AspectLEDA: Un Lenguaje de descripción arquitectónica orientado a aspectos[1]

Amparo Navasa, Miguel A. Pérez, Juan M. Murillo

Quercus SEG http://quercusseg.unex.es/QuercusProy
Departamento de Informática. Universidad de Extremadura
{amparonm, toledano, juanmamu}@unex.es

Abstract. El incremento de la complejidad de los sistemas software hace necesario la utilización de nuevas técnicas que permitan manipularlos adecuadamente. Por una parte, la arquitectura del software es una parte del diseño que ayuda a controlar la complejidad de los grandes sistemas. Por otra parte, el DSOA es uno de los paradigmas que se han propuesto para gestionar la complejidad de los mismos. En este artículo se considera un modelo para el desarrollo de sistemas desde un punto de vista estructural teniendo en cuenta conceptos de separación de aspectos. Se muestran los pasos en que se basa el modelo, incidiendo en el punto de vista del arquitecto software que ha de desarrollar y mantener sistemas complejos. La propuesta resulta ser una herramienta para facilitar la evolución y el mantenimiento de los sistemas a través de la definición de una arquitectura OA. Para ello, se presenta un lenguaje arquitectónico OA describiendo su utilización por el arquitecto software

1 Introducción

Al aumentar la complejidad de los sistemas a desarrollar, se hace necesario disponer de técnicas que permitan manipularlos considerando no sólo su complejidad, sino teniendo en cuenta que, a lo largo de su vida, la estructura de los sistemas cambia y evoluciona para adaptarse a nuevas situaciones. Dos conceptos ayudan al desarrollador a obtener sistemas complejos bien diseñados y fácilmente adaptables: la arquitectura del software (AS) durante la cual se define la estructura de un sistema, y el desarrollo de sistemas orientados a aspectos (DSOA) que se está mostrando como uno de los mejores paradigmas para desarrollar sistemas complejos.

Un buen momento para considerar los aspectos es durante la AS porque es cuando se realiza la definición estructural del sistema. La conveniencia de definir un diseño arquitectónico OA surge cuando se observa que los *crosscutting concerns*[2] cruzan los elementos arquitectónicos, contaminando componentes y conectores, complicando así el diseño final. Durante el diseño arquitectónico los aspectos se pueden considerar

[1] Trabajo parcialmente patrocinado por los proyectos CICYT TIC2002-04309-C02-01 y PRI 2PR04 B0111

[2] Se mantiene la expresión inglesa por considerarse más adecuada que su traducción: preocupaciones transversales

elementos de primera clase. Además, hay que definir la interacción entre ellos y los otros elementos arquitectónicos.

En este artículo se presenta un Lenguaje de Descripción Arquitectónica Orientado a Aspectos (LDA-OA) que denominamos *AspectLEDA*, que se obtiene extendiendo el LDA LEDA [2]. El nuevo lenguaje se define sobre un modelo arquitectónico [7] que permite integrar aspectos en sistemas ya diseñados. El artículo se estructura como sigue: en la sección 2 se reflexiona sobre los conceptos de DSOA, arquitectura del software, y la conveniencia de su uso conjunto; en la sección 3 se resume nuestro modelo arquitectónico OA, incidiendo en las características del lenguaje que se define; en los apartados 4 y 5 se presentan algunos trabajos relacionados y conclusiones.

2 Sistemas complejos, DSOA y arquitectura del software

En esta sección se comentan los conceptos en los que se apoya el modelo descrito brevemente en la sección 3.

El ingeniero de software debe enfrentarse a demandas de la sociedad cada vez más exigentes, que solicita el desarrollo de sistemas con unos requisitos cada vez más estrictos. Además, el mundo real evoluciona y los sistemas software deben adaptarse a él para seguir siendo útiles. Por esta razón, los sistemas deben poder rediseñarse para capturar los cambios tanto del entorno como de los requisitos. Esta necesidad de que el software cambie y evolucione continuamente supone un importante reto para los ingenieros de software que deben buscar técnicas y herramientas que faciliten el mantenimiento de los sistemas complejos. Por ello, los paradigmas que se utilicen en su desarrollo tienen que considerar la evolución como una parte del ciclo de vida ya que, por una parte la evolución es inevitable, y por otra, se van degradando, reduciéndose la confiabilidad del sistema. Aproximaciones tradicionales (programación estructurada, o el paradigma OO) sólo permiten una reconfigurabilidad y reutilización parcial del sistema. Esto sólo es adecuado parcialmente en el caso de cambios en sistemas preexistentes, grandes, multiplataformas, o cambios que provienen de múltiples fuentes.

DSOA y complejidad. Para gestionar la complejidad de los sistemas se han considerado diversas soluciones, siendo una de ellas el DSOA, que permite la identificación temprana de aspectos, su extracción y composición, para facilitar el diseño y la obtención de un código más modularizado, evitando los efectos del enmarañamiento y la dispersión. Para ello, los aspectos se pueden considerar entidades de primera clase manipulables durante el desarrollo.

Dentro del paradigma de DSOA, la definición de aspectos es un mecanismo de abstracción que se puede añadir de algún modo a un sistema existente, de manera que el elemento incorporado no quede distribuido entre diversos módulos del sistema sino que se mantiene independiente de ellos. Dado que el DSOA permite modelar como artefactos software las propiedades que atraviesan los sistemas, estos se pueden adaptar más fácilmente a los cambios y a la evolución de estas propiedades, frente a la utilización de paradigmas tradicionales en los que el código transversal queda disperso por los diferentes módulos o componentes del sistema.

Arquitectura del software y complejidad. La arquitectura del software (AS) es una actividad del diseño que ayuda los desarrolladores a controlar la complejidad de los sistemas, así como a definir su estructura de modo que el mantenimiento y evolución de los mismos sean sencillos. Esto es así porque durante la AS se define la estructura de un sistema como un conjunto de componentes que realizan las computaciones, y un conjunto de conectores que definen la interacción entre ellos.

Control de la complejidad de los sistemas desde un punto de vista arquitectónico dentro del paradigma del DSOA. Consideramos que es interesante tratar conjuntamente los conceptos de AS y DSOA, pues ambas aproximaciones facilitan la gestión de la complejidad de los sistemas: la AS estudia la estructura de los sistemas cuando ésta se está definiendo, y el DSOA permite generar sistemas con código limpio y más fácilmente mantenible. El uso conjunto de ambos conceptos potencia las características de cada uno de ellos.

3 Definición de un LDA-OA.

En esta sección, primero se describe someramente un modelo arquitectónico OA que requiere para su formalización de un LDA que lo soporte. Para ello, en el segundo apartado de esta sección se define un LDA orientado a aspectos (LDA-OA): se hace una descripción detallada del lenguaje de descripción arquitectónica que se define para obtener un sistema OA (*sistema extendido*) a partir de otro LDA (LEDA [2]).

3.1 Un modelo arquitectónico OA

En este apartado se describe brevemente una propuesta [7] que define un modelo arquitectónico OA. Los pasos de la propuesta se resumen a continuación:
- Se considera el desarrollo del sistema inicial (*sistema básico*) desde las primeras etapas: especificación de requisitos y diseño de alto nivel, definiendo su diagrama de casos de uso, los diagramas de secuencia y expresando la descripción arquitectónica en un LDA. El lenguaje LEDA [2] se elige por la fuerte base formal en la que se apoya y que permite la ejecución de un prototipo desde el diseño arquitectónico.
- Se redefinen los requisitos para añadir los aspectos. Para ello se describen y se añaden al sistema incluyéndolos en el diagrama de casos de uso del *sistema básico*. Los aspectos se añaden como "*use case extension*[3]"[5], indicándose los puntos donde interaccionan con los casos de uso del *sistema básico*. Luego se redefinen los diagramas de secuencia asociados a los casos de uso extendidos.

Además, el *sistema extendido* también debe expresarse formalmente en un LDA. Como LEDA no dispone de primitivas para soportar aspectos, se ha definido un juego de instrucciones (*AspectLEDA*) que lo extienden (apartado 3.2). Estas instrucciones se traducen a LEDA, de modo que se puede asegurar que se mantienen las características formales del lenguaje.

[3] Se mantiene la expresión inglesa por ser de uso común

Del estudio realizado a lo largo de los pasos anteriores se obtiene la información necesaria para llevar a cabo la extensión[4].

Finalmente cabe decir que esta descripción del *sistema extendido* se ha obtenido a partir de un modelo arquitectónico abstracto (figura 1) de 2 niveles independientes [7]: *nivel de componente* en el que se sustenta el *sistema básico* (componentes e interacciones); y *nivel de aspecto* que representa un espacio de trabajo independiente del anterior, que contiene los componentes de diseño asociados al aspecto, así como la gestión de su interacción. Esto permite que la evolución de un sistema, siguiendo la propuesta, suponga una modificación mínima sobre la arquitectura del *sistema básico.*

El elemento más interesante del nivel de aspecto es el coordinador que se define asociado a un componente (servidor) y a un aspecto. La ocurrencia de un evento hace posible la ejecución del aspecto si se satisfacen ciertas condiciones. El coordinador actúa sólo si el evento detectado está asociado al aspecto coordinado por él. Su comportamiento se define a partir de la información deducida tras la aplicación de la propuesta[4].

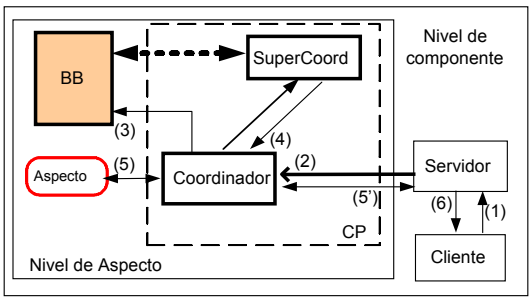

Fig. 1. *Sistema Extendido* Descrip. arquitectónica.

El funcionamiento del coordinador se puede describir del siguiente modo: cuando el coordinador detecta la ocurrencia de un evento, determina si dicho evento está asociado al aspecto cuya ejecución coordina. En este caso, actúan los diversos elementos que componen el nivel de aspecto. El componente de aspecto se ejecuta finalmente o no dependiendo del valor de los *Common Items*[4] y de ciertas reglas que definen el comportamiento del coordinador. Este componente se define siguiendo un modelo de coordinación exógeno orientado a control (Coordinated Roles [9]) y que hace posible la coordinación de dos componentes de modo transparente a ellos.

Tanto los coordinadores, como el supercoordinador (o coordinador de coordinadores que se define para reducir la carga de trabajo a los primeros) forman el proceso de control (CP) que gestiona la extensión del sistema (serán los *extraelements* definidos en el siguiente apartado).

3.2 AspectLEDA

Después de seguir los pasos de la propuesta anterior, considerando las informaciones que se han ido obteniendo[4], el diseño arquitectónico del *sistema*

---

[4]Estas informaciones se almacenan en una estructura denominada *Common Items* y son las siguientes:
Insertion Point –IP- Es cada punto de corte identificado en un diagrama de secuencia. IP debe estar en el interfaz del componente de diseño.
Nombre del *componente* afectado por el IP.
Nombre del componente de *aspecto* a aplicar.
El *tipo de evento* que dispara la aplicación del aspecto.
Las *condiciones* que se deben satisfacer para permitir la ejecución del aspecto.
Una cláusula *When* que indica cuando el aspecto se debe aplicar (antes o después).

extendido se obtiene a partir del conjunto de instrucciones en *AspectLEDA* (cuadro 1). Los elementos de esta descripción se detallan a continuación.

```
Component extendedsystem {
composition
  basicsystem: System;
  aspect: Aspect;
  extraelements: Component;
parameters
  c2:= component_name;
  et:=  eventtype;
  ip(param):=c2operation(param);
  cond_expres[]:= conditions;
  whc:= {before,after};
case et=RMA then coor:=asyn-coor
case et=RMS then coor:=syn-coor;}
```

Cuadro 1. *S. extendido*: descripción en AspectLEDA.

Descripción de `extendedsystem`

El *sistema extendido* se define como un componente formado por:
- Tres componentes: el *sistema básico*, el aspecto a añadir (*aspect)* y otros elementos (*extraelements*),
- Un conjunto de parámetros que se obtienen del estudio detallado del sistema y de la extensión a aplicar.

`basicsystem: System`

La descripción del *sistema extendido* se realiza a partir de la descripción arquitectóni-ca del *sistema básico* (en LEDA). La figura 2 representa 2 componentes de un sistema mayor que forman un cierto *sistema básico*, así como su descripción arquitectónica parcial[5].

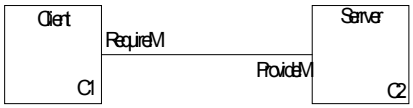

Fig 2a. *Sistema Básico* Rep gráfica parcial.

```
component basicsysten{
        interface none;
        composition
                c1: Client;
                c2: Server;
attachments
c1.requireM(ip,param)<>
c2.provideM(ip,param);
}
component Client{
        interface
           requireM:RequireM;
        }
component Server{
        interface
           provideM:ProvideM;
        }
role ProvideM(ip,param){
  spec is
ip?(answer).(value)answer!(value).
ProvideM(ip,param);
        }
 role RequireM(ip,param) {
  spec is
(answer)ip!(answer).answer?(value).
RequireM(ip,param);
        }
```

Fig 2b. *Sistema Básico* en LEDA

aspect: Aspect

La descripción del *sistema extendido* también incluye el componente de diseño que representa el aspecto. En LEDA, su descripción genérica se muestra en la figura 3. La figura, además de la descripción genérica de alto nivel del componente *aspect,* contiene la descripción del rol que ejecuta[5].

```
component Aspect {
interface
   modify:Modify;
}
role Modify(aspectop,param,ip,x){
spec is
aspectop?(reply).t.(val)reply!(val).Modi
fy(asp ectop,param,ip,x);
}
```

Fig 3. Aspecto en LEDA

`extraelement: Component`

Este componente contiene el conjunto de elementos que es necesario incluir en el *sistema extendido* para poder incorporar el aspecto de forma efectiva. Los elementos que lo forman son fundamentalmente los componentes *coordinadores* que se definen en el modelo, y la descripción de los roles que realiza cada uno de ellos en su interac-

[5] Es un fichero LEDA de entrada al generar el *sistema extendido* en LEDA

ción, entre otros, con los componentes del *sistema básico* o con el aspecto añadido. Figura 4 representa la descripción LEDA de estos elementos[5]. No se detalla su contenido ya que es transparente al arquitecto software que realiza la extensión.

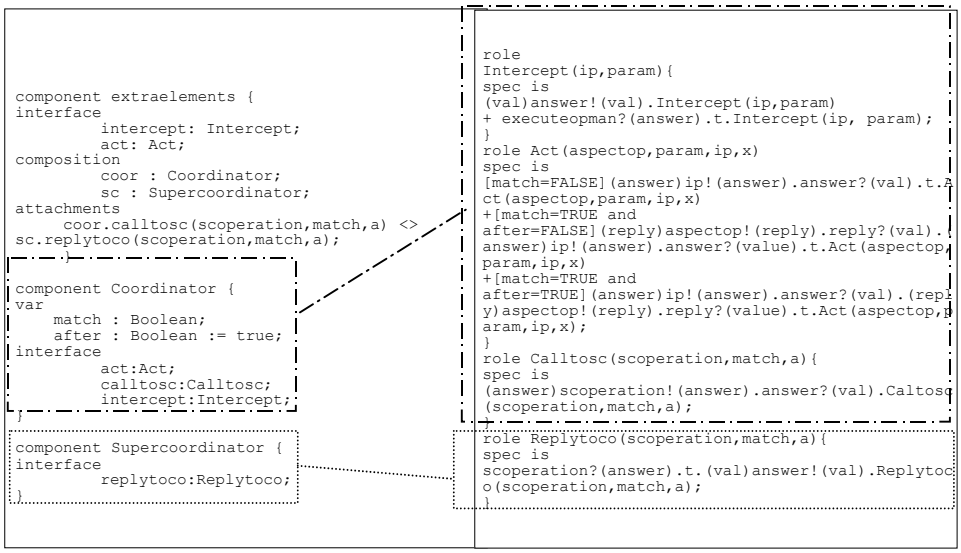

Fig 4. Descripción en LEDA de *extraelements.*

Descripción de `Parameters`

Este apartado de la descripción arquitectónica del *sistema extendido* en *AspectLEDA* contiene los parámetros que hay que considerar para incluir un aspecto en el sistema.

La información de los parámetros se deduce del estudio detallado del sistema que se va a extender, así como de la extensión a incluir, cuando se siguen los pasos del modelo [7] y se almacena en la estructura *Common Items* ya mencionada. No se detalla la descripción de cada uno por razones de espacio.

Obtención del sistema extendido.

Las instrucciones que se acaban de describir amplían LEDA y permiten incorporar aspectos a un sistema existente, considerados como entidades de primera clase, sin modificar los componentes iniciales, y manteniendo el principio de inconsciencia [4].

Un traductor traduce las instrucciones de *AspectLEDA* a LEDA, manteniéndose las ventajas de utilizar este lenguaje, en lo que se refiere a las características formales que lo sustentan. Se obtiene así la descripción arquitectónica del *sistema extendido*, en LEDA (figura 5) que a su vez se traduce a Java.

Descritos los elementos del *sistema extendido,* la relación entre los componentes implicados en la extensión, desde un punto de vista externo se representa según la figura 6a, e internamente, según 6b en la que se representa cómo el coordinador es el que gestiona la asociación del aspecto al componente del *sistema básico* (servidor) en tiempo de diseño.

```
component es {
interface none;
composition
         c1 : Client;
         c2: Server;
         audit :Aspect;
         coor : Coordinator;
         sc : Supercoordinator;
attachments
      c1.requireM(ip,param) <>
coor.intercept(ip,param);
     coor.calltosc(scoperation,match,a) <>
sc.replytoco(scoperation,match,a);
     coor.act(aspectop,param,ip,x) <>
c2.provideM(aspectop,param,x)<>
audit.modify(aspectop,param,ip,x);
}
component Client{
interface
   requireM:RequireM;
}
component Server{
interface
   provideM:ProvideM;
}
component Aspect {
interface
   modify:Modify;
}
component Coordinator {
var
    match : Boolean;
    after : Boolean := true;
interface
         act:Act;
         calltosc:Calltosc;
         intercept:Intercept;
}
component Supercoordinator {
interface
         replytoco:Replytoco;
}
role ProvideM(aspectop,param,ip,x){
spec is
ip?(answer).t.(val)answer!(val).ProvideM(aspectop
,param,ip,x);
}
```

```
role RequireM(aspectop,param) {
spec is
(answer)executeopc2!(answer).answer?(val).t.Requi
reM(aspectop,param);
}
role Modify(aspectop,param,ip,x){
spec is
aspectop?(reply).t.(val)reply!(val).Modify(aspect
op,param,ip,x);
}
role Act(aspectop,param,ip,x){
spec is
[match=FALSE](answer)ip!(answer).answer?(val).t.A
ct(aspectop,param,ip,x)
+[match=TRUE and
after=FALSE](reply)aspectop!(reply).reply?(val).(
answer)ip!(answer).answer?(value).t.Act(aspectop,
param,ip,x)
+[match=TRUE and after=TRUE](answer)executeop
man!(answer).answer?(val).(reply)aspectop!(reply)
.reply?(value).t.Act(aspectop,param,ip,x);
}
role Intercept(ip,param){
spec is
(val)answer!(val).Intercept(ip,param)
+ ip?(answer).t.Intercept(ip, param);
}
role Replytoco(scoperation,match,a){
spec is
scoperation?(answer).t.(val)answer!(val).Replytoc
o(scoperation,match,a);
}
role Calltosc(scoperation,match,a){
spec is
(answer)scoperation!(answer).answer?(val).Caltosc
(scoperation,match,a);
}


instance es:Extemded-System;
```

Fig 5. Descripción LEDA del *sistema extendido*

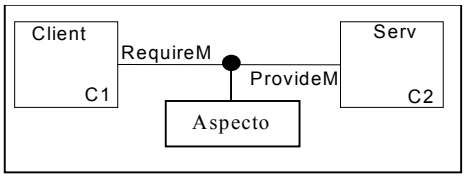

Fig 6a. Relación entre componentes. Punto vista externo.

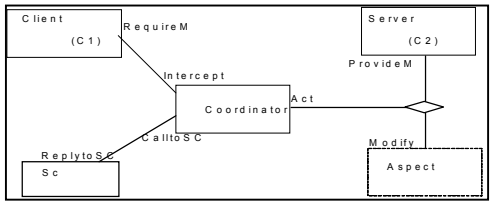

Fig 6b. Relación entre componentes. Punto de vista interno.

4 Trabajos relacionados

La definición de LDA orientados a aspecto es una línea de trabajo que están siguiendo diversos investigadores. En esta sección se mencionan algunos. Todos ellos tienen una concepción diferente de la arquitectura y del modelo de aspectos. Igualmente los lenguajes en los que se apoyan tienen características diferentes.

DAOP [8] es una plataforma dinámica y distribuida para sistemas OA en la que tanto aspectos como componentes se consideran entidades de primera clase que se componen en tiempo de ejecución, mediante un nivel de middleware, que almacena la

información de la arquitectura y de los componentes. La plataforma se apoya en un modelo de componentes (CAM) y un lenguaje de definición arquitectónica (DAOP) orientado a aspectos y basado en XML.

En [1] se define un *framework* extendiendo un modelo de componentes y un LDA basado en XML con conceptos de OA. Presenta una aproximación arquitectónica de tres niveles para realizar la evolución del software basándose en la separación entre computación, coordinación y configuración. Se define una primitiva de modelado para encapsular las interacciones entre componentes de modo transparente a ellos.

En [6] se expresa una arquitectura OA considerando un modelo de coordinación exógeno, y como LDA una extensión de Rapide (AO-Rapide). En este caso, los aspectos son conectores arquitectónicos, no componentes.

5 Conclusiones y cuestiones abiertas

En este artículo se presenta un LDA orientado a aspectos (*AspectLEDA*), basado en otro (LEDA) con una fuerte base formal. La definición del lenguaje se apoya en un modelo arquitectónico [7] que permite incorporar aspectos observando el principio de inconsciencia [4].

Los sistemas ya desarrollados necesitan evolucionar y adaptarse al mundo que representan. El modelo que se referencia [7] y el LDA-OA que se propone podrían utilizarse durante la evolución de los sistemas si los nuevos requisitos se pudieran considerar como aspectos. Cuándo y qué nuevos requisitos se pueden considerar como aspectos arquitectónicos son cuestiones abiertas para la discusión.

Referencias

[1] Andrade, L. et al. Separating Computation, Coordination and Configuration. Formal Foundations of Software Evolution WS. Software Maintenance and Re-engineering Conf. Portugal 2001

[2] Canal, C, Pimentel, E., Troya, M. Compatibility and Inheritance in Software Architecture. Review Scene and Computing Programming. Vol. 41, nº2. 2001 105-130

[3] Early Aspects net http://www.early-aspects.net

[4] Filman, R. E., Friedman, D. P. Aspect-Oriented Programming is Quantification and Obliviousness Workshop on Advanced Separation of Concerns. OOPSLA 2000, USA

[5] Jacobson, I. Weing, P. Aspect-Oriented software Development with Use Cases. Addison Wesley 2005.ISBN 0321268881

[6] Palma, K. Using a coordination model to specify aspect-oriented software architectures. Santiago, Chile, 2004. Master Tesis Ciencias de la Ingeniería. Pontif U. Católica de Chile.

[7] Navasa, A. Perez, M. A., Murillo, J. M. Aspect Modelling at Architecture Design. Second European Workshop on Software Architecture (EWSA 2005). Pisa. Italia. Junio 2005. LNCS vol nº3525 pp 41-58. Eds R. Morrison, F. Oquendo. Springer Verlag (2005)

[8] Pinto, M., et al. Separation of Coordination in a Dynamic Aspect Oriented Framewrok. AOSD Conf.. Enschede. Netherlands. 2002

[9] Murillo,J. M. et al. Coordinated Roles: Promoting Re-Usability of Coordinated Active Objects Using Event Notification Protocol. Coordination Proceeding, pp53-68, 1999

.