PRISMA: PlatafoRma OASIS para Modelos Arquitectónicos

Jennifer Pérez, Isidro Ramos, Ángeles Lorenzo, Patricio Letelier, Javier Jaén

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia – España
{jeperez|iramos|alorenzo|letelier|fjaen}@dsic.upv.es

Resumen. El desarrollo de sistemas software actuales es necesario abordarlo mediante plataformas que permitan describir modelos de arquitectura complejos, distribuidos, evolutivos y reutilizables. PRISMA es un modelo arquitectónico basado en aspectos y componentes que utiliza un lenguaje de definición de componentes (componentes, conectores y sistemas) para definir los elementos de la arquitectura a un elevado nivel de abstracción y un lenguaje de configuración para diseñar la arquitectura de sus sistemas software. El lenguaje de definición de componentes de PRISMA potencia la reutilización, permite la importación de COTS y aglutina dos aproximaciones de desarrollo de sistemas software de actualidad, el Desarrollo Software Basado en Componentes (DSBC) y el Desarrollo Software Orientado a Aspectos (DSOA). El lenguaje de configuración permite definir instancias a partir de los elementos definidos o de las COTS importadas, especificando la topología existente para un sistema software específico. El metanivel del PRISMA y las propiedades reflexivas de los lenguajes diseñados permiten tanto la evolución de los elementos, como la reconfiguración dinámica de la topología.
Palabras clave: Arquitecturas software, lenguaje de definición de componentes, lenguaje de configuración, componentes, aspectos, reutilización, evolución.

1. Introducción

Los sistemas de información cada vez son más difíciles de desarrollar debido a sus complejas estructuras y a la dinámica y competitividad del mercado que implica una evolución constante de los requisitos del sistema. Por este motivo, los entornos de desarrollo deben permitir construir aplicaciones con arquitecturas complejas, distribuidas, evolutivas y reutilizables. El enfoque orientado a objetos por sí solo no permite abordar los requisitos de los sistemas software complejos actuales. Debido a esto, han surgido dos tendencias en el desarrollo de sistemas software, el Desarrollo de Software Basado en Componentes (DSBC) y el Desarrollo de Software Orientado a Aspectos (DSOA).

En la actualidad es fundamental que los procesos de producción de sistemas software minimicen el tiempo y el coste de desarrollo y mantenimiento. Por este

motivo, no sólo es importante que los entornos software permitan definir elementos reutilizables y evolutivos, sino que también han de posibilitar la importación de componentes COTS (comercial *off-the-shelf*).

Este trabajo presenta un enfoque para la construcción de modelos arquitectónicos basado en aspectos y componentes llamado PRISMA. Dicho enfoque utiliza una extensión de OASIS para la definición formal de la semántica de los modelos de arquitectura software. Dicha extensión ha sido necesaria porque OASIS [Let98] es un lenguaje formal para definir modelos conceptuales de sistemas de información orientados a objetos y no permite especificar arquitecturas software. Sin embargo, OASIS permite validar y generar las aplicaciones automáticamente a partir de la información capturada en los modelos (compilación de modelos).

La extensión de OASIS ha dado lugar a un lenguaje de definición de componentes para definir los elementos de la arquitectura y a un lenguaje de configuración para especificar modelos arquitectónicos. Ambos pueden ser compilados para generar automáticamente el código del producto siguiendo el paradigma de la programación automática.

El lenguaje de definición de componentes de nuestro PRISMA potencia la reutilización, permite la importación de COTS y aglutina dos aproximaciones para el desarrollo de sistemas: el DSBC y el DSOA. Por ello, un elemento de nuestro modelo arquitectónico esta constituido por un conjunto de aspectos: funcional, distribución, coordinación, etc.

El metanivel de PRISMA y las propiedades reflexivas de los lenguajes diseñados dan soporte tanto para la evolución de los elementos, como para la reconfiguración dinámica de la topología. Esta característica del lenguaje permite definir un metanivel por aspecto que reifica las propiedades que deseen ser evolucionadas basándose en la reflexión de OASIS ([Car99]).

El modelo arquitectónico PRISMA que presentamos a continuación incorpora las ventajas de los DSBS y DSOA, reduciendo el tiempo de desarrollo y mantenimiento gracias a que sus elementos son reutilizables y evolutivos, y que además, es capaz de importar componentes COTS.

La estructura del trabajo es la siguiente: en primer lugar se presenta el modelo arquitectónico PRISMA y el metamodelo asociado. En el apartado 3, se presenta el ejemplo que se va a desarrollar a lo largo del documento. En los apartados 4 y 5 se describen los lenguajes de definición de componentes y de configuración respectivamente. En el apartado 6 se comenta el estado del arte del desarrollo de arquitecturas basadas en componentes y aspectos. Finalmente, en el apartado 7 se presentan las conclusiones y trabajos futuros.

2. Modelo de Arquitectura

PRISMA es un modelo para definir arquitecturas de sistemas software complejos. Dicho modelo permite definir los elementos mediante la composición de aspectos: funcional, distribución, coordinación, etc. Dependiendo del tipo de elemento de la arquitectura que se defina y del tipo de sistema que se esté desarrollando, el elemento tendrá especificados unos aspectos u otros.

El problema de la integración sin solape de código que existe en la programación por aspectos [Kic01] se resuelve en nuestro modelo gracias a que la integración se realiza a un alto nivel de abstracción y a que la generación de código de cada uno de estos aspectos se realiza de forma separada.

A continuación se detallan algunos de los aspectos que puede incorporar un elemento PRISMA, que son los que se van a considerar en este trabajo:

Aspecto Funcional: El aspecto funcional captura la semántica de la organización, mediante la definición de su estructura y su comportamiento.

Aspecto de Coordinación: El aspecto de coordinación nos define las reglas de negocio y la sincronización entre elementos durante su comunicación. Así como la coreografía o protocolo, extendiendo el concepto de *"contrato"* de Andrade y Fiadeiro ([And99]).

Aspecto Distribución: El aspecto de distribución captura aquellas propiedades que caracterizan el mecanismo de ubicación dinámico de las instancias de un elemento dentro del modelo de arquitectura software definido.

Aspecto de Evolución: El aspecto de evolución refleja la evolución del elemento debido al cambio de los requisitos de la organización. Este aspecto proporciona la dinámica del elemento, es decir, la capacidad de cambio de su estructura y comportamiento basándose en la evolución del software que plantea la reflexión de OASIS.

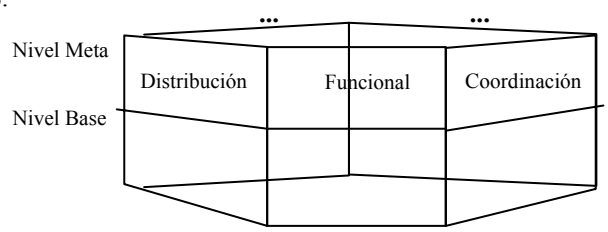


Fig. 1. PRISMA: Plataforma OASIS para Modelos Arquitectónicos

Un elemento (componente, conector o sistema) de nuestro modelo de arquitectura puede ser visto como un prisma con tantas caras como aspectos considere el tipo de elemento que representa, teniendo en cuenta que el aspecto de evolución no es una cara del prisma, sino la parte superior de la división en niveles base y meta de cada cara (ver figura 1). Cada una de ellas tendrá definido un nivel meta, siempre y cuando en el elemento se reifique alguna de las propiedades del nivel base del aspecto que representa dicha cara. La reificación puede afectar a cualquier sección de la especificación OASIS del aspecto, es el que le proporciona al lenguaje de definición de arquitecturas su capacidad reflexiva.

La reificación se realiza a partir del nivel base, lo que implica que no puede existir un elemento que tenga un aspecto especificado a nivel meta y no a nivel base. Por lo tanto, la relación entre el nivel meta y el nivel base de un aspecto es uno a muchos.

Nuestro modelo arquitectónico consta de distintos tipos de elementos. Todos los elementos de nuestro modelo arquitectónico forman parte del metamodelo (ver figura 2) para así poder modificarlos de forma dinámica a través de los mecanismos de reflexión del lenguaje. A continuación se citan y definen cada uno de ellos.

Interfaz: Una interfaz proporciona un conjunto de servicios con su aridad. Existen diferentes tipos de interfaces, uno para cada tipo de aspecto.

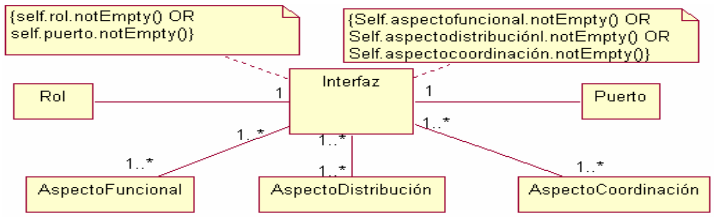


Fig. 2. Sección de interfaces del metamodelo de PRISMA

Aspecto: Un aspecto define de forma completa la estructura y el comportamiento de la componente desde un determinado punto de vista. Un aspecto puede verse como la unión de un conjunto de interfaces del tipo de dicho aspecto, más la especificación de la semántica de su estructura y comportamiento (cambios de estado, disparos, protocolos, etc). Existen diferentes tipos de aspectos: funcional, distribución, coordinación, etc. Definiéndose cada uno de ellos de forma disjunta.

Componente: Una componente es un elemento del sistema de información que no actúa como coordinador entre otros elementos. Está formado por un conjunto de aspectos (funcional, distribución, etc) y por uno o más puertos de entrada y de salida, cuyo tipo es una determinada interfaz. Las componentes interaccionan con el resto de elementos de la arquitectura a través de sus puertos.

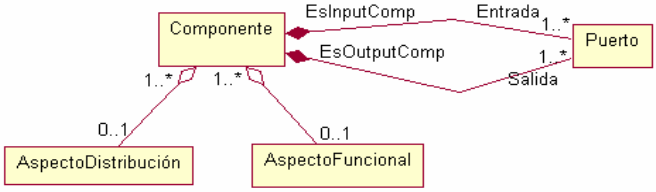


Fig. 3. Sección de componentes del metamodelo de PRISMA

Conector: Un conector (ver figura 4) es un elemento del sistema de información que actúa como coordinador entre varios elementos. Se compone de un aspecto de coordinación, un aspecto funcional, un aspecto distribuido y un conjunto de roles de entrada y salida que tienen como tipo una interfaz de carácter funcional o distribuido. Los conectores conectan y sincronizan componentes, conectores o sistemas a través de sus roles.

Sistema: Un sistema (ver figura 5)es una componente que encapsula un conjunto de conectores, componentes y otros sistemas correctamente conectados, pudiendo surgir de dicha composición propiedades emergentes. Además, se caracteriza por la definición de *binding links* entre los puertos del sistema y los puertos de las componentes que encapsula.

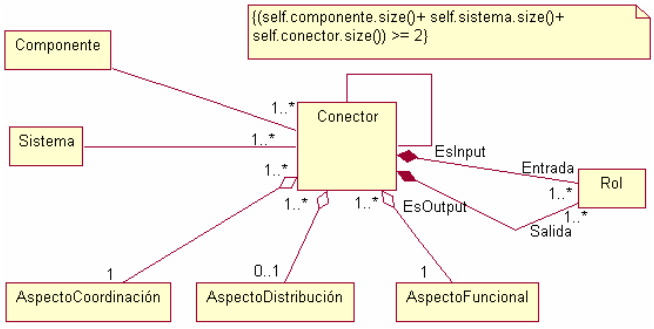


Fig. 4. Sección de conectores del metamodelo de PRISMA

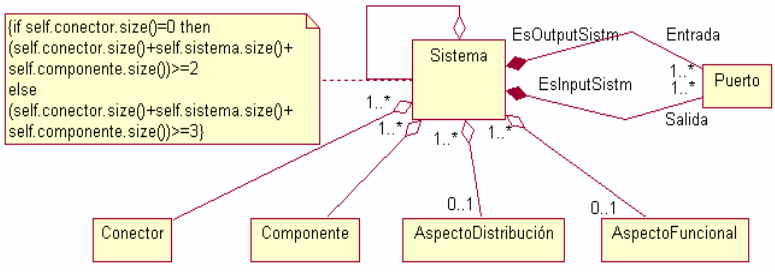
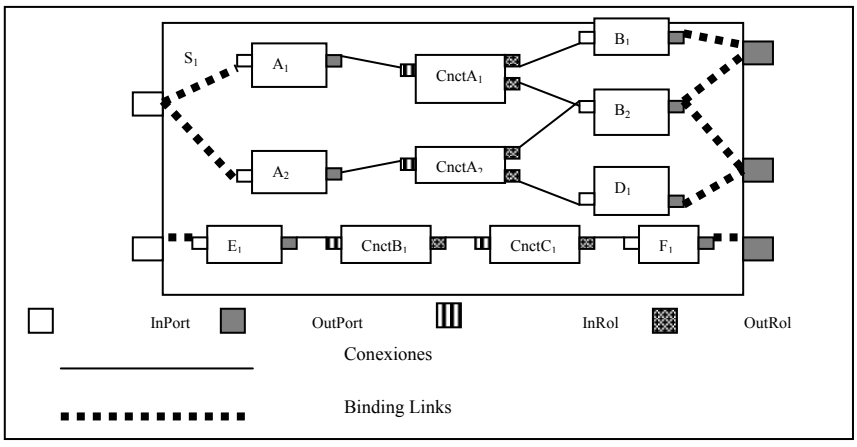


Fig. 5. Sección de sistemas del metamodelo de PRISMA

3. Ejemplo

A continuación se presenta un pequeño ejemplo definido en base a los conceptos de nuestro modelo arquitectónico. Este ejemplo se utilizará en el resto del documento para mostrar una posible especificación de arquitectura utilizando los lenguajes que se presentarán en los siguientes apartados.



S_1: Instancia del sistema S definida mediante el lenguaje de definición de componentes
A_1, A_2: Instancias del componente A definido mediante el lenguaje de definición de componentes
B_1, B_2: Instancias del componente B definido mediante el lenguaje de definición de componentes
D_1: Instancia del componente D definido mediante el lenguaje de definición de componentes
$CnctA_1$, $CnctA_2$: Instancias del conector CntcA definido mediante el lenguaje de definición de componentes
E_1: Instancia del componente E definido mediante el lenguaje de definición de componentes
F_1: Instancia del componente F definido mediante el lenguaje de definición de componentes
$CnctB_1$: Instancia del conector CntcB definido mediante el lenguaje de definición de componentes
$CnctC_1$: Instancia del conector CntcC definido mediante el lenguaje de definición de componentes

Fig. 6. Ejemplo de Sistema

4. Lenguaje de Definición de Componentes

El lenguaje de definición de componentes PRISMA permite definir los tipos de elementos de la arquitectura. Este lenguaje tiene como ciudadanos de primer orden: interfaces, aspectos, componentes y conectores, potenciando así la reutilización. De este modo, una interfaz puede ser utilizada por varios aspectos y un aspecto por varios elementos. Así mismo, tanto componentes como conectores y sistemas podrán reutilizarse en diferentes modelos arquitectónicos.

La importación de COTS se expresa gracias al carácter opcional de los aspectos, es decir, a la especificación de la estructura y comportamiento de las distintas facetas de la componente o el conector. Esta opcionalidad permite ver a estos elementos como cajas negras y con ello la posibilidad de incorporar componentes COTS.

4.1. Interfaces y Aspectos

En la especificación de una interfaz funcional se incluyen un conjunto de servicios con una signatura, describiendo la visibilidad parcial, que tiene un elemento del sistema o un elemento externo, de un determinado aspecto de un elemento. Un aspecto funcional se especifica mediante la plantilla OASIS presentada en [Let98], de forma que se definen los atributos, servicios, evaluaciones, derivaciones, precondiciones, disparos, restricciones de integridad y el **protocolo** de la componente que incluya este aspecto.

A continuación se muestra el extracto del código perteneciente a las interfaces funcionales y al aspecto funcional que especifica los servicios que incluyen dichas interfaces entre otras características funcionales.

```
Functional Interface I1(someone)
    Services(service1, service2)
End Functional Interface I1;
Functional Interface I2(someone)
    Services(service3)
End Functional Interface I2;
Functional Interface I3(someone)
    Services(sevice4)
End Functional Interface I3;
```

```
Functional Aspect F1
    Attributes …
    Services
        service1(…);
        service2(…);
        service3(…);
        service4(…);
         …
End Functional Aspect F1;
```

La especificación de una interfaz distribución es análoga a una de tipo funcional, la única diferencia es que los servicios ofrecidos en dicha interfaz contribuyen a la gestión del carácter distribuido de un elemento. A nivel de componentes dicha interfaz ofrece servicios de obtención de valores de métricas para la implementación de estrategias de distribución complejas tales como las que se describen en [Zin97, Che02]. A nivel de conectores y sistemas dichas interfaces ofrecen servicios de gestión del carácter distribuido de una colección de componentes. Un aspecto distribución especifica las políticas y estrategias de distribución a seguir en función de los valores obtenidos en tiempo real de las métricas establecidas.

La especificación de una interfaz de coordinación es análoga a las dos anteriores. Un aspecto de coordinación especifica los atributos, servicios, sincronizaciones entre interfaces y la coreografía de un conector, ya que este aspecto no se puede dar en las componentes o en los sistemas.

Las **sincronizaciones** definen las conexiones permitidas entre tipos de rol y de puerto. Esta especificación se realiza mediante interfaces, y no mediante puertos y roles de los elementos, para conseguir así un mayor nivel de abstracción y que un aspecto de coordinación pueda ser utilizado por distintos tipos de conecto, de no ser así, el aspecto de coordinación dejaría de ser un ciudadano de primer orden del lenguaje.

La **coreografía** especifica el proceso de coordinación que aplica el conector para sincronizar a los tipos de elementos que conecta. Al igual que las sincronizaciones, se especifica mediante interfaces.

Un ejemplo de especificación de una interfaz de coordinación y de un aspecto de coordinación con sus sincronizaciones, atributos, servicios y coreografía es el que se muestra a continuación:

```
Coordination Interface
I4(someone)
   services(all)
End Coordination Interface I4;

Coordination Aspect Co1
   Attributes …
   Services …
   Synchronizing Strategies
   {
    I1 << I1;
    I2 >> I3;
    I3 >> Ik;
    I3 >> Ir;
   }
   Coreography(…);
End Coordination Aspect Co1)
```

4.2. Componentes, Conectores y Sistemas

La **componente** contiene una series de puertos de entrada y de salida cuyos tipos son una interfaz funcional o distribuida. De forma que tenemos dos tipos de puertos, los que sirven para acceder aspectos funcionales de la componente y los que permiten acceder a aspectos distribuidos. Los puertos de entrada especifican los atributos y servicios que oferta la componente a otros elementos (comportamiento servidor). Los puertos de salida especifican los atributos y servicios externos que utiliza de otros elementos (comportamiento cliente). Siempre que se defina una componente se ha de especificar su aspecto funcional, excepto cuando son COTS. Mientras que la especificación de su aspecto distribución únicamente se realizará cuando se quiera dotar a la componente de un comportamiento distribuido. Se ha de tener en cuenta que

los tipos de los puertos de una componente únicamente podrán ser aquellas interfaces que formen parte del aspecto funcional o distribuido de dicha componente.

La definición de **conector** contiene un conjunto de roles de entrada y de salida, que tienen la misma funcionalidad y semántica que los puertos en la componente. Siempre que se defina un conector se ha de especificar su aspecto funcional y de coordinación, mientras que el de distribución es opcional tal y como ocurre en la componente. Al igual que en las componentes, únicamente se podrá asociar un tipo a los roles del conector con aquellas interfaces que formen parte del aspecto funcional o distribuido que incluye dicho conector. Sin embargo, las interfaces de coordinación no se utilizan en los roles, ya que expresan la visibilidad del aspecto de coordinación a elementos externos al sistema. Además, un conector incluye los tipos de componentes, sistemas y conectores que puede conectar.

A continuación se muestra la especificación del tipo de conector CntcA del ejemplo, incluyendo los aspectos funcional y de coordinación anteriormente definidos, los tipos de componentes A, B y D que puede conectar este conector, un rol de entrada y dos roles de salida, cuyas interfaces han sido definidas previamente.

```
Connector_type CntcA
   InRols IRol1: I1; End InRols;
   OutRols
        ORol1: I2;
        ORol2: I3; End OutRols;
   Connected Elements_Type: A, B, D;
   Functional Aspect F1;
   Coordination Aspect Co1;
End Connector_type CntcA;
```

Finalmente, la definición de **sistema** es equivalente a la de componente, ya que un sistema es una componente compleja. Por lo tanto un sistema es una componente formada por un conjunto de tipos de elementos conectados entre sí, cuya composición puede hacer emerger un aspecto funcional y un aspecto distribuido. Por ese motivo, un sistema incluye el conjunto de conectores, componentes y sistemas que conecta. También se ha de tener en cuenta que si el sistema incluye un tipo de conector deberá de contener todos los tipos de componentes, sistemas y conectores que conecte dicho tipo de conector. La especificación de la conexión entre los elementos que incluye el sistema y el propio sistema se realiza mediante los ***binding links***, que al igual que las sincronizaciones de los conectores se especifican mediante las interfaces.

4.3. Aspecto de Evolución

El lenguaje de definición de componentes puede construir automáticamente el metanivel a partir del nivel base, reificando solamente aquellos elementos de la especificación que sean susceptibles de evolucionar en el contexto del problema. Para expresar dicha reificación se hará uso del operador ***reify***, teniendo en cuenta que dicha reificación se realizará de forma automática.

La semántica del operador *reify* en cada uno de los niveles de granularidad de la especificación de los tipos de elementos será diferente y por lo tanto, se deberá asociar una generación diferente en el metanivel para cada uno de los casos detectados, que serán especificados mediante patrones. La evolución puede ser estructural o de comportamiento. La evolución estructural se consigue reificando las

propiedades de los elementos que definen a los elementos de una especificación. Mientras que la evolución de comportamiento se consigue reificando aquellas secciones de la especificación que definen el comportamiento del elemento. Este tipo de reificación nos da la propiedad cinemática del lenguaje (cambio de comportamiento sin cambio de estructura).

La especificación de los elementos puede expresarse como un término, pudiendo cualificar con el operador *reify* a cada uno de los subtérminos que constituyen ese término. Por lo tanto, una especificación puede verse como un árbol con diferentes niveles de profundidad, tantos como tipos de elementos de diferente granularidad existan en una especificación. Por esta razón, *reify* es un operador que tiene como entrada el árbol o subárbol de nivel base que se quiere reificar y produce como salida el árbol a nivel meta correspondiente. La función del operador consistirá en aplicar el patrón de generación del tipo de elemento que representa el árbol para generar las metapropiedades y metaeventos correspondientes y posteriormente, enraizarlos en el árbol de nivel meta existente que representa la especificación meta adecuadamente.

5. Lenguaje de Configuración

El lenguaje de configuración permite definir las instancias y especificar la topología del modelo arquitectónico. Para ello, se han de importar todos aquellos tipos de conectores, componente y sistemas, definidos mediante el lenguaje de definición de componentes, que se necesiten para un determinado modelo de arquitectura. Después se define el conjunto de instancias necesarias de cada uno de los tipo importados.

Finalmente se debe definir la topología del modelo, interconectando las instancias del modelo. Para ello se deberán instanciar las sincronizaciones de los conectores y los *binding links* de los sistemas.

```
Architecture Model  Ejemplo
Import
S,A,B,D,E,F,CnctA,CnctB,CntcC;
S1: S;
A1, A2: A;
B1,B2: B;
Connector CnctA1
  Coordination Aspect
   Connected Components
   { A1: A;
     B1,B2: B; }
   Synchronizing Strategies
   {IRol1 << A1.OPort1;
    ORol1 >> B1.IPort1;
    ORol2 >> B2.IPort1;}
  End Coordination Aspect;
End Connector CnctA1;
Connector CnctA2
  Coordination Aspect
   Connected Components
   { A2: A;
     B2: B;
     D1: D; }
D1: D;
E1: E;
F1: F;
CnctA1,CnctA2: CnctA;
CnctB1: CnctB;
CnctC1: CnctC;
   Synchronizing Strategies
   {IRol1 << A2.OPort1;
    ORol1 >> B2.IPort1;
    ORol2 >> D1.IPort1;}
  End Coordination Aspect;
End Connector CnctA2;
Connector CnctB1
  Coordination Aspect
   Connected Components
   { E1: E;
     CnctC1: CnctC;}
   Synchronizing Strategies
   {IRol1 << E1.OPort1;
    ORol1 >> CnctC1.IRol1;}
  End Coordination Aspect;
End Connector CnctB1;
```

```
Connector CnctC1
  Coordination Aspect
   Connected Components
   { F1: F;
     CnctB1: CnctB;}
   Synchronizing Strategies
   {IRol1 << CnctB1.ORol1;
    ORol1 >> F1.IPort1;}
  End Coordination Aspect;
End Connector CnctC1;
System S1
 A1, A2: A;
 B1,B2: B;
 D1: D;
 E1: E;
 F1: F;
 CnctA1,CnctA2: CnctA;
 CnctB1: CnctB;
 CnctC1: CnctC;

 Binding Links
 {S1.IPort1 >> A1.IPort1;
  S1.IPort1 >> A2.IPort1;
  S1.IPort2 >> E1.IPort1;
  S1.OPort1 << B1.OPort1;
  S1.OPort1 << B2.OPort1;
  S1.OPort2 << B2.OPort1;
  S1.OPort2 << D1.OPort1;
  S1.Oport3 << F1.OPort1;}
End System S1;
End Architectural Model Ejemplo;
```

6. Trabajos Relacionados

En la actualidad existen una gran variedad de trabajos relacionados con modelos arquitectónicos, componentes, lenguajes de definición de arquitecturas, reconfiguración dinámica de arquitecturas y todo aquello que guarda relación con el desarrollo de sistemas software complejos. Este gran esfuerzo de investigación persigue el objetivo de encontrar un consenso en los conceptos básicos de sistemas software complejos, así como su definición precisa y la metodología a aplicar para su desarrollo. Debido a que no se ha alcanzado un consenso, nuestra propuesta ha tenido en cuenta varios trabajos que incorporan las distintas áreas de interés para nuestro modelo. Cada uno de ellos aporta una parte de nuestra propuesta, mientras que nuestro trabajo integra dichas partes en un modelo arquitectónico único y hace emerger propuestas innovadoras.

El trabajo realizado por Garlan [Gar01] es un punto de referencia muy importante a la hora de establecer los elementos de un sistema software complejo, en el cual se introducen conceptos como sistema, conector, componente, puerto de entrada y puerto de salida entre otros. Nuestra propuesta ha considerado los trabajos de Andrade y Fiadeiro acerca del concepto de contrato, ya que hemos definido nuestros conectores como una extensión del contrato propuesto por su trabajo. Dicha extensión se produce debido a la incorporación del concepto de coreografía en el conector, concepto utilizado con una semántica distintita en los trabajos de [Mon02] y [Bra02].

En una línea cercana a la nuestra, se encuentra Model-Driven Architecture (MDA), un enfoque propuesto por la OMG (www.omg.org) para la integración e interoperabilidad de los sistemas. MDA enfatiza el modelado (con UML) independiente de las plataformas de implementación. MDA está basado en el modelado de diferentes aspectos y niveles de abstracción, explotando las interrelaciones y estableciendo transformaciones hacia su implementación.

Existe una gran variedad de lenguajes de definición de arquitecturas, cada uno de ellos presenta ventajas e inconvenientes, tal y como presenta el estudio realizado en [Med00]. Uno de los lenguajes que más se aproxima a los nuestros es el propuesto por Loques y Leite [Loq00] en su modelo R-RIO, en el cual incorporan aspectos de reconfiguración dinámica a través del metanivel. También, se encuentra en enfoques

orientados a un nivel de abstracción inferior, es decir, orientados a implementación como [McG02]. En GUARANÁ [Oli98] también se presenta un metanivel complejo que hay que definir durante la compilación, siendo su estructura ajena al modelo. Otro marco de referencia en nuestro trabajo es el Diseño Software Orientado a Aspectos (DSOA), en el que se encuentran trabajos como [Aos02]. Esta aproximación se aplica principalmente en la implementación y no se aplica para la especificación a un alto nivel de abstracción de sistemas arquitectónico que integren componentes y aspectos.

7. Conclusiones y Trabajos Futuros

Se ha propuesto un marco de trabajo para modelado arquitectónico basado en aspectos y componentes llamado PRISMA. Dicho marco se caracteriza por la integración que realiza de las aproximaciones de desarrollo DSBC y DSOA y las propiedades reflexivas a través de un metanivel que le da capacidad de evolución. PRISMA se presenta como un enfoque integrado y flexible para describir modelos de arquitectura complejos, distribuidos, evolutivos y reutilizables.

Los modelos arquitectónicos que siguen el PRISMA se especifican mediante dos lenguajes, un lenguaje de definición de componentes y un lenguaje de configuración, ofreciendo la posibilidad de importar componentes COTS y reutilizar aquellos componentes que se especifiquen.

Las líneas del trabajo futuras son muchas debido a la amplitud y complejidad del tema. Los trabajos a corto plazo se centran en la reconfiguración dinámica de los modelos arquitectónicos que se especifiquen y en los aspectos del prisma, tanto para incorporar nuevos aspectos como para especificar de una forma más detallada los aspectos de distribución, coordinación y evolución. Los trabajos a largo plazo consistirán en la aplicación de este modelo a distintos campos del desarrollo de software y la construcción de un compilador de modelos para PRISMA.

Referencias

[And99] Andrade L., Fiadeiro J., "*Interconnecting Objects via Contracts*". OOPSLA'99. Tutorial 56.

[Aos02] Aspect Oriented Software Development, http://aosd.net

[Bra02] Bracciali A., Brogi A., Canal C., "*Sistematic Component Adaptation*", In Proc IDEAS, La Habana, 2002.

[Car99] Carsí J. A., "OASIS como marco conceptual para la evolución del software". Ph.D. Thesis 1999. DSIC
(Universidad Politécnica de Valencia).

[Che02] Cheng S., Garlan D., Schmerl B., Steenkiste P., Hu N., *"Software Architecture-based Adaptation for Grid Computing"* , IEEE Conference on High Performance Distributed Computing. Scotland, July, 2002.

[McG02] McGurren F., Conroy D., "*X-ADAPT: An Arquitecture for Dynamic Systems"*, Workshop on Component-Oriented Programming, ECOOP, Málaga, Spain, 2002.

[Gar01] Garlan D., Cheng S., Kompanek A.J., "*Reconciling the Needs of Architectural Description with Object-Modeling Notations*", Science of Computer Programming Journal, Special UML Edition, Elsevier Science, 2001.

[Kic01] Kiczales G., Hilsdale E., Huguin J., Kersten M., Palm J., Griswold W.G., *"An Overview of AspectJ"*, proceeding of the European Conference on Object-Oriented Programming, Springer-Verlag, 2001.

[Let98] Letelier P., Sánchez P., Ramos I., Pastor O. OASIS 3.0: "*Un enfoque formal para el modelado conceptual orientado a objeto*". Universidad Politécnica de Valencia, SPUPV - 98.4011, ISBN 84-7721- 663-0, 1998.

[Loq00] Loques O., Sztajnberg A., Leite J., Lobosco, M., "*On the Integration of Meta-Level Programming and Configuration Programming*", In Reflection and Software Engineering (special edition), Editors: Walter Cazzola, Robert J. Stroud, Francesco Tisato, V. 1826, Lecture Notes in Computer Science, pp.191-210, Springer-Verlag, Heidelberg, Germany, June, 2000.

[Med00] Medvidovic N., Taylor R.N., "*A classification and Comparison Framework for Software Architecture Description Languages",* IEEE Transactions of SW Engineering, Vol. 26, nº 1, January 2000

[Mon02] Monge R., Alves C. Vallecillo A., "*A Graphical Representation of COTS-based Software Architectures*", In Proc IDEAS, La Habana, 2002.

[Oli98] Oliva A., Garcia I.C., Buzato L.E.,"*The Reflective Architecture of Guaraná"*, Technical Report IC-98-14. Instituto de computación, Universidad de Campiñas, abril, 1998.

[Zin97] Zinky J., Bakken D., Schantz R., *"Architectural Support for Quality of Service for CORBA Objects,"* Theory and Practice of Object Systems 3(1), 1997.