Desarrollo de Software Orientado a Aspectos, DSOA 2006
Asociado a XV Jornadas de Ingeniería del Software y Bases de Datos
J. Araújo, J. Hernández, E. Navarro y M. Pinto (Eds)
Sitges (Barcelona), Octubre - 2006

DAOPxADL: UNA EXTENSIÓN DEL LENGUAJE DE DESCRIPCIÓN DE ARQUITECTURAS xADL CON ASPECTOS *

Lidia Fuentes, Nadia Gámez y Mónica Pinto

Departamento de Lenguajes y Ciencias de la Computación
E.T.S.I. Informática
Universidad de Málaga
Boulevard Louis Pasteur 35, Campus de Teatinos, Málaga (España)
{lff,nadia,pinto}@lcc.uma.es, http://caosd.lcc.uma.es/index.htm

Palabras clave: Arquitectura del Software, LDA, DSOA.

Resumen. *La Arquitectura del Software promulga la importancia de especificar la estructura de un sistema, fundamentalmente utilizando los Lenguajes de Descripción de Arquitecturas (LDAs). El Desarrollo de Software Orientado a Aspectos aboga por identificar "aspectos" entremezclados en varios componentes en la fase de diseño arquitectónico. En este artículo se presenta DAOPxADL, una extensión con aspectos de un LDA de propósito general, xADL.*

1. INTRODUCCIÓN

La Arquitectura del Software (AS) es una disciplina que ayuda a comprender y mantener más fácilmente los sistemas software. La AS estudia la estructura de una aplicación, formada por un conjunto de componentes, sus propiedades visibles y las relaciones entre ellos [1]. Para describir la arquitectura de un sistema es habitual utilizar los Lenguajes de Descripción de Arquitecturas (LDAs), que permiten la descripción, análisis y reutilización de arquitecturas software [2]. Mediante un LDA, la arquitectura software de un sistema se define como un conjunto de componentes, conectores y las conexiones entre ellos.

Por otra parte, el Desarrollo de Software Orientado a Aspectos (DSOA) propugna la separación de aspectos en todas las fases del ciclo de vida del software, desde requisitos y diseño arquitectónico (*early aspects*) hasta implementación. Sin embargo, los LDAs tradicionales, aunque separan eficientemente la computación de la comunicación, fallan a la hora de proporcionar el soporte necesario para separar de forma adecuada cualquier comportamiento que atraviese varios elementos del sistema (*crosscutting concerns*). Se necesita disponer, por tanto, de lenguajes y herramientas para expresar los aspectos en la etapa

* Este trabajo ha sido financiado por los proyectos IST-2-004349-NOE AOSD-Europe y MCYT TIN2005-09405-C02-01

de diseño arquitectónico.
Una tendencia es la definición de lenguajes que usan esquemas XML para describir los elementos del lenguaje. Dentro de estos últimos se encuentra xADL 2.0 [3], un LDA de propósito general diseñado para ser usado por sí mismo, así como para ser extendido soportando nuevas aplicaciones y dominios [4]. Este LDA es útil para especificar un sistema con componentes y conectores clásicos, pero no incorpora el concepto de aspecto. Hay otros LDAs que sí proporcionan elementos para abordar la separación avanzada de conceptos, como DAOP-ADL [5] o PRISMA [6]. Sin embargo, estos LDAs exhiben una estructura diferente a los LDAs tradicionales y su soporte de herramientas es limitado. Sería deseable disponer de un LDA con aspectos con la misma estructura que los LDAs tradicionales, como AO-Rapide [7], para definir y reutilizar arquitecturas con o sin aspectos sin necesidad de cambiar de lenguaje.
En este trabajo se presenta una extensión de xADL, llamada DAOPxADL, a la que se le ha incorporado el concepto de aspecto. En la definición de DAOPxADL nos hemos apoyado en las nociones del LDA basado en componentes y aspectos DAOP-ADL.
Este artículo se estructura como sigue: en la sección 2 se exponen los LDAs escogidos para este trabajo, en la sección 3 se presenta la propuesta de extensión de xADL para añadirle aspectos, y en la sección 4 se comentan algunas consideraciones finales y trabajos futuros.

2. LENGUAJES DE DESCRIPCIÓN DE ARQUITECTURAS

En nuestra propuesta hemos escogido xADL, un lenguaje de propósito general, que usa esquemas XML, para extenderlo con los conceptos que nos proporciona DAOP-ADL. xADL 2.0 es muy adecuado para nuestro propósito ya que es altamente extensible y tiene un conjunto de herramientas que apoyan la creación y manipulación de documentos xADL. Su alta extensibilidad permite añadir fácilmente los conceptos de la orientación a aspectos. Para la extensión del lenguaje habrá que añadir nuevos esquemas XML con las nociones que queramos incorporarle.
En lugar de partir de cero para extender xADL añadiéndole aspectos, DAOPxADL se apoya en los conceptos ya definidos en DAOP-ADL, ya que al integrar el desarrollo de software basado en componentes y el orientado a aspectos, posee características muy útiles para nuestra propuesta. Otra ventaja de escoger DAOP-ADL es que está definido mediante esquemas XML, por lo que la extensión de xADL, a través de nuevos esquemas XML, será más sencilla.

2.1. xADL

Los elementos básicos que proporciona xADL son los componentes, los conectores, las interfaces y los enlaces (*links)* (Figura 1.a). Los componentes y conectores están definidos genéricamente, incluyen un identificador único y un conjunto de interfaces. Estas interfaces son los puertos (*ports*) de los componentes y conectores, y también son

genéricas, pues sólo especifican su dirección (indicador de si la interfaz es proporcionada, requerida o ambas). Los *links* son las conexiones entre los elementos del sistema. Para que las propiedades comunes de los elementos puedan ser especificadas una sola vez en lugar de definirlas en cada elemento, los componentes, conectores o interfaces pueden corresponder a un tipo que especifica dichas propiedades. Una descripción completa de xADL se encuentra en [3], de donde se ha tomado el ejemplo que se utilizará como caso de estudio en este artículo.

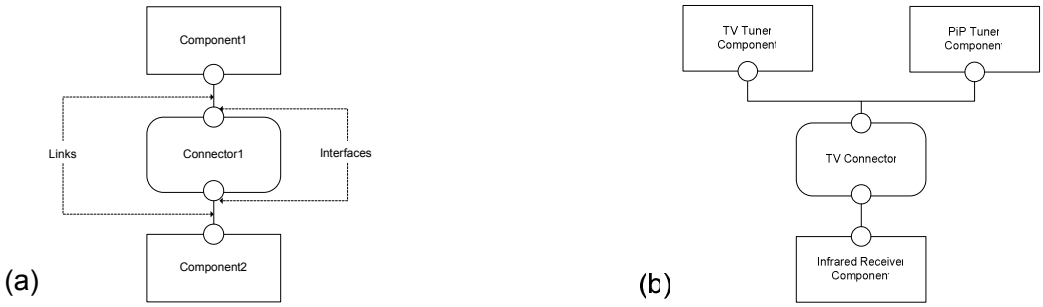


Figura 1. (a) Diagrama de relaciones entre elementos de xADL [4]. (b) Ejemplo del sistema de TV [3].

El ejemplo mostrado en la Figura 1.b es un sistema de televisión de características mínimas, que consta de un receptor de infrarrojos (*Infrared Receiver*), un sintonizador del canal de televisión (*TV Tuner*) y otro sintonizador *picture-in-picture* (*PiP Tuner*), para mostrar dos canales al mismo tiempo en la televisión. Estos componentes se conectan a través de un conector software. Los sintonizadores son del mismo tipo, por lo que tienen las mismas interfaces, una interfaz proporcionada que se enlaza (*link*) con la interfaz requerida del conector (*TV Connector*). Y la interfaz requerida del receptor de infrarrojos se enlaza con la proporcionada del mismo conector.

2.2. DAOP-ADL

DAOP-ADL es un lenguaje para especificar la arquitectura de un sistema orientado a aspectos [5]. Los elementos básicos de DAOP-ADL son: los componentes y aspectos que forman parte de la aplicación; las propiedades que utilizan (para separar accesos a datos compartidos por varios componentes); las restricciones de composición entre ellos; la información de despliegue que indica donde se instancia cada entidad; y el contexto inicial que define los componentes y las propiedades que se crearán al iniciar la aplicación [8].

DAOP-ADL identifica por su nombre de rol a cada componente y aspecto de una aplicación, y los describe en términos de sus interfaces, los atributos que definen su estado, las propiedades que usan y una lista de posibles implementaciones. Las interfaces que describen a un componente son la *interfaz proporcionada*, la *interfaz requerida* y la *interfaz de eventos* lanzados por el componente. La interfaz que describe a un aspecto es la *interfaz evaluada*, que define los mensajes, eventos, creaciones o destrucciones de componentes (*joinpoints*) que

es capaz de interceptar y cuándo. El cuándo o tipo de *advice* en DAOP-ADL indica si el aspecto se aplica antes o después del punto de intercepción. Un tipo especial de aspecto es el de coordinación, que incorpora un protocolo de coordinación para gestionar los eventos lanzados por los componentes.

Las restricciones de composición están formadas por las reglas de composición de componentes y las reglas de evaluación de aspectos, donde se define cómo y cuándo aplicar los aspectos. Las reglas de composición de componentes resuelven dos incompatibilidades. La primera puede surgir entre el nombre de rol que aparece en la interfaz requerida de un componente (*formalRole*) y el nombre de rol del componente destino (*realRole*). Si el rol real y el formal no tienen el mismo nombre habrá que definir una regla que ponga de manera explícita que el rol formal corresponde al real. La segunda incompatibilidad es entre las propiedades compartidas. Puede darse el caso de que dos nombres de propiedad distintos se refieran a una misma propiedad, entonces habrá que especificar que se refieren a la misma propiedad. O que haya dos propiedades con el mismo nombre pero con distinto significado, en cuyo caso habrá que renombrar una de ellas.

En DAOP-ADL las reglas de composición entre componentes y aspectos, que definen cómo y cuándo aplicar los aspectos a los componentes, se llaman reglas de evaluación de aspectos, y en lo sucesivo en este artículo también serán llamadas así. Una regla de evaluación tiene dos partes: el punto de corte capturado y la definición de los aspectos que se aplican cuando ese punto sea interceptado. Se define una regla por cada mensaje enviado, evento lanzado, componente creado o componente finalizado, que se quieran interceptar.

3. EXTENSIÓN DE xADL PARA ADAPTAR LAS NOCIONES DE DAOP-ADL

Para extender xADL, integrando los elementos y conceptos necesarios para adaptarlo a DAOP-ADL y convertirlo en un LDA orientado a aspectos (DAOPxADL), hay que añadir los elementos que existen en DAOP-ADL y no están incluidos en xADL. Para los elementos que tienen en común, se solventarán las diferencias que haya entre ellos, extendiendo los elementos de xADL. Como se indica en [9], los elementos de xADL pueden usarse o no dependiendo de las necesidades, por lo que los elementos de xADL que no se usen en DAOP-ADL se podrán omitir de las descripciones arquitectónicas en DAOPxADL.

La descripción de una arquitectura en DAOPxADL (*AppArchType* en Figura 2) consta de las estructuras de xADL (componentes y conectores con sus interfaces y los *links*) y sus tipos. En DAOP-ADL, además se consideran las propiedades y las reglas que resuelven las incompatibilidades entre ellas, la información de despliegue y el contexto inicial, por lo que todo esto se añade tal cual al nuevo lenguaje.

En la Tabla 1 se puede ver un resumen de las diferencias entre xADL y DAOP-ADL y cómo DAOPxADL las resuelve. Se ha creado *ComponentDAOPType* (Figura 3) como extensión del *ComponentType* de xADL, al igual que se hace en [9], para añadir los

atributos de estado y las propiedades a los componentes. El componente de xADL tiene un atributo identificador único, éste será usado como se usa el nombre de rol en DAOP-ADL. En el *ComponentType de xADL* no se incluyen la interfaces directamente si no que se apunta a un tipo de interfaz, mediante unos constructores llamados signaturas. Por tanto, no será necesario incluir en el *ComponentDAOPType* nada relativo a si la interfaz es proporcionada, requerida o de eventos. Aprovechando esta característica, los aspectos se modelarán como un componente de este tipo, ya que no habrá que indicar nada sobre su interfaz evaluada. Así transformamos un lenguaje asimétrico (DAOP-ADL) en uno simétrico, pues sólo se ha dejado un elemento arquitectónico básico de primer nivel, el componente. Esto tiene la ventaja de que cualquier componente podrá ser usado como aspecto dependiendo del contexto de su uso.

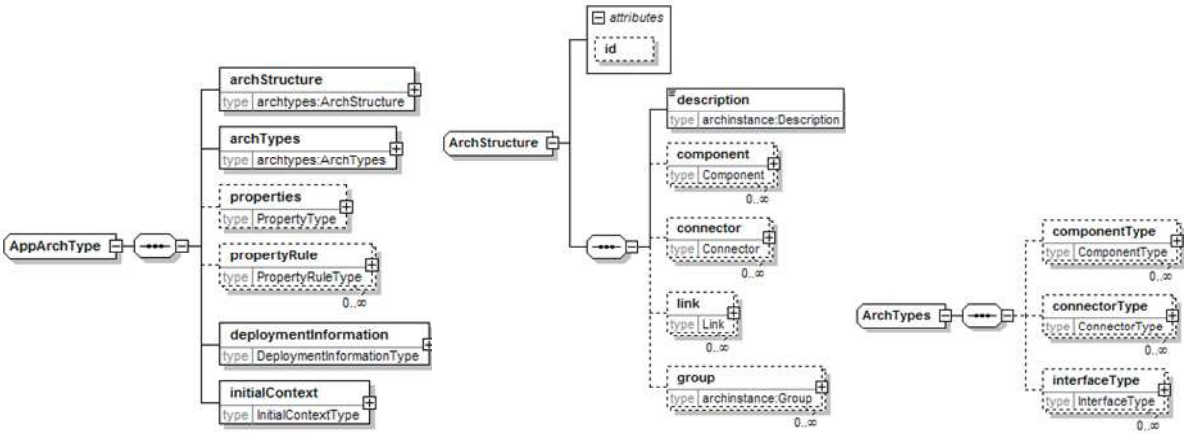


Figura 2. Esquema XML de los elementos básicos en DAOPxADL.

Las reglas de composición de componentes de DAOP-ADL resolvían dos tipos de incompatibilidades. Para resolver la incompatibilidad entre los nombres de rol se ha extendido *link* con los atributos *formalRole* y *realRole*. Para las incompatibilidades entre las propiedades se crea un constructor (*PropertyRule*) que se ha adaptado directamente de DAOP-ADL.
Los links no deben ser extendidos para añadirles semántica puesto que serían conectores [4]. Entonces, para las definir reglas de evaluación de aspectos, extendemos del *ConnectorType* de xADL y le añadimos las reglas de DAOP-ADL. Así se encapsulan en los conectores las reglas de composición entre componentes y aspectos.
En resumen, en DAOPxADL los componentes pueden conectarse directamente, mediante links si hay alguna incompatibilidad, o si se quiere insertar un comportamiento aspectual se incluye un conector. Por lo tanto, cualquier componente podrá tener comportamiento aspectual en un momento dado, siempre que haya una regla en un conector que indique que ese componente se aplica de modo aspectual sobre otros componentes. A modo de aclaración, cabe decir que cualquier componente o conector (del tipo *ComponentType* o *ConnectorType*) descrito en xADL también será valido en DAOPxADL, ya que en este

lenguaje se han definido dos tipos nuevos (*ComponetDAOPType* y *AspectualConnectorType*) como extensión de los anteriores. Pero en la arquitectura de la aplicación, nada restringe a que los elementos que se describan sean de esos tipos.

Elementos	**xADL**	**DAOP-ADL**	**DAOPxADL**
Componentes	*ComponentType*: Elemento con signaturas que apuntan a interfaces genéricas.	*Component*: Elemento con atributos de estado, propiedades e interfaz proporcionada, requerida o de eventos.	*ComponentDAOPType*: Extensión de *ComponentType*, con atributos de estado y propiedades.
Conectores	*ConnectorType*: Elemento con signaturas que apuntan a interfaces genéricas.	No tiene conectores.	No se usarán los conectores con el fin descrito en xADL, se extenderán para encapsular las reglas de evaluación.
Aspectos	No tiene.	*Aspect*: Elemento con atributos de estado, propiedades e interfaz evaluada.	Se usa *ComponentDAOPType* para describir un aspecto.
Links	*Links* que enlazan interfaces.	No tiene *links* explícitos.	Se extienden los *links* para las reglas de composición de componentes.
Interfaces	*InterfaceType*: Elemento genérico que indica una dirección (*in*, *out* o *inout*).	*ProvidedInterface*, *RequiredInterface*, *EventInterface* y *EvaluatedInterface*.	Se extiende *InterfaceType* para especificar cada interfaz definida en DAOP-ADL.
Aspecto de coordinación	No tiene aspecto de coordinación.	*CoordinationAspectType*: extensión de aspecto con protocolo de coordinación e interfaz requerida.	*CoordinationAspectType*: extensión de *ComponentDAOPType* con protocolo.
Reglas de composición de componentes	No tiene reglas de composición de componentes.	Resuelven incompatibilidades: de componentes y de propiedades.	*ComponentRuleType* extiende de *link* y *PropertyRules* adaptado de DAOP-ADL.
Reglas de evaluación de aspectos	No tiene reglas de evaluación de aspectos.	*AspectEvaluationRuleType*: reglas que definen cómo y cuándo aplicar los aspectos a los componentes.	*AspectualConnectorType*: extiende de *ConnectorType* para añadirle las reglas.

Tabla 1. Elementos básicos de xADL, DAOP-ADL y DAOPxADL.

En la Figura 3 se muestran los esquemas XML del tipo de componente (*ComponentDAOPType*) y del conector aspectual (*AspectualConnectorType*).

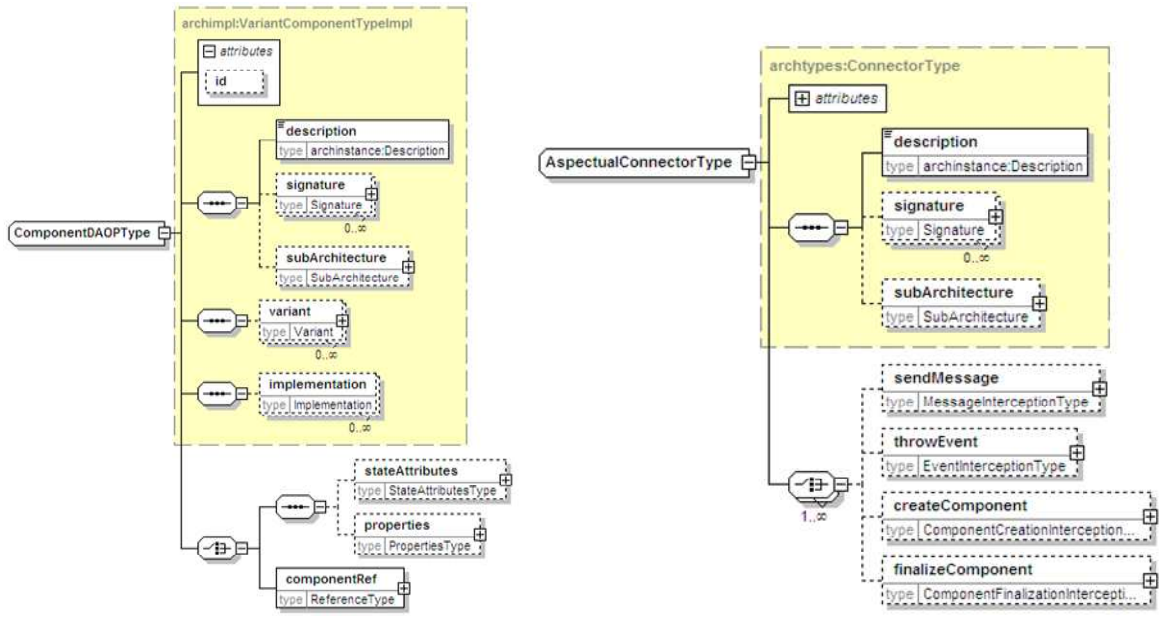
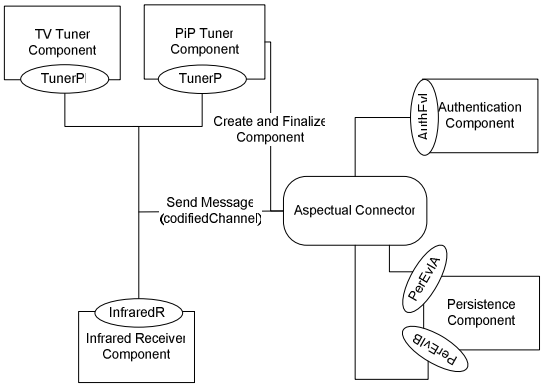


Figura 3. Esquema XML de ComponentDAOPType y AspectualConnectorType

3.1. Descripción de la arquitectura del sistema de televisión en DAOPxADL

Describiremos la arquitectura del sistema de televisión en DAOPxADL para ilustrar la utilidad de separar las propiedades que atraviesan elementos del sistema. Modelamos las propiedades de persistencia y autenticación como aspectos. Si queremos que cuando se active el *picture-in-picture* se sintonice el canal que estaba antes de apagarlo, debemos aplicar el aspecto de persistencia para guardar el estado del canal del *PiP* cada vez que se destruya dicho componente, y recuperarlo cuando se vuelva a crear. Para definir canales que sólo pueden ser visualizados introduciendo previamente un código correcto, se debe aplicar el aspecto de autenticación, encargado de solicitar el código y de comprobar la corrección del mismo.
Cómo DAOPxADL sólo usa los conectores para las reglas de evaluación de aspectos, en nuestro ejemplo (Figura 4.a) desaparece el conector de la TV. La interfaz proporcionada de los componentes de tipo sintonizador contiene los métodos para cambiar de canal y para tratar los canales codificados (*codifiedChannel)*. Estos métodos son los que necesita el receptor de infrarrojos, y por tanto, se encuentran en su interfaz requerida. El aspecto de persistencia tiene dos interfaces evaluadas, una para recuperar el estado del *PiP* después de crear el componente y otra para guardarlo antes de destruirlo. El aspecto de autenticación tiene una interfaz evaluada para capturar el mensaje *codifiedChannel*. El conector aspectual (Figura 4.b) define una regla para que después de que se envíe el mensaje (*AFTER_SEND*) *codifiedChannel* se aplique el aspecto de autenticación. Además, proporciona dos reglas para aplicar el aspecto de persistencia: una para después de crear un componente (*AFTER_NEW*) *piptuner*, y otra para antes de destruirlo (*BEFORE_DESTROY*).
La ventaja de extender xADL con aspectos, es que para el ejemplo se han reutilizado los componentes que se describieron antes. Así, sin tener que cambiar de lenguaje, se han añadido dos componentes que se comportan como aspectos y un conector con las reglas de evaluación.

(a)



(b)

```
<types:connectorType types:id="AspConnType"
  xsi:type="daopxADL:AspectualConnectorType">
  <daopxADL:sendMessage … >
      <daopxADL:sourceComp daopxADL:id=infrared …/>
      <daopxADL:targetComp daopxADL:id=TunerType … />
      <daopxADL:targetMessages daopxADL:name="codifiedChannel"/>
      <daopxADL:AFTER_SEND xsi:type="daopxADL:ListOfAspectsType">
            <daopxADL:concurrent xsi:type="daopxADL:ConcurrentType">
                  <daopxADL:aspect daopxADL:criticism="CRITICAL"
                                   daopxADL:id="authentication"/>
            </daopxADL:concurrent>
      </daopxADL:AFTER_SEND>
  </daopxADL:sendMessage>
  <daopxADL:createComponent daopxADL:id="piptuner" … >
      <daopxADL:AFTER_NEW xsi:type="daopxADL:ListOfAspectsType">
            <daopxADL:concurrent xsi:type="daopxADL:ConcurrentType">
                  <daopxADL:aspect daopxADL:criticism="CRITICAL"
                                   daopxADL:id="persistence"/>
                  ...
  </daopxADL:createComponent>
  <daopxADL:finalizeComponent daopxADL:id="piptuner" … >
                  … daopxADL:id="persistence"...
    </daopxADL:finalizeComponent>
</types:connectorType>
```

Figura 4. (a) Diagrama de ejemplo de la arquitectura del sistema de televisión con aspectos. (b) Definición en XML de las reglas de evaluación en el conector aspectual.

3.2. Soporte de herramientas

La utilidad de un LDA está directamente relacionada con el tipo de herramientas de apoyo que proporcione [2]. En la Figura 4.b se observa que la definición del conector aspectual no es demasiado legible y que escribir documentos XML de este tipo es algo tedioso. Para la descripción de la arquitectura del ejemplo, se ha usado *ArchEdit*, un editor gráfico con estructura de árbol que esconde todos los detalles de los documentos XML. Este editor forma parte de *ArchStudio* [10] (conjunto de herramientas que ofrece xADL) y su principal ventaja es, que se puede usar para descripciones arquitectónicas hechas en extensiones de xADL. Después, para comprobar que el documento esté bien formado y es válido con respecto al esquema XML que define DAOPxADL, se puede utilizar cualquier validador de XML.

4. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha mostrado cómo incorporar la separación de aspectos a un LDA estándar. Nuestra propuesta es simétrica, por lo que cualquier componente podrá se utilizado como aspecto. Se ha extendido la semántica del conector para encapsular las reglas de evaluación de los aspectos. Y los *links* se usan para resolver las reglas de incompatibilidad entre los nombres de rol de los componentes.

Actualmente estamos desarrollando un generador de código, para solventar el salto que se produce entre el diseño y la implementación de sistemas. Esta aplicación generará código Java, AspectJ y AOP/JBoss a partir de documentos XML que contengan descripciones arquitectónicas en DAOPxADL.

REFERENCIAS

[1] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 6ª ed. (1999).

[2] Medvidovic, N., Taylor, R.N.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, 26(1):70–93 (2000).

[3] xADL Home Page, "xADL 2.0: A Highly Extensible Architecture Description Language for Software and Systems", The Regents of the University of California (2000-2005). http://www.isr.uci.edu/projects/xarchuci/.

[4] Dashofy, E., Van Der Hoek, A., Taylor R.: A Comprehensive Approach for the Development of Modular Software Architecture Description Languages. ACM Transactions on Software Engineering and Methodology, Vol. 14, No. 2, April 2005, Pages 199–245.

[5] Pinto, M., Fuentes, L., Troya, J.M.: DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development. F. Pfenning and Y. Smaragdakis (Eds.): GPCE 2003, LNCS 2830, pp. 118–137, 2003. Springer-Verlag

Berlin Heidelberg.

[6] PRISMA Home Page, "Model Compiler of Aspect-Oriented Component-Based Software Arquitecture". http://prisma.dsic.upv.es/

[7] Navasa, A., Palma, K., Murillo, J., Eterovic, Y.: Dos modelos arquitectónicos para el DSOA. In: DSOA, JISBD, Málaga, España (2004).

[8] Pinto, M.: CAM/DAOP: Modelo y Plataforma Basados en Componentes y Aspectos, Tesis Doctoral. Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga (2004).

[9] Dashofy, E., Van der Hoek, A.: Representing Product Family Architectures in an Extensible Architecture Description Language. Fourth International Workshop on Product Family Engineering, October 2001, pages 330–341.

[10] ArchStudio Home Page, "ArchStudio 3: A Software Architecture-Based Development Environment". http://www.isr.uci.edu/projects/archstudio/