Implementación Orientada a Aspectos en C# de un Sistema Robótico Tele-operado[1]

Rafael Cabedo, Jennifer Pérez, Nour Ali, Isidro Ramos, Jose Ángel Carsí

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46071 Valencia, España
{rcabedo | jeperez | nourali | iramos | pcarsi }@dsic.upv.es

Resumen. El desarrollo de software basado en componentes y orientado a aspectos esta despertando un gran interés, no sólo en el ámbito académico, sino también en el industrial. El modelo prisma se caracteriza por la combinación que realiza del Desarrollo de Software Basado en Componentes (DSBC) y el Desarrollo de Software Orientado a Aspectos (DSOA) y por su metanivel que permite la evolución de sus arquitecturas. El modelo PRISMA ha sido aplicado a un caso de estudio de ámbito industrial, el robot TeachMover. Se eligió este caso de estudio con el objetivo de abordar un sistema real que permitiera comprobar que realmente el modelo reúne los requisitos necesarios para la construcción de arquitecturas software complejas, distribuidas, dinámicas y reutilizables. Concretamente, este trabajo presenta la implementación en lenguaje C# de los elementos arquitectónicos, aspectos y *weavings* (tejidos, sincronizaciones) entre aspectos del robot mediante el uso del *middleware* de PRISMA, llamado PRISMANET. La implementación del robot ha dado como resultado una aplicación orientada a aspectos sobre .NET que permite mover a un sistema robótico tele-operado de una forma segura.

1. Introducción

En la construcción de sistemas software actuales, cada vez cobran más relevancia características que no forman parte del proceso de negocio. Este cambio, junto con el hecho de que es necesario dar un soporte a la evolución de los sistemas software de la forma más dinámica y menos costosa posible, ha hecho plantearse nuevos enfoques de desarrollo software como el propuesto por PRISMA.

PRISMA [3] es un enfoque de desarrollo de software cuyo modelo combina el Desarrollo de Software Basado en Componentes (DSBC) [8] y el Desarrollo de Software Orientado a Aspectos (DSOA) [1]. El modelo da la misma relevancia tanto a

[1]Este trabajo ha sido financiado por el CICYT (Centro de Investigación Científica Y Tecnológica), Proyecto DYNAMICA (DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures), TIC 2003-07804-C05-01. Además, su desarrollo ha sido financiado por Microsoft Research Cambridge, proyecto "PRISMA: Model Compiler of Aspect-oriented component-based software architectures".

requisitos funcionales como a requisitos no funcionales. Esto se consigue especificándolos de la misma manera mediante el concepto de aspecto. Por otro lado, la orientación a aspectos aporta simplicidad en el proceso de evolución, ya que cuando se desean cambiar propiedades que afectan a un solo interés del producto software, solo es necesario considerar las propiedades del aspecto que especifica el interés (*concern*) que se desea evolucionar.

Además, la combinación del DSBC y el DSOA permite tener diferentes niveles de reutilización y adaptación que permiten mejorar tanto los tiempos de desarrollo como los de mantenimiento.

Dicho enfoque está siendo desarrollado sobre la plataforma .NET [6]. Si bien es cierto, la tecnología .NET no proporciona directamente la programación orientada a aspectos. Por este motivo, ha sido necesario desarrollar un *middleware*, llamado PRISMANET, que proporciona todas estas funcionalidades necesarias para la ejecución de arquitecturas software PRISMA sobre la plataforma .NET.

Se ha especificado [5] e implementado un caso de estudio real haciendo uso del enfoque PRISMA, el robot TeachMover. El objetivo de este trabajo es mostrar la implementación de los aspectos de las componentes del robot y cómo se han implementado los *weavings* (tejidos, sincronizaciones) entre aspectos, para finalmente dar como resultado una aplicación C# que permite el movimiento del robot con una implementación orientada aspectos siguiendo el modelo PRISMA y sobre tecnología .NET.

La estructura del trabajo es la siguiente: en primer lugar se presenta brevemente el modelo PRISMA y su *middleware* PRISMANET. En el apartado 3, se presenta el caso de estudio real del TeachMover. En el apartado 4, se presenta la implementación de aspectos y *weavings* haciendo uso de PRISMANET. Finalmente, en el apartado 5 se presentan las conclusiones y trabajos futuros.

2. PRISMA sobre PRISMANET

El modelo PRISMA [3] permite definir arquitecturas de sistemas software complejos. Principalmente, se caracteriza por la integración que realiza del DSOA y del DSBC. Esta integración se realiza definiendo las características relevantes (distribución, coordinación, seguridad, etc.) (*concerns*) de un elemento arquitectónico como aspectos.

Un aspecto PRISMA es un *concern* que es común y compartido por el conjunto de tipos del sistema (*crosscutting concerns*). Los tipos de aspectos que forman un elemento arquitectónico varían dependiendo del sistema software.

Un elemento arquitectónico PRISMA (componente o conector) está formado simétricamente [2] por un conjunto de aspectos de distinto tipo, las relaciones de sincronización entre estos aspectos (*weavings*) y uno o más puertos o roles. Los puertos se utilizan para definir componentes y los roles para definir conectores. Estos representan los puntos de interacción entre los elementos arquitectónicos. El *weaving* establece las sincronizaciones necesarias entre los distintos aspectos que conforman un elemento arquitectónico. El *weaving* define que la ejecución de un servicio de un aspecto puede generar la invocación de un servicio de otro aspecto.

Uno de los objetivos de PRISMA es el de generar automáticamente el código de sus arquitecturas software orientadas a aspectos sobre la plataforma .NET. Para ofrecer aquellas características PRISMA que .NET no ofrece de forma directa, ha sido desarrollado un *middleware* llamado PRISMANET [4]. Dicho *middleware* permite la ejecución de arquitecturas dinámicas y orientadas a aspectos PRISMA. El *middleware* implementa el modelo PRISMA mediante un conjunto de clases base que se corresponden con los conceptos PRISMA. De este modo, el desarrollo de una aplicación PRISMA específica se realiza mediante la extensión de dichas clases.

3. CASO DE ESTUDIO: Robot TeachMover

Con el objetivo de validar el lenguaje de descripción de arquitecturas de PRISMA, se ha realizado la especificación e implementación de un caso de estudio real. Este caso de estudio se basa en el control de un brazo robot conocido como TeachMover [7]. Este robot está formado por un conjunto de articulaciones (*Base, Shoulder, Elbow, Wrist*) y una pinza que posibilitan la recogida, el desplazamiento y la deposición de objetos.

Cada una de las articulaciones está compuesta por un actuador y un sensor. Los actuadores son los encargados de mandar acciones a las articulaciones o a la herramienta del robot, para que procesen la acción que éstos especifican. Mientras que los sensores realizan la lectura del resultado de dichas acciones, para saber si éstas se han ejecutado correctamente. La función del robot es la de moverse, bien sea por pasos simples o especificando la posición en grados que debe alcanzar una articulación. Además, se debe de realizar un control antes de cada movimiento puesto que las articulaciones sólo pueden desplazarse entre un rango predeterminado. Así, se comprueba que los movimientos son seguros, que pueden ejecutarse y que ni el robot, ni cualquier otro elemento que forme parte del entorno corre peligro.

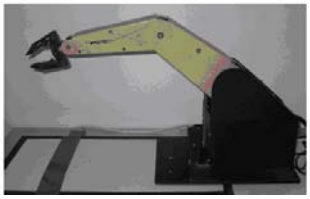

Fig. 1. Robot TeachMover

Concretamente, en este artículo se va a utilizar como ejemplo la implementación que se ha realizado del aspecto de seguridad de cada una de las articulaciones. A continuación se muestra parcialmente su especificación:

```
Safety Aspect SMotion
  Attributes
    Constant
     minimum: integer, NOT NULL;
```

```
    maximum: integer, NOT NULL;
  Services    ... ...
    in check(input Degrees: integer, output Secure: boolean);
    Valuations
    {(Degrees >= minimum) and (Degrees <= maximum)}
    [check(Degrees, Secure)] Secure := true};
    {(Degrees < minimum) or (Degrees > maximum)}
    [check(Degrees, Secure)] Secure := false;
  Protocol ... ...
End_Safety Aspect SMotion;
```

4. IMPLEMENTACIÓN

En PRISMA, un aspecto se implementa como una clase de C# definida dentro de un ensamblado para que sea totalmente reutilizable. Para dar soporte a la implementación de los aspectos se emplean las clases definidas en PRISMANET [4]. Estas proporcionan la funcionalidad genérica y común para todos los aspectos. En PRISMA existen un número indefinido de tipos de aspectos (coordinación, seguridad...), cada tipo tiene ciertas propiedades que lo caracterizan pero comparte una funcionalidad común con el resto de aspectos. En PRISMANET existe una clase para cada tipo de aspecto, estas clases extienden la clase raíz *AspectBase* en la que está definida la funcionalidad común. Para definir un aspecto concreto se debe extender por herencia una clase que representa un tipo de aspecto. En la implementación del caso de estudio, el aspecto de seguridad *SMotion* extiende la clase *SafetyAspect* de PRISMANET. La figura 2 muestra los distintos tipos de aspecto que se han utilizado en el caso de estudio. Concretamente, la figura muestra cómo se define el aspecto de seguridad *SMotion* extendiendo la clase *SafetyAspect* de PRISMANET. El propósito del aspecto *SMotion* es el de asegurar que los movimientos que se solicitan a las articulaciones del robot son seguros.

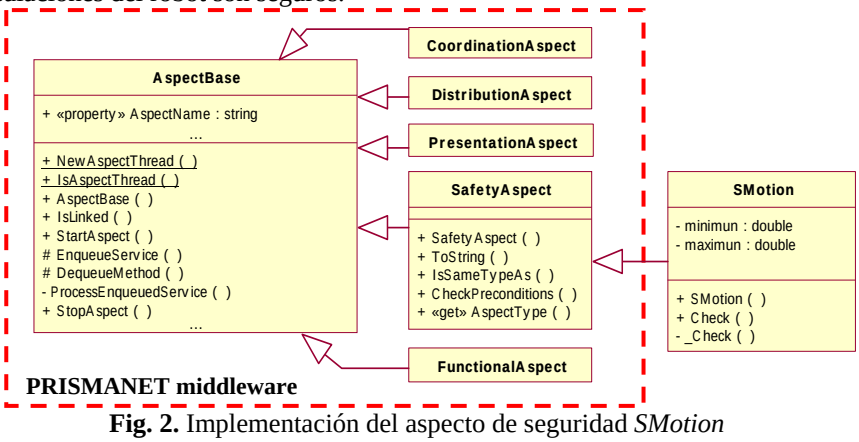

Fig. 2. Implementación del aspecto de seguridad *SMotion*

Los mecanismos heredados de *AspectBase* permiten a los aspectos ejecutarse concurrentemente y de forma asíncrona a través de un hilo de ejecución propio y de una cola, donde se almacenan las invocaciones dirigidas a los servicios del aspecto.

En implementación, los atributos de los aspectos se implementan como variables privadas de clase, por ejemplo los atributos *minimun* y *maximun* son los valores máximo y mínimo en grados que el movimiento de una articulación puede alcanzar.

Los servicios de los aspectos se implementan a través de dos métodos, uno público y otro privado. El método privado implementa el comportamiento definido en el servicio y el método público da soporte a la ejecución de los servicios de forma asíncrona. Cuando se solicita un servicio de un aspecto, se invoca el método público. Éste encola, en forma de delegado, la petición del método privado en la cola de petición de servicios (e*nqueueService*). De esta forma, cuando el hilo de ejecución del aspecto no esté procesando ninguna otra petición, este desencolará la petición y ejecutará el método privado asociado. En el aspecto *Smotion*, el servicio *Check* toma como parámetro los grados en los que se va a situar una articulación y devuelve si el ángulo es válido o no. Este comportamiento se ha implementado del siguiente modo:

```
public AsyncResult Check(double newAngle) {
   return EnqueueService(cinematicsMoveJointDelegate,newAngle);}
private AsyncResult _Check(double newAngle, out bool secure){
   if ((newAngle >= minimun) && (newAngle <= maximun))
      secure = true;
   else secure = false;}
```

Los aspectos son instanciados y agregados dinámicamente dentro de los elementos arquitectónicos. En implementación, un componente o un conector se han traducido por una clase que extiende la clase base *ComponentBase* de PRISMANET. Esta clase, además de integrar en su interior los aspectos y los puertos de comunicación, también realiza dinámicamente los *weavings* entre los aspectos. Se ha de tener en cuenta, que no sólo se da soporte a la agregación de aspectos y weavings a un componente en tiempo de carga, también es posible realizar su composición dinámica de aspectos o weavings emergentes en tiempo de ejecución. Si bien es cierto, dicha agregación dinámica no es persistente porque el middleware todavía no da soporte a la emisión de código en tiempo de ejecución.

En el caso de estudio existe un conector que se encarga de llevar a cabo la coordinación de los componentes Actuador y Sensor a través de su aspecto de coordinación. El componente Actuador es el que envía los comandos de movimiento al robot y el componente Sensor es el que recupera, después del movimiento, si se ha efectuado correctamente. Mediante el servicio *cinematicsMoveJoint* del aspecto de coordinación, el conector indicará al Actuador que efectúe un movimiento. Pero, antes de que se efectúe el movimiento es necesario comprobar que es seguro. Por este motivo, el aspecto de seguridad *SMotion* se añade al conector cuando este es construido.

En PRISMANET existe un conjunto de clases para dar soporte a los *weavings*. Un *weaving* se asocia a un servicio origen de algún aspecto del elemento arquitectónico. Este servicio desencadenará la invocación de servicios destino de otros aspectos del elemento. La asociación entre un servicio origen y un servicio destino se establece a

través de una lista dinámica de delegados. El servicio origen puede tener asociados hasta tres tipos de *weavings* distintos: *after*(o *afterIf*), *before*(o *beforeIf*) e *instead*.

Cuando se invoca el servicio de un aspecto, si tiene *weavings* asociados, la dependencia de los servicios requiere que el *weaving* se ejecute de forma síncrona, por lo que su ejecución se realiza en un hilo de ejecución independiente.

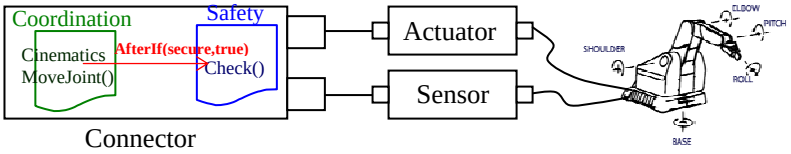

Fig. 3. *Weaving* entre el aspecto de seguridad y el aspecto de coordinación

Mediante el *weaving* se define que el servicio *cinematicsMoveJoint* se ejecutará después del servicio *Check* si el movimiento es seguro. Es decir, la invocación de *cinematicsMoveJoint* del aspecto de coordinación desencadena la ejecución (*weaving*) del servicio *Check* definido en el aspecto de seguridad (ver figura 3). De esto modo, los *weavings* hacen de sincronizadores entre los distintos aspectos del componente que se están ejecutando concurrentemente.

La definición de un *weaving* se realiza en el cuerpo de constructor del elemento arquitectónico. Su definición se realiza mediante el método *AddWeaving*, codificado en la superclase que da soporte a los elementos arquitectónicos de PRISMANET, y requiere como argumentos el nombre del servicio origen y la instancia del aspecto donde está definido el servicio, el nombre del servicio destino y la instancia del aspecto donde está definido, así como el tipo de *weaving*. A continuación se muestra la instrucción que asocia el *weaving* entre el aspecto de seguridad y el aspecto de coordinación del conector.

```
AddWeaving(coordinationAspect,"CinematicsMoveJoint",
WeavingType.AFTERIF("secure",true),safetyAspect,"Check");
```

5. CONCLUSIONES Y TRABAJOS FUTUROS

El trabajo presenta una introducción de como han sido implementados los aspectos y *weavings* del modelo PRISMA mediante C#. NET. Para ello, se ha utilizado como ejemplo el caso de estudio de un sistema robótico tele-operado, el robot TeachMover. El resultado de este trabajo ha permitido ejecutar una arquitectura orientada a aspectos sobre la plataforma .NET que ha posibilitado el movimiento del robot TeachMover.

A corto plazo, este trabajo va a servir para la identificación de los patrones de generación automática de código. Dichos patrones se utilizarán para desarrollar el compilador de modelos PRISMA a lenguaje C#.

Bibliografía

[1] Aspect-Oriented Software Development, http://aosd.net
[2] Cuesta, C. E., Romay, P., De La Fuente, P., Barrio-Solórzano, M., Architectural Aspect of Architectural Aspects. Second European Workshop on Software Architecture (EWSA'05), Lecture Notes in Computer Science 3527, Pisa, June 2005.
[3] Pérez J., Ali, N., Carsí J.A., Ramos I., Dynamic Evolution in Aspect-Oriented Architectural Models. Second European Workshop on Software Architecture (EWSA'05), Lecture Notes in Computer Science 3527, Pisa, June 2005.
[4] Pérez J., Ali N. , Costa C., Carsí J.A., Ramos I., Executing Aspect-Oriented Component-Based Software Architectures on .NET Technology, Journal of .NET Technologies, Vol.3, No.1-3, 2005 ISSN 1801-2108.
[5] Pérez J., Ali N., Carsí J.A., Ramos I.,"Arquitectura PRISMA para el Caso de Estudio Robot 4U4", DSIC-II/13/04, Universidad Politécnica de Valencia, 2004.
[6] Robinson S. et al. , Professional C# 2nd Edition, Wrox Programmer to Programmer.
[7] TeachMover Robot, http://www.questechzone.com/microbot/teachmover.htm
[8] Szyperski, C., *Component software: beyond object-oriented programming*, (New York, USA: ACM Press and Addison Wesley, 2002).