cabo las operaciones de navegación dentro de la aplicación (entrada de *room*, salida del *room*, entrada en el *building*, salida del *building*, etc) y *ToolManager* para la gestión de las herramientas que se tratan en la sesión.

Más concretamente las aplicaciones clientes y los experimentos trabajan con las clases *ClientEnvironment* y *ServerEnvironment* respectivamente, las cuales heredan de la clase *Environment.* De esta manera, cada una de estas nuevas clases incluye los métodos necesarios en cada una de las partes, recogiendo por un lado aquellos métodos que afectan a los clientes y por otro lado aquellos que tienen su aplicación en los servidores de experimentos.

Como hemos analizado anteriormente, la librería ANTS.COMM maneja las notificaciones y suscripciones mediante el uso de un *hashtable* incluyendo los pares clave-valor. Con el fin de facilitar la tarea al programador, se han definido un conjunto de eventos para el tratamiento interno de los *hashtable* de tal forma que el desarrollador puede trabajar con los eventos mediante el manejo de objetos.

Para el envío de eventos a través del bus de comunicación, el *Environment* proporciona el método *sendEvent* en el cual se le debe indicar el responsable del evento, el evento y el tipo de evento. En el caso de los eventos que se origina en el interior de la plataforma se usa la palabra clave *"Internal"* como responsable de los eventos.

A continuación exponemos el conjunto de eventos que hemos incorporado para facilitar la gestión de estos al programador de aplicaciones.

Manejo de Eventos

Con el fin de facilitar el manejo de los eventos dentro de la plataforma de aprendizaje, se proporcionan una serie de tipos de eventos que entre otras cosas sirven para

encapsular el uso de las notificaciones manejadas por ANTS.COMM. Estos eventos quedan representados en la tabla 6.

EVENTOS	DESCRIPCION
ColabActionEvent	Usado para informar de las acciones que tienen que realizar los *Phenomenon* como el reseteo de estos.
ControlEvent	Usado para manejar eventos de control que se producen en el entrono como pueden ser cambio de localización (changeLocation), realizar cualquier acción dentro del entorno como puede ser la de resetear los valores de una simulación (performAction), transferLeaderShip, grantFloor, revokeFloor, claimFloor, para recoger el estado de las herramientas setStatus, evento para indicar que un objeto ha sido guardado objectSaved.
LoadObjectEvent	Este evento es usado para indicar que se requiere que se carguen los valores de un objeto.
VariableEvent	Usado para manejar los eventos de ciertas variables de las que se tengan cierto interés para ser manejadas de manera independiente.
VectorEvent	Usado para manejar los eventos de vectores, pudiendo así tratar una lista de valores.
ColabEvent	Usado para representar los eventos de las aplicaciones.

Tabla 6. Tabla de Eventos.

De igual manera se ha incluido un listener por cada uno de los tipos de eventos manejados, ofreciendo una serie de métodos *adddListenerXXXX* desde el *Environment* para realizar la suscripción a los listeners correspondientes. Siguiendo el modelo de programación impuesto por el uso de la librería ANTS.COMM, estos nuevos listeners implementarán la clase *RemoteEventListener* y en el método *eventReceived* se tratará el evento recibido. Así pues, se notificará la llegada de un evento mediante el uso de *updateEvent* en cada una de las entidades que hayan suscrito su interés en dicho evento.

En cuanto a la suscripción y desuscripción de eventos se proporciona un conjunto de métodos *addListenerXXXX* o *removeListenerXXXX* los cuales se ofrece a través del uso del *Environment* (véase los ejemplos mostrados en la figura 84).

Mediante el uso de listeners, diferentes recursos pueden escuchar los mismos tipos de notificaciones, pudiendo así ofrecer la misma información con diferente visualización, como sería el caso de una tabla y una gráfica. De esta manera, distintos recursos del sistema pueden trabajar sobre la misma base, es decir sobre el mismo **Marco de Referencia**. Esto se debe sobre todo a que la información en el bus de eventos es transferida usando el estándar XML.

Figura 84. Interfaces para la gestión de los eventos y listeners.

Mediante el uso del *Environment*, se ofrece la comunicación a los escenarios síncrono como el asíncrono dando soporte a los diferentes ***Escenarios colaborativos.***

2.4. Algoritmo de Presence Awareness

Como ya hemos comentado en anteriores secciones y capítulos, una de las características importantes en cualquier sistema que permita la colaboración síncrona es la gestión de los usuarios que están conectados en un cierto instante del tiempo. Para ello es necesario incorporar en el sistema colaborativo un servicio que lleve la gestión de dichos usuarios.

El principal problema que surge en esta gestión es la identificación de aquellos usuarios que han perdido su conexión en el sistema, de tal manera que no pueden comunicarse. Anteriormente en el desarrollo del sistema colaborativo ANTS hemos aportado un esquema en el cual la herramienta desarrollada era la encargada de implementar un mecanismo de envío de ping para indicar su presencia.

Sin embargo en este caso, puesto que las herramientas elaboradas se construyen haciendo uso del *Environment* para facilitar la comunicación a las fuentes de datos así como al bus de eventos, es aquí dónde se realizará la notificación del estado de la actividad de la aplicación cliente. Además, para que la comunicación de dicha información sea más segura y el cliente tenga una certeza de que se ha realizado, se invocará a un método del *Broker* mediante el uso de comunicación síncrona, de esta forma se asegura que no se pierde el evento.

Puesto que un usuario sólo podrá estar activo en una sesión, no habrá ningún bloqueo en la actualización de las instancias *UserBean* correspondientes a cada uno de los usuarios que estén invocando el cambio en el parámetro *isAlive*.

Para ello, cuando desde una aplicación cliente se valide un usuario, se arranca un Thread llamado *IsAlive* el cual cada cierto tiempo invoca al método *isAlive* del *Broker*, indicando el usuario responsable de dicha acción. Dicho método recoge el timestamp de cuando se ha realizado dicha llamada y se actualiza en un campo de la correspondiente instancia *UserBean* del usuario.

Las llamadas a los métodos del *Broker* siguen una semántica de errores *"at most once invocation"*, con lo cual si en la primera llamada no ha obtenido un resultado no se reintenta una nueva llamada. Con el objetivo de que se realice un número de reintentos hemos comprobado el tipo de excepción y forzado a realizar la operación un número limitado de veces, evitando de esta manera una desconexión si en la primera invocación ha fallado. Si tras ese número de reintentos no se ha podido realizar la llamada *isAlive* en el *Broker*, se procede a la salida (logout) del usuario, y a informar acerca de la pérdida de conexión.

Con el fin de llevar la cuenta del estado de los usuarios, se ha incluido un servicio llamado *OnlineChecker* el cual es un Thread que se arranca en el inicio del servidor. Cada cierto tiempo recoge la lista de usuarios conectados al sistema y comprueba que la diferencia entre la última vez que se hizo un *isAlive* y el tiempo actual del sistema no sea inferior a cierto límite. Si este es el caso, se supone que el usuario ha perdido la conexión al sistema y se realiza un *logout* del mismo, liberando todos los recursos que dicho usuarios había reservado.

Más concretamente, el esquema de funcionamiento de dicho servicio es mostrado mediante el diagrama de actividad en la figura 85.

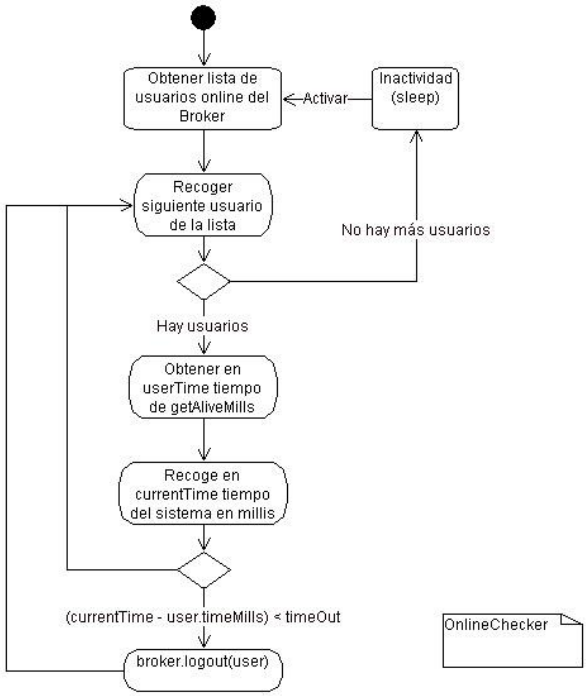

Figura 85. Diagrama de Actividad del servicio Online Checker para la gestión de usuario online.

Una de las características necesarias en la interfaz gráfica es mostrar la lista de los usuarios conectados al sistema así como la localización de éstos en el área compartida. Con este fin se ha incorporado en la aplicación COLAB un panel que muestra los usuarios que pertenecen al grupo y mediante círculos de colores se muestra quien está activo, como muestra la figura 86. Más concretamente el color verde del círculo denota al usuario coordinador en dicha *Room* y el color rojo denota el resto de usuarios. Gracias al servicio de presencia que se ejecuta en el servidor se mantiene una lista fiable de los usuarios conectados al sistema.

Además de mostrar qué usuarios están conectados dentro de una sesión, también es útil conocer la localización de dichos usuarios, detallando a un nivel mayor la característica de presencia. Para ello hace uso de diferentes iconos con forma de triángulo con diferentes colores para las diferentes habitaciones o *Room*. Para escuchar los eventos de cambio dentro de las habitaciones de los diferentes usuarios esta herramienta deberá estar suscrita a los eventos que ocurren al nivel de *Floor*, de tal forma que ante un evento de cambio de *Room* actualice los valores adecuados en el panel de información. Toda esta información facilita las características de *Social Awareness* de un sistema colaborativo.

Figura 86. Presence Awareness

2.5. Sistema de Logs.

Con el fin de recoger los eventos de interés generados por la colaboración para su análisis posterior, se ha implementado un servicio de log, llamado *OnlineLogger*. Este servicio establece una conexión mediante el uso de las librerías ANTS.COMM suscribiendo su interés a los eventos etiquetados con *applicationLog* a través de un listener llamado *OnlineLoggerListener* dónde trata los eventos recibidos acorde con la acción requerida para ello.

Con el objetivo de que las herramientas envíen aquellos eventos de los que se requieren mantener cierta información, se provee desde el *ClientEnvironment* el método *sendEventLog*. A través de esté método, en la creación de la notificación se incluye un campo denominado *applicationLog* que indica que este evento sólo deberá ser tratado por el servicio de log.

Este servicio de logs puede guardar la información en base de datos mediante el uso de *entity beans* o bien en ficheros.

Más concretamente, en la aplicación COLAB desarrollada sobre nuestra plataforma se optó por el uso de ficheros XML, de tal forma que se guarda uno por grupo. Desde la parte pedagógica de este proyecto se especificó la recogida de la información de los eventos que vienen ilustrados en la tabla 7.

Tool	Action
PC	Addgoal
	AddSubgoal
	ShowDescription
	ShowLinks
	ShowHints
	ShowNotes
	ShowHistory
	ShowReport
	ShowGoalTree
	CompleteGoal
	GoalName
	AddNote
General	Tools opening/closing
	LoginFloor#Building#
	LogoutFloor#Building#
	LeaveRoomMeeting
	LeaveRoomHall
	LeaveRoomTheory
	LeaveRoomLab
	EnterRoomHall
	EnterRoomMeeting
	EnterRoomTheory
	EnterRoomLab
WhiteBoard	DrawShape
	MoveShape
	ResizeShape
	ChangeColor
	EditTextColor
ModelEditor	SelectOpenModel
	AddRelation
	DeleteRelation
	AddFlow
	DeleteFlow
	CheckSpecification
	RunModel
	OneStepRun
	StopModel
	ResetModel
Chat	Sendmsg
	SendToRoom
	SentToFloor

Tabla 7. Eventos de interés dentro del sistema COLAB.

En la figura 87 se muestra el esquema seguido en la monitorización de los eventos de logs del sistema. Se puede observar como a través del bus de eventos se transfiere toda la información que fluye desde los recursos del sistema hacia el sistema de log. Dicho sistema recoge las notificaciones de interés y las almacena en diferentes ficheros según el grupo dónde se haya originado.

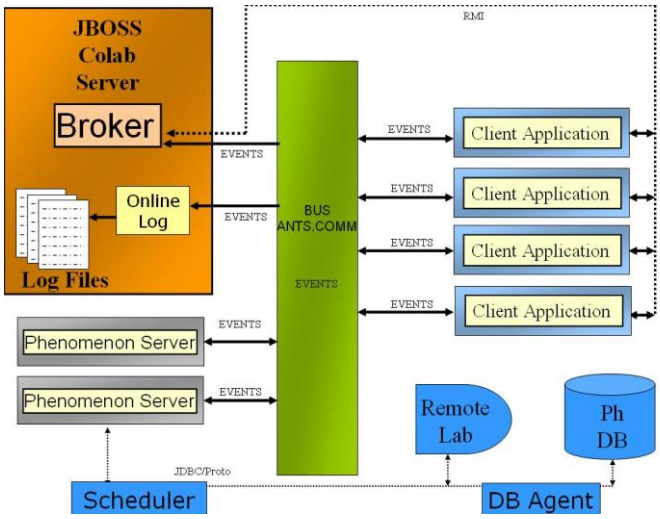

Figura 87. Esquema de logs dentro de la plataforma colaborativa.

Una vez obtenida dicha información el análisis será realizado mediante el uso de la herramienta Analog, de la cual la universidad de Twente ha incluido un módulo para tratar los ficheros generados por el sistema de logs implementado en Colab [Anj04] (véase figura 88). Desde esta herramienta se ofrece la siguiente información:

- Ver la secuencia de pasos que han ido realizando los colaboradores dentro del entorno, como puede ser los movimientos entre las *Room*.
- Numero de veces que cada usuario visita una herramienta.
- Categorizar las acciones

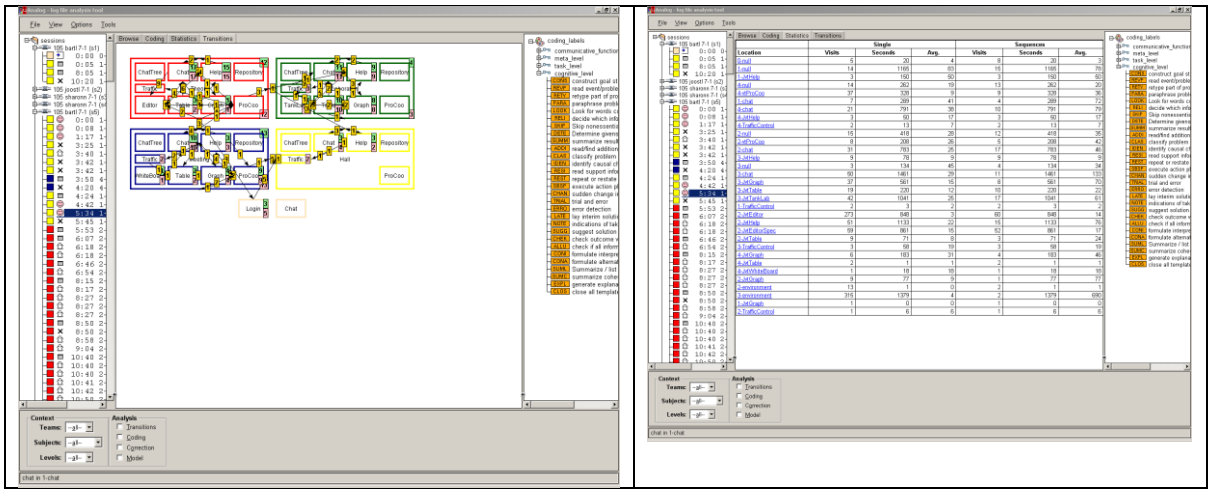

Figura 88. Aplicación Analog con los ficheros de logs recogidos en COLAB.

Mediante esta herramienta se puede realizar un análisis cuantitativo de la actividad del alumno pudiendo observar la frecuencia de uso de cada una de las herramientas así como la duración del trabajo en ellas.

En el caso de la herramienta de chat también se puede realizar un análisis cualitativo, puesto que se puede seguir la conversación establecida por los alumnos. Esto es debido

a que en la definición del sistema de logs, en el caso del chat también se guardan los mensajes que fluyen a través de la colaboración.

2.6. Floor Control.

Una de las características esenciales en cualquier entorno que permita la colaboración síncrona es la inclusión de las políticas que permitan el acceso coordinado a los recursos por parte de los usuarios.

Como hemos podido ver en el capítulo de antecedentes existen una gran variedad de mecanismos y de políticas. El uso de uno u otro dependerá de la aplicación a desarrollar.

En el desarrollo de la aplicación COLAB la parte pedagógica indicó que la colaboración llevada a cabo entre el grupo de estudiantes debía ser similar a la que transcurre en las colaboraciones cara a cara, en la cual uno de los estudiantes del grupo es el que coordina el trabajo decidiendo quién y cuando se tiene acceso a los recursos.

De hecho, este es uno de los fundamentos sobre los que se basa el aprendizaje colaborativo, que sean los propios estudiantes mediante la experiencia compartida los que construyan su conocimiento. Debido a esto, hemos optado por implementar la política **Chair-guidance** para coordinar el acceso a los recursos. De tal forma el *chair* decide cuando y qué usuarios tienen acceso a los recursos. Como la clave fundamental es que los propios estudiantes sean los responsables de su aprendizaje, el *chai*r es uno de los colaboradores de la sesión.

Más concretamente en la integración de un sistema de *Floor Control* dentro de nuestra plataforma hemos optado por el mecanismo **token asking**, de tal forma que los usuarios preguntan por el floor en lugar de pasarlo de unos a otros.

Para implementar el token hemos optado por un mecanismo centralizado para el control de la concurrencia. Más concretamente hemos hecho uso de un *entity bean* el cual sigue la estructura mostrada en la figura 89.

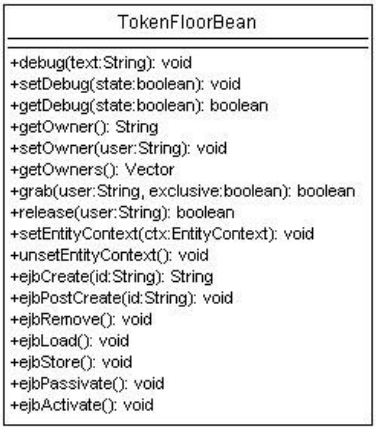

Figura 89. Diagrama del Entity bean TokenFloorBean.

Entre las operaciones más características de un elemento token podemos destacar la operación *grab* que permite a un usuario convertirse en el propietario, indicando si este token es de uso exclusivo o puede tener más de un propietario. Otra de las operaciones

características es la operación *release* que permite a un usuario abandonar su estado de propietario.

Con el objetivo de que estas operaciones queden accesibles al *Environment* sin necesidad de trabajar directamente con el *entity bean* anteriormente ilustrado, se han incluido una serie de métodos de tal manera que permitan la creación y gestión del token a través del uso del *Broker*.

Una de las cuestiones abarcadas a la hora de desarrollar los gestores de *Floor Control* dentro del desarrollo de la aplicación COLAB, así como en las herramientas gráficas para su gestión fue la del nivel al cual establecer la gestión de recursos.

Se plantearon dos situaciones: una de ellas conlleva la gestión a nivel de cada una de las herramientas o bien llevarla a nivel de sesión. En la primera de ellas se debe indicar los recursos a los cuales se le concede acceso al *floor holder* y en la segunda de ellas el *floor holder* tiene acceso a todos los recursos de la sesión.

En el caso de que el *Floor Control* se gestione a nivel de habitación (*Room*), manteniendo el control en cada uno de los recursos de ésta, para la creación de los diferentes tokens es necesario especificar la localización (Group, *Building*, *Floor* y *Room*) del usuario que solicita realizar una operación sobre el token. De esta manera, el identificador del token será la concatenación de los parámetros que nos da la localización del usuario, es decir, las diferentes sesiones.

En el caso de que se necesite establecer el control en cada uno de los recursos de una misma sesión, se puede incluye además el identificador de la herramienta y de esta manera se mantendrá un control por cada recurso pudiendo así poder tener diferentes usuarios manejando diferentes herramientas.

Más concretamente en el desarrollo de COLAB se llevó a cabo la implementación a nivel de sesión, de tal forma que el *floor holder* puede manejar cualquier recurso disponible en dicha sesión.

La interfaz de un *Floor Control* debe contener básicamente los métodos que se exponen a continuación:

- **claim**. Un usuario pregunta por el floor.
- **create**. Crea el floor.
- **expand**. Añade un usuario nuevo al floor.
- **grant**. Concede el floor a un usuario.
- **info**. Muestra información acerca del floor.
- **release**. El floor holder libera el floor.
- **remove**. Borra el floor.
- **revoke**. Fuerza al floor holder a liberar el floor.
- **track**. Realiza un seguimiento de la actividad llevada a cabo dentro del floor.
- **shrink**. El usuario abandona el floor.
- **throwOut**. Fuerza a un usuario a que abandone el floor.

Con el objetivo de hacer la gestión del *Floor Control* lo más extensible posible dentro de cualquier arquitectura se hace necesario incorporar una interfaz genérica, de tal

forma que cada nuevo gestor que se incorpore dentro de la arquitectura implemente dicha interfaz.

Siguiendo los principios del aprendizaje colaborativo la colaboración entre usuarios debe ser llevada a cabo de una manera similar a las colaboraciones "face to face". Desde la parte pedagógica se instruye que sea un miembro del grupo el que coordine el trabajo de los demás (chair o representante).

Teniendo en cuenta lo anterior, en la integración de los gestores de *Floor Control* introducidos dentro de nuestra plataforma sólo se ha considerado aquellos que contemplan la política chair-guidance.

En cuanto a los gestores de *Floor Control* se han introducido los siguientes tipos dentro de la arquitectura COLAB:

- **ListFloorControlManager**. El chair determina que usuarios pueden acceder a los recursos, así como los que no pueden tener acceso. Así pues los usuarios a los cuales se les ha concedido el privilegio de acceso a los recursos serán capaces de acceder a los datos compartidos. Sin embargo este acceso no es exclusivo, es decir, varios usuarios pueden estar modificando los datos de los recursos compartidos.

- **TurnAskingFloorControlManager**. El chair determina qué usuario puede acceder a los recursos compartidos cada vez. De esta manera el usuario al que se le conceden los privilegios podrá modificar los datos de los recursos. El acceso a los recursos se hace de manera exclusiva, de tal forma que sólo un usuario puede estar modificando el estado de las herramientas cada vez.

Ambas implementaciones se basarán en el uso del token de tal forma que dentro de esta estructura se indicará que usuarios tienen acceso al token y si este ha sido grabado de manera exclusiva o no. En cuanto a la información de los solicitantes del floor es mantenida en cada una de las aplicaciones clientes.

Una de las bases fundamentales en la implementación del *Floor Control* es el bus de eventos, el cual servirá para propagar los cambios de estado del floor a cada uno de las aplicaciones cliente, favoreciendo de esta forma la gestión de la interfaz gráfica en la notificación al usuario de los estados de cada uno de los usuarios respecto al control del floor.

Con el objetivo de que la información de los cambios de floor se propague a los diferentes usuarios del sistema, algunas de estas operaciones envían ciertos eventos en el bus los cuales son recibidos por aquellos usuarios que se encuentran en la misma sesión. De esta forma ante la llegada de cierto evento, los usuarios actualizan sus representaciones de *Floor Control* con la información recibida. Esta información también nos servirá para mostrar los elementos gráficos apropiados dentro de la herramienta de control del floor manejada por el chair, la cual sirve para garantizar o revocar los permisos de acceso a los recursos.

En la tabla 8 se muestra como se han implementado los dos gestores de *Floor Control* anteriormente expuestos.

	ListFloorControl	TurnAskingFloorControl
floor_claim	El usuario solicita el acceso al recurso compartido, a partir de entonces es tarea del chair concedérselo o no. A su vez se propaga un evento para hacer saber a los demás la petición.	El usuario solicita el acceso al recurso compartido, a partir de entonces es tarea del chair concedérselo o no. A su vez se propaga un evento para hacer saber a los demás la petición.
floor_create	Crea el floor.	Crea el floor.
floor_expand	Añade un usuario al floor. Esta operación se suele invocar tras recibir un evento de claim de un usuario.	Añade un usuario al floor. Esta operación se suele invocar tras recibir un evento de claim de un usuario.
floor_grant	El chair permite a un usuario que acceda al recurso compartido. En este gestor el chair no pierde el acceso al recurso puesto que este no es exclusivo. Esta operación propaga un evento en el bus para indicar que a un usuario se le ha concedido el acceso al recurso.	El chair le concede el turno de acceso exclusivo sobre el recurso compartido a un nuevo usuario. Por lo tanto el chair abandonará el uso de éste. En este gestor el chair pierde el acceso a los recursos puesto que el acceso se concede de manera exclusiva. Esta operación propaga un evento en el bus para indicar que a un usuario se le ha concedido el acceso al recurso.
floor_info	El usuario recibe información acerca de quien es el chair y una lista con los usuarios conectados, indicando cuales están autorizados para acceder al recurso compartido y cuales no.	El usuario recibe información acerca de quien es el chair, quien tiene el turno de acceso sobre el recurso compartido en ese momento, una lista con los usuarios conectados y otra con cuales de ellos han solicitado el turno de acceso al recurso.
floor_release	Este método no es necesario porque es el chair quien se encarga de conceder / revocar permisos.	Este método no es necesario porque es el chair quien se encarga de conceder / revocar permisos.
floor_remove	Elimina el floor.	Elimina el floor.
floor_revoke	El chair quita el privilegio de acceso al recurso compartido a un determinado usuario. Esta operación propaga un evento	El chair le quita el turno de acceso exclusivo sobre el recurso compartido al usuario que lo tiene para concederlo a otro o así mismo.

	que es transmitido a los demás usuarios de la sesión.	Esta operación propaga un evento que es transmitido a los demás usuarios de la sesión.
floor_shrink	Un usuario abandona voluntariamente el floor. Si es precisamente el chair el que se va debe dejar el mando a otro usuario (de esto ya se encarga el *environment*) revocando los privilegios concedidos a los usuarios participantes de la sesión.	Un usuario abandona voluntariamente el floor. Si es precisamente el chair el que se va debe dejar el mando a otro usuario (de esto ya se encarga el *environment*) revocando los privilegios concedidos a los usuarios participantes de la sesión.
floor_throwOut	El chair obliga a un usuario a abandonar el floor. Tiene su aplicación cuando un claimer está solicitando el acceso al recurso y el chair decide negárselo.	El chair obliga a un usuario a abandonar el floor. Tiene su aplicación cuando un claimer está solicitando el acceso al recurso y el chair decide negárselo.

Tabla 8. Métodos y acciones de los distintos gestores de Floor Control.

En la figura 90 se muestra el esquema UML de los gestores de *Floor Control* implementados.

Más concretamente en COLAB la elección del chair (leader) se hace de forma automática, tal que el primer usuario que entra en la sesión se convierte en el chair y en el *floor holder*.

Como hemos visto anteriormente, todas las herramientas de nuestra plataforma colaborativa manejan el *Environment* para llevar a cabo las operaciones dentro del entorno. Con el objetivo de que las herramientas accedan a los métodos para gestionar el floor, así como obtener información acerca de él, la clase *Environment* implementa la interfaz *FloorControl*, que proporciona los métodos necesarios para llevar a cabo la gestión de *Floor Control*. Más concretamente es la clase *ClientEnvironment* la que implementa dichos métodos.

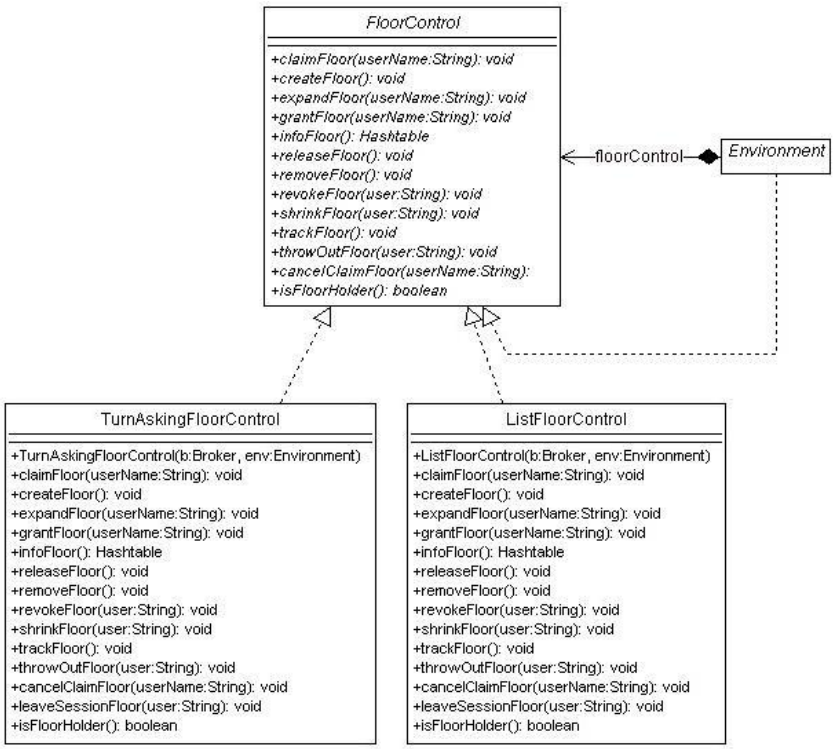

Figura 90. Gestores de Floor Control implementados en COLAB.

Desde el *ClientEnvironment* se gestionan los procesos de creación de sesiones cambio de chair, y abandono de la sesión. Por lo tanto en estas fases es dónde se debe manejar el ciclo de vida del gestor *Floor Control* para cada usuario.

Debido a que los gestores de *Floor Control* implementados se basan en la figura del chair, es necesario establecer como se elegirá dicho chair. Existen multitudes de opciones para la elección del leader o coordinador por parte de los usuarios de la sesión, como por ejemplo los esquemas de votación. Sin embargo, en la aplicación COLAB se optó por elegirlo de manera automática, siendo el primer usuario que entra en una habitación el chair o leader dentro de esa sesión.

Como hemos comentado anteriormente las sesiones se establecen a nivel de habitación (*Room*) o a nivel de planta (*Floor*). Sin embargo, los usuarios acceden a los recursos por medio de las sesiones establecidas en las habitaciones, sesión a nivel de *Room*. Por esta razón, en la aplicación COLAB la creación del gestor de *Floor Control* será llevada a a cabo en la operación *getIntoRoom()*. Una vez creado (*createFloor*) si el usuario es el chair este se encargará del floor invocando a la operación *grantFloor*.

Cuando un usuario abandona la sesión (*getOutOfRoom*) invocará el método *shrinkFloor* y en el caso de ser el último usuario además se borra el floor.

Otro de los aspectos que se han citado anteriormente cuando se exponían los métodos de cada uno de los gestores es qué hacer cuando el chair o leader de la sesión abandona ésta. En la implementación de los gestores del floor hemos optado por llevar a cabo un cambio de chair, enviando un evento indicando esta situación. Si bien el cambio de leader o chair es una operación llevada a cabo por el sistema, en la operación

shrinkFloor el leader libera el token al igual que revoca el privilegio de acceso al floor a todos los *floor holder* hasta el momento.

Como ya hemos comentado anteriormente, la gestión del *Floor Control* es replicada. Por lo tanto, cada cambio es propagado a los demás usuarios de la sesión para que actualicen sus representaciones con el icono correspondiente. Cuando un usuario recibe una solicitud de *claimFloor* sobre el floor realiza una operación *expandFloor*, añadiendo dicho usuario como un claimer dentro del floor. Si es el leader el que recibe dicho evento además incluye dicho usuario en la visualización de los que solicitan el floor.

En la tabla 8 se explica la implementación de operaciones de los gestores de *Floor Control* incluidos en la plataforma. Algunas de las operaciones expuestas son llevadas a cabo por el chair, por lo cual para que los demás colaboradores sepan los nuevos cambios, estas operaciones deben generar eventos a través del bus. Entre estas operaciones podemos destacar las operaciones de *grant*, *revoke* y *throwOut*.

A continuación expondremos cuál es el comportamiento implementado dentro de COLAB ante la llegada de los eventos que se propagan por la llamada de algunas de las operaciones llevadas a cabo dentro del gestor de *Floor Control*.

En el caso de que se reciba un evento de *grantFloor*, si se trata del usuario al que se le concede el floor, éste anula la petición de turno y notifica a los demás dicho cambio. Los demás colaboradores actualizan las representaciones gráficas, el estado del learner y se visualiza el icono apropiado para indicar qué el usuario es uno de los *floor holder*.

Más concretamente en la figura 91 se muestra el esquema de funcionamiento de una operación de grant realizada por el chair. Previo al envío del evento *grantFloor* el chair debe realizar la operación grab en el token de la sesión para el usuario al cual se le concede el privilegio.

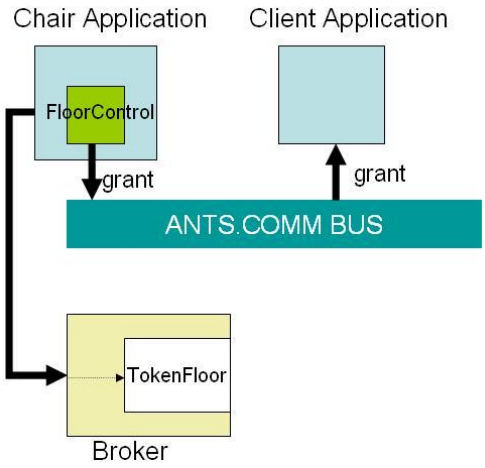

Figura 91. Comunicación de una operación grant.

En el caso de que se reciba un evento *revokeFloor* todos los usuarios que reciban este evento actualizan la representación de dicho usuario suprimiendo el icono de *floor holder*. El esquema del revoke es similar al del grant, con la diferencia del cambio de

eventos. De igual manera, previo al envío del evento de *revokeFloor* el chair debe realizar la operación *release* sobre el token de la sesión.

En el caso de recibir un evento *throwOut* (véase figura 92) se cancela la petición de floor de este usuario en todas las representaciones de los colaboradores. En el caso del throwOut sólo se envía el evento a través del BUS para que todos los colaboradores anulen la petición de *claimFloor* del usuario al que el chair ha expulsado. De esta forma se podrá actualizar esta información tanto en la instancia de *Floor Control* así como en la representación gráfica.

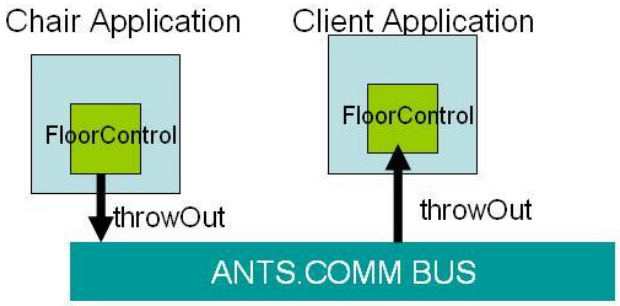

Figura 92. Comunicación de una operación throwOut.

En la gestión de las herramientas dentro de la aplicación Colab, cuando el *Environment* recibe un evento de *grantFloor* o *revokeFloor* se encarga de modificar el acceso poniéndolo a fuerte o débil respectivamente. En cuanto a las herramientas que tienen control fuerte, como son las de ejecución de experimentos o dibujo en el editor, deben de controlar en su implementación si el usuario que esta accediendo es un *floor holder* o no. Para ello el *Environment* incorpora una función *isFloorHolder()* para saber si el colaborador actual es uno de los usuario que posee el acceso a los recursos de la sesión.

Figura 93. Pantalla gráfica de la representación en COLAB del Floor Control.

Como hemos podido ver en el capítulo de antecedentes una de las partes fundamentales en el diseño del *Floor Control* es la incorporación de gráficos para informar del estado del floor a los usuarios de la sesión. Más concretamente en la aplicación COLAB hemos elegido el símbolo de la mano levantada para simbolizar que un usuario está solicitando el acceso al recurso compartido, como se puede apreciar en la figura 93. Para indicar que un usuario es el *floor holder*, hemos usado el icono de la pluma.

En cuanto a la ventana que maneja el chair para conceder o revocar permisos a los usuarios es mostrada en la parte derecha de la figura 93, con la lista de los usuarios que están dentro del *Floor Control*. Más concretamente, los que tienen el botón de revoke es porque son los *floor holder*s y los que tienen el botón de grant son aquellos usuarios que solicitan el acceso a los recursos pero todavía no se les ha concedido.

2.7. Llegada Tarde.

Uno de los aspectos fundamentales a tener en cuenta en el desarrollo de cualquier sistema que implique colaboración síncrona es la llegada tarde de los usuarios a la sesión. Se requiere la implementación de algoritmos para que un usuario que llegue tarde pueda recibir la información relevante para colaborar con el resto de los usuarios. Además, la implementación de estos algoritmos debe tener en cuenta múltiples factores para que la entrada del nuevo usuario se haga lo más pronto posible, sin la apariencia de quedarse bloqueado.

En el capítulo de antecedentes hemos visto como Jürgen Vogel, Martin Mauve proponen un algoritmo totalmente distribuido para abordar el problema de los usuarios que llegan tarde. En su propuesta los usuarios responsables del envío del estado de la sesión inicializarn un timer a un valor dado mediante el uso de un algoritmo exponencial con realimentación. Sin embargo, dicho diseño tiene la desventaja de que dos aplicaciones pueden consumir sus retardos al mismo tiempo, enviando por tanto los estados a la vez. O bien, que uno de los usuarios no reciba el evento de envío de estado y por tanto no pare su timer enviando el estado.

En aplicaciones dónde la transferencia de datos no requiera un ancho de banda considerable, dicho diseño es totalmente adecuado, puesto que la replicación del evento de estado no supone un alto consumo de ancho de banda de la red. Sin embargo, en aquellas aplicaciones dónde el envío del estado supone una transferencia de datos considerablemente grande se hace necesario controlar la replicación de los eventos de estado.

Más concretamente en plataformas colaborativas dónde se llevan a cabo experimentaciones y simulaciones se producen grandes cantidades de datos a transferir a cada una de las aplicaciones colaboradora.

Por lo tanto, como uno de los campos para los que puede ser usada nuestra plataforma colaborativa es el ámbito de la experimentación, en el diseño del algoritmo de *late join* para el control del envío de datos se ha hecho necesario la inclusión de mecanismos de sincronización basados en control centralizados, como son el uso de token. De tal forma que los servidores de estado cuando reciben el evento de llegada de un nuevo usuario intentan grabar un token asociado a dicha sesión. Dicho token debe ser creado por cada una de las sesiones, indicando también que es el usado para el envío del evento de estado. Además es creado en modo exclusivo de tal manera que sólo un usuario accede a grabarlo. Así pues, sólo el usuario que graba el token será el responsable de enviar el estado.

Uno de los problemas que puede surgir cuando se da un gran retardo en la red en el envío de datos, es que un usuario reciba el evento de llegada tarde de otro usuario cuando ya se le haya enviado el estado, con lo cual el token estaría libre para grabarlo y

el estado podría enviarse de nuevo. Para evitar este problema hemos optado por incorporar en la plataforma, un *entity bean* denominado *SessionUse*r con los estados de las herramientas de cada usuario dependiendo de la sesión. Por lo tanto en este nuevo *entity bean* se incluyen un registro por cada una de las herramientas y *datasets* con los que puede trabajar el usuario en esa sesión. En cada una de estas entradas se puede marcar los siguientes valores:

SENT.- Indica que el estado ha sido recibido.
NULL.- Indica que el estado de la herramienta no ha sido enviado.

En la definición del algoritmo de *late join*, uno de los primeros pasos es determinar quienes van a ser los servidores de estado ante la llegada tarde de los usuarios. Debido a que las aplicaciones de aprendizaje colaborativo se basan en sesiones con un grupo de usuarios reducido, siendo este número no superior a cinco usuarios, es aconsejable que todos ellos sean servidores del estado. Por lo tanto todos ellos guardan la información del estado.

Esto también plantea la necesidad de que los usuarios que se conecten tarde reciban cuanto antes el estado de la sesión dónde se están conectando, evitando al mismo tiempo un retardo bastante acusado en la entrada en la aplicación.

El algoritmo de llegada tarde implementado en nuestra plataforma colaborativa se basa en la idea propuesta por Jürgen Vogel, Martin Mauve con las variantes que se exponen a continuación.

En cuanto a la forma de elegir quien será el encargado de enviar el estado, todos los usuarios recibirán el evento de llegada tarde y lo tratarán en el *eventArrived* del *RemoteEventListener*. En el momento que escuchan este evento recogen el token de sesión (*TokenStateSessionEvent*), el cual es creado en modo exclusivo, y comprueban si ha sido enviado el estado a ese usuario. En el caso de que no haya sido enviado, se envía el estado de las herramientas y se marcan en la tabla *SessionUser*.

```
If (env.grabTokenStateSessionEvent(userName)){
  If (Broker.askSessionUserState(user,session,tools) = = NULL)  {
        env.sendStateEvent();
         broker.markSessionUserState(user,session,tools, SENT);
     }
      env.releaseTokenStateSessionEvent(userName);
}
```

Un usuario que se conecte al mismo tiempo que se está enviando el estado lo puede recibir, marcando en la tabla *SessionUser* con estado SENT las herramientas de las que ha recibido el estado. De tal manera que cuando los servidores de estado escuchen el evento de llegada de este nuevo usuario y comprueban que tiene el estado SENT, no lo enviarán.

Otra de la cuestiones a abordar es cómo enviar el estado. Enviar todo el estado de forma inmediata puede implicar un retardo bastante considerable en la entrada del usuario al sistema, sobre todo en los casos en los que los *dataset* o conjunto de valores de un experimento sean de un tamaño considerablemente grande. En nuestro caso los datos

que más ocupan ancho de banda y más tiempo necesitan para ser retransmitidos son aquellos que se generan en la experimentación.

Por tanto la solución que hemos propuesto para nuestra plataforma implica dos fases en el envío del estado. En la primera sólo se envía el estado de aquellas herramientas que no dependan de los *dataset* y posteriormente cuando el usuario ya ha entrado en el sistema se le envía los resultados de los *dataset* en el caso de que tratemos con una colaboración basada en la experimentación. El uso de otro token diferente viene determinado por el hecho de que el envío del *dataset* no interfiera con el envío del estado de las demás herramientas.

```
If (env.grabTokenDataSetStateSessionEvent(userName)){
  If (Broker.askSessionUserState(user,session,dataset) = = NULL){
        env.sendDataSetStateEvent();
       broker.markSessionUserState(user,session,dataset, SENT);
     }
     env.releaseTokenDatasetStateSessionEvent();
}
```

2.8. Diseño de las herramientas visuales y experimentos.

En el diseño e implementación de los recursos del sistema, es necesario abarcar las fases de desarrollo de los componentes, interacción entre ellos y por último la integración de estos componentes en el entorno.

A continuación se detallarán cada una de estas fases.

Desarrollo de los componentes.

En cuanto a los recursos del sistema, hemos podido ver que en nuestra plataforma colaborativa es necesario tanto abarcar aquellas herramientas visuales que muestran al usuario final el estado de la colaboración así como los experimentos que pueden ser lanzados en la plataforma para ejecutar ciertas acciones y cuyos resultados se propagan a través del bus de comunicación.

Todas las herramientas que deseen ser incorporadas en la plataforma de aprendizaje por descubrimiento deben implementar la interfaz *ColabTool*, cuyo diagrama UML queda representado en la figura 94.

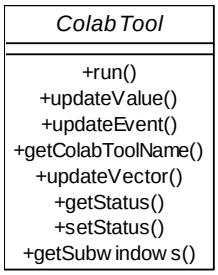

Figura 94. Interfaz ColabTool.

A través del método *run*, las herramientas recibirán la instancia del *Environment* en el cual han sido creadas, así cómo una configuración del estado de dichas herramientas. Este estado puede ser de una configuración previamente guardada en el caso de que se inicializa por primera vez la sesión, o bien el estado que se está manejando en la sesión. De esta manera, las herramientas podrán enviar eventos a través de la instancia del *environment* que se le ha proporcionado, así como el tratamiento de ciertos eventos mediante los métodos *updateEvent* o *updateValue.* Como se ha comentado anteriormente, las herramientas visuales contenidas en la aplicación cliente manejarán los métodos proporcionados en la clase *ClientEnvironment.*

Cuando se recibe el evento por uno de los listeners dónde se ha suscrito la herramienta, se invocará al método *update* para que dicha herramienta visualice el nuevo evento. De esta manera las herramientas pueden comunicarse entre ellas propagando los nuevos cambios a través del bus de eventos. Así pues, en el desarrollo de estas herramientas se recomienda el modelo replicado.

Así pues mediante el uso del método *sendEvent* proporcionado en el *Environment* así como el conjunto de métodos *update* implementados en las herramientas visuales se puede establecer la comunicación entre diferentes herramientas. El uso de estas interfaces favorecen la integración de herramientas colaborativas que permitan a los usuarios intercambiar razonamientos, hipótesis, y en general toda clase de comunicación. De esta forma esta plataforma ofrece soporte para la creación de ***Herramientas Colaborativas.***

Con el fin de homogeneizar la interfaz gráfica de cada herramienta así como de poder manejar el control de las acciones a realizar desde el entorno gráfico, en el proyecto Colab se requirió que todas las herramientas visuales de la aplicación extendieran de la clase *ColabWindow* la cual es mostrada en la figura 95. De esta manera atendiendo al tipo de *Floor control* que se requiera, se habilitará o deshabilitará ciertas opciones o el uso de ciertas herramientas.

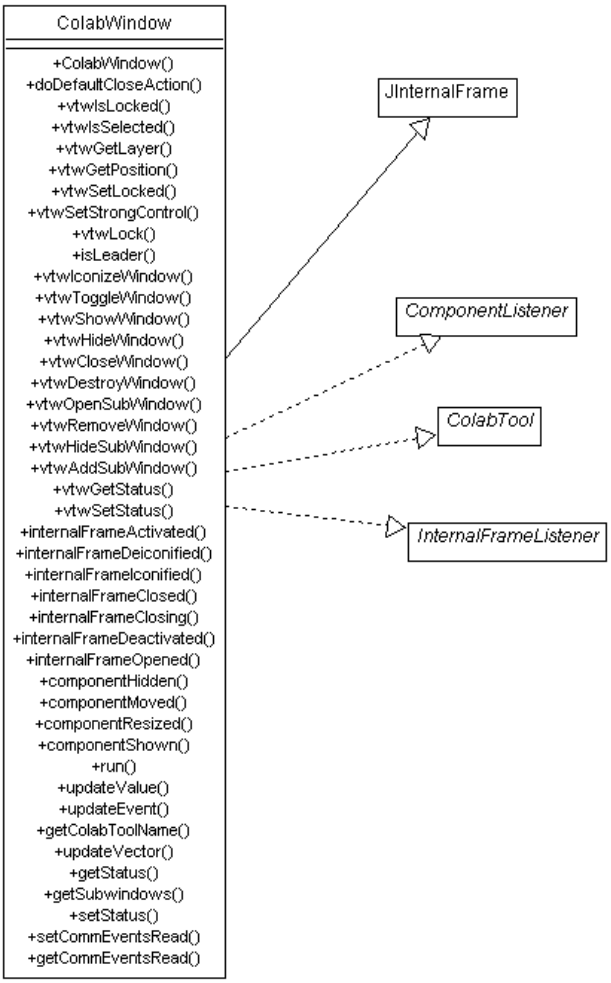

Figura 95. Descripción UML de la clase ColabWindow

En cuanto a los experimentos, bien sean simuladores bien sean interfaces para laboratorios remotos, implementan la interfaz *Phenomenon* cuyo diagrama UML puede ser aprecidado en la figura 96. De igual manera que ocurría en las herramientas visuales, se incorpora un método *run* a través del cual se pasa el *Environment* y un String con la configuración para inicializar los valores correspondientes de dicho *Phenomenon.*

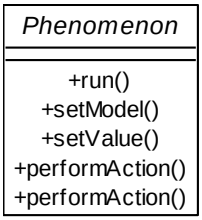

Figura 96. Descripción UML del Interfaz Phenomenon

En el desarrollo de nuestra plataforma hemos contemplado tanto simuladores como laboratorios remotos. En el caso del simulador, la comunicación se realiza sólo a través del BUS de eventos, de tal forma que el experimento se ejecuta siguiendo un programa java y se envía los eventos a través del *Environment* mediante el método *sendEvent*. Para la ejecución, como la parada de las simulaciones, las herramientas visuales comunican estas operaciones a través del envío de eventos los cuales son notificados al

experimento mediante el método *performAction*. Como hemos visto con anterioridad, en el caso del tratamiento de eventos de los *Phenomenon*, se trabajará con la clase *ServerEnvironment.*

En cuanto a los laboratorios remotos es necesaria la implementación de esta fachada Java que implemente la interfaz *Phenomenon,* de tal manera que la comunicación entre la fachada y las herramientas visuales siga el esquema anteriormente descrito.

Más detalladamente en la implementación de dicha fachada se invoca ciertos métodos del experimento remoto mediante el modelo de comunicación síncrono. Las respuestas de estas invocaciones se propagan a través del Bus de comunicación, pudiendo ser por tanto escuchadas por cualquier elemento que haya suscrito su interés a dichos eventos. De esta manera se evita el problema de la de-sincronización que habíamos analizado en los esquemas anteriores, puesto que todos los clientes reciben el mismo valor en el evento. Además el manejo de estos laboratorios no depende de una aplicación cliente, sino que ante el colaborador dicho experimento es tratado como cualquier simulador más.

Esta comunicación entre herramientas clientes y experimentos establece el ***Espacio de Experimentación*** a través del cual se puede visulizar los resultados de un experimento en las herramientas de los colaboradores. Como hemos visto anteriormente, este espacio constituye la parte central sobre la cual se basa el aprendizaje por descubrimiento.

Con el fin de gestionar el arranque dinámico de los experimentos se ha creado un gestor de experimentos, tal que cuando se detecte que un grupo entra a un *Floor* en el que hay configurados una serie de experimentos, se arranca los servidores de la simulación para esa sesión, bien sean los propios simuladores o los servicios fachada para conectarse con los laboratorios remotos. Igualmente cuando se detecta que una sesión no está activa, el propio gestor será el encargado de parar los servidores asociados a dicha sesión.

En la Figura 97 se muestra este gestor dinámico, el cual lanza su ejecución cuando se arranca todo el sistema y permanece a la escucha de los eventos de entrada de cualquier grupo en *Floor*. Así pues cuando llega el evento de entrada de un nuevo grupo se comprueba si para dicho grupo y para dicho *Floor* se han arrancado los *Phenomenon* que han sido configurados. En caso afirmativo no se realiza nada. Si dicho grupo en dicha planta no tiene inicializado ningún *Phenomenon* se instancia todos los *Phenomenon* que hay configurados para dicho *Floor*, los cuales se obtienen mediante consultas al *Broker*.

Con el fin de liberar los recursos de los *Phenomenon* cuando dejen de ser utilizados se gestiona una lista por cada grupo y *Floor* en el cual se indican el número de usuarios que ha entrado a dicha sesión, contando las entradas a *Floor* de los usuarios. Dicho número se va decrementando conforme a las salidas de los usuarios en dicha sesión, y una vez que este número llega a cero se para la ejecución del servidor de *Phenomenon* (o experimento) y se libera el espacio en memoria.

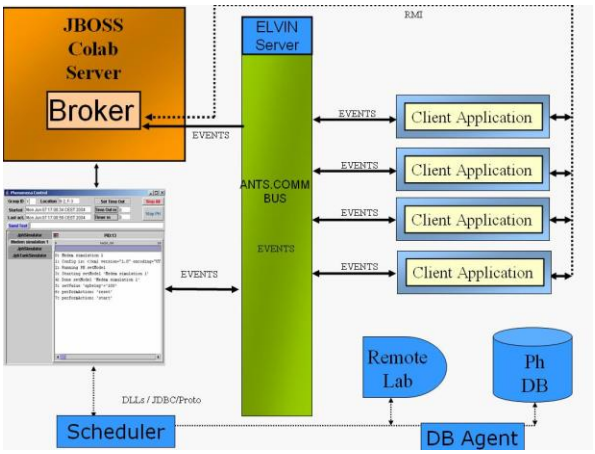

Figura 97. Esquema de conexión del gestor de experimentos dinámico.

Como ya hemos comentado anteriormente los simuladores representan programas que se ejecutan en la máquina servidora y en cuanto a los laboratorios remotos éstos siguen el modelo vista control, en los cuales las ejecuciones de las acciones se llevan a cabo en otras localizaciones. De esta manera, el laboratorio remoto proporciona una interfaz de comunicación que permita a la interfaz Java invocar ciertos métodos y obtener una respuesta. Más concretamente para realizar dicha comunicación dentro de la aplicación COLAB, se ha usado WinComm, ODBC, aunque podría utilizarse cualquier tecnología de comunicación síncrona de las que hemos visto anteriormente como puede ser XML-RPC o SOAP.

En cuanto al modelo de programación en el desarrollo de las aplicaciones, se sigue un modelo replicado, de tal forma que las herramientas colaborativas mantienen una copia local de su estado actual y los nuevos cambios que surjan se propagan a través del bus de eventos modificando la copia local acorde a estos eventos.

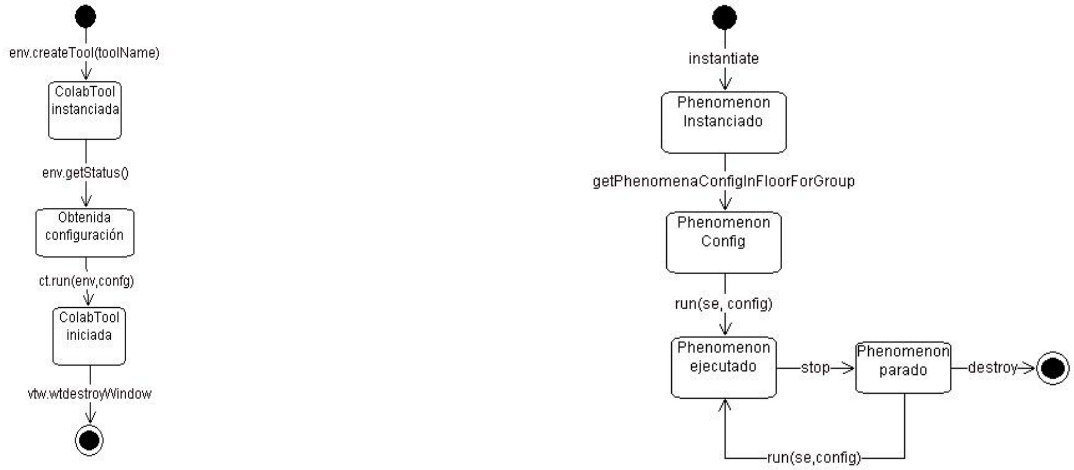

Figura 98. Creación de los componentes ColabTool y Phenomenon.

En la figura 98 se muestran los pasos requeridos para la inicialización y ejecución tanto de las *VisualTools* como de los *Phenomenon*. Como se puede observar, en el método run se le pasa un fichero de configuración a las *VisualTools* y a los *Phenomenon*. Dicho fichero es una descripción XML de las variables de inicialización junto con los valores a visualizar o a tener en cuenta en la ejecución del experimento.

Toda la información que fluye en cuanto a la configuración, estado y guardado de los recursos, *VisualTools* y *Phenomenon*, fluye bajo el estándar XML.

En el desarrollo de la aplicación COLAB, se definió un modelo datos de tal forma que se pueda interpretar en cualquier recurso del sistema mediante el uso del parser creado para este modelo. Un ejemplo de configuración siguiendo este modelo es mostrado en la figura 99.

```
<?xml version="1.0" encoding="UTF-8"?>
<phenomenonContext                    xmlns="http://www.co-lab.nl/xml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.co-lab.nl/xml PhenomenonContext.xsd">
      <descriptor>
            <symbol>RCLab</symbol>
      </descriptor>
      <variables>
            <variable>
                  <descriptor>
                        <symbol>voltage</symbol>
                  </descriptor>
                  <type>double</type>
            </variable>
            <variable>
                  <descriptor>
                        <symbol>voltageC</symbol>
                  </descriptor>
                  <type>double</type>
            </variable>
            <variable>
                  <descriptor>
                        <symbol>time</symbol>
                  </descriptor>
                  <type>double</type>
            </variable>
      </variables>
      <actions>
            <action>
                  <descriptor>
                        <symbol>run</symbol>
                  </descriptor>
                  <parameters>
                        <parameter>
                              <descriptor>
                                    <symbol>rc</symbol>
                              </descriptor>
                              <type>string</type>
                        </parameter>
                        <parameter>
                              <descriptor>
                                    <symbol>cmd</symbol>
                              </descriptor>
                              <type>string</type>
                        </parameter>
                  </parameters>
            </action>
      </actions>
</phenomenonContext>
```

Figura 99. Fichero de configuración del experimento RCLab.

Diseño de la interacción de las herramienta visuales y los experimentos.

La notificación de los nuevos cambios dentro de las herramientas visuales se realiza mediante los métodos *sendEvent*. La información recibida es comunicada a través de los métodos *updateEvent* a través de los cuales las herramientas realizan los pasos necesarios para actualizar sus vistas. Véase la figura 100 para ver el diagrama de secuencia de dicho esquema de comunicación.

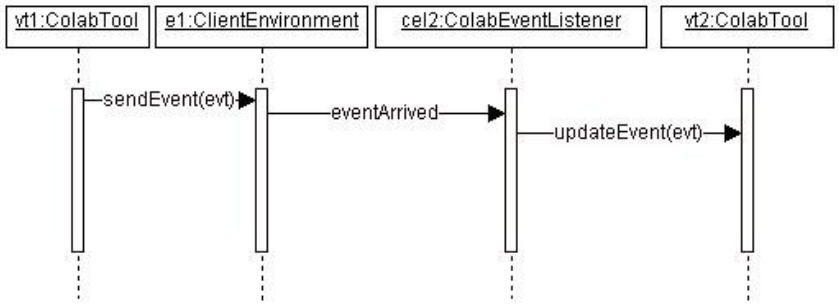

Figura 100. Diagrama de secuencia de la comunicación de un evento generado desde la VisualTool vt1 el cual es recibido por la VisualTool vt2.

En la figura 101 se muestra el esquema de comunicación entre una *VisualTool* y un experimento, mostrando la secuencia de invocaciones de métodos en cada objeto cuando se solicita la ejecución de una acción al experimento en cuestión.

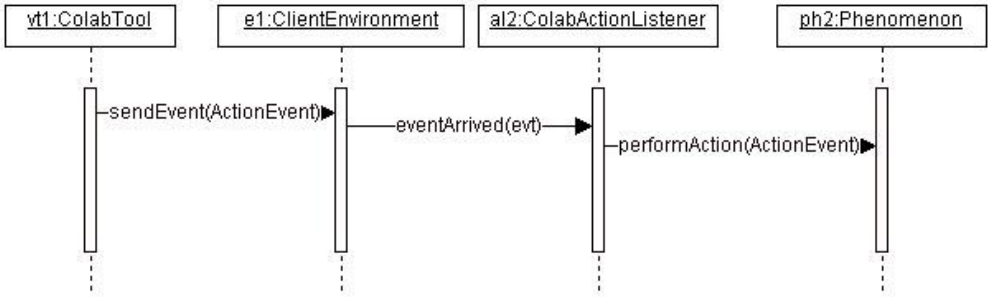

Figura 101. Diagrama de secuencia de la comunicación de un evento generado desde la VisualTool el cual es recibido por el phenomenon ph2.

En la figura 102 se muestra la secuencia de invocación de métodos entre un *Phenomenon* y una *VisualTool* en el caso de comunicación de resultados del experimento. Como podemos comprobar el *environment* del cliente mantiene una copia del experimento o *dataset* empleados, de tal manera que cuando una herramienta visual borre los datos pueda posteriormente volverlos a visualizar obteniendo estos valores del *Environment.*

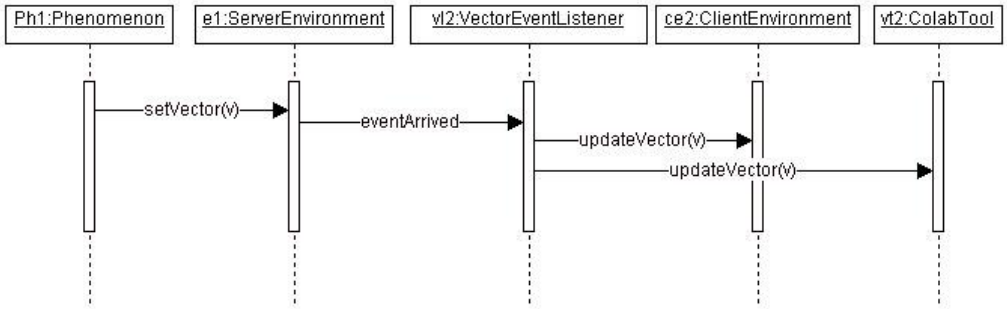

Figura 102. Diagrama de secuencia de la comunicación de un evento generado desde el phenomenon ph1 el cual es recibido por la visual tool vt2.

Para llevar a cabo la monitorización de ciertos eventos, las herramientas harán uso del método *sendEventLog* pudiendo de esta forma el servicio *OnlineLogger* recoger estos eventos. Este intercambio de eventos queda reflejado en el diagrama de secuencia mostrado en la figura 103.

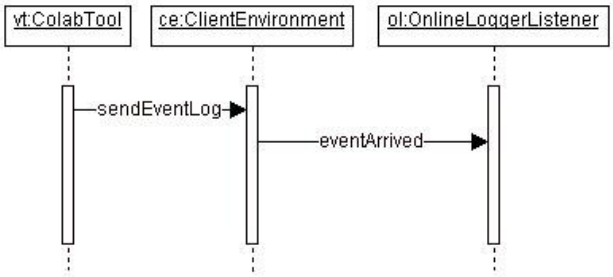

Figura 103. Diagrama de secuencia de la comunicación de un evento de log generado desde la VisualTool vt el cual es recibido por listener OnlineLoggerChecker

Otra de las operaciones relevantes a realizar por las herramientas colaborativas es el guardado de los valores del resultado o resultados de los experimentos que estan visualizando en un determinado momento mediante el uso del método *saveColabObject*. Posteriormente dichos valores pueden ser recuperados por las herramientas mediante el método *loadColabObject.* En la figura 104 se muestra la secuencia de métodos tanto en el caso de los colaboradores que escuchan este nuevo evento así como del *Phenomenon* que lo reciba.

Un proceso de *loadColabObject* implica cargar un modelo en el *Phenomenon*. En la siguiente figura se muestran los pasos llevados a cabo ante un evento de este tipo.

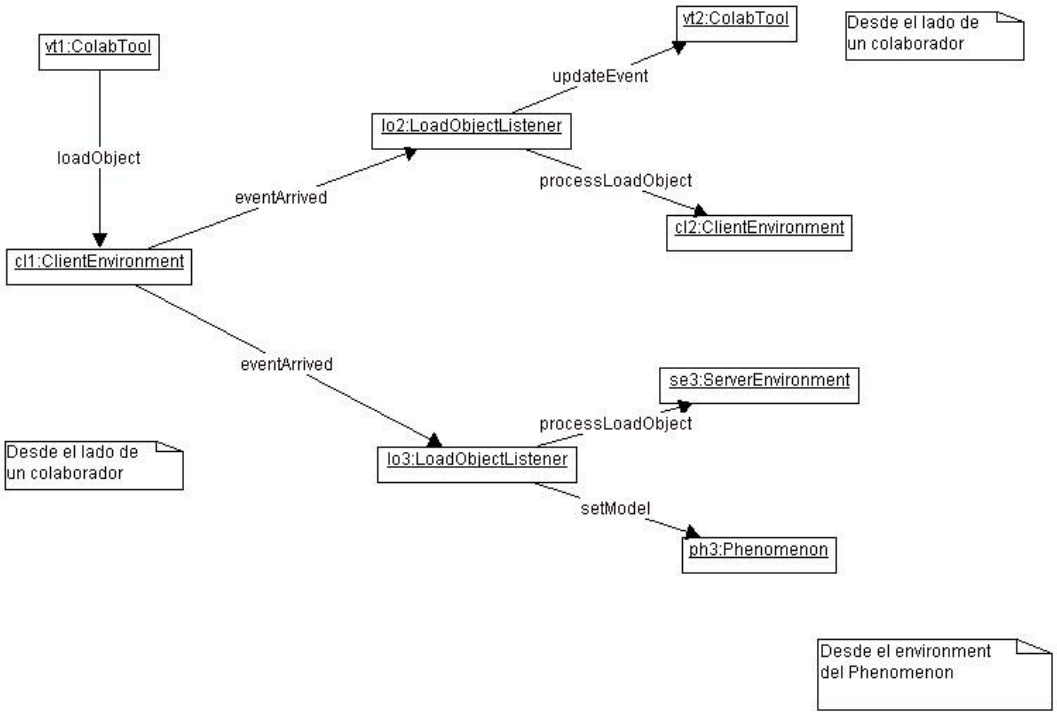

Figura 104. Diagrama de secuencia ante un evento de loadObject.

Instalación en la plataforma

Debido al uso de las especificaciones J2EE se favorece el uso del empaquetado de las aplicaciones mediante el uso de descriptores XML y jars. Para la distribución de todos los recursos de la plataforma a través de la web hemos hecho uso de la tecnología JavaWebStart. Más concretamente mediante el uso del fichero jnlp se le indican los componentes necesarios para descargar, con el fin de ejecutar la aplicación. Este fichero debe ser integrado también junto con el paquete .ear generado de la aplicación, el cual se desplegará en el servidor J2EE, en nuestro caso JBOSS.

De tal forma los usuarios para conectarse harán uso del navegador, se conectarán a través de una página Web y a través de un enlace activarán el uso de JavaWebStart el cual comprueba si el usuario posee los jars de la aplicación en cache. Si es así y los ficheros del servidor no han sido modificados ejecuta la aplicación. Si los jars de la aplicación son más recientes que los que dispone el usuario en su copia local, se procederá a la descarga de los nuevos ficheros y posteriormente se ejecutará la aplicación. Dicho esquema es mostrado en la figura 105.

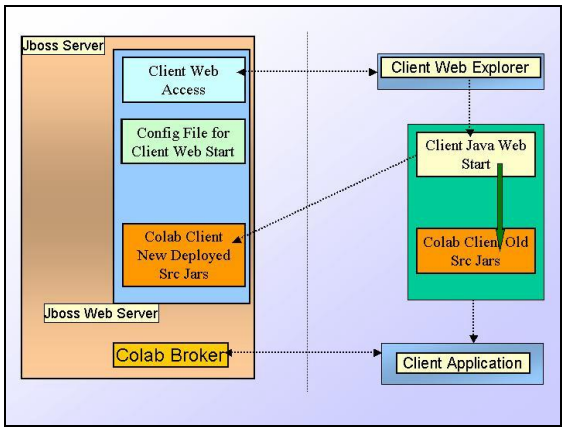

Figura 105. Esquema de funcionamiento de la tecnología JavaWebStart previo a la ejecución de la aplicación COLAB.

La forma de integrar una nueva herramienta se realizará mediante el empaquetamiento de esta herramienta a través del empaquetado *jar*. Con el fin de comunicar al *JavaWebServer* los ficheros a descargarse se hace uso del fichero *ColabClient.jnlp*

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="$$codebase">
      <information>
            <title>Colab Client</title>
            <vendor>Colab Consortium</vendor>
            <homepage href="http://colab.edte.utwente.nl"/>
            <description>Colab Client Java Application</description>
            <icon href="colab.gif" />
      </information>
      <resources>
      <!-- Warning: changes in heap params will affect Tomcat http
configuration -->
      <j2se version="1.4+"
href="http://java.sun.com/products/autodl/j2se" Xmx="128" initial-
heap-size="64m"/>
            <!-- <jar href="jbossall-client.jar" /> -->
            <jar href="jboss-client.jar" />
            <jar href="jboss-j2ee.jar"/>
            <jar href="jaas.jar" />
            <jar href="jbosssx-client.jar" />
            <jar href="jnp-client.jar" />

            <jar href="jndi.jar" />
            <jar href="jboss-common.jar" />
            <jar href="log4j.jar" />

            <jar href="jbcl.jar" />
            <jar href="utwente.jar"/>
            <jar href="<your_package>.jar"/>
            <jar href="je-4.0.0.jar" />

            <jar href="Jama-1.0.1.jar" />

            <jar href="jcommon-0.9.1.jar" />
            <jar href="jfreechart-0.9.16.jar" />
            <jar href="gnujaxp.jar" />
            <!-- <jar href="servlet.jar" /> -->
```

```
            <jar href="ColabDevelopment.jar" />
            <jar href="ph.jar" />
            <jar href="tools.jar" />
            <jar href="parser.jar" />
            <jar href="xml.jar" />
            <jar href="amstel.jar" />
            <jar href="teos.jar" main="true" download="eager"/>
            <jar href="resources.jar" />
            <jar href="draw.jar" />
            <jar href="vt.jar" />
            <jar href="all.jar" />

      </resources>
      <application-desc main-
class="colab.teos.CoLabApplet.CoLabApplet">
            <argument>$$context</argument>
      </application-desc>
</jnlp>
```

De esta manera se hacen accesibles las nuevas herramientas y/o experimentos en el sistema. Con el objetivo de crearlos como herramientas o como experimentos se hará uso de la herramienta de Gestión de Recursos (véase figura 106), indicando campos como el nombre de la herramienta (*VisualTool*) o del experimento (*Phenomenon*), una descripción y la clase java en la que se define la herramienta. En esta creación se indica al sistema si la *VisualTool* a añadir iniciará su sesión a nivel de *Floor* (marcando casilla) o sólo será a nivel de *Room*. En cuanto a los *Phenomenon* se puede especificar ficheros los cuales indiquen los parámetros de inicialización de los valores del experimento.

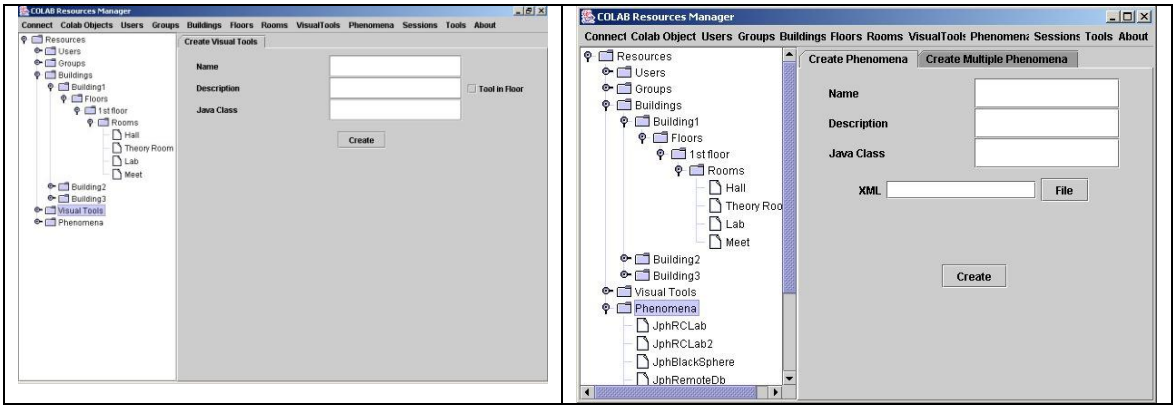

Figura 106. Aplicación ResourcesManager para la gestión de los recursos dentro de la plataforma colaborativa.

De esta forma se integran los nuevos componentes en la lista de componentes disponibles de la plataforma (lista de *VisualTool* y lista de *Phenomenon*). Como ya hemos comentado, las sesiones de una aplicación de aprendizaje por descubrimiento, se hará atendiendo a *Building, Floor, Room* para el caso de las herramientas y en *Building, Floor* en el caso de los experimentos y herramientas especiales. Esta información también se indica en la integración de las herramientas dentro de la plataforma.

Para la configuración de las herramientas dentro de una habitación se puede especificar el fichero de configuración de dicha herramienta, así como los valores iniciales en la de las variables.

Parsers y Modelos de datos.

Todos los recursos del sistema serán configurados mediante el uso del estándar XML, de tal forma que la inicialización de los valores y opciones de las herramientas y experimentos vendrá dado por ficheros de configuración basados en dicho estándar.

Esto confiere al sistema extensibilidad y adaptabilidad ya que las herramientas no responden a unos parámetros fijos, sino a los definidos por el fichero XML interpretando estos modelos. Como comentábamos anteriormente este hecho confiere un Marco de Referencia común, permitiendo así que varias herramientas puedan interpretar la misma información.

Con el fin de establecer un marco común en el diseño del formato de los ficheros descriptores, en el desarrollo de la aplicación COLAB la parte pedagógica instruyó en el seguimiento del modelo especificado en la figura 107.

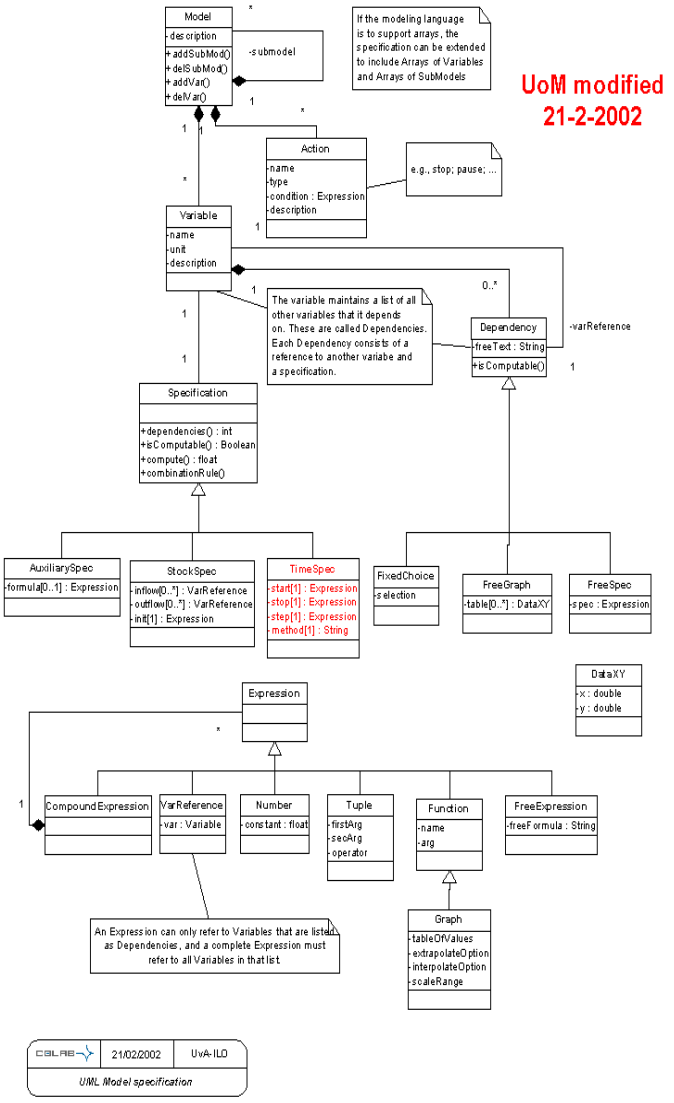

Figura 107. Especificación UML del modelo.

Con el fin de manejar ficheros XML de los modelos dentro de COLAB el grupo COLOS de la Universidad de Murcia se realizó el siguiente diagrama XSD, proveyendo una API para realizar el parser de dichos modelos que viene indicado en la figura 108.

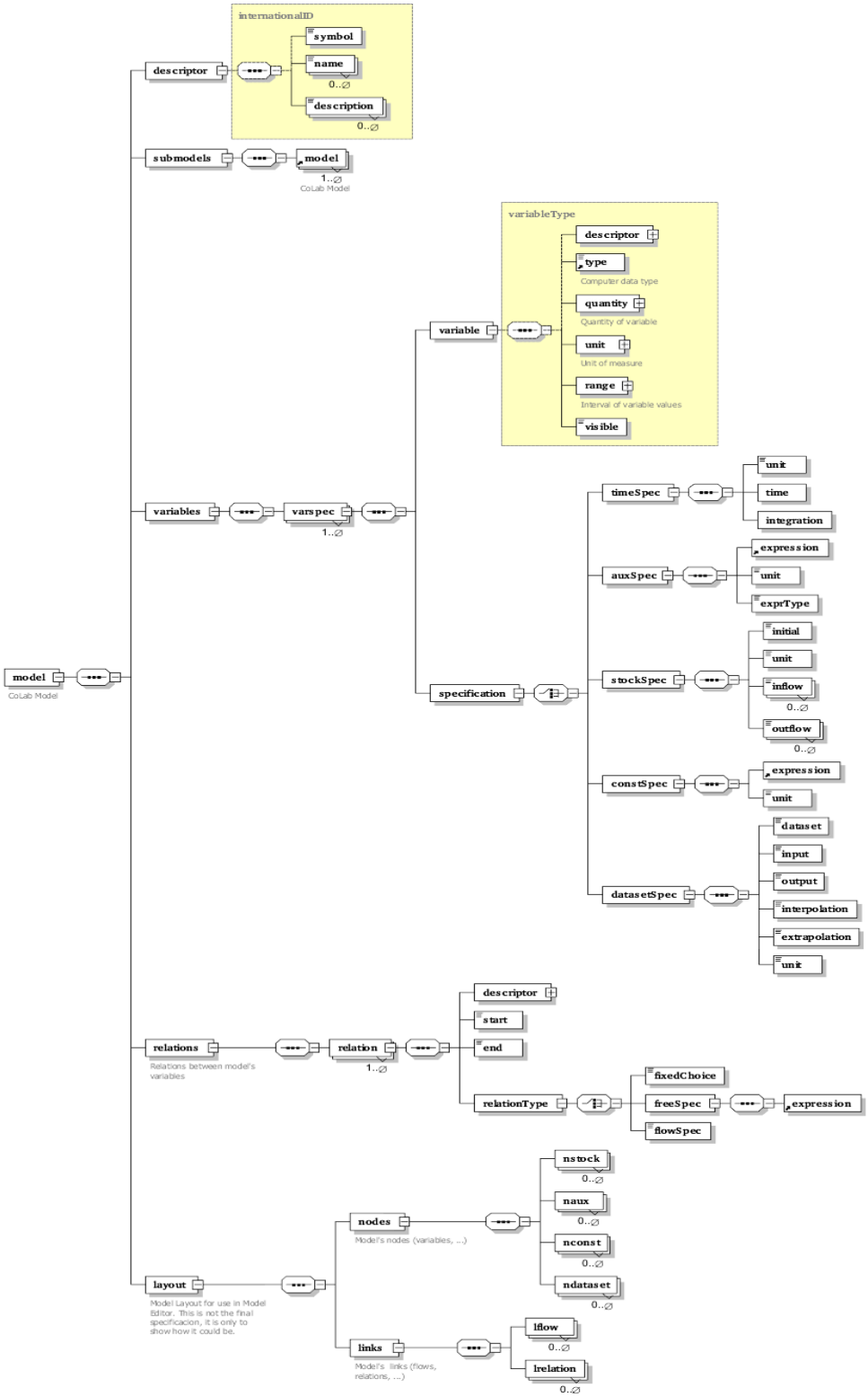

Figura 108. Diagrama XSD para los modelos de COLAB.

Mediante el uso del gestor de recursos que se proporciona dentro de nuestra plataforma colaborativo, se puede configurar el entorno asociando a cada elemento un fichero descriptor, pudiendo por tanto configurar herramientas con diferentes ficheros de configuración atendiendo a la sesión dónde se van a inicializar.

Como ya hemos visto anteriormente, una herramienta visual es instanciada por el *Environment* de la aplicación cliente y mediante el método *run* se transmite la información en XML con la que se ha configurado dicha herramienta o bien con la información guardada de una sesión previa. Mediante el uso del parser se obtendrán aquellos campos requeridos para la inicialización de dicha herramienta.

De igual manera, un *Phenomenon* o experimento recibe en su método *run* el fichero XML para recoger la información con la que ha sido configurado inicialmente y mediante el uso del parser recoge los valores de las variables necesarias para la ejecución de dicho experimento. Estos valores pueden ser posteriormente cambiados por el uso de las herramientas visuales.

La definición de este modelo y la posibilidad de leer sus valores a través del parser, permite que los recursos de la plataforma puedan interpretar los mismos ficheros, dando lugar a un mismo marco de referencia.

2.9. Aplicación Colab.

A continuación expondremos los aspectos más generales de la aplicación práctica llevada a cabo mediante el uso de nuestra plataforma colaborativa que hemos expuesto a lo largo de este capítulo. Dicha aplicación constituye el resultado del proyecto COLAB [Mart04, Mart05]. Algunos de los aspectos más específicos en la construcción de esta aplicación ya han sido abordados conforme hemos ido tratando cada uno de los aspectos de nuestra plataforma.

En el desarrollo de la aplicación hemos proporcionado una serie de ventanas a través de las cuales el usuario elige el área dónde quiere colaborar. Básicamente el desarrollo de la aplicación sigue el esquema mostrado en la Figura 109. Con el objetivo de realizar de una manera más rápida y eficaz la gestión gráfica entre el cambio de habitaciones y la salida y entrada de la plantas se han incorporado dos hilos que son *EnterFloorThread* y *EnterRoomThread.*

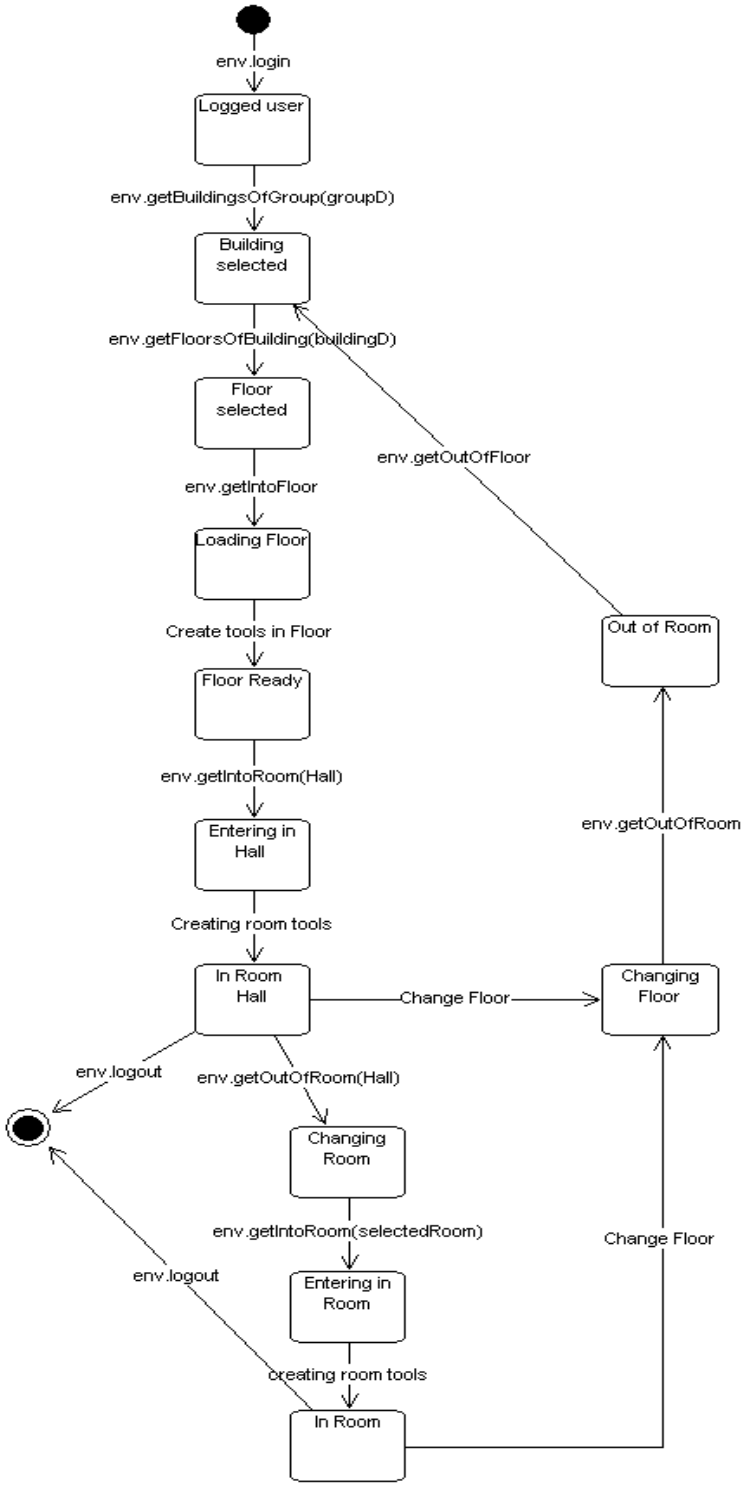

Figura 109. Diagrama de secuencia del manejo de la Aplicación Colab.

Del conjunto de herramientas disponibles, existen algunas de ellas como es el Proccess Coordinator que inicializan sus instancias cuando se entra en el *Floor*, puesto que suscriben la recepción de eventos a nivel de *Floor*. De tal forma que independientemente del *Room* dónde se encuentren mostraran el mismo estado. Estas instancias se destruyen cuando el usuario salga o cambie de *Floor*, realizando previamente el guardado del estado de dichas herramientas. Esta gestión de las herramientas se realiza desde el método *getIntoFloor* proporcionado desde el

Environment, que a su vez invoca al del *Broker* para la creación de las estructuras correspondientes para almacenar la información relevante a nivel del *Floor*. La salida del *Floor* se realiza mediante el método *getOutFloor* proporcionado desde el *Environment*, el cual a su vez invoca su correspondiente método en el *Broker*. Esta secuencia de entrada al *Floor* queda representada en la figura 110.

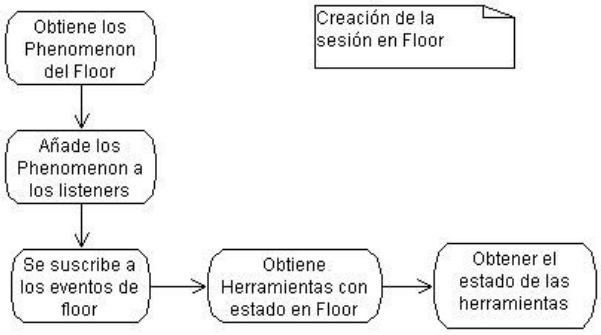

Figura 110. Diagrama de secuencia en la entrada a un Floor en COLAB.

Cada cambio de habitación o de *Room* supone la terminación de los threads de las herramientas que se han activado para dicha sesión y por tanto el guardado de la información de éstas. Al iniciar una nueva sesión en la nueva *Room* se realiza la instanciación de aquellas herramientas configuradas en dicho espacio. Si bien todas las herramientas no aparecen con estado visible, todas ellas están inicializadas, y una vez que se seleccionan se cambia su estado de visualización.

Como se puede apreciar de la figura, la gestión de *Floor* se lleva a cabo en cada cambio de *Room*, indicándose diferentes gestores de *Floor Control* dependiendo de las configuraciones de sesiones de nivel *Room*. La secuencia de entrada en el *Room* viene representada por el diagrama de la figura 111.

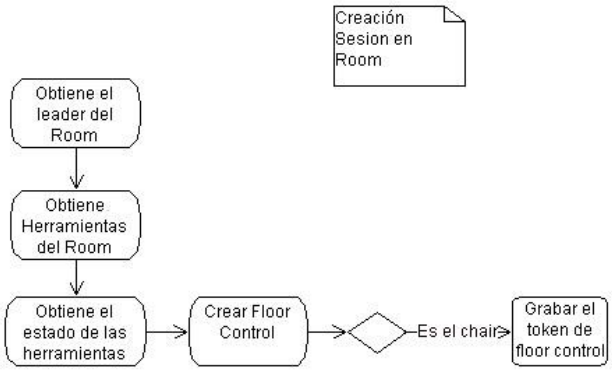

Figura 111. Diagrama de secuencia en la entrada a una Room en COLAB.

Siguiendo todos los pasos que se han mostrado anteriormente para la incorporación de herramientas dentro del marco, los diferentes colaboradores han desarrollado un conjunto de recursos tanto de *VisualTools* como de *Phenomenon*.

Más concretamente el conjunto de herramientas colaborativas es mostrado a continuación:

- **Chat.** Permite la comunicación textual entre los diferentes colaboradores.
- **Pizarra.** Permite la comunicación gráfica entres los diferentes colaboradores.
- **Process Coordinador.** Mediante esta herramienta el profesor puede establecer una serie de objetivos o de pasos a realizar por los aprendices. Así tras la experimentación pueden resolver ciertas preguntas o tareas.
- **Editor de Modelos.** Esta herramienta permite modelar simulaciones de igual manera que realizar la simulación con ellas. Para ello se requiere el uso de un simulador especial, el cual debe ser configurado en el *Floor* para que pueda ser cargado automáticamente por el gestor de sesiones.
- **Navegador Colaborativo.** Dicho navegador recoge la página html que están configuradas para la sesión en la que se encuentra *Building/floor/Room* dónde es arrancada dicha herramienta. Con el objetivo de favorecer la importación de dichas páginas en el servidor, hemos tenido que proveer desde el *Broker* con un método *deployHtml* que sea el encargado de desplegar la información html en el servidor.

Se desarrollaron un amplio conjunto de simuladores los cuales requieren la implementación de una herramienta visual de tal forma que los usuarios colaboradores pudieron visualizar el estado de la simulación de una manera gráfica.

En cuanto a los experimentos remotos se trabajó con los siguientes:

- Capacitor lab
- Mechanics lab
- Tank lab
- Desktop Greenhouse

Por cada uno de estos experimentos se desarrollaron herramientas visuales, la cuales permiten la comunicación entre la aplicación cliente, y la interfaz Java del *Phenomenon* que se está ejecutando en el servidor. Dichas herramientas hacen uso de una herramienta *Scheduler* la cual lleva el control de reservas del laboratorio remoto en cuestión. De esta forma no se permite la ejecución de dicho laboratorio a aquellos usuarios que no hayan hecho una reserva.

Más concretamente para el desarrollo de las herramientas visuales se hace uso de cámaras de video mediante las cuales se puede visualizar el objeto en cuestión sobre el que se está experimentando. Para ello se instala en el servidor de la aplicación COLAB el programa camproxy que permite al cliente visualizar las imágenes obtenidas.

Otra de las características importantes en esta aplicación es el uso de *Internacionalización,* de tal forma que los usuarios de los diferentes países dónde se realizaron las pruebas pudieron tener personalizado los mensajes de la aplicación al idioma que se elige en la entrada.

Una de las partes fundamentales en todo entorno es el manejo de interfaces (HCI), puesto que mediante ellas el usuario recibe la información adecuada del sistema. Para la visualización gráfica tanto de iconos como del manejo de colores se ha hecho uso del paquete *utwente.gui*, desarrollado por la Universidad de Twente, bajo la cual se implementa un formato común en cuanto a colores, iconos y fondos. De esta manera, un

cambio en el color de la plataforma se realiza sólo mediante la modificación de las clases pertinentes en dicho paquete.

Mediante el uso de la herramienta de gestión de recursos que posee la plataforma se planificó cuatro *Buildings*: Introducción al sistema, Electricity, Water Management y Mecanics, cada uno de ellos destinado a la experimentación en un campo concreto. El conjunto de experimentos accesibles en cada *Floor* así como el conjunto de herramientas disponibles en cada *Room* se configuró también mediante la herramienta de gestión. De esta forma también se configuró el estado inicial de cada una de las herramientas dentro de cada una de las habitaciones.

En la figura 112 se muestra el aspecto visual de la aplicación Colab. En esta imagen se muestra una experimentación llevada a cabo dentro de la habitación *Laboratory (Lab)* en la cual se muestran los resultados de la simulación de Medem tanto en tablas como en graficas. En el lado izquierdo de dicha imagen se proporcionan las herramientas de las que disponen los colaboradores, los controles para la navegación entre diferentes habitaciones, la lista de usuarios conectados así como su localización representada mediante triángulos de colores. En la parte inferior de la aplicación se muestra lo que son las herramientas de repositorio y la del chat.

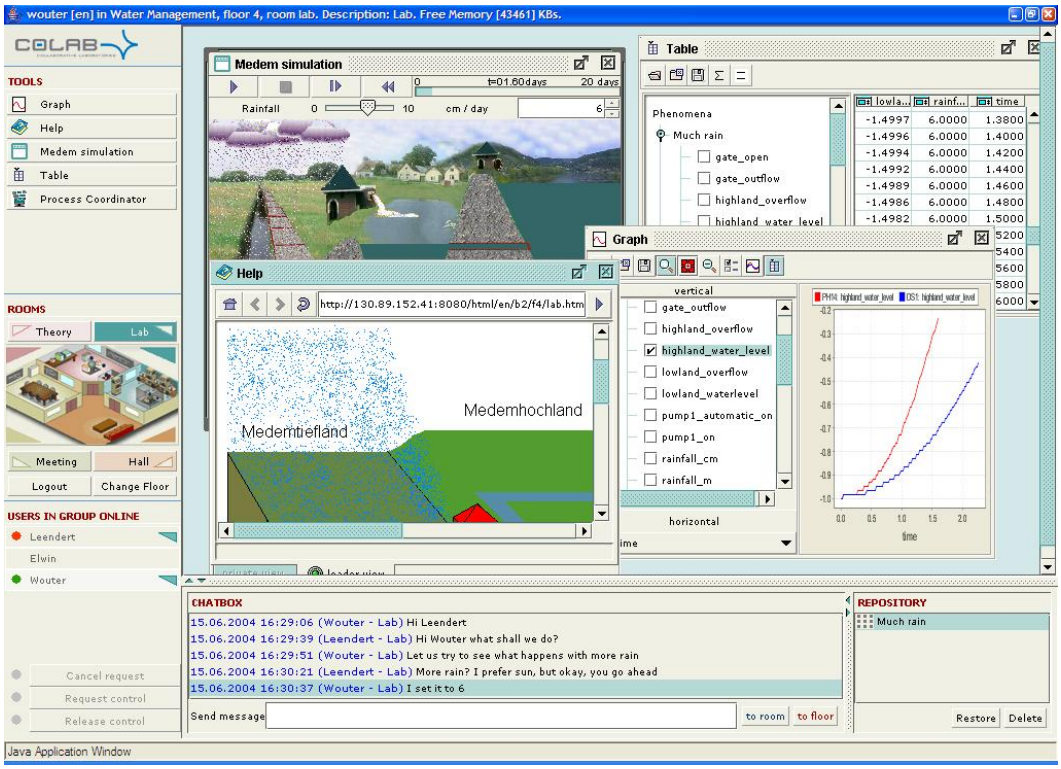

Figura 112. Interfaz dentro de la habitación Laboratory de la aplicación COLAB.

2.10. Evaluación Pedagógica de Colab

Una de las partes interesantes en toda arquitectura propuesta para la colaboración entre los usuarios es la evaluación de dicha plataforma, tanto en las herramientas que ofrece así como en el diseño. A través de su uso se obtiene una realimentación indicando ventajas y desventajas de la plataforma sobre la que se desarrolla la colaboración.

Uno de los puntos importantes en el desarrollo de la aplicación fueron las evaluaciones que se desarrollaron por parte de los estudiantes tanto de colegios como de universidades.

En la tabla 9 se muestra parte de la evaluación pedagógica llevada a cabo a través de una serie de pruebas con el entorno Colab [Laz04]. En dicha tabla se muestra sobre todo la evaluación del estudiante con respectos a los temas generales del manejo de Colab así como del uso de la colaboración. De la información que se muestra en la tabla se deduce que el estudiante esta bastante satisfecho del uso de Colab, tanto de la facilidad de uso así como del conjunto de herramientas que ofrece. De esta forma se obtiene uno de los objetivos que se requeren en el proyecto, motivar el uso de estas herramientas mostrando una interfaz y manejo fácil para el usuario final.

Esta evaluación también mostró que en general los estudiantes encuentran gratificante la colaboración en el grupo, indicando positivamente la colaboración y el trabajo realizado por el grupo.

Co-Lab in general	Mean (SD)
1. Overall, I am satisfied with how easy it is to use Co-Lab	3.00 (.93)
2. It was simple to use Co-Lab	3.16 (1.00)
3. Is was difficult to make the assignment in Co-Lab	2.60 (.90)
4. It was easy to learn use Co-Lab	4.05 (.72)
5. I believe I became productive quickly using Co-Lab	3.74 (.98)
6. Whenever I made a mistake using Co-Lab, I recovered easily and quickly	3.40 (.98)
7. The information provided with Co-Lab is clear	3.00 (.98)
8. It was easy to find the information I needed	3.07 (.94)
9. The information was effective in helping me complete the assignment	3.16 (.81)
10. The organization of information (i.e., the interface) on the Co-Lab screen was clear	3.42 (.98)
11. The interface of Co-Lab was pleasant	3.19 (.91)
12. I liked working with Co-Lab	3.53 (.96)
13. Co-Lab has all the functions and capabilities I expect it to have	3.49 (1.14)
14. I understood the error messages Co-Lab generates	3.18 (1.06)
Co-Lab navigation	
15. I liked the floor metaphor in Co-Lab	3.33 (1.06)
16. The floor metaphor helped me in navigating between rooms	3.51 (1.01)
17. I understood the separation of rooms in Co-Lab	4.05 (.87)
18. It was easy to navigate between room	3.77 (1.27)
19. I understood the function of the different rooms in Co-Lab	4.09 (1.04)
Collaboration in Co-Lab	
20. I was able to efficiently collaborate during the assignment	2.93 (1.22)
21. I liked the collaboration in our group	3.88 (1.03)
22. Our group was able to effectively work together during the assignment	3.30 (1.21)
23. The chat was a good way to collaborate with the students in my group	3.14 (1.34)
24. The sentence openers were useful during collaboration	2.08 (1.04)

Note. minimum score = 1; maximum score = 5)

Tabla 9. Tabla de la evaluación pedagógica de la aplicación COLAB.

2.11. Conclusiones.

En este apartado hemos expuesto tanto la creación de una plataforma colaborativa así como la creación de la aplicación COLAB sobre ella, la cual constituye una plataforma

para el aprendizaje por descubrimiento y es el resultado del proyecto europeo COLAB *"Collaborative Laboratories for Europe"*.

Con el objetivo de obtener una plataforma colaborativa aplicable a muchos ámbitos hemos tenido que abordar los elementos necesarios en un entorno colaborativo así como los requerimientos para que dicha plataforma también pueda constituir la base para la construcción de plataformas orientadas al aprendizaje por descubrimiento. Una de las particularidades de nuestra plataforma es que ofrece tanto la posibilidad de colaboración síncrona como asíncrona. Por lo tanto ha sido necesario estudiar ciertos algoritmos y características de ambos tipos de colaboración.

El soporte para la colaboración asíncrona así como la definición de toda la comunicación síncrona de nuestra plataforma la hemos desarrollado basándonos en las especificaciones J2EE mediante el uso de *entity beans* y *session beans*. El uso de estas especificaciones confiere a nuestro sistema las características de portabilidad, extensibilidad y reusabilidad.

Como ya hemos mostrado a lo largo de este trabajo de investigación, el bus de eventos es uno de los principales puntos para la colaboración síncrona del sistema, por lo tanto debe ser robusto y dar una alta fiabilidad en el reparto de los mensajes. Con el fin de que nuestra plataforma adquiera la característica de independencia de la tecnologías subyacentes, se ha mostrado como se ha construido dicho Bus basándonos en el módulo ANTS.COMM que se ofrece en la arquitectura colaborativa ANTS expuesta en la tesis del doctor Pedro García.

En la creación de nuestra plataforma colaborativa se han definido todos los elementos necesarios en un entorno colaborativo, como son los usuarios (U) que se agrupan en grupos accediendo a las áreas (*Building, Floor, Room)* constituyendo diferentes sesiones en estas áreas (S) a las que se les ha dado acceso. Dentro de estas áreas o sesiones los usuarios disponen de una serie de herramientas o recursos (R) a los que tienen acceso mediante el uso de gestores de *Floor Control* (F).

En cuanto a la gestión de sesiones hemos provisto dos tipos de sesiones, una a nivel de *Floor* dónde la información será compartida por todos los colaboradores que se encuentren localizados en ese *Floor*, independientemente de la *Room* dónde estén llevando a cabo su colaboración. El otro nivel de sesión se establece a nivel *Room* en la cual los eventos a través de esta sesión sólo serán recibidos por aquellos colaboradores que se encuentren en la misma.

Hemos identificado los recursos del sistema, que en general será el acceso a las herramientas visuales que mostrará la aplicación cuando los colaboradores se unan a una sesión.

Con el fin de definir una serie de políticas de acceso de los usuarios a los diferentes recursos hemos establecido un conjunto de políticas (F). Debido a que estas colaboraciones pretenden mantener grandes semejanzas a las colaboraciones "face to face" sólo se han incluido a gestores los cuales están basados en la figura de un coordinador que será un usuario del grupo.

Otra tarea importante a abordar en el desarrollo del entorno colaborativo son los algoritmos de presencia, para asegurar que la lista de usuarios online refleja la lista real de usuarios dentro del sistema. Por ellos hemos propuesto un algoritmo que trate de averiguar con la mayor rapidez que usuarios están conectados en un determinado punto en el tiempo y que además sea capaz de detectar que usuarios han quedado en un estado inconsistente. Si el usuario esta en un estado inconsistente dicho algoritmo provocará la liberación de los recursos que tenía asignado dicho usuario y la salida de dicho usuario del sistema.

Con el objetivo de abordar el problema de la llegada tarde de los usuarios, se han estudiado una serie de algoritmos adaptando estos a la arquitectura general de nuestra plataforma. Más concretamente hemos adaptado el algoritmos de Jürgen Vogel, Martin Mauve eliminando el problema de la replicación de estado basándonos en el uso de mecanismos centralizados para la coordinación entre los diferentes servidores del estado.

De esta forma en el entorno que hemos desarrollado queda definida la tupla expuesta por Dommel y García Aceves, en cuando a la definición de un sistema colaborativo.

Puesto que este sistema colaborativo pretende ofrecer también soporte al aprendizaje por descubrimiento, hemos ofrecido las características que según Wouter, se requieren para dichos sistemas.

Nuestro sistema proporciona un ***Marco de Referencia*** en el cual diferentes herramientas pueden trabajar con los mismos datos a través del uso del estándar XML así como la definición de un modelo común para la configuración de *VisualTools* y *Phenomenon*. Por lo tanto todas las herramientas serán configuradas mediante ficheros que sigan el modelo, de tal forma que distintos recursos pueden interpretar la misma información. De igual manera, la suscripción de una serie de listeners permite a los distintos recursos recibir las mismas notificaciones, pudiendo mostrar de esta manera la misma información y trabajar sobre la misma base de conocimiento.

El uso del *Environment* así como de las interfaces *ColabTool* proporcionan un marco para la creación e inclusión de ***Herramientas Colaborativas*** a través de las cuales los colaboradores puedan razonar, establecer hipótesis y comunicarse en tiempo real.

Además mediante el uso del *Environment y Broker* se proporciona los dos tipos de ***Escenarios de Colaboración*** ofreciendo de una manera conjunta la colaboración síncrona como la asíncrona, enriqueciendo de esta manera la colaboración de los usuarios.

Principalmente mediante las interfaces *Environment* y *Phenomenon* se proporciona un marco para la comunicación ente experimentos y herramientas visuales de los colaboradores, favoreciendo de esta manera la creación de ***Espacios para la experimentación.*** Además este espacio constituye la base de la información a través de la cual los colaboradores argumentarán, establecerán nuevas hipótesis. En general a través de los resultados obtenidos y de esta colaboración obtendrán el conocimiento mediante la experiencia compartida de los colaboradores.

Con el objetivo de facilitar la integración de herramientas y experimentos sin necesidad del conocimiento acerca de las tecnologías empleadas, hemos ofrecido un conjunto de intefaces como son *ColabTool* y *Phenomenon* que mediante el uso del *Environment* se abstraen del uso de Listeners y del manejo del *Broker*. Más concretamente el *Broker* nos proporciona un modelo de comunicación síncrono para el guardado y obtención de datos y el *Environment* nos ofrece un modelo de comunicación asíncrono el cual gestiona la información a través de la sesión.

Hemos visto como integrar recursos dentro de la plataforma, mostrando el esquema de comunicación entre los diferentes elementos para la difusión de los eventos entre herramientas así como entre los experimentos y la aplicación cliente.

Con el objetivo de facilitar el manejo de experimentos así como la introducción de datos se han creado dos aplicaciones que son el gestor dinámico de experimentos y el gestor de los recursos del sistema. Ambos forman parte de nuestra plataforma y son independientes de los recursos a introducir.

Con todo lo anteriormente podemos concluir que hemos obtenido un sistema colaborativo para el aprendizaje por descubrimiento, favoreciendo de esta manera las teorías colaborativas del aprendizaje.

Hemos mostrado además el desarrollo de la aplicación COLAB sobre nuestra plataforma constituyendo de esta forma una aplicación para el aprendizaje por descubrimiento en el área de la Física.

Todos los recursos de COLAB están desarrollados bajo la implementación de las interfaces que constituyen parte del marco. De esta forma se ha creado un conjunto de recursos formando el conjunto de las herramientas colaborativas. Además, las herramientas de colaboración han sido desarrolladas basadas en esquemas de colaboración síncrona y asíncrona, de tal manera que los usuarios conectados a la misma sesión podrán comunicarse de manera online. Del mismo modo, mediante la opción asíncrona, se puede guardar la información actual de las herramientas y posteriormente se pueden recuperar.

Mediante el uso del servicio del presencia implementado en nuestro marco se favorece la muestra de dicha información al usuario desde la interfaz gráfica mostrando no sólo quien está conectado así como la localización dentro de la sesión.

El mecanismo de *Floor Control* constituye una de las bases fundamentales para la colaboración síncrona, estableciendo turnos entre los usuarios que colaboran sobre los mismos recursos. Por tanto en el desarrollo de Colab se han incorporado los dos gestores de *Floor Control* expuestos en la plataforma y se han incorporado una serie de elementos gráficos para que el usuario sea consciente del estado actual del floor.

La recogida de información del sistema de monitorización constituye un elemento clave para la evaluación desarrollada por la parte pedagógica pudiendo realizar un análisis cuantitativo de la actividad mediante el uso de Analog-Colab.

Mediante este entorno hemos mostrado la construcción de una aplicación para la experimentación desarrollada sobre nuestra plataforma colaborativa. Debido al carácter

genérico con el que se definen los recursos, tanto de herramientas visuales como experimentos, la arquitectura desarrollada podría ser aplicada a otros ámbitos de la colaboración.

Además, puesto que son las herramientas y el uso de los parsers los que le dan valores y la configuración del campo de trabajo, la inclusión de diferentes herramientas así como los diferentes valores en los modelos de datos darían lugar a otro ámbito de aplicación. Incluso se podría usar la misma aplicación cliente, puesto que la información es manejada atendiendo a una previa configuración del sistema.

El resultado de COLAB constituye uno de los sistemas base y original, el cual pretende ser usado en el trabajo en colegios y universidades de diferentes países, facilitando la experimentación y el compartimiento de recursos así como la motivación de los alumnos en el ámbito de la física, mediante el uso de las nuevas tecnologías.

De las evaluaciones pedagógicas se infiere una actitud positiva por parte del alumno en cuanto al uso de estas plataformas, valorando positivamente tanto las facilidades de uso, opciones y herramientas esperadas así como la posibilidad de colaborar con el resto del grupo. Por lo tanto, se cumple uno de los objetivos que se pretenden con estos entornos colaborativos, estimular el aprendizaje de las ciencias mediante el uso de las nuevas tecnologías.

Capítulo Quinto.

Conclusiones y Vías futuras.

1. Conclusiones.

El resultado de este trabajo de investigación es proporcionar el diseño de entornos colaborativos que puedan ser aplicables en el ámbito educacional. Uno de los principales objetivos dentro de nuestra tarea investigadora dentro de estos entornos es la de facilitar la colaboración síncrona entre un grupo de usuarios.

Más concretamente uno de los resultados es un sistema colaborativo síncrono que es fácilmente integrable en cualquier plataforma ya existente y el otro constituye una plataforma colaborativa que nos sirve como base para la creación de cualquier aplicación, ya sea simplemente mediante el uso de una colaboración entre herramientas, ya sea mediante el uso de experimentos y simulaciones.

A través de este trabajo hemos estudiado diferentes tecnologías existentes analizando las ventajas y desventajas de cada una de ellas y cual se adecuaba mejor a nuestra solución. Así pues hemos concluido que en la elaboración de aplicaciones muchos a muchos la mejores tecnologías de comunicación eran las asíncronas y para la comunicación y guardado de información dentro del sistema las tecnologías más adecuadas son la síncronas.

Hemos analizado los diferentes principios así como arquitecturas para la creación de aplicaciones distribuidas concluyendo que la más adecuada para la creación de lo que es nuestra plataforma distribuida para la colaboración eran aquellas basadas en las especificaciones J2EE, puesto que forman un conjunto de interfaces más maduras, ofrecen interoperabilidad con otras tecnologías como CORBA, sus especificaciones resultan más fácil de manejar y existen implementaciones no basadas en software propietario.

Además hemos estudiado los principios más importantes de los sistemas cooperativos analizando los conceptos CSCW, para tener en cuenta tanto en el diseño de las aplicaciones así como el uso de interfaces adecuadas que ofrezcan la información necesaria al usuario. De este estudio hemos concluido que el mejor modelo de distribución de datos para la colaboración síncrona es el modelo replicado, manteniendo así cada usuario una copia local de la información y propagando los cambios a través de un BUS de eventos.

Debido a que nuestro trabajo está enfocado en el ámbito de la colaboración síncrona y que el modelo de distribución de datos será un modelo replicado, en el estudio de los sistemas colaborativos hemos analizado problemas típicos de este tipo de colaboración como son el establecimiento de turnos en el manejo de recursos, *Floor Control*, y la llegada tarde de un usuario, *Late join*. En el primer caso hemos estudiado una serie de teorías y propuestas obtenidas por Dommel y García-Luna-Aceves. En el segundo caso hemos estudiado dos propuestas bastantes similares llevadas a cabo por Greyer, Vogel y Mauve.

Puesto que la aplicación de nuestro trabajo está centrada en entornos educativos, hemos presentado las dos grandes tendencias actuales en el aprendizaje. Hemos visto como dentro del área de la educación se diferencia el término cooperativo y colaborativo, cosa que no ocurre dentro de otras áreas. Hemos visto que el aprendizaje cooperativo es aquel que es guiado por el profesor y el colaborativo es aquel aprendizaje que se obtiene de la experiencia compartida de los estudiantes y profesor, siendo el profesor un colaborador más dentro del sistema. Con relación a este tipo de aprendizaje emerge el concepto de sistema CSCL, en el cual el estudio va enfocado en proveer un conjunto de herramientas de comunicación a los estudiantes tal que dichas sirvan de base en la construcción del conocimiento. Dentro del campo CSCL emerge el Aprendizaje por Descubrimiento, que es aquel basado en la experimentación compartida. En los entornos CSCL es fundamental tanto ofrecer la información de los usuarios conectados así como herramientas para el análisis de las interacciones del usuario con el sistema.

Para la creación de una plataforma colaborativa que pueda soportar el Aprendizaje por Descubrimiento es necesario establecer unos escenarios de colaboración, conjunto de herramientas colaborativas, un espacio para la experimentación así como un marco de referencia común de tal forma que usando diferentes herramientas se pueda visualizar la misma información. En el desarrollo de nuestra plataforma colaborativa mostraremos cómo hemos abarcado estos cuatro puntos.

Previo al desarrollo de nuestra solución se realizó una serie de investigaciones tanto en el entorno educacional JLE como en la arquitectura ANTS. En la integración de una pizarra colaborativa dentro del entorno educacional JLE se hizo uso de la API JSDT sobre la cual se desarrolló la pizarra colaborativa. A través de esta integración concluimos que el desarrollo de una aplicación en una determinada tecnología, implica la modificación de la aplicación en el caso de cambio de la tecnología subyacente. Las tecnologías cambian constantemente, ofreciendo nuevas posibilidades así como mejoras en el reparto de información de unas a otras. Por lo tanto se hace necesario la creación de una API genérica que permita el desarrollo de aplicaciones colaborativas independientemente de la tecnología subyacente para el reparto y suscripción de eventos.

A través del capítulo tercero hemos mostrado la arquitectura ANTS así como hemos mostrado la integración de una aplicación en dicha plataforma, intentando que dicha arquitectura fuera la base de la creación de una plataforma de Aprendizaje por Descubrimiento. Sin embargo ANTS impone un modelo de programación muy sofisticado mediante el uso de modelos inadecuados en una arquitectura basada en el envío de datos binarios, lo cual implica al programador analizar cada uno de los eventos obtenidos, conllevando además el conocimiento de la parte tecnológica del sistema. Esto conlleva la dificultad en la integración de nuevos componentes puesto que se requiere el conocimiento de gran parte tecnológica. Todos estos problemas junto con el requerimiento de la parte pedagógica de la creación de conjunto de APIS que facilite la creación e integración de nuevas herramientas, nos obligó al diseño de otra plataforma colaborativa.

Del estudio de ANTS hemos apreciado como su sistema modular ofrece un conjunto de APIs para la creación de herramientas colaborativas sin la necesidad de conocer las tecnologías subyacentes.

Más concretamente el uso de la API ANTS.CORE y lo que es más, el uso de ANTS.COMM permite el uso de diferentes tecnologías MOM sin que las aplicaciones se vean afectadas por esto. Así pues, estas APIs han constituido las bases para la creación de entornos colaborativos en nuestro trabajo de investigación.

De este capítulo hemos concluido que uno de los puntos centrales en cualquier sistema que requiera la colaboración síncrona es el BUS de eventos, constituyendo la base fundamental tanto para la colaboración entre los usuarios transfiriendo la información de las herramientas así como en la recogida de los eventos de logs para posteriormente analizar/monitorizar las actividades de los usuarios.

Basándonos en nuestra investigación previa hemos desarrollado los resultados de nuestro trabajo, el sistema colaborativo ANTS y una plataforma colaborativa.

En el desarrollo de nuestro sistema colaborativo ANTS hemos hecho uso tanto de la API ANTS.COMM para el bus de eventos así como nos hemos basado en la API ANTS.CORE para crear una nueva sin necesidad del uso de la tecnología EJB sobre la que trabaja ANTS. Así pues mediante está API se han desarrollado dos herramientas colaborativas, el Maptool y el Instant Messages, los cuales han sido integrados en dos sistemas educacionales Web BSCL y FLE. Hemos visto además como estas herramientas colaborativas pueden guardar la información en el entorno siempre y cuando este ofrezca una API para la invocación de una serie de operaciones. Así pues mediante el uso de http en el caso del BSCL y el uso de una interfaz basada en JPE en el caso de FLE, nuestro sistema desarrollado se favorece también de la colaboración asíncrona, puesto que las herramientas pueden guardar y recuperar información de sesiones previas del entorno subyacente.

Otro de las características que se ofrecen en estas integraciones es un sistema de logs de las herramienta colaborativas síncronas. Se ha mostrado como dicho sistema de logs tiene como base el bus de eventos del sistema, a través del cual recoge la información relevante y la guarda en las estructuras apropiadas. A través de este estudio hemos ofrecido herramientas para realizar un análisis cuantitativo como cualitativo.

Mediante este trabajo hemos obtenido un sistema colaborativo síncrono que nos permite la construcción de herramientas colaborativas basadas en una API de comunicación genérica no dependiente de la tecnología subyacente. Mediante la integración de este sistema en dos entornos, hemos podido comprobar que dicho sistema no sobrecarga al entorno existente, produciéndose una total conexión si dicho entorno ofrece una API de comunicación.

En nuestro segundo resultado hemos mostrado como mediante el uso de la tecnología J2EE y el bus de eventos ANTS.COMM hemos construido una plataforma colaborativa que puede ser también el soporte para el Aprendizaje por Descubrimiento. La base principal de esta arquitectura la ofrecen las interfaces *Broker* y *Environment*, mediante las cuales se lleva a cabo todo al comunicación síncrona y asíncrona del sistema. Esta plataforma proporciona un conjunto de interfaces bien definidos para la creación tanto de herramientas colaborativas así como los experimentos, favoreciendo con esto último el soporte para una aplicación de aprendizaje basado en la experimentación. Tanto herramientas como experimentos se construyen de manera independiente de la tecnología subyacente, ya que todos los métodos necesarios para la comunicación de información así como la obtención de ésta son ofrecidos en la clase *Environment*.

Esta arquitectura ha sido la base para la creación de la aplicación COLAB que es una aplicación para la experimentación colaborativa en el área de la física.

Esta arquitectura ofrece un *Marco de referencia* común mediante el uso de modelos XML y un conjunto de eventos bien definidos manejables por todas las aplicaciones del sistema. Ofrece dos *Escenarios de colaboración*, el síncrono basado en el bus de eventos y el asíncrono basado en el manejo de información del servidor J2EE, guardando la información de una sesión y su posterior recuperación. Para ello, cuando el último usuario abandona la sesión el sistema hace el guardado de toda la información actual. En el desarrollo de COLAB se ofrecen un conjunto de *Herramientas colaborativas* que permiten la colaboración a los usuarios. El diseño de los experimentos así como las herramientas visuales para el control de estos son el marco para el *Espacio de experimentación* el cual está basado en la colaboración síncrona, siendo su principal eje el Bus de eventos a través del cual las herramientas pueden controlar el curso de la simulación y los experimentos pueden distribuir los resultados actuales a todos los clientes interesados en dicho experimento. De esta forma, nuestra plataforma colaborativa también soporta los cuatro puntos esenciales que Wouter proponía en el diseño de una plataforma para el Aprendizaje por Descubrimiento.

Esta arquitectura también ofrece soporte para la recogida de los eventos más relevantes que se originen de la interacción entre usuarios y el sistema, permitiendo su almacenamiento para su posterior análisis. Para ello se ha incluido un listener, OnlineLogger, para la recogida de eventos y su guardado en el fichero adecuado así como un método en *Environment* que permita etiquetar este evento como evento de log.

Mediante este entorno hemos mostrado la construcción de una aplicación para la experimentación desarrollada sobre nuestra arquitectura. Debido al carácter genérico con el que se definen los recursos, tanto de herramientas visuales como experimentos, la arquitectura desarrollada podría ser aplicada a otros ámbitos. Hemos analizado los posibles escenarios de colaboración que pueden llevarse a cabo dentro de la plataforma, mostrando como se comunican aquellas herramientas colaborativas para comunicar

información entre ellas, así como la comunicación entre herramientas colaborativas y experimentos.

Además, puesto que son las herramientas y el uso de los parsers los que le dan valores y la configuración del campo de trabajo, la inclusión de diferentes herramientas así como los diferentes valores en los modelos de datos darían lugar a otro ámbito de aplicación. Incluso se podría usar la misma aplicación cliente, puesto que la información es manejada atendiendo a una previa configuración del sistema.

Puesto que trabajamos con herramientas de colaboración síncrona, en ambos resultados se han creado algoritmos tanto para el manejo de los usuarios conectados, la coordinación de turnos dentro del área (*Floor Control*) así como la llegada tarde de un usuario.

Más concretamente en este trabajo de investigación se ofrecen dos algoritmos de *Presence Awareness* que permiten detectar cuando un usuario se ha quedado en un estado inconsistente liberando de esta forma los recursos de los que disponía. Uno de ellos creando estructuras de usuarios sin el acceso a la base de datos del entorno y otro de ellos favoreciéndose de las estructuras creadas en el entorno.

El diseño de *Floor Control* que hemos propuesto está basado en el modelo replicado, propagando los cambios de estados de floor entre los diferentes interesados en dicho floor. Basándonos en este diseño hemos incluido dos políticas basadas en la figura del chair. Este diseño es génerico y podría se adoptado por cualquier entorno.

En cuanto a la llegada tarde, en cada solución hemos propuesto diferentes algoritmos, todos ellos tienen en común que todos los colaboradores de la sesión forman parte de los servidores de estado, de tal forma que En el desarrollo del Maptool para la integración en BSCL y FLE, hemos propuesto un algoritmo basado en la lista de usuarios conectados, de tal forma que el usuario que llega tarde solicita el estado actual a uno de los usuarios conectados.

En el algoritmo desarrollado para nuestra plataforma colaborativa hemos adaptado el algoritmo de Jürgen Vogel, Martin Mauve eliminando el problema de la replicación de estado basándonos en el uso de mecanismos centralizados para la coordinación entre los diferentes servidores del estado. De esta forma ofrecemos un nuevo diseño para aquellos sistemas en los que la duplicación del envío de información produzca un fuerte impacto en el ancho de banda del sistema.

2. Vías Futuras.

Una de las ramas de investigación que queda abierta es el análisis de la información de logs, de aquellos eventos relevantes que nos permita inferir información más concreta acerca del comportamiento de los usuarios, así como la clasificación de estos. Para ello el uso de técnicas de DataMining y WebMining que mediante el análisis de los ficheros de logs originados por la interacción del usuario y el sistema, nos podrían ofrecer patrones y tendencias de los usuarios, de tal forma que esta información sirviera de retroalimentación para una mejor planificación de las sesiones de trabajo.

Los avances en tecnologías de dispositivos móviles producen nuevos entornos de colaboración. Todo esto, complementado por la ubicuidad de las redes móviles de dispositivos, presenta un escenario de aplicación especialmente interesante para aplicaciones de colaboración. Por lo tanto, otra de las ramas que se abren es cómo combinar el uso de dispositivos móviles tanto con el sistema colaborativo así como en la plataforma.

Otro de las investigaciones futuras que se abren desde nuestro trabajo, es la comunicación de nuestra plataforma con otras aplicaciones, permitiendo la comunicación de información de plataformas heterogéneas entre sí, pudiendo por ejemplo visualizar datos ofrecidos por otras herramientas en nuestras aplicaciones y viceversa. Más concretamente, a través del proyecto NetCOIL (Scientific Network for Collaborative Inquiry Learning) financiado por DFG (Deutsche Forschungsgemeinschaft / German Science Foundation) se está estudiando el desarrollo de una especificación SAIL llevada a cabo por el grupo TELS [TELS] para la creación de una plataforma que permita la comunicación entre diferentes resultados como son COLAB, WISE [Lin03], Pedagogica y CoolModes [Pin03]. En este proyecto colaboran las siguientes instituciones: IPN Kiel (Germany), University of Oslo, University of Amsterdand, University of Twente (Netherlands), University of Berkeley (USA) y la Universidad de Murcia.

El uso de sistemas de video y audío podría favorecer y enriquecer las herramientas de colaboración síncrona. Más concretamente la aparición del protocolo SIP y de la API JAIN [JAIN] podrían ser una de las tecnologías subyacentes sobre la cual basar el transporte de información de las herramientas visuales, aportando nuevos servicios subyacentes debajo de ANTS.COMM. De igual manera, otra de las vías futuras sería incluir Jabber como un posible servicio de comunicación asíncrona en la capa superior de ANTS.COMM. Con esta integración, la pila de protocolos quedaría como se muestra en la figura 113, quedando todos estos servicios a disposición de todos los trabajos desarrollados sobre la API ANTS.COMM.

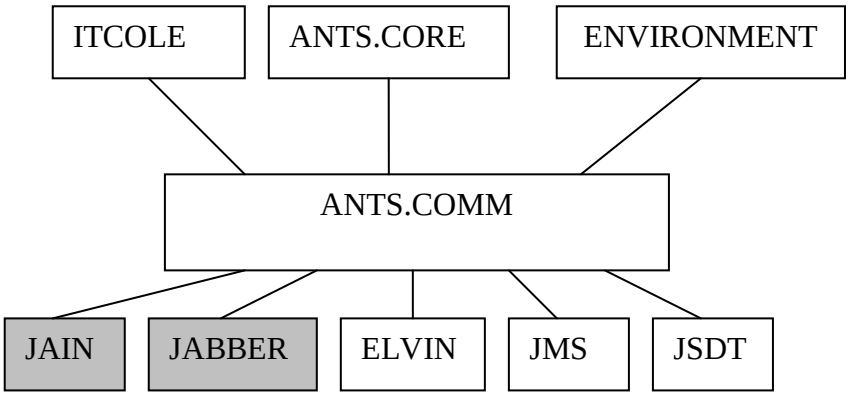

Figura 113. Protocolo de Servicios posibles para la API ANTS.COMM.

Otra de las tendencias en el ámbito colaborativo es seguir un modelo descentralizado dónde recursos de diferentes partes de la red estén disponibles para diversos miembros. Más concretamente uno de los protocolos más conocido para el compartimiento de recursos es el protocolo P2P "peer to peer". Aplicaciones de este tipo funcionan con gran éxito como son Napster, Gnutella, KaZaA o eMule. El p2p no sólo ofrece lo que es

la distribución de recursos streaming, web hosting, etc, sino que provee otros servicios como puede ser el uso de Hashtables distribuidos (DHTs).

Más concretamente, la plataforma DERMI [Pai04] basándose en DHT-p2p, provee un middleware para la construcción de herramientas colaborativas favoreciendo el compartimiento de recursos a través de la red. Una de las vías futuras que se abren en nuestro trabajo, es el estudio de estas nuevas tecnologías basadas en el compartimiento de recursos y cómo poder aplicarlo en los diferentes ámbitos colaborativos como puede ser el mundo de los negocios o un entorno educacional.

Referencias.

[Ack96] Ackerman, M. and Starr, B. “Social Activity Indicator for GroupWare”, IEEE Computer, 1996

[Anj04] Anjewierden A, Analog for Colab, May 15, 2004

[Ayu01] Ayuso Pérez, M. J. and Gómez Skarmeta, A.F. “Proyecto Informático: Herramienta de Tutorware para el entorno de aprendizaje JLE”

[Bac03] Bacon J. and Harris T. (2003). Operating Systems: Concurrent and Distributed Software Design. Ed: Addison Wesley

[Beg98] James Michael Allen Begole (1998), “Flexible Collaboration Transparency: Support Worker Independence in Replicated Application-Sharing Systems”. Dissertation submitted in the Faculty of Virginia Polytechnic Institute and State University.

[Bel97] Bell, P. (1997). “Using argument representations to make thinking visible for individuals and groups”. In R. Hall, N. Miyake, & N. Enyedy (Eds.), Proceedings of CSCL '97: The Second International Conference on Computer Support for Collaborative Learning, (pp. 10-19). Toronto: University of Toronto Press. http://www.kie.berkeley.edu/sensemaker/

[Bor00] Borghoff, Uwe M. and Schlichter, Johann H. (2000) “Computer-Supported Cooperative Work”. Editorial Springer-Verlag.

[Bru95] Bruffee, Kenneth A. “Sharing our toys- Cooperative learning versus collaborative learning”, Change, Jan/Feb, 1995 pp12-18.

[BSCW] BSCW link http://bscw.gmd.de/bscw/

[BSCL] BSCL link http://bscl.gmd.de/bscl2/

[Budd] Link cliente de mensajería BuddySpace http://buddyspace.sourceforge.net/

[Bur99] Burridge R, Sun Java Shared Data Toolkit ver 2.0, http://java.sun.com/products/java-media/jsdt, 1999

[Ces03] Donatella Cesareni, Bruno Emans, Vassilis Kollias, Minna Lakkala, Jiri Lallimo, Beatrice Ligorio, Ilaria Mancini, Francesca Martini, Marjaana Rahikainen, Essi Ryymin, Wilfred Rubens, Henk Sligte, Xanthi Vamvakousi “Final test and evaluation report” ITCOLE Deliverable 7.5 (2003)

[Cha99] Chabert A., Grossman E., Jackson L., Pietrovicz S, "NCSA Habanero-Synchronous collaborative framework and environment", 1999. http://havefun.ncsa.uiuc.edu/habanero/Whitepapers/ecscw-habanero.html

[Cha01] Chad Vawter and Ed Roman, "J2EE vs Microsoft .NET". The Middleware Company. June 2001

[Col04] Colab web link http://www.co-lab.nl

[Cou01] Coulouris G.F., Dollimore J. and Kindberg T. (2001). Distributed Systems, Concepts and Design, Ed: Addison-Wesley.

[Cro90] Crowley T, "MMConf: An Infraestructure for Building Shared Multimedia Applications". ACM CSCW`90 Conference on Computer-Supported Cooperative Work Systems Infraestructure for CSCW, 1990

[Cur92] P. Curtis, "Mudding: Social Phenomena in Text-Based Virtual Realities", in the Proceedings of the 1992 Conference on the Directions and Implications of Advanced Computing, Berkeley, 1992. ftp://ftp.lambda.moo.mud.org/pub/MOO/papers/DIAC92.ps

[Cur93] Pavel Curtis. LambdaMOO Programmer's Manual. XEROX Palo Alto Research Center, 1996. ftp://ftp.lambda.moo.mud.org/pub/MOO/

[DAO] Data Access Object pattern in J2EE. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

[DeB03] De Benito, B.y Pérez, A "La evaluación de los aprendizajes en entorno de aprendizajes cooperativos", Redes de comunicación en la enseñanza, Ed. Paidós, capítulo 8 pp 209-226, 2003

[Dew91] Dewan, Choudhary "Primitives for Programming Multi-user Program", ACM Transactions on Computer-Human Interaction, 1991

[Dew98] P. Dewan y R. Choudhary (1998) "Coupling the User Interfaces of Multiuser Program". ACM Transactions on Computer-Human Interaction, Marzo,1998

[Dom97] Dommel, Hans-Peter and Garcia-Luna-Aceves, JJ. "Floor Control for Multimedia Conferencing and Collaboration". Multimedia Systems, 5, pp 23-38. Springer-Verlag 1997

[DU01] Diversity University. 2001. http://www.du.org/

[Edu] Enlace de la plataforma EDUSTANCE http://www.edustance.com

[Ege00] Egea, J., Skarmeta, A. G., García, P., Gisbert, M. and Rallo, R. (2000) "Un entorno para la enseñanza abierta y a distancia: JLE". Boletín RedIris nº 52 pp 17-24

[FLE03] Future Learning Environment. 2003 http://fle3.uiah.fi/

[Gaim] Link Cliente de Mensajería Gaim http://gaim.sourceforge.net/

[Gar01] García López P.,Gómez Skarmeta A., Rallo Molla, A.. "*ANTS: a new Collaborative Learning Framework".* Proc. Of the European CSCL. Maastricht (Holanda) Marzo 2001

[Gar02] García López P., Gómez-Skarmeta A. "Plataforma ANTS: Una arquitectura software basada en componentes para el desarrollo de Aplicaciones Distribuidas de Trabajo Colaborativo" Tesis Doctoral (2002).

[Gar02b] García López P., Gomez Skarmeta, A., Martínez Carreras, M. A. "AWS: A bridging the gap between awereness and proactive Tutorware" AACE EDMEDIA Colorado (USA) Junio 2002

[Gey00] Geyer W, Vogel J, Mauve M, "An efficient and flexible late join algorithm for interactive shared whiteboard". 2000

[Gil96] an L. and Schreiber R. (1996). Distributed Computing with IBM MQSeries. New York: Wiley.

[Gom00] Gómez-Skarmeta, A. F., García Parens, E., Martínez Carreras, A. "Caracterización de entornos de formación: Experiencias", boletín RedIris nº 54-55, 2000-2001

[Gom03] Gómez-Skarmeta, A. F., García Parens, E., Martínez Carreras, A. "Nuevas Tecnologías y herramientas en la Teleformación", Redes de Comunicación en la Enseñanza. Las nuevas perspectivas del Trabajo Corporativo. Ed. Paidós, pp 227-258, 2003.

[Gra96] T.C. Nicholas Graham, Tore Urnes and Roy Nejabi. Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware . In Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'96). ACM Press, Seattle, USA, pp. 1-10, Noviembre 1996.

[Gra97] Graham Hamilton, "Java Beans", Sun Microsystems 1997.

[Gro] GroupKit link
http://www.cpsc.ucalgary.ca/grouplab/project_snapshots/groupkit/groupkit.html

[Gut96] C. Gutwin, M. Roseman, S. Greenberg, "A Usability Study of Awareness Widgets in Shared WorkSpace GroupWare Systems" ACM Conference on Computer Supported Work, 1996

[Hil94] Hill, R.D., T. Brinck, S.L. Rohall, J.F. Patterson and W. Wilner, 'the rendezvous architecture and language for constructing multiuser applications'. *ACM Transactions on Computer-Human Interactio*n, 1 (June 1994), 2, p. 81-125, 1994. http://www.acm.org/pubs/articles/journals/tochi/19941-2/p81-hill/p81-hill.pdf.

[Hon00] Honkela T, Leinonen T, Lonka K, Raike A. "Self Organizing Maps and Constructive Learning". Proceedings of ICEUT, IFIP, Beijing, August, pp 339-343.

[Hum02] Humphrey Sheil and Michael Montero “Rumble in the jungle: J2EE versus .Net”, JavaWorld, June 2002.

[IEE00] IEEE 1484 Learning Objects Metadata (IEEE LOM). http://www.ischool.washington.edu/sasutton/IEEE1484.html

[IMS] IMS (Instructional Management System) http://www.imsproject.org/

[IMS01] IMS Content Packaging Specification. http://www.imsproject.org/content/index.html

[IMS01b] IMS Question & Test Interoperability Specification. http://www.imsproject.org/question/index.html

[J2EE] Java 2 TM Platoform Enterprise Edition Specification, v1.4, release November 24, 2003. Sun Microsystems, Inc.

[Jabb] Link Jabber Foundation http://www.jabber.org

[Jab04] Link “Instant Messaging Interoperability moves forward” http://www.jabber.com

[JAIN] Enlace de JAIN http://java.sun.com/products/jain/index.jsp

[JBOSS] JBoss link http://www.jboss.org/index.html

[JEP] JEP-0060 Publish-suscribe Draft http://www.jabber.org/jeps/jpe-0060.html

[JJ81] Peter Johnson-Lenz, Trudy Johnson-Lenz. "Consider the Groupware: Design and Group Process Impacts on Communication in the Electronic Medium," in Hiltz, S. and Kerr, E., Studies of Computer-Mediated Communications Systems: A Synthesis of the Findings, Research Report #16, Computerized Conferencing and Communications Center, New Jersey Institute of Technology, Newark, New Jersey, 1981.

[Joh88] Johansen, R., 'Current user approaches to groupware'. In R. Johansen (ed.), *Groupware : Computer support for business team*s. Free Press, New York, 1988, p. 12-44. 1988.

[Jon01] Jonassen, D. H., Kown, H. (2001) ”Communication Patterns in Computer Mediated Versus Face-to-Face Group Problem Solving”, ETR&D, Vol 49, No 1, 2001, pp 35-51.

[Jon98] Jong, T. de, & Joolingen, W.R. van (1999). “Discovery Learning with computer simulations of conceptual domains”. Review of Educational Research, 68 179-201.

[Joo00] Joolingen, W.R. van (2000). “Designing for collaborative discovery learning”. In G. Gauthier, C. Frasson and K. VanLehn (Eds). Intelligent Tutoring systems. (pp. 202-211).Berlin: Springer.

[JTA] Java Transaction Api http://java.sun.com/products/jta

[JXTA] Proyecto JXTA http://www.jxta.org

[KF301] Knowledge Forum 3. 2001. http://www.learn.motion.com/lim/kf/kf3info1.html

[Kli01] Kligyte G., Leinonen T. "Study of functionality and interfaces of existing CSCL/CSCW systems" Deliverable 3.1. ITCOLE project, 2001

[Laz04] Lazonder, A., Carrata, A., Jong, T., Joolingeng, W., Manlove, S., Miani, L., et Al (2004). "Report on Evaluation Studies" Deliverable 8, Colab project , 2004

[Lei01] Leinonen T., Hakkarainen K., Appelt W., Dean P., Gómez-Skarmeta A., Ligorio B., Lipponen L., Merisaari L., Mielonen S., Pontecorvo C., Sligte H., & Vosniadou S. (2001) "ITCOLE Project: Designing innovative technology for collaborative learning and knowledge building". Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications. Tampere. Finland, June 25-30, 2001.

[Lei02] Leinonen, T.; Kligyte, G (2002). Future Learning Environment for Collaborative Knowledge Building and Design. "Development by Design" Conference (DYD02), Bangalore, India 2002. Published online at http://www.thinkcycle.org/.

[Li98] Li, Du and Muntz, Richard. "COCA: Collaborative Objects Coordination Architecture". Department of Computer Science University of California, Los Angeles, 1998

[LIMU] Limu Link http://www.limu.com.

[Lin03] Linn, M.C., Clark, D.B. & Slotta, J.D. "WISE Design for Knowledge Integration". In S. Barab (Ed.) Building Sustainable Science Curriculum: Acknowledge and Accommodating Local Adaptation. Special Issue of Science Education, 2003

[Mar99] Martin Erzberger and Marcel Altherr. "Every Dad needs a Mom: Message-oriented Middleware" White Paper. Softwired, 1999.

[Mark97] Mark Day. "What Synchronous Groupware Needs: Notification Services". Proceedings of the 6th Workshop on Hot Topics in Operating Systems. IEEE pp 118-122, 1997

[Mart02] M. A. Martínez Carreras, A. F. Gómez Skarmeta, J. Tavira Moreno, P. García López "Integración de herramientas síncronas en un sistema BSCW" IE-2002, Vigo 2002

[Mart03] Martínez Carreras, M. A., Gómez Skarmeta A. F, Pérez Sánchez, J. A., Tavira Moreno, J., García López, P. "Integration of the ANTS Collaborative System inside the educational environment BSCL and FLE". Proocedings of the MIcte Conference, 2003

[Mart04] Martínez Carreras M. A., Gómez Skarmeta A. F., Martínez Graciá E., Mora González M. "COLAB: Una plataforma para la simulación en entornos colaborativos en

laboratorios virtuales". Inteligencia Artificial, Invierno 2004, Volumen 8º, Nº 24 pp 45 – 53.

[Mart05] Martínez Carreras M. A., Gómez Skarmeta A. F., Martínez Graciá E., Mora González M. "COLAB: A Platform Design for Collaborative Learning in Virtual Laboratories". IFIP TC3 Technology Enhanced Learning Workshop World Computer Congress, Ed. Springer, pp 95-109, 2005

[Mid02] The Middleware Company, "J2EE and .NET Application server and Web services Benchmark", October 2002.

[Mic03] Microsoft, "Arquitectura de aplicaciones para .NET", McGraw Hill, 2003.

[ODP] Reference Model for Open Distributed Processing http://uml.fsarch.com/RM-ODP/index.html

[Pai04] Pairot, C., García P., Mondéjar R., Gómez Skarmeta, A. "Towards a Peer-to-Peer Object Middleware for Wide-Area Collaborative Application Development". Inteligencia Artificial, Invierno 2004, Volumen 8º, Nº 24, pp 55-65

[Pan97] Panitz, T. "Collaborative Versus Cooperative Learning: Comparing the Two Definitions Helps Understand the nature of Interactive learning", **Cooperative Learning andCollege Teaching**, V8, No. 2, Winter, 1997

[Pao+95] M. Paolucci, D. Suthers, and A. Weiner. Belvedere: Stimulating students' critical discussion. In CHI95, Denver, CO, May 1995.

[Pat96] Patterson, J. F., Day, M., and Kucan, J., "Notification Servers for Synchronous Groupware", in Proceedings of the 6 th ACM Conference on Computer- Supported Cooperative Work, ACM Press, 1996, pp. 122-129.

[Pin03] Pinkwart, N. "A Plug-In Architecture for Graph Based Collaborative Modeling Systems". In U. Hoppe, F. Verdejo, J. Kay (eds.): Shaping the Future of Learning through Intelligent Technologies. Proceedings of the 11th Conference on Artificial Intelligence in Education. Amsterdam, IOS Press, 2003

[Rap91] Rapaport, M. (1991) "Computer Mediated Communication" New York, John Wiley & Sons, Inc.

[Ros96] Roschelle, J. (1996) Learning by collaborating: Convergent conceptual change. In T. Koschmann (Ed.) *CSCL: Theory and Practice of an Emerging Paradigm*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 209-248.

[SC84] Sluizer, S. and P. Cashman, 'XCP: An experimental tool for supporting officeprocedures'. In R.W. Taylor (ed.), *Proceedings of the IEEE first international conference on office automation, NewOrleans, LA, USA, 198*4. IEEE Computer Society Press, Silver Spring, MD, USA, 1994, p. 73-80.

[SG85] Sarin, S. and I. Greif, 'Computer-based real time conferencing systems'. *Compute*r, 18 (October 1985), 10, p. 33-45.

[Sim] SimQuest www.simquest.nl

[Sin99] Singh, Munindar P. “Write asynchronous, run synchronous”. IEEE Internet Computing. March/April 1999 pp 4-5

[Smi92] Smith, Randall B. “What You See Is What I Think You See”. SIGCUE Outlook, 21(3), 18-23, Presented at ACM Conference on Computer Supported Collaborative Learning

[Spr99] Springer, L., Stanne, M. E., Donovan, S.S. “Effects of small-group learning on undergraduates in science, mathematics, engineering and technology: a meta-analysis”. Review of Educational Research, 69, 21-51

[Sta00] Stahl, G. “A Model of Collaborative Knowledge-Building". International Conference of the Learning Sciences. 2000.

[Ste87] Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., Suchman, L. (1987): “Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings”. Communication of the ACM 30:1, 32-47

[Szy98] Szyperski, C. “Component Software. Beyond Object-Oriented Programming”. Addison Wesley, 1998.

[TELS] Enlace de TELS http://docs.telscenter.org/

[Vog03] Vogel, J., Mauve, M., Hilt V. and Effelsberg,W. “Late join algorithm for distributed interactive applications”, Multimedia Systems, 2003, 9 pp 327 - 336. Springer-Verlag 2003

[W3C00] http://www.w3.org/TR/SOAP

[Wol02] Wolfgang Appelt, Rudolf Ruland, Gerry Stahl, Antonio F. Gómez Skarmeta, Mª Antonia Martínez Carreras, Javier Tavira Moreno, Jose Antonio Perez Sanchez, “ITCOLE PROJECT DELIVERABLE 4.3 Revised system specification for Synergeia Version 2”, Octubre 2002

[WMFC] WorkFlow Management Council http://www.wfmc.org/

[X-BSCW] X-BSCW: XML-RPC Application Programming Interface to BSCW. Fraunhofer FIT, April 2004

[Yeo95] Yeong, W. et al. Lightweight Directory Access Protocol. RFC 1777. 1995.

http://www.umich.edu/~dirsvcs/ldap/doc/rfc/rfc1777.txt

Anexo I. Sistema de Mensajería instantánea en Jabber con autenticación LDAP.

Introducción.

Jabber [Jabb] es un conjunto de protocolos XML y de tecnologías que facilitan a dos clientes de internet el intercambio de mensajes, posibilidad de ver que clientes están conectados, si están presentes o ausentes así como otras características de la colaboración en tiempo real. Entre las características que ofrece Jabber cabe destacar:

- Ofrece servidores y clientes de Open sources.
- Basado en estándares, el IETF ha formalizado los estándares de los protocolos de XML para streaming bajo el nombre de XMPP (Extended Message Presence Protocol). Dichos protocolos son en los cuales se basa las especificaciones de Jabber.
- Estabilidad, las primeras implementaciones llevan desde 1998 y en la actualidad hay servidores bastante estables y millones de personas usando Jabber para Mensajería Instantánea (MI).
- Descentralizado, la arquitectura de Jabber permite la instalación de un servidor propio de Jabber habilitando a organizaciones, individuos, etc tomar el control sobre la mensajería Instantánea.
- Seguridad, cualquier servidor Jabber puede ser aislado de la red publica Jabber.
- Flexibilidad, las aplicaciones Jabber de MI incluye gestión de red, herramientas colaborativas, compartición de ficheros, etc
- Diversidad, hay una gran cantidad de proyectos open-source que usan los protocolos Jabber para crear aplicaciones y servicios de tiempo real.

Debido a las características de descentralización, flexibilidad y diversidad que ofrece Jabber, este servidor se presenta como el mejor candidato para que formar parte del soporte del servicio de mensajería instantánea, puesto que ofrece la posibilidad de crear un servidor propio, aislado de la red general y con una licencia General Public License.

Diseño de comunicación Jabber con autenticación LDAP.

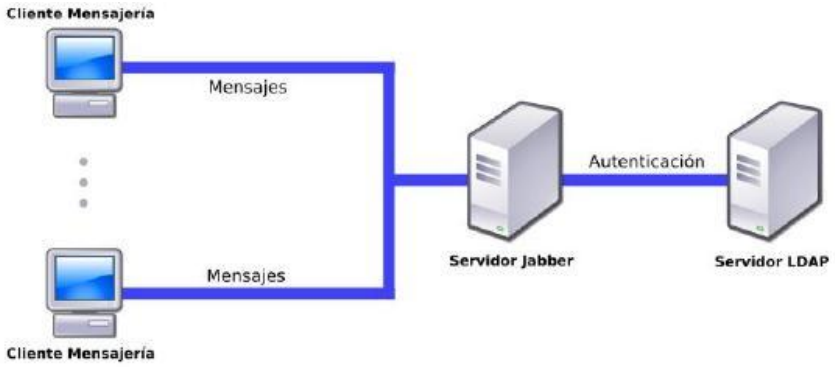

Figura 114. Esquema de mensajería instantánea basada en Jabber y autenticada bajo LDAP.

El esquema de la figura 114 muestra el diagrama de conexión entre los clientes y el servidor Jabber, así como la conexión de este para la autenticación de los clientes en un servidor LDAP. El servidor Jabber elegido para la propagación de eventos entre los clientes ha sido la versión 1.4.3. Para la conexión con el servidor LDAP configurado en la Universidad de Murcia ha sido necesario instalar el módulo xdb_ldap basado en programación C.

Con el fin de que las conexiones de autenticación a dicho servidor se realizen siguiendo los patrones definidos dentro de nuestro esquema LDAP ha sido necesario modificar los ficheros xdb_ldap_jud.c y xdb_ldap_common.c

Así para establecer la conexión entre el servidor Jabber y el servicio de directorio LDAP se ha incluido los siguientes valores en el fichero de configuración Jabber.

```
<xdb id="xdb_ldap">
  <ns>jabber:iq:auth:0k</ns>
  <ns>jabber:iq:auth</ns>
  <ns>vcard-temp</ns>
  <ns>jabber:jud:users</ns>
  <host/>
  <load>
    <xdb_ldap>./xdb_ldap/src/xdb_ldap.so</xdb_ldap>
  </load>
  <xdb_ldap xmlns="jabberd:xdb_ldap:config">
    <connection>
      <host>jtavira.atica.um.es</host>
      <port>389</port>
      <rootdn>o=Universidad de Murcia,c=es</rootdn>
      <uniqattr>uid</uniqattr>
      <binddn>cn=jtavira, o=Universidad de Murcia, c=es</binddn>
      <bindpw>password</bindpw>
      <suffix>ou=usuarios, o=Universidad de Murcia, c=es</suffix>
</connection>
    <spool><jabberd:cmdline flag='s'>./spool</jabberd:cmdline></spool>
  </xdb_ldap>
</xdb>
<xdb id="xdb_other">
```

```
  <ns>jabber:iq:roster</ns>
  <ns>jabber:iq:private</ns>
  <ns>jabber:iq:register</ns>
  <ns>jabber:iq:filter</ns>
  <ns>jabber:x:offline</ns>
  <host/>
  <load>
    <xdb_file>./xdb_file/xdb_file.so</xdb_file>
  </load>
  <xdb_file xmlns="jabber:config:xdb_file">
    <spool><jabberd:cmdline flag='s'>./spool</jabberd:cmdline></spool>
  </xdb_file>
</xdb>
```

Más concretamente, para la prueba de este servicio se hizo uso de los clientes de mensajería Gaim [Gaim] escrito en C y BuddySpace [Budd] el cual está escrito en Java. El uso del estándar XMPP en Jabber hace posible la comunicación entre dichos clientes así como del control de presencia, favoreciendo la interoperabilidad.

Jabber Software Foundation surge con el objetivo de potenciar el uso de Jabber así como la definición del protocolo XMPP. Igualmente esta fundación pretende potenciar la extensibilidad de dicho protocolo y de realizar una serie de extensiones para su interoperabilidad con otras tecnologías. Así pues se han llevado a cabo extensiones como son la comunicación con XML-RPC así como el estudio de SOAP, comandos ad-hoc, actividad del usuario, transferencia de ficheros, así como la definición de un gateway para permitir la comunicación XMPP-SIP.

Más concretamente, se está anunciado el desarrollo de un gateway XMPP-SIP [Jab04] que permita la comunicación entre ambos protocolos, habilitando de esta forma la creación de herramientas de vídeo conferencia que permitan la transmisión de audio y vídeo así como de mensajería.

Anexo II. Características de un entorno educacional a distancia.

Las características de un entorno educacional a distancia deben ser tales que permitan a dicho entorno suplir las funciones de un sistema de educación tradicional y además dotarlo de otra serie de funciones adicionales propias de un sistema de educación a distancia.

Generalmente los entornos de aprendizaje a distancia van asociados con los sistemas gestores de recursos. Esto es debido a que estos sistemas son realmente usados por aquellas personas que necesitan de una flexibilidad de tiempo para llevar a cabo la tarea de aprendizaje. A diferencia de los sistemas CSCL, no es la experiencia compartida la que genera el aprendizaje, sino el estudio de una serie de recursos en el entorno así como pruebas de nivel mediante algunos de ellos.

Si bien el entorno debe poseer las suficientes herramientas para que el alumno pueda realizar su tarea de aprendizaje dentro de él, también se requiere un compromiso y responsabilidad por parte del alumno, puesto que el profesor no es el que dicta las pautas de aprendizaje, ni el ritmo de la clase, sino es una entidad encargada en suministrar los recursos didácticos y en ayudar al alumno en las dudas que se le plantea mediante el trabajo con esta plataforma.

Más concretamente este tipo de aprendizaje está cobrando relativa importancia en el ámbito profesional, puesto que de esta forma se pueden distribuir un conjunto de recursos a los trabajadores de una empresa, manteniendo ellos el ritmo del aprendizaje.g

Roles y grupos

Al igual que en cualquier sistema educacional, no todos los usuarios que accedan al sistema tendrán los mismos intereses y privilegios. Cada uno de ellos accederá con unos fines distintos, jugará un papel diferente. Para ello el entorno debe permitir asignar un papel distinto a cada uno de ellos, un **rol**. A nuestro juicio tres son los roles básicos en todo entorno educacional, a saber:

- **Administrador**. El entorno debe contar con un único administrador, que será el que se encargue de gestionar al resto de usuarios y recursos del sistema.

- **Profesor**. El entorno contará con tantos profesores como sea necesario, los cuales se encargarán de elaborar los recursos para la formación de los alumnos, de estudiar el seguimiento de éstos ante los recursos, de prestar la atención necesaria y personalizada a cada uno de los alumnos,...

- **Alumno**. Los alumnos deben constituir el componente básico de todo entorno educacional. Los esfuerzos de los profesores se deben centrar en elaborar recursos apropiados para ellos y el sistema debe ocuparse de poner a la

disposición del profesor todos los medios necesarios para llevar a cabo dicha misión.

Por otro lado, no todos los usuarios que accedan al sistema deberán tener acceso a los mismos recursos y para ello será necesaria la creación de diversos **grupos**, de forma que cada uno de los usuarios forme parte de uno de ellos. De esta forma se podrá limitar el acceso de los usuarios a los diversos recursos del sistema.

Contenidos

Como hemos comentado anteriormente la base de estos sistemas de aprendizaje son los contenidos. Debido a que estos sistemas a distancia van a requerir que el momento de aprendizaje así como la consecución de éste sea llevado con total flexibilidad por parte del alumno, es necesario diseñar con todo detalle los contenidos educativos de dicho sistema.

Cuando en lo sucesivo nos refiramos a los **contenidos** del sistema, estaremos haciendo referencia a aquellos materiales elaborados por el profesor y transmitidos a los alumnos a través de la plataforma educacional, permitiéndoles así ir evolucionando en su aprendizaje. Por lo tanto, podremos entender como tales los contenidos teóricos relacionados con un determinado tema, las actividades, los exámenes,... pudiendo distinguir en este último caso entre exámenes de autoevaluación y exámenes de evaluación.

- Los **exámenes de autoevaluación** se deben generar de forma aleatoria a partir de una batería de preguntas elaboradas por el profesor, relacionadas con el tema en cuestión. A partir de ellos el alumno conocerá cual está siendo su evolución a lo largo de un determinado tema, permitiéndole así decidir en qué contenidos debe hacer mayor hincapié. Estos exámenes deben ser corregidos en su totalidad por el sistema, sin que sea necesaria la intervención del profesor. Lo ideal sería que éste tuviera conocimiento de que el alumno ha realizado un examen de autoevaluación pero no de su resultado, lo cual condicionaría el comportamiento del alumno.

- Los **exámenes de evaluación**, por el contrario, serán generados por el profesor y todos los alumnos que formen parte de un determinado curso estarán obligados a su realización. Además, en condiciones normales, el profesor deberá establecer una fecha y una hora para la realización de dicho examen por parte de todos los alumnos y determinar su duración. En este caso, a partir del sistema de logs del entorno en cuestión, el profesor podrá conocer el momento de inicio y de fin del examen de cada uno de los alumnos para saber así si se ha realizado de acuerdo a lo establecido. En lo que respecta a la nota del examen de evaluación, la máquina proporcionará al profesor una nota 'aproximada' como resultado de la corrección realizada de aquellas preguntas que así lo permitan. Dicha nota no debe ser conocida por el alumno, que únicamente tendrá noticias de la nota obtenida cuando el profesor corrija el examen y establezca la nota definitiva.

Herramientas básicas en un entorno educacional

Es conveniente que todos los contenidos del sistema respondan a una estructura común. Para ello, a nuestro juicio el sistema debe contar con las siguientes herramientas básicas:

- Un **editor de contenidos** que permita a los profesores del entorno elaborar los contenidos teóricos de un determinado tema dotándolos de la estructura propia de los contenidos del sistema. Dichos contenidos deben ser elaborados introduciendo una serie de etiquetas las cuales nos ayuden tanto en la búsqueda del contenido en el recurso así como en el seguimiento de las partes de dicho contenido. Actualmente se están llevando a cabo numerosas investigaciones de que deben ofrecer los recursos y como enlazar cada una de sus partes (o temas) facilitando la tarea de aprendizaje al alumno. En estas investigaciones se están usando estándares como IMS (Instructional Management Systems) [IMS]. IMS ha tratado de crear especificaciones que abarquen los diferentes módulos de un sistema de tele-enseñanza. Más concretamente en el módulo de gestión de contenidos es dónde el uso de estándares ha tenido más relevancia y se ha plasmado en estándares aceptados por organismos internacionales. Dicho módulo abarca el ciclo completo de vida de los contenidos educativos definiendo desde el etiquetado y empaquetado de unidades didácticas hasta su publicación en un servidor de contenidos, y las búsquedas y accesos a este repositorio de contenidos. Los estándares aceptados por organismos internacionales y aceptados dentro de la comunidad educativa han sido LOM (*Learning Object Model*) [IEE00]que se encarga del etiquetado de cursos y algo menos popular *IMS Content packaging* [IMS01] para el empaquetado e indexación de estos contenidos.

  También se está investigando sobre la forma en que un aprendiz o estudiante pasa de una parte de recursos a otra, dando lugar a un aprendizaje inteligente de tal forma que si un alumno demuestra suficientes conocimientos como para pasar cierta información de los recursos se le muestre otra ruta en el árbol de aprendizaje. De esta forma los alumnos avanzados podrán seguir de una forma más rápida los contenidos y los alumnos que tengan más problemas en ciertas partes indagarán más siguiendo otra rama en el árbol de los contenidos. De tal forma se podría obtener recursos adaptativos al nivel del alumno. Esta forma de aprendizaje entra en la rama del **"Learningware"**.

- Un **editor de exámenes** que permita a los profesores del entorno elaborar los exámenes tanto de evaluación como de autoevaluación dotándolos de la estructura propia de los contenidos del sistema. Básicamente, la herramienta proporcionará al profesor los mecanismos necesarios para elaborar una batería de preguntas relacionadas con un determinado tema. A partir de dichas preguntas el profesor podrá generar un examen de evaluación cuando así lo crea oportuno y al mismo tiempo serán las preguntas a partir de las cuales la máquina generará los exámenes de autoevaluación que el alumno solicite.

  El profesor deberá disponer de la opción de reservar algunas de estas preguntas para evaluación de forma que dichas preguntas 'reservadas' no puedan aparecer en ningún examen de autoevaluación generado por la máquina. En cuanto a este tipo de recursos IMS dispones de un módulo de Evaluación en el que se define

estándares de intercambio de datos tanto a nivel de preguntas como de resultados para herramientas de evaluación en un entorno formativo. Dicho módulo también ha cristalizado en estándares concretos como QTI (*Question Test Interface*) [IMS01b] que ya están implementados en diferentes aplicaciones de evaluación.

En cuanto a la evaluación del alumno por parte del profesor son necesarias, al menos, dos herramientas. A saber:

- **Herramienta de evaluación**. Dicha herramienta deberá permitir al profesor corregir los exámenes de evaluación realizados por el alumno. Además de establecer la nota de dicho examen, el profesor podrá realizar 'anotaciones' en dicho examen que permitan al alumno conocer el motivo de sus fallos así como cualquier otra apreciación que el profesor estime oportuna.

- **Herramienta de revisión de exámenes**. Dicha herramienta deberá permitir al alumno acceder a la corrección de sus exámenes de evaluación por parte del profesor.

Sistema de logs.

Llegados a este punto hemos hecho un rápido recorrido por lo que deberían ser, a nuestro juicio, las principales características de un entorno educacional. Pero de poco nos servirían si el entorno no contara con un sistema de logs que se encargue de registrar la interacción de los usuarios con el sistema y, en especial, la interacción de los alumnos con el sistema.

La decisión de cuales deben ser los datos recogidos por este sistema de logs es una tarea en la que se debe poner un cuidado especial. A partir de dichos datos, el sistema debe ser capaz de extraer información que permita, entre otras cosas, que el profesor sea capaz de controlar la evolución de los alumnos a lo largo de un determinado tema y de los contenidos relacionados con él. Dicha evaluación de la evolución de los alumnos puede ser realizada en grupo o de forma individual.

- A partir de una **evaluación en grupo** el profesor tendrá una visión general del comportamiento del grupo pudiendo deducir de dicha información, entre otras cosas, si los contenidos elaborados para un determinado tema son adecuados o si, por el contrario, los alumnos (en general) tienen problemas para seguirlos.

- A partir de una **evaluación individual** el profesor podrá conocer la evolución de un alumno en particular y compararla a su vez con la evaluación del grupo en general. Para sacar el máximo provecho de esta información, el sistema debe ofrecer la posibilidad de personalizar en un momento dado el recorrido que el profesor desea que realice un alumno en concreto a través de los contenidos del sistema, para lo cual podrá elaborar contenidos complementarios que se adapten especialmente a sus necesidades.

La información obtenida de ambos tipos de evaluación debe ser tal que permita al profesor determinar la nota conseguida por cada uno de los alumnos a lo largo de una

determinada asignatura o curso, teniendo en cuenta para ello los objetivos a alcanzar establecidos por el profesor para dicha materia.

Herramientas complementarias en un entorno educacional

Para hacer posible que el entorno educacional desarrolle su tarea de la forma más sencilla y cómoda para el alumno debe contar con una serie de herramientas para tal fin. Teniendo en cuenta sus requerimientos en cuanto al momento de su uso, podemos clasificarlas en asíncronas y síncronas. Veamos las características de cada una de ellas.

Asíncronas

En cuanto a las herramientas asíncronas interesantes a incluir en un sistema de este tipo son las siguientes:

- **Correo electrónico**.
- **Tablón de anuncios**. A través del tablón de anuncios el profesor puede comunicar a todos los alumnos de un determinado grupo una cierta información, si bien es posible también que alguno de los alumnos de dicho grupo quiera hacer algún comunicado o petición al resto de compañeros de su grupo.
- **Foro de discusión**. Mediante el foro de discusión el profesor tiene la posibilidad de plantear un tema para que los alumnos discutan sobre él. De esta forma, los alumnos expresarán su opinión acerca de dicho tema y el resto tendrá acceso a ella cuando lo desee, sin que sea necesario una interacción en tiempo real entre todos los usuarios.
- **Zona compartida**. Aunque esta herramienta no es fundamental en un entorno a distancia, si que fomenta la comunicación entre usuarios así como puede ayudar en el trabajo cooperativo, en cuanto a realización de ciertas tareas en grupo.
- **Editor colaborativo**. Al igual que la anterior, este tipo de herramientas no suelen ser la base de estos entornos puesto que esta se refieren más al trabajo en grupo que al trabajo individual. Sin embargo este tipo de herramientas pude servir en el caso de que hayan grupos o que la actividad se oriente en el uso de participación de alumnos.

Síncronas

- **Chat**. Puede ser muy útil en la comunicación con el profesor, pudiendo establecerse tutorías a través de dicha herramienta.

- **Videoconferencia**. La principal utilidad de este tipo de herramienta es permitir la emisión de clases por parte del profesor para que los alumnos sean testigos de ellas. Dichas clases pueden ser grabadas, siendo así posible que el alumno pueda asistir en directo a la emisión de dicha clase o asistir a ella en diferido cuando le sea posible o cuando así lo necesite.

- **Pizarra**. Este tipo de herramienta complementará a la anterior en la emisión de las clases y, al igual que en el caso anterior, deberá ser posible grabar una determinada sesión para que el alumno pueda acceder a ella cuando lo crea oportuno. Lo ideal es que tanto esta herramienta como la anterior estén sincronizadas.

- **Herramienta de presentaciones**. El uso ideal de esta herramienta es para la emisión de clases por parte del profesor junto con las dos anteriores.