Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Amor, Mercedes; Fuentes, Lidia; Jimenez, Daniel; Pinto, Mónica
Adaptive Collaborative Virtual Environments: a component and aspect-based approach
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 8, núm. 24, 2004, pp. 33-43
Asociación Española para la Inteligencia Artificial
Valencia, España

Adaptive Collaborative Virtual Environments: A Component and Aspect-based Approach

Mercedes Amor, Lidia Fuentes, Daniel Jimenez, Mónica Pinto

Dep. Lenguajes y Ciencias de la Computación, Universidad de Málaga
Campus de Teatinos, s/n.
Málaga 29071
{pinilla,lff,priego,pinto}@lcc.uma.es

Resumen

Nowadays, the interest in collaborative virtual environments has increased considerably, mainly due to the current technological advances that make it possible the use of computers to collaborate. One of the main challenges in the development of these systems is to achieve a high degree of adaptability and extensibility in order to deal with system evolution. We achieve this goal combining component-based and aspect-based software technologies. In this paper we are going to highlight the innovative design of CoopTEL, a component and aspect platform specially suited for developing adaptive distributed applications.

Palabras clave: Adaptabilidad, Entorno Virtual Colaborativo, Plataforma de Componentes y Aspectos, Composición Dinámica.

1. Introducción

Los avances tecnológicos de los últimos años hacen posible el uso de los ordenadores para comunicarse y colaborar con personas geográficamente dispersas formando grupos de trabajo virtuales. Esto es posible si se proporciona un Entorno Virtual Colaborativo (EVC) [Gutwin00] [Sakai02] que integre los recursos necesarios para colaborar – usuarios, herramientas colaborativas, documentos.

El desarrollo de estos EVCs, como el de cualquier otro sistema distribuido, es inherentemente complejo. Para hacer frente a esta complejidad, la Ingeniería del Software ha evolucionado adoptando nuevas tecnologías software que evitan el desarrollo de un sistema desde cero. Su principal objetivo es conseguir un alto grado de *configurabilidad*, *extensibilidad* y *adaptabilidad*. De esta forma es posible incorporar, de forma fácil y rápida, los nuevos requisitos y cambios que afectan continuamente a este tipo de sistemas, adaptándolos tanto de forma estática como dinámica.

Con este objetivo nuestra propuesta aborda la construcción de este tipo de sistemas aplicando de forma conjunta nuevos paradigmas de programación, como son el desarrollo de software basado en componentes [Brown98] (DSBC, Component-Based Software Development en inglés) y el desarrollo de software orientado a aspectos [WebDSOA] (DSOA, Aspect-Oriented Software Development en inglés). Estos dos paradigmas sirven de base para la definición de un marco de trabajo (MT) de EVCs [Amor03] que se ejecuta sobre una plataforma de componentes y aspectos [Pinto02a].

El DSBC permite desarrollar aplicaciones mediante la reutilización y composición de componentes software prefabricados. Sin embargo, para conseguir que las aplicaciones basadas en componentes sean

realmente extensibles y adaptables, es necesario realizar una descomposición adecuada del sistema en componentes totalmente independientes, lo que no es una tarea precisamente fácil. Dentro de un mismo componente es posible encontrar, además de código relativo a su funcionalidad básica, otras propiedades como comunicación, coordinación, seguridad, etc., que reducen drásticamente la reutilización y adaptabilidad de un componente a nuevos entornos de ejecución. La separación de aspectos propone una solución a este problema, que se ha dado en llamar el problema del "código enmarañado" [Kickzales97], separando en diferentes entidades, denominadas aspectos, las propiedades que están presentes en más de un componente. El DSOA modela los componentes y aspectos como dos entidades separadas donde los aspectos se mezclan o componen de forma automática con el comportamiento funcional del sistema.

El principal objetivo de combinar el DSBC y el DSOA es conseguir una mejor modularización de los EVCs haciéndolos más adaptables y configurables. La utilización conjunta de estas tecnologías sirve de base para desarrollar CoopTEL [Pinto02a], nuestra propia plataforma de componentes y aspectos sobre la que se define el MT de EVCs.

Una de las características más relevantes de la plataforma CoopTEL es la forma en la que maneja la información sobre la Arquitectura de la aplicación. En nuestra propuesta, la Arquitectura de la aplicación se describe como parte de un fichero de despliegue en XML [Pinto03] en términos de un conjunto de componentes, un conjunto de aspectos y las reglas de composición entre ellos. A diferencia de otras plataformas tipo CORBA/CCM o J2EE/EJB, en CoopTEL esta información no está codificada en los componentes ni en los aspectos, sino que se almacena en las estructuras de datos de la plataforma. Esto permite añadir información sobre nuevos componentes y aspectos, modificar los ya existentes, e incluso cambiar las reglas que rigen su composición sin necesidad de modificar la implementación de los mismos. Otra característica importante de la plataforma CoopTEL es la utilización, en tiempo de ejecución, de esta información para llevar a cabo la composición dinámica de los componentes y aspectos. Esto la hace especialmente adecuada para desarrollar sobre ella aplicaciones distribuidas como los EVCs, con grandes necesidades de adaptabilidad y extensibilidad.

Este artículo se organiza de la siguiente forma. En la sección 2 describimos las características más importantes de la plataforma CoopTEL, detallando las posibilidades de adaptación que ofrece. Las ventajas de utilizar la plataforma CoopTEL en el desarrollo de aplicaciones de EVCs se muestra en la sección 4.

2. Plataforma CoopTEL

CoopTEL está basado en un nuevo modelo en el que los componentes y los aspectos son entidades independientes que existen en tiempo de ejecución. Los componentes interaccionan intercambiando mensajes y emitiendo eventos. Los aspectos pueden aplicarse a los mensajes (de entrada o de salida), a los eventos, y a los métodos de creación y finalización de componentes. Una característica relevante de CoopTEL es que los componentes y los aspectos no codifican referencias directas entre ellos. En su lugar se identifican dentro de la plataforma por un nombre de rol único, que describe el papel que desempeña ese componente o aspecto en la aplicación. Dicho rol se asigna como parte de la descripción de los componentes y los aspectos en el fichero de despliegue. En esta sección describimos los servicios más importantes ofrecidos por la plataforma CoopTEL (las interfaces que aparecen sobre la clase *CoopTEL Platform* en la Figura 1), y la infraestructura que soporta la implementación de estos servicios (las clases por debajo de la clase *CoopTEL Platform* en la misma Figura 1).

2.1. Servicios de CoopTEL

De forma similar a otras plataformas de componentes, CoopTEL proporciona un conjunto de servicios requeridos por la mayoría de las aplicaciones distribuidas, como son la instanciación de componentes y aspectos, el envío y multidifusión de mensajes, la gestión de eventos, la evaluación de aspectos, y la persistencia de datos. Además de estos servicios básicos, CoopTEL ofrece un servicio de configuración de la arquitectura de la aplicación en tiempo de ejecución para conseguir adaptabilidad dinámica.

Instanciación de Componentes y Aspectos. Las interfaces *ComponentFactory* y *AspectFactory* de la Figura 1 permiten la creación de componentes y aspectos. La característica más relevante de este servicio es que para crear un nuevo componente o aspecto sólo es necesario proporcionar su nombre de rol y no su clase de implementación. La plataforma se encargará de resolver que componente o aspecto proporciona ese rol y de instanciar la clase que lo implementa. Esta característica ofrece una clara ventaja de cara a la adaptabilidad como veremos en la siguiente sección.

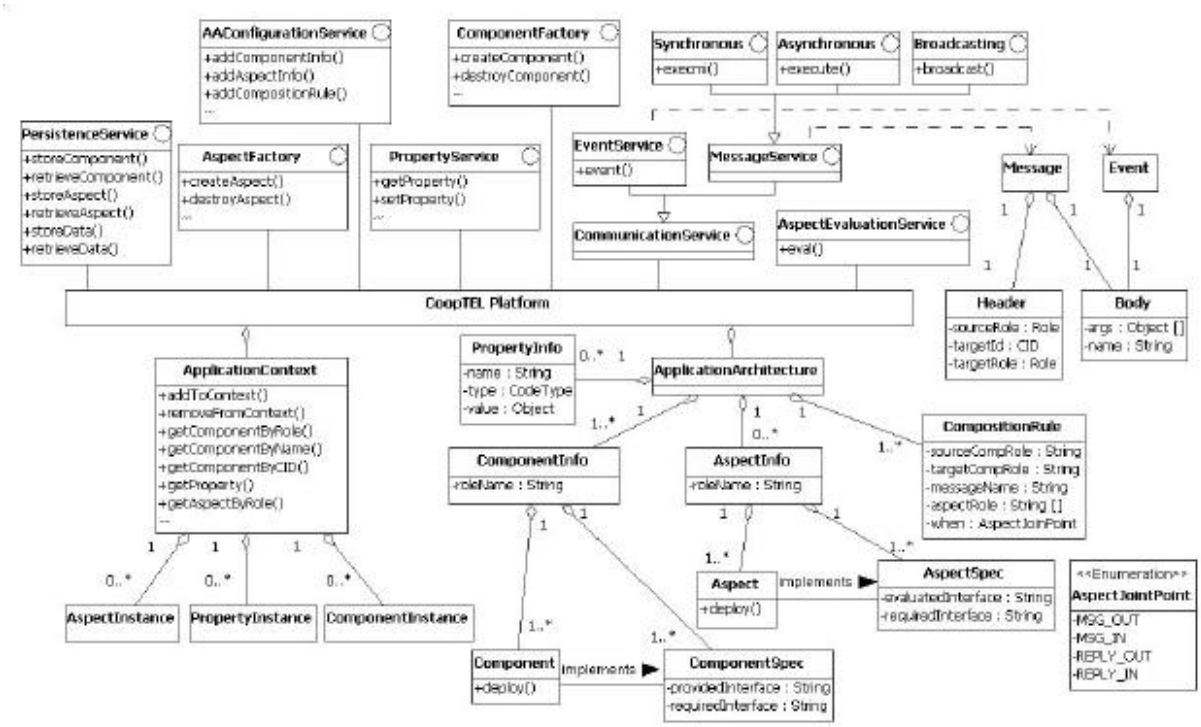

Figura 1. Servicios e Infraestructura de la Plataforma CoopTEL.

Comunicación entre Componentes. La interfaz *CommunicationService* de la Figura 1 proporciona cuatro primitivas para la comunicación entre componentes: mediante el envío de mensajes asíncronos (*execute()*); síncronos (*execmi()*); la difusión de mensajes (*broadcast()*); *y* la transmisión de eventos (*event()*). En las primitivas para el envío de mensajes el componente destino se identifica por su nombre de rol, haciendo a los componentes independientes del resto de componentes con los que interaccionan. De nuevo, la plataforma será la encargada de resolver que instancia(s) de los componente(s) destino debe(n) recibir el mensaje. En el caso de los eventos, el componente destino se omite, siendo un aspecto de coordinación [Pinto02b] el encargado de manejarlos según la información interna acerca de quien(es) debe(n) gestionarlo.

Evaluación de Aspectos. En CoopTEL el mecanismo de composición de componentes se extiende para incorporar la evaluación dinámica de aspectos, de forma que cuando un componente crea o finaliza otro componente, o envía un mensaje o un evento utilizando alguna de las primitivas de comunicación anteriores, este servicio lo intercepta y se encarga de evaluar los aspectos correspondientes. El proceso de evaluación viene determinado por una serie de reglas que indican cuándo y qué aspectos deben ser evaluados. Para proporcionar un mayor grado de adaptabilidad, estas reglas no están codificadas en los componentes ni en los aspectos, sino que se definen como parte de la arquitectura de la aplicación. Esta característica nos diferencia de otros trabajos como AspectJ [Kickzales01], el lenguaje de aspectos más utilizado. Para que se lleve a cabo la evaluación, los aspectos deben implementar la interfaz *AspectEvaluationService*, ya que esta interfaz define el método *eval()*, que será invocado por la plataforma para iniciar la evaluación del aspecto.

Servicio de Persistencia. En la implementación actual de CoopTEL se utiliza el servidor de directorios LDAP [LDAP] como almacén de datos persistentes. En el caso de los EVCs, los componentes y los aspectos pueden utilizar este servicio (interfaz *PersistenceService*) para almacenar y recuperar su información de configuración y de estado.

Configuración de la Arquitectura de la Aplicación. La interfaz *AAConfigurationService* proporciona un conjunto de métodos para modificar la arquitectura de la aplicación en tiempo de ejecución. Es posible añadir o modificar la descripción de componentes y aspectos, así como sus reglas de composición. Este es el servicio

utilizado para adaptar los EVCs de forma dinámica tal y como se describe en posteriores secciones.

2.2. Infraestructura de CoopTEL

A continuación explicamos la estructura interna de CoopTEL, haciendo especial hincapié en el tipo de información que almacena la plataforma sobre la arquitectura de la aplicación.
Básicamente, CoopTEL organiza su información sobre la arquitectura en dos objetos. El objeto *ApplicationArchitecture* que almacena la descripción arquitectónica de los componentes y aspectos en la aplicación; y el objeto *ApplicationContext* que mantiene la lista de referencias a los componentes y aspectos instanciados en un momento determinado.

```
<componentDescription>
  <component role="c1">
    <interfaces>...</interfaces>
    <implementations>
        <implementation name="impl1">
                java c1Impl1.class
        </implementation>
        <implementation name="impl2">
                java c1Impl2.class
        </implementation>
    </implementations>
  </component>
  <component role="c2">...</component>
  ...
</componentDescription>

<aspectDescription>
  <aspect role="a1">
    <interfaces>...</interfaces>
    <implementations>...</implementations>
  </aspect>
  <aspect role="a2">...</aspect>
  ...
</aspectDescription>

<compositionConstraints>
   <componentCompositionRules>
     ...
   </componentCompositionRules>

   <aspectEvaluationRules>
        <aspectRule>
          <compRole>c1</compRole>
          <inputAspects>
              <messages>m1</messages>
              <aspectList>a1 a2</aspectList>
          </inputAspects>
          <outputAspects>
              <messages>m2</messages>
              <aspectList>a2</aspectList>
          </outputAspects>
        </aspectRule>
        ...
   </aspectEvaluationRules>
</compositionConstraints>
```

Figura 2. Descripción en XML de la AA

Arquitectura de la Aplicación (AA). Como se mencionó en la introducción y se muestra en la Figura 2, la AA describe la información relativa a los componentes y aspectos que constituyen la aplicación, y las reglas que determinan su composición y evaluación, respectivamente. La AA se define durante la fase de diseño [Pinto03] y se almacena en un almacén de datos, que en la implementación actual de CoopTEL es un servidor de directorios LDAP. Cuando un usuario se conecta a una aplicación de EVC, la información sobre la arquitectura se descarga y se almacena en el objeto *ApplicationArchitecture* (Figura 1) y en las clases que lo componen. La plataforma utiliza esta información para proporcionar los servicios de instanciación de componentes y aspectos, comunicación de componentes, y evaluación de aspectos descritos anteriormente, proporcionando la adaptabilidad y extensibilidad requerida por las aplicaciones desarrolladas sobre CoopTEL.

2.3. Adaptabilidad en CoopTEL

Aunque la descomposición de la funcionalidad de un sistema en componentes y aspectos incrementa su reutilización y extensibilidad, CoopTEL proporciona un mayor grado de adaptabilidad debido principalmente a la definición explícita de la arquitectura de la aplicación (
Figura 2) y a su utilización en tiempo de ejecución.

Por un lado, los componentes y aspectos son reutilizables en distintas aplicaciones ya que no mantienen referencias directas entre sí. En su lugar utilizan el nombre de rol como identificador, como se explicó anteriormente (ej: nombres de rol “c1”, “c2”, “a1” y “a2” en la
Figura 2). Además, la información sobre qué aspectos son aplicados y cuándo son aplicados no está codificada como parte ni de los componentes ni de los aspectos. En su lugar se describe mediante en las reglas de composición de la
Figura 2 (elemento XML *compositionConstraints*). Asimismo, la composición de componentes y la evaluación de aspectos no se lleva a cabo hasta la ejecución.

Por tanto, para adaptar el comportamiento de una aplicación sólo es necesario modificar la información sobre su arquitectura de forma adecuada. Podemos adaptar la aplicación de forma estática modificando directamente el fichero XML que describe su arquitectura. De esta forma conseguimos describir nuevas aplicaciones modificando simplemente un fichero de texto. Más interesante aún es adaptar la aplicación de forma dinámica. Una vez que se inicia la ejecución de la aplicación y que la información sobre la AA ha sido cargada en las estructuras de la plataforma, ésta se puede modificar utilizando el servicio de configuración de la AA descrito en la sección anterior. Los cambios en la información sobre la AA realizados de forma dinámica, se hacen persistentes

actualizando el fichero XML original. Tanto si se modifica la arquitectura de forma estática como dinámica, CoopTEL proporciona las herramientas necesarias [Pinto03] para comprobar que la información proporcionada es correcta.

La adaptabilidad proporcionada por CoopTEL permite entre otras cosas:

- Incorporar nuevos componentes y aspectos no contemplados durante el desarrollo de la aplicación. Para ello basta determinar el nombre de rol del nuevo componente o aspecto, cuales son los mensajes y/o eventos que recibe y envía, etc., añadiendo esta información al fichero XML de la
- Figura 2, mediante un nuevo elemento de tipo *component* o *aspect*.
- Reemplazar componentes y aspectos ya existentes por nuevas versiones, añadiendo o modificando la información sobre sus clases de implementación (elemento *implementations* dentro de la descripción del componente o aspecto).
- Modificar las reglas que determinan la composición de componentes y aspectos. Para modificar el número y tipo de aspectos que son aplicados a un componente, será suficiente añadir, eliminar o modificar la información definida en el elemento *aspectEvaluationRule* de la
- Figura 2. En el ejemplo mostrado en la
- Figura 2, la plataforma CoopTEL evaluará los aspectos con nombre de rol "a1" y "a2" antes de que el componente con nombre de rol "c1" reciba el mensaje "m1". Evaluará el aspecto con nombre de rol "a2" después de que el componente reciba y ejecute el código asociado al mensaje "m2". Es importante resaltar que estas reglas se describen en base a los nombre de rol de los componentes y los aspectos, por lo que los cambios realizados en la descripción de los mismos no afectará a su composición.

En la siguiente sección se mostrará, sobre un ejemplo, como una aplicación de EVC saca partido de la adaptabilidad proporcionada por CoopTEL.

3. Adaptabilidad aplicada a Entornos Colaborativos

Hoy en día el desarrollo de software se caracteriza por la necesidad de desarrollar de forma rápida y fiable aplicaciones complejas y especializadas en campos concretos. En muchas ocasiones los equipos de desarrollo no disponen de los conocimientos técnicos o del tiempo necesario para implementar una solución completa partiendo de cero. Debido a este problema, en los últimos años se ha tendido al uso cada vez más generalizado de los MTs que son diseñados específicamente para cubrir estas carencias, aumentando la productividad y reutilización del software desarrollado.

Aunque hoy en día existen diversos MTs para el desarrollo de aplicaciones de trabajo colaborativo (CSCW) o EVCs, como por ejemplo Sametime [Sametime], estos no suelen proporcionar una implementación completa de su entorno colaborativo, por lo que se ha de desarrollar este entorno independientemente del MT. Además, estos MTs no suelen permitir la integración de aplicaciones desarrolladas por terceros a su entorno de desarrollo. Por esta razón y para dar solución a este tipo de problemas, hemos desarrollado un MT para la construcción de aplicaciones de EVCs [Amor03] que se ejecutan sobre la plataforma CoopTEL (Figura 3). Las aplicaciones desarrolladas utilizando nuestro MT son muy adaptables y flexibles, pudiendo ser totalmente personalizadas para cada usuario. Esto permite que los usuarios puedan modificar su funcionamiento y utilizarlas de maneras en las que no fueron originalmente pensadas por los desarrolladores. En las siguientes secciones mostraremos algunas de estas propiedades al ir desarrollando los ejemplos.

3.1 Arquitectura del MT de Entornos Virtuales Colaborativos

Para demostrar la adaptabilidad y flexibilidad de CoopTEL utilizaremos nuestro MT de EVCs para desarrollar una aplicación de Aula Virtual (AV). Este MT proporciona el soporte necesario para el desarrollo de nuevos tipos de EVCs y de aplicaciones individuales o colaborativas, fomentándose la reutilización de los aspectos y componentes desarrollados. La Figura 3, muestra un diagrama de clases UML de nuestro MT para EVCs (para obtener más detalles sobre el MT véase [Amor03]).

Los EVCs tratan de resolver el problema de la integración de todos los recursos -usuarios, documentos, aplicaciones colaborativas, etc.- necesarios para colaborar. Para ello, proporcionan un entorno compartido (componente *Entorno* en la Figura 3) donde todos los recursos de la aplicación sean fácilmente accesibles. Este entorno compartido se organiza en espacios de colaboración (componente *EspacioColaboración* en la Figura 3).

Los usuarios conectados al entorno pueden navegar entre los distintos espacios de colaboración interactuando con los usuarios y recursos

(componente *Recurso* en la Figura 3) que en él se encuentren.

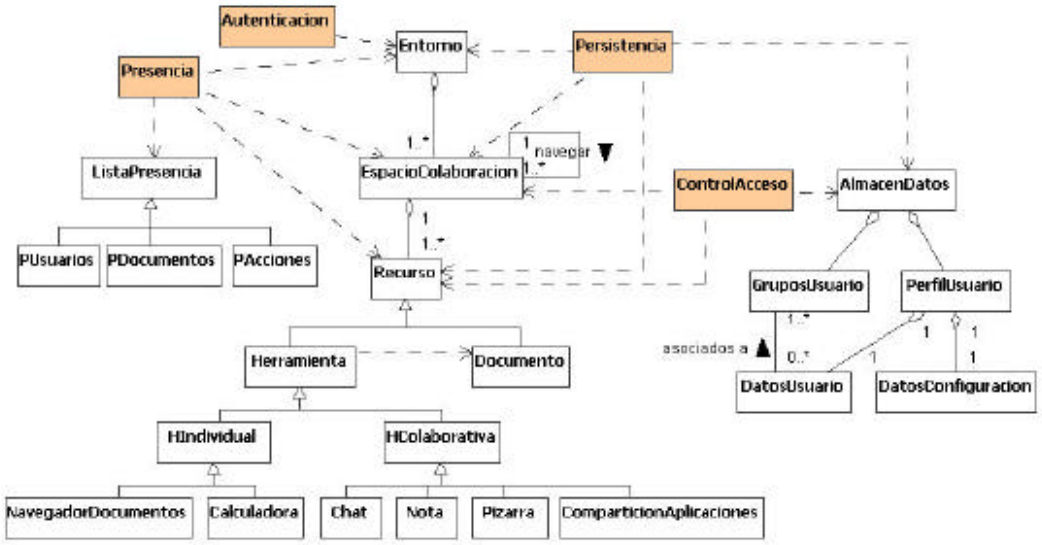

Figura 3. Marco de Trabajo para derivar EVCs

En la aplicación de AV que hemos desarrollado sobre este MT, estos espacios de colaboración modelan, por ejemplo, las distintas aulas de un centro virtual.Todos los alumnos y el profesor de una determinada asignatura se conectarán al espacio de colaboración asignado a dicha asignatura para colaborar. Los recursos que encontramos en un espacio de colaboración se clasifican en dos tipos. Por una parte los documentos (*Documento*) y por otra las herramientas (*Herramienta*). Los primeros modelan un documento compartido del MT. Las segundos representan las aplicaciones dentro del MT y se dividen a su vez en dos tipos: colaborativas (*Hcolaborativa*),, como por ejemplo una aplicación de pizarra compartida, un chat, un sistema de notificación de mensajes, una utilidad de compartición de aplicaciones o un sistema de videoconferencia; y no colaborativas (*Hindividual*), por ejemplo una calculadora o un navegador de documentos web.

Además, el MT modela como aspectos una serie de propiedades comunes a todos los EVCs como son: la *autenticación* de usuarios; la información de *presencia* de los mismos; la *persistencia* de datos; y el *control de acceso* a los recursos. Las relaciones de dependencia en la Figura 3 muestran como estos aspectos interaccionan con distintos componentes del MT, lo que justifica su separación en entidades independientes.

En esta sección, pretendemos demostrar la extensibilidad y adaptabilidad que nuestra plataforma proporciona en el desarrollo de aplicaciones de EVCs. Para lograrlo mostraremos un ejemplo de cómo crear una nueva aplicación no colaborativa e incorporarla al AV y como modificar el comportamiento básico de esta mediante los aspectos. Finalmente, veremos como es posible convertirla en una aplicación colaborativa sólo mediante la incorporación de aspectos.

3.2 Ejemplo de aplicación: Agenda personal

Dado que inicialmente nuestra AV no tenía la utilidad de agenda personal, decidimos incorporarla y ver como la plataforma soportaba este cambio. La agenda personal será una aplicación disponible en los despachos de los profesores permitiéndoles planificar cuando llevar a cabo sus clases mediante videoconferencia y otras actividades en el AV. Como se ha dicho anteriormente, nuestro modelo de componentes y aspectos y los servicios ofrecidos por nuestra plataforma facilitan no sólo la incorporación de nuevos componentes y aspectos al MT para utilizarlos en el desarrollo de nuevas aplicaciones, sino también la incorporación de los mismos a aplicaciones en ejecución, sin necesidad de detenerlas o recompilarlas.

```
<component role="Agenda">
  <interfaces>...</interfaces>
    <implementations>
      <implementation name="impl1">
        java CAgenda1.class
      </implementation>
    </implementations>
</component>
```

Figura 4. Definición del componente Agenda.

3.2.1 Incorporación de un nuevo componente

Para añadir un nuevo componente o aspecto al AV, sólo hemos de añadir información sobre los mismos a la AA utilizando los servicios que la plataforma ofrece. La Figura 4 muestra la definición en XML del componente *Agenda* que añadiremos a la información de configuración de la AA para incorporar el nuevo componente a la aplicación. En nuestro caso, el componente debe almacenar entradas del usuario y recuperarlas posteriormente. Analizando su funcionamiento vemos que podemos modelar estos procesos mediante un aspecto de persistencia. Este aspecto ya existe en el MT como muestra la Figura 3, y por tanto no tenemos que desarrollar uno nuevo.

```
<aspect role="Persistencia">
 <interfaces>...</interfaces>
  <implementations>
   <implementation name="impl1">
     java APersistencia1.class
   </implementation>
 </implementations>
</aspect>
```

Figura 5. Definición del aspecto de Persistencia.

La Figura 5 muestra su definición. Pasaremos ahora a describir el proceso de instanciación de un componente de tipo *Agenda* en el AV. Antes de instanciarse el componente *Agenda* (Figura 6), se ejecutará el aspecto de persistencia (paso 1) que recuperará las entradas de la agenda y si no ocurre ningún problema (paso 2), inicializará el contenido de la misma a partir de los datos recuperados (paso 3).

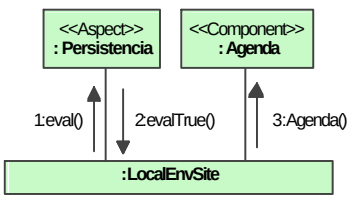

Figura 6. Componente Agenda básico.

Cada vez que se realicen cambios en una entrada de la agenda (véase Figura 7), el componente lanzará un mensaje "*modifyEntry*" (paso 1). Este mensaje es interceptado por el aspecto de persistencia (paso 2) que se encarga de almacenar el nuevo estado del componente en un almacén de datos. Finalmente si no ocurre ningún error (paso 3), el componente ejecuta el método que modifica la entrada en su representación gráfica (paso 4). Para llevar a cabo estos cambios, sólo necesitamos dar de alta el nuevo componente y asignarle los aspectos adecuados como se indicó en la sección anterior. De esta forma, el nuevo componente será dado de alta como una nueva aplicación del AV pudiendo comenzar a utilizarse sin problemas. La Figura 8 muestra las reglas de composición añadidas a la AA.

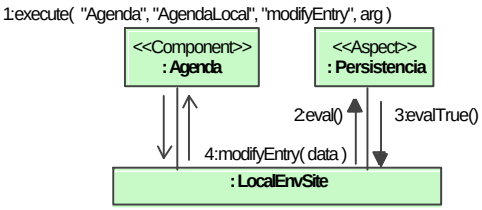

Figura 7. Modificando una entrada de la agenda.

```
<aspectRule>
 <compRole>Agenda</compRole>
 <inputAspects>
  <messages>modifyEntry</messages>
  <aspectList>Persistencia</aspectList>
 </inputAspects>
</aspectRule>
```

Figura 8. Reglas de composición.

3.2.2 Modificación de la implementación de un aspecto

La AA permite la definición de diferentes implementaciones del mismo aspecto (ver Figura 2). Esta característica nos permite seleccionar que implementación debe utilizar la aplicación.

A continuación, mostraremos un ejemplo de cómo utilizar este mecanismo para reemplazar el aspecto de persistencia por una nueva versión. Como se indicó, este aspecto se encarga de almacenar y recuperar las entradas de la agenda. Su implementación actual en el AV utiliza un servidor de directorios LDAP [LDAP], pero también podría implementarse mediante otros sistemas, como por ejemplo un sistema de ficheros remotos como NTFS, una base de datos, etc. Así dispondremos de diferentes sistemas de almacenamiento dependiendo de la implementación del aspecto elegido sin que esta elección afecte a la funcionalidad del resto de la aplicación.

Supongamos que nos interesa cambiar la implementación actual de este aspecto haciendo persistentes las entradas de la agenda en una base de datos Oracle. En primer lugar debemos desarrollar la nueva implementación del aspecto que almacene y recupere las entradas de una base de datos Oracle. Después simplemente es necesario añadir la nueva implementación a la definición del aspecto de persistencia incluida en la AA. Esta implementación debe ser capaz de gestionar los mismos mensajes que la implementación original.

Finalmente debemos indicar en la AA que ésta es la nueva implementación del aspecto que sustituye a la anterior. Es importante resaltar que todos estos cambios se realizan en tiempo de ejecución, y que hemos logrado modificar el funcionamiento de la aplicación sin tener que modificar el componente, ya que sólo hemos cambiado la implementación del aspecto, dejando el resto de la AA intacta.

3.3 Modificación del comportamiento utilizando aspectos

Con el ejemplo de la sección anterior, hemos demostrado la flexibilidad del AV a la hora de adaptarse a cambios en los requisitos del sistema seleccionando la implementación del aspecto más adecuado a nuestras necesidades. Ahora daremos un paso más y mostraremos como es posible modificar el comportamiento del componente mediante el uso de aspectos.

Supongamos que los profesores desean notificar a sus alumnos cuando se llevarán a cabo las clases virtuales mediante videoconferencia. La tarea de enviar una notificación a cada uno de los alumnos del AV no estaba dentro de las capacidades originales de la agenda. Además, realizar esta tarea a mano resulta tedioso y es fácil cometer errores. Ahora bien, el AV dispone de un componente encargado de gestionar mensajes de correo electrónico a listas de usuarios denominado *ListaCorreo*. Mostraremos a continuación como es posible combinar estos dos componentes mediante un aspecto para obtener la funcionalidad deseada. En primer lugar crearemos un nuevo aspecto denominado *Notificación*, que será aplicado a los mensajes emitidos por el componente *Agenda* tras el aspecto de persistencia. Este aspecto capturará los mensajes emitidos por el componente y en el caso de ser una videoconferencia enviará un mensaje al componente de notificación. Una vez implementado el aspecto de notificación, añadimos su definición a la AA y modificamos las reglas de evaluación para indicar la aplicación del nuevo aspecto, como se muestra en la Figura 9.

```
<aspectRule>
 <compRole>Agenda</compRole>
 <outputAspects>
  <messages>modifyEntry</messages>
  <aspectList>
   Persistencia Notificación
  </aspectList>
 </outputAspects>
</aspectRule>
```

Figura 9. Reglas modificadas.

Tras añadir la información indicada, el conjunto de interacciones del componente cambiará al mostrado en la Figura 10, donde se puede ver que tras emitir el componente *Agenda* un mensaje comunicando un cambio en una entrada (paso 1), el aspecto de persistencia almacena el nuevo estado (paso 2) y si no ocurre ningún problema (paso 3) se aplicará el aspecto de notificación (paso 4). Este aspecto determina si la entrada modificada es del tipo videoconferencia y en caso afirmativo (paso 5) envía un mensaje *"mailTo"* (paso 6) que será recibido por el componente *ListaCorreo* (paso 7). Finalmente la agenda actualiza su contenido mostrándolo al profesor (paso 8).

La solución tradicional a este problema hubiera sido modificar el código del componente para añadirle esta funcionalidad o bien crear una nueva implementación del componente *Agenda* que incluyera esta opción, y sustituir el componente antiguo, siendo necesario además reiniciar todo el sistema para que los cambios surtieran efecto. En nuestra propuesta los cambios son registrados en la arquitectura de la aplicación en tiempo real y no necesitamos detener la aplicación para que el componente cambie su comportamiento. Además, este aspecto queda disponible para su uso por otras aplicaciones.

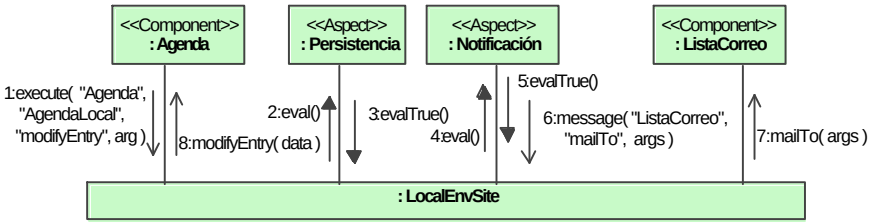

Figura 10. Aplicando aspecto de notificación.

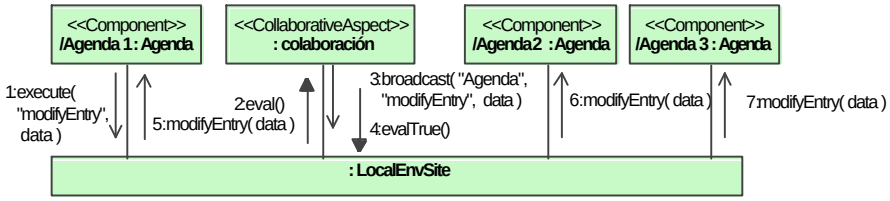

Figura 11. Aspecto de colaboración.

3.4 Conversión de una aplicación no colaborativa en colaborativa

Supongamos ahora que se desea añadir una nueva funcionalidad al aula virtual, de forma que los profesores puedan coordinarse para realizar la reserva de aulas, la organización conjunta de videoconferencias, etc. Para llevar a cabo esta coordinación, sería útil utilizar una agenda colaborativa, en la que la planificación realizada por cada profesor pueda visualizarla el resto. Dado que el componente *Agenda* incorporado anteriormente era de uso individual por cada profesor, tenemos dos opciones. Implementar e incorporar un nuevo componente de *Agenda* colaborativa, o convertir el componente *Agenda* que ya tenemos en uno colaborativo. En CoopTEL esto es posible gracias a la utilización de aspectos. En concreto, haremos uso del aspecto de colaboración ya definido en el MT cuya función es la de coordinar todas las instancias de un componente colaborativo notificando los cambios que se produzcan en su estado.

En las secciones anteriores, cada usuario instanciaba un componente *Agenda* y estas instancias no tenían ninguna relación entre sí. Para hacerlas colaborativas, consideramos que todas son instancias locales de la misma agenda y utilizaremos un aspecto de colaboración para difundir de forma transparente los mensajes generados por una instancia al resto.
En el caso de la agenda colaborativa, el aspecto de colaboración se encargará de notificar los cambios cuando los usuarios añadan, modifiquen o eliminen entradas. La Figura 11 muestra como un componente *Agenda* emite un mensaje de modificación de una entrada (paso 1) y este mensaje es interceptado por el aspecto de colaboración (paso 2), que lo notifica al resto de instancias del componente *Agenda* del AV (pasos 3, 4, 5, 6 y 7). Por último, para que el cambio de comportamiento del componente sea efectivo hemos de registrar en la AA los cambios realizados, de la misma forma que se ha hecho en las secciones anteriores. A partir de ese momento, el componente *Agenda* pasará a ser colaborativo. Realizando esta sencilla actualización ya hemos transformado el componente *Agenda* en colaborativo. Sin embargo, en el caso de la agenda debemos hacer algunas consideraciones adicionales, ya que ahora todas las entradas de la agenda son accesibles por cualquier usuario, por lo que sería necesario limitar de algún modo el acceso a dichas entradas mediante permisos de acceso.

```
<aspectRule>
 <compRole>Agenda</compRole>
 <inputAspects>
  <messages>modifyEntry</messages>
  <aspectList>
   ControlAcceso Persistencia Colaboración
  </aspectList>
 </inputAspects>
</aspectRule>
```

Figura 12. Configuración final de la AA.

Esto se consigue fácilmente mediante la aplicación de un aspecto de control de acceso, el cual ya está definido en el MT (*ControlAcceso)*. A partir de este momento, antes de crear, modificar o borrar cualquiera de las entradas de la agenda, este aspecto se encargará de asegurar que el usuario posee los permisos necesarios para realizar esa operación. La

nueva configuración de la AA se muestra en la Figura 12.

Pasaremos ahora a mostrar el comportamiento de la aplicación con el aspecto de control de acceso. Supongamos que un usuario modifica una entrada de la agenda (ver Figura 13). El mensaje es interceptado por el aspecto de control de acceso *ControlAcceso* (paso 2) que determina si el usuario posee los permisos necesarios para modificar la entrada, en caso negativo se notifica al usuario que no puede modificar la entrada y no continua la evaluación del resto de aspectos. En otro caso (paso 3), el proceso de modificación de la misma continuará, aplicándose el aspecto de persistencia (paso 4) y el de colaboración (paso 6) como se mostró anteriormente.

En resumen, tras realizar todos estos cambios se ha modificado el comportamiento de la aplicación, hemos reutilizado todos los componentes y aspectos iniciales sin modificarlos y se ha hecho sin detener la ejecución de la aplicación.

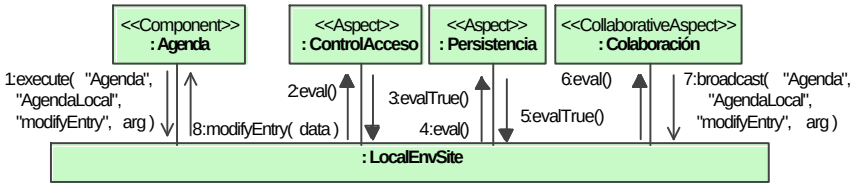

Figura 13. Modificando una entrada en la agenda colaborativa.

4. Conclusiones

Este artículo propone la utilización de la plataforma CoopTEL para el desarrollo de EVCs. CoopTEL proporciona un conjunto de servicios que facilitan la adaptación de los EVCs con un mínimo esfuerzo. Simplemente modificando el documento XML que contiene la descripción de la AA, se puede cambiar su comportamiento tanto estática como dinámicamente. Tomando como ejemplo un AV se ha mostrado la potencia y grado de adaptabilidad conseguido con nuestra propuesta.

Hemos logrado mostrar en primer lugar que CoopTEL permite modificar el comportamiento de cualquier componente mediante la aplicación de aspectos sin modificar el código del componente. En segundo lugar que estos cambios se pueden realizar en tiempo real sin tener que reiniciar la aplicación y por último que es posible convertir con facilidad cualquier componente no colaborativo en colaborativo mediante la aplicación de un aspecto de colaboración sobre el mismo. Estas tres características reflejan la flexibilidad y adaptabilidad del marco de trabajo basado en nuestra plataforma CoopTEL.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia y Tecnología bajo el proyecto TIC2002-04309-C02-02

Referencias

[Amor03] M. Amor, L. Fuentes, D. Jiménez, M. Pinto. “Marco de Trabajo de Componentes y Aspectos para el desarrollo de Entornos Virtuales Colaborativos”. Próxima publicación en las actas de las Jornadas JITEL 2003, Gran Canarias, Septiembre 2003.

[Brown98] A.W. Brown, K.C. Wallnau. “The Current State of CBSE”, IEEE Software, Septiembre/Octubre 1998.

[WebDSOA] Aspect-Oriented Software Development Web Site. http://www.aosd.net.

[Gutwin00] C. Gutwin, S. Greenberg. “The mechanics of collaboration: developing low cost usability evaluation methods for shared workspaces”, Actas de IEEE 9th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.

[Kickzales97] G. Kickzales y otros, “Aspect-Oriented Programming”, Actas de ECOOP’97, Junio 1997.

[Kickzales01] G. Kickzales y otros, “An Overview of AspectJ”, Actas de ECOOP’01, Junio 2001.

[LDAP] LDAP protocol RFC-1777. http://www.ietf.org/rfc/rfc1777.txt

[Pinto02a] M. Pinto, L. Fuentes, J. M. Troya, Plataforma para la Composición Dinámica de Componentes y Aspectos. Actas de JISBD’02, Noviembre 2002.

[Pinto02b] M. Pinto, L. Fuentes, M. Fayad, J. M. Troya, Separation of Coordination in a Dynamic Aspect-Oriented Framework. Actas de AOSD'02, Holanda, Abril 2002.

[Pinto03] M. Pinto, L. Fuentes, J.M. Troya, DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development, Actas de la International Conference on Generative Programming and Component Engineering (GPCE'03), Septiembre 2003.

[Sakai02] S. Sakai, y otros. "An integrated distance learning system capable of supporting interactions for asynchronous distance learning", Actas del 22 International Conference on Distributed Computing Systems, 2002.

[Sametime] Real-time collaboration with Sametime. ftp://ftp.lotus.com/pub/lotusweb/product/sametime/Real-timeCollab.pdf