ICT-UNPA-42-2012
ISSN: 1852-4516
Aprobado por Resolución Nro. 0375/12-R-UNPA

MACS: herramienta para la gestión de Conexiones Aspectuales

Estrategias para la conexión e integración de reglas de negocio con aspectos

Cecilia Fuentes y Sandra Casas

Universidad Nacional de la Patagonia Austral

Unidad Académica Río Gallegos

Campus Universitario, Av. P. Rivero S / N

9400 - Río Gallegos, Argentina

cfuentes277@gmail.com

1. INTRODUCCIÓN

Las reglas de negocio son elementos atómicos que componen el marco estructural, la política, la estrategia y la operatividad de una empresa u organización. Es decir que reflejan las políticas de negocio, que tienen como finalidad alcanzar los objetivos del negocio, satisfacer a los clientes, hacer un buen uso de los recursos, y respetar las leyes o convenciones de la empresa u organización [1]. Pueden encontrarse en todo tipo de aplicaciones tales como finanzas y seguros, negocios electrónicos, transportes, servicios basados en Web y personalización; telecomunicaciones entre otros. Las reglas de negocio son dinámicas, están sujetas a cambios en el tiempo. Estos cambios constantes exigen que la implementación de la regla de negocio este separada de la lógica central del negocio para responder a este entorno dinámico.

Las principales estrategias de implementación de reglas de negocio, Patrones de Diseño [2] y Sistemas Manejadores de Reglas de Negocio [3], se abocan a hacer las reglas explícitas y separadas de la funcionalidad central. Esto permite la separación de las reglas del dominio, sin embargo, este tipo de implementación requiere código que conecte e integre a la regla de negocios de la funcionalidad central. En consecuencia se producen inconvenientes para la reutilización y mantenimiento de estos componentes bases, además de convertir al dominio en volátil, dado que un simple cambio en la interfaz con la regla implica modificaciones en el código de la conexión.

La Programación Orientada a Aspectos (AOP) [4] es un paradigma que permite encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando la dependencia entre cada uno de los módulos. Con esto se consigue evitar la dispersión del código y módulos cuyo código responde a diversas funciones, y que las implementaciones sean más comprensibles, adaptables y reusables.

La AOP ha sido propuesta para mejorar la integración de reglas de negocio con la funcionalidad central OO [5] [6][7][8], estos trabajos se enfocan en la implementación de las conexiones con aspectos, indicando las dificultades y posibilidades que los diferentes lenguajes AOP ofrecen. El uso de aspectos para conectar reglas de negocios, es lo que denominamos “conexiones aspectuales”. En el diseño e implementación de las mismas, hemos observado, principalmente en aquellas aplicaciones que requieren importante cantidad de reglas de negocio, que hacen falta técnicas para facilitar el desarrollo y mantenimiento de las mismas.

En nuestros trabajos previos hemos aportado una serie de estrategias con el objeto de mejorar el manejo de las conexiones aspectuales a los desarrolladores. Se han provisto medios sistemáticos para documentar, analizar, clasificar las conexiones aspectuales y generar código [9][10][11]. El presente trabajo rescata estos aportes y presenta MACS, una herramienta que automatiza estas estrategias, además de incorporar mecanismos de mapeo automático a código AspectJ [12].

La estructura de este informe se encuentra conformada en base a los siguientes elementos que se detallan: la Sección 2 corresponde a una introducción a Programación orientada a aspectos. En la Sección 3 se presentan estrategias para analizar y clasificar las conexiones aspectuales para componer reglas de negocio. En la Sección 4 se muestra la plantilla de conexiones aspectuales y la definición de cada una de sus componentes. La Sección 5 es un detalle de las

operaciones y funciones de la herramienta. Las reglas para generar el código de los aspectos se exhibe en la Sección 6. La sección 7 corresponde al diseño y las vistas de la herramienta. En la Sección 8 se expone el caso de estudio en cuestión, las reglas de negocio y las conexiones referentes al mismo, además del diagrama y las vistas de algunas de las operaciones que se pueden realizar con Macs. En la Sección 9 se presentan los trabajos relacionados. Y por último las conclusiones se detallan en la Sección 10.

2. PROGRAMACIÓN ORIENTADA A ASPECTOS

Una aplicación de software está compuesta por un conjunto de "*concern*" (asunto, competencia, incumbencia). En el desarrollo de software un concern será un servicio o una funcionalidad del sistema que puede ser identificada y limitada. Los concerns pueden ser de diferente naturaleza y especie: temas referidos a las reglas de negocio, la interfaz de usuario, la seguridad, la persistencia, el registro de actividades, las comunicaciones, etc. son algunos ejemplos de concerns.

Las herramientas de desarrollo (lenguajes de programación, notaciones de diseño, etc.) están centradas en la modularización de unidades funcionales (clases, objetos, funciones, etc.), por lo tanto, los demás concerns entrecruzan y atraviesan las unidades funcionales para adicionar comportamiento que responde principalmente a requerimientos de diseño e implementación. Los mecanismos de abstracción e implementación de las herramientas de desarrollo convencionales no soportan la separación de aquellos concerns diferentes a la funcionalidad básica. De esta manera, el código resultante de cada módulo responde a varios objetivos y se torna confuso; simultáneamente el comportamiento de los concerns esta disperso y duplicado en distintos módulos de la aplicación. Las desventajas más significativas son: código de mala calidad, trazabilidad pobre, baja productividad, baja reusabilidad, dificultades en el testing, adaptabilidad mala y evolución pobre. [13]

La AOP [4][14] permite a los programadores codificar estas propiedades técnicas, que se entrecruzan en unidades llamadas aspectos, separadas de la funcionalidad básica. Luego un proceso de composición combina los aspectos y las unidades funcionales generando la aplicación final. Los beneficios que se obtienen al aplicar una mayor modularización es que las aplicaciones de software resultan más fáciles de diseñar, codificar, mantener y reusar, superando los problemas de código mezclado y código diseminado [6].

Los Lenguajes de Programación Orientados a Aspectos (AOL) son extensiones de lenguajes de programación existentes que incorporan los mecanismos necesarios para dar soporte a los aspectos. Un proceso de tejido (realizado por un nuevo tipo de compilador o intérprete) combina los componentes de funcionalidad básica con los aspectos, para generar la aplicación ejecutable [15].

El tejedor permite que en ciertos puntos de la ejecución de los componentes funcionales se inserte el código de los aspectos. Estos puntos se denominan puntos de unión (join points) y podrían interpretarse como eventos que al ocurrir, activan la ejecución de un aspecto. Desde la perspectiva de los aspectos, éstos incluyen puntos de cortes (pointcuts) que se asocian a los puntos de unión del código funcional para adicionar cierto comportamiento (advise).

En la mayoría de los AOL, es posible definir entidades de primera clase que representen a los aspectos. Esta construcción es, una clase o una entidad muy parecida a una clase, en esencia es una unidad de código con un nombre y con variables y métodos propios.

2.1 ASPECTJ

Actualmente el AOL más popular, maduro y estable es AspectJ [12]. AspectJ es una extensión compatible de Java TM orientada a aspectos y de propósito general, con una nueva clase de módulos llamado *aspectos*. Los aspectos pueden interceptar las clases, las interfaces y a otros aspectos. Mejora la separación de competencias permitiendo localizar de forma aislada los conceptos de diseño del corte. A diferencia de las clases, no se pueden crear instancias de los aspectos. Un aspecto es una clase, exactamente igual que las clases Java, pero con la diferencia que pueden contener constructores de corte, que no existen en Java, como se muestra en la Figura 1.

Figura 1. Componentes de un aspecto en *AspectJ*.

Los *Aspects* (aspectos) son constructores que trabajan al cortar de forma transversal la estructura de las clases de forma limpia y cuidadosamente diseñada.

Pointcuts : también llamados cortes, capturan colecciones de eventos en la ejecución de un programa. Estos eventos pueden ser invocaciones de métodos, invocaciones de constructores y señalización y gestión de excepciones. Los cortes no definen acciones, simplemente describen eventos. Un *Pointcut* está formado por dos partes, separadas ambas por dos puntos. En la parte izquierda (a) se define el nombre del corte y el contexto del corte. La parte derecha, del Ejemplo 1, (b,c) define los eventos de corte.

```
pointcut corte(): call(void Line.moveBy(int, int));
```
(a) (b) (c)

Ejemplo 1.

Los cortes se utilizan para definir el código de los aspectos utilizando avisos. A los descriptores de eventos de la parte derecha de la definición del corte se les llama designadores. Un designador puede ser: Un método o Un constructor o Un manejador de excepciones. Se pueden componer utilizando los operadores o (“||”), y (“&”) y no (“!”).

Advices: define la asociación de los "*pointcuts*" para insertar la implementación del aspecto que se ejecuta en esos puntos bien definidos. Estos puntos pueden identificarse por cortes con nombre (b), en el Ejemplo 2 o por cortes anónimos.

```
   (a)          (b)
before() : corte() {

System.out.println("call-before sobre Line.moveBy()");(c)
}
```

Ejemplo 2.

El cuerpo de un *advice* se puede añadir en distintos puntos del código(a), cada uno de los cuales se define de mediante alguna de las siguientes palabras clave: **befote (**se ejecuta justo antes de que lo hagan las acciones asociadas con los eventos del corte) ; **alter (**se ejecuta justo después de que lo hayan hecho las acciones asociadas con los eventos del corte); **catch (**se ejecuta cuando durante la ejecución de las acciones asociadas con los eventos definidos en el corte se ha producido una excepción del tipo definido en la apropiada cláusula catch); **finally (**se ejecuta justo después de la ejecución de las acciones asociadas con los eventos del corte, incluso aunque se haya producido una excepción durante la misma); **around (**atrapan la ejecución de los métodos designados por el evento. La acción original asociada con el mismo se puede invocar utilizando thisJoinPoint.runNext( )).

Introducciones y declaraciones: Las introducciones "*Introduction*" se utilizan para añadir elementos completamente nuevos, permitiendo modificar la estructura de una clase dada. Es decir afecta la estructura estática de las clases. Entre los elementos que se pueden añadir se encuentran: Un nuevo método a la base; Un nuevo constructor; Un atributo; Varios de los elementos anteriores a la vez.; Varios de los elementos anteriores en varias clases.

3. TAXONOMIA DE CONEXIONES ASPECTUALES

Una conexión aspectual entre una regla de negocio y la funcionalidad central es todo mecanismo de implementación que permite encapsular en uno o mas aspectos: la invocación al objeto que representa a la regla de negocio, la obtención y transmisión de la información que requiere la regla de negocio, la resolución de la interacción ante la posible simultaneidad de aplicación de más de una regla y el retorno de la información generada por la aplicación de la misma [9]. Hemos analizado diversas situaciones en las cuales reglas de negocio y lógica de negocio son conectados con aspectos, y hemos identificado claramente cuatro tipos de conexiones aspectuales: básica, consulta, cambio y compleja.

Conexión aspectual básica*:* la conexión activa la regla de negocio en un punto específico de la funcionalidad principal (evento), la información requerida por la regla de negocio está disponible en el contexto de eventos o es información global del sistema. Este tipo de conexión requiere conocimiento acerca de la regla de negocios, el evento que dispara la regla de negocio.

Conexión aspectual de consulta: la conexión activa la regla de negocio en un punto específico de la funcionalidad principal, pero la información requerida por la regla de negocio

no está disponible en el contexto del evento. Entonces la conexión primero debe recuperar la información a fin de que esté disponible cuando la regla de negocio se aplique. En este caso, la conexión aspectual debe gestionar dos eventos (pointcuts) y dos advices, cada uno con un propósito diferente. Esta conexión requiere el mismo conocimiento que la conexión básica y además información acerca del evento del cual debe recuperar la información que la regla demanda.

Conexión aspectual de cambio: esta conexión debe agregar nuevas propiedades (campos / métodos) a los componentes de las funciones principales, a fin de que la regla de negocio se pueda disparar. Esto significa que la regla de negocio nueva requiere adaptar el vocabulario del dominio. La conexión debe soportar la adaptación de dominio, tal como la adición de nuevos campos y métodos en las clases existentes. La descripción de la conexión de cambio incluye los mismos elementos que la conexión básica y se le incorporan información de las propiedades que debe adicionar al dominio.

Conexión aspectual compleja: Esta conexión tiene las mismas características que la conexión de consulta y la de cambio. La conexión tiene que actualizar el dominio para que nuevas reglas de negocios sean aplicadas, pero la información necesaria para la condición de la regla de negocio no está disponible en el contexto del evento que desencadena la regla de negocio. Requiere los mismos conocimientos que las conexiones de consulta y de cambio.

4. PLANTILLA DE CONEXIONES ASPECTUALES

Inicialmente la plantilla conexiones aspectuales (PCA) fue elaborada con el objeto de identificar los elementos necesarios para el posterior diseño e implementación de los aspectos. Sin embargo, la planilla resulto ser un artefacto probo para aplicar la taxonomía en forma automática, identificar potenciales interacciones, y como se verá más adelante para la generación del código en forma automática. En la Figura 2 se presenta la plantilla, y a continuación se describen las secciones identificadas con letras.

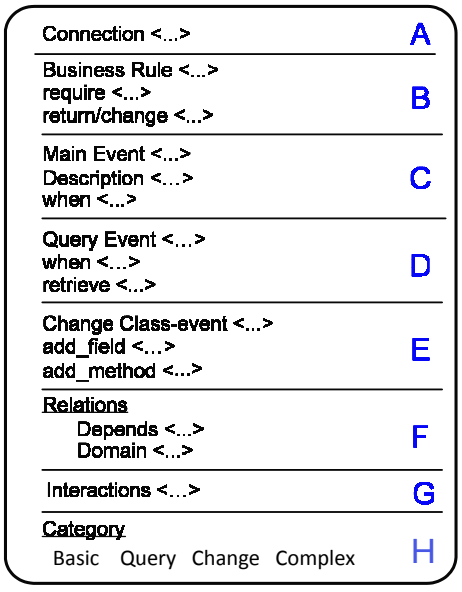


Figura 2. Plantilla de conexiones Aspectuales

A: Identificación de la conexión aspectual.

B: Elementos de la reglas de negocio, identificación de la regla que se debe disparar, se explicita el identificador de la clase y método que la encapsula, los argumentos que requiere y lo que la regla de negocio retorna. Se deberá explicitar si se retorna un nuevo valor u objeto o se modifica el estado de los objetos pasados a la regla de negocio como argumentos.

C: La clase y método que representan el evento que dispara la regla de negocio, una descripción textual y el momento en cual la regla debe ser disparada respecto de la ejecución del evento (before/after/around).

D: Esta sección debe ser completada sólo en aquellos casos que la información requerida por la regla de negocio no está disponible en el contexto del evento principal (C), por lo que en este apartado se documenta el evento que debe ser interceptado y la información que del mismo debe ser recuperada.

E: Esta sección debe ser completada en aquellas situaciones en el que el dominio debe ser adaptado, primero para que se aplique una determinada regla de negocio (B) que configure las nuevas propiedades (estados y comportamientos), y luego para que otras reglas de negocio puedan ser aplicadas en función de estas nuevas propiedades adicionadas.

F: Cuando la aplicación de una determinada regla de negocio es un requisito para que otra/s puedan ser aplicadas, se especifican las relaciones entre las conexiones aspectuales en este apartado.

G: Se completa con el identificador de aquella/s conexiones aspectuales con las que se establece una interacción, es decir, dos o más conexiones diferentes están asociadas al mismo evento, declarado en la secciones C o D.

H: Corresponde a la clasificación de las conexiones aspectuales será: si se han completado los apartados A, B y C, es una conexión básica; si se ha completado la sección A, B, C y D es una conexión de consulta, si en cambio se han completado los apartados A, B, C y E es de cambio y si se han completado las secciones A, B, C, D y E se trata de una conexión compleja.

5. GESTION DE CONEXIONES ASPECTUALES

La gestión de las conexiones aspectuales es una tarea ineludible y crítica cuya responsabilidad recae totalmente en los desarrolladores. Lo cual se complica en aquellas aplicaciones software grandes, que encarnan diversas políticas organizaciones, y por ende la cantidad de reglas de negocios a diseñar, implementar, y que constantemente se deben modificar o cambiar por otras, desborda los esfuerzos convencionales. En este sentido, entendemos que la gestión refiere al conjunto de operaciones y funciones que asisten al desarrollador y facilitan la tarea de desarrollo y posterior mantenimiento, de manera tal de minimizar los costos, acortando los tiempo. Paralelamente, la gestión indica una forma sistemática de realizar acciones, que aunque no sea rígida, tampoco puede ser diferente para cada aplicación, en este sentido se busca aplicar un método en cierta medida repetible.
A partir de la automatización de las PCA, es posible aplicar diversas funciones y operaciones, en un momento previo a su implementación. Así, disponibles las conexiones aspectuales en un contenedor (repositorio), y descriptas según la PCA, encontramos viables y útiles las funciones y operaciones indicadas en la Tabla 1, cuya formalización hemos presentado [11].

Tipos	Operación	Descripción
Actualización	`Add`	Añade una nueva conexión al contenedor
	`Remove`	Elimina una conexión desde el contenedor
Consultas	`BCL (Basic Connection List)`	Retorna la lista de conexiones básicas existente en el contenedor
	`QCL (Query Connection List)`	Retorna la lista de conexiones de consulta existente en el contenedor
	`CHCL (Change Connection List)`	Retorna la lista de conexiones de cambio existente en el contenedor
	`CCL (Complex Connection List)`	Retorna la lista de conexiones complejas existente en el contenedor
	`BRL (Business Rules List)`	Retorna la lista de elementos de reglas de negocio en conexión
	`EL (Event List)`	Retorna la lista de eventos en conexión
	`CRL (Connection Relations List)`	Retorna lista de relaciones de dependencia de en la lista
Análisis	`Dominate`	Verifica si la conexión es miembro de la lista de conexiones dominantes.
	`Depend`	Verifica si la conexión es miembro de la lista de conexiones dependientes.
	`Analisys Dominate`	Despliega lista de todas las conexiones, específica si cada conexión es o no miembro de la lista de conexiones dominantes.
	`Analisys Depend`	Despliega lista de todas las conexiones, específica si cada conexión es o no miembro de la lista de conexiones dependientes.
Métricas	`NOC (Number of Connections)`	Número de conexiones
	`NOBC (Number of Basic Connections)`	Número de conexiones básicas
	`NOQC (Number of Query Connections)`	Número de conexiones de consultas

	NOCHC (Number of Change Connections)	Número de conexiones de cambio
	NOCC (Number of Complex Connections)	Número de conexiones complejas
	NOBR (Number of Business Rules in Connection)	Número de reglas de negocio en conexión
	NOE (Number of Event in Connection)	Número de eventos en Conexión
	CARDBR (Cardinality of Business Rules)	Calcula la cardinalidad de una regla de negocio, es decir cuantas conexiones existen para una regla de negocio específica.
	CARDEV (Cardinality of Event)	Calcula la cardinalidad de un evento, es decir cuantas conexiones existen para un evento específico.
	List	Despliega un listado de todas las métricas
	Bar Graph	Muestra gráfico de barras de todas las métricas
Interacción	Individual Connection	Indica cuáles son las conexiones que coinciden en su elemento de evento, a partir del identificador de la conexión ingresado.
	Total Connections	Despliega un listado con todas las conexiones en el contenedor y con cuales coincide cada una de ellas en su elemento de evento.
Resumen	Resume	Presenta un resumen de cada una de las conexiones aspectuales del contenedor, visualizando todos los componentes de las mismas (identificador, tipo, elemento de reglas de negocio, evento, etc.)
Mapeo	Mapping	Genera código AspectJ o Spring para cada conexión aspectual seleccionada.

Tabla 1. Funciones y operaciones de gestión

6. GENERACION DE CODIGO ASPECTJ

La implementación del código se considera una de las actividades mas duras en el desarrollo de software. Se pretende por este motivo proveer métodos que sistemáticamente permitan generar el código, ya sea en forma automática o semi-automática.
En [10] hemos probado que a partir de las PCA (en formato XML) y la taxonomía es posible generar automáticamente el código de un aspecto en Spring AOP Framework. A continuación presentamos las reglas de mapeo para generar código AspectJ a partir de las PCA y la taxonomía. Por cada tipo de conexión definida en la taxonomía se presentan las reglas de mapeo o generación de código AspectJ y un ejemplo sencillo de su aplicación.

Se asume que las reglas de negocio son clases que implementan la interfaz *condition ()*, *action ()* y *apply ()* , como sugiere el patrón Object Rule [2] y se presenta en el Ejemplo 3.

```
interface BusinessRule {
 boolean condition (// lista de parámetros)
  {// evalua un estado}
 void action (// lista de parámetros)
  { // aplica }
 void apply(//lista de parámetros)
   { if condition (//lista de parámetros)
       then action(//lista de parámetros) }
}
```

Ejemplo 3.

6.1 CONEXIÓN BÁSICA

En la Tabla 2, se presentan las reglas para crear el aspecto correspondiente a una conexión básica a partir de los elementos de la PCA descripta en la Sección 4.

Paso	Sección y Elemento PCA	Elemento de Aspecto
1	Sección A – elemento <connection>	Identificador del aspecto
2	Sección B - elemento <business rule>	Declarar y crear objeto correspondiente a la clase de la Regla de Negocio como atributo del aspecto
3	Sección C - elemento <main event>	Crear pointcut *main*
4	Sección C – elemento <when>	Crear advice y sentencia de invocación a la regla de negocios.
Consideraciones generales: - el identificador de la regla de negocio siempre se denomina “br” - el identificador del pointcut siempre se denomina “main”		

Tabla 2. Reglas para generar aspectos a partir de conexión básica

En la Figura 3 se presenta un ejemplo detallado para generar el código del aspecto de una conexión básica a partir de los elementos de la PCA, siguiendo los pasos establecidos en la Tabla 2.

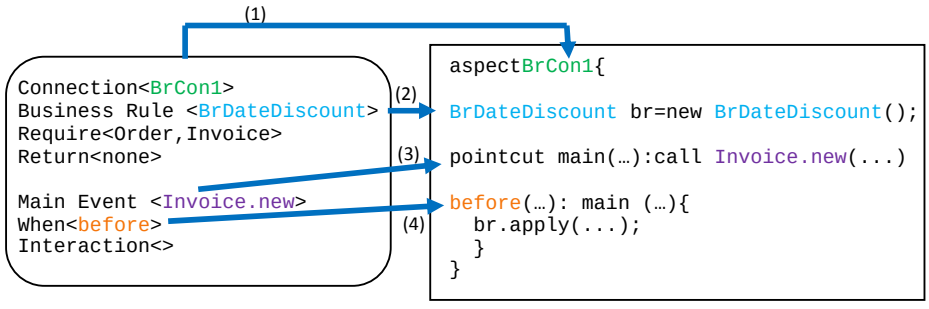


Figura 3. Generación de código del aspecto desde la PCA para una conexión básica.

6.2 CONEXIÓN DE CONSULTA

En la Tabla 3, se presentan las reglas para crear el aspecto correspondiente a una conexión de consulta a partir de los elementos de la PCA descripta en la Sección 4.

Paso	Sección y Elemento PCA	Elemento de Aspecto
1	Sección A – elemento <connection>	Identificador del aspecto
2	Sección B - elemento <business rule>	Declarar y crear objeto correspondiente a la clase de la Regla de Negocio como atributo del aspecto
3	Sección C - elemento <main event>	Crear pointcut *main*
4	Sección C – elemento <when>	Crear advice y sentencia de invocación a la regla de negocios.
5	Sección D – elemento <retrieve>	Declarar y crear objeto correspondiente al tipo de dato que devuelve el elemento de consulta como atributo del aspecto
6	Sección D – elemento <query event>	Crear pointcut query
7	Sección D – elemento <when>	Crear advice y recuperar información del contexto en atributo creado en paso 5.
Consideraciones generales: - el identificador de la regla de negocio siempre se denomina “br” - un identificador de pointcut denominado main - un identificador de pointcut denominado query		

Tabla 3. Reglas para generar aspectos a partir de conexión de consulta

En la Figura 4 se presenta un ejemplo detallado para generar el código del aspecto de una conexión de consulta a partir de los elementos de la PCA, siguiendo lo establecido en la Tabla 3.

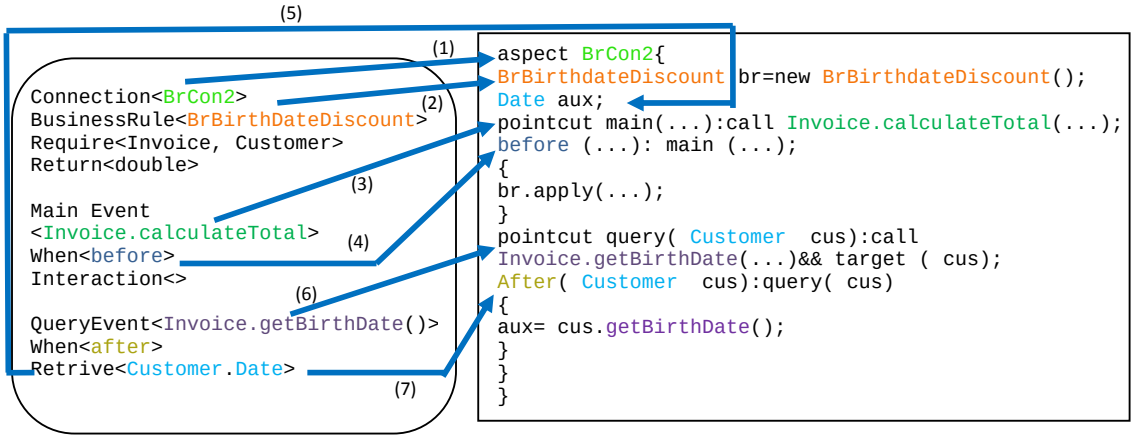


Figura 4. Generación de código del aspecto desde la PCA para una conexión de consulta.

6.3 CONEXIÓN DE CAMBIO

En la Tabla 4, se pueden observar las reglas para crear el aspecto correspondiente a una conexión de cambio a partir de los elementos de la plantilla descripta en la Sección 4.

Paso	Sección y Elemento PCA	Elemento de Aspecto
1	Sección A – elemento <connection>	Identificador del aspecto
2	Sección B - elemento <business rule>	Declarar y crear objeto correspondiente a la clase de la Regla de Negocio como atributo del aspecto
3	Sección E- elemento <add field>	Por cada elemento field, declarar una introducción de atributo en el aspecto.
4	Sección E- elemento <Add Method>	Por cada elemento method, declarar una introducción de método en el aspecto.
5	Sección C - elemento <main event>	Crear pointcut *main*
6	Sección C – elemento <when>	Crear advice y sentencia de invocación a la regla de negocios.
Consideraciones generales: - el identificador de la regla de negocio siempre se denomina “br” - un identificador de pointcut denominado main - en los métodos introducidos (paso 4) el código debe ser implementado manualmente		

Tabla 4. Reglas para generar aspectos a partir de conexión de cambio

En la Figura 5 se presenta un ejemplo detallado para generar el código del aspecto de una conexión de consulta a partir de los elementos de la PCA, siguiendo lo establecido en la Tabla 4.

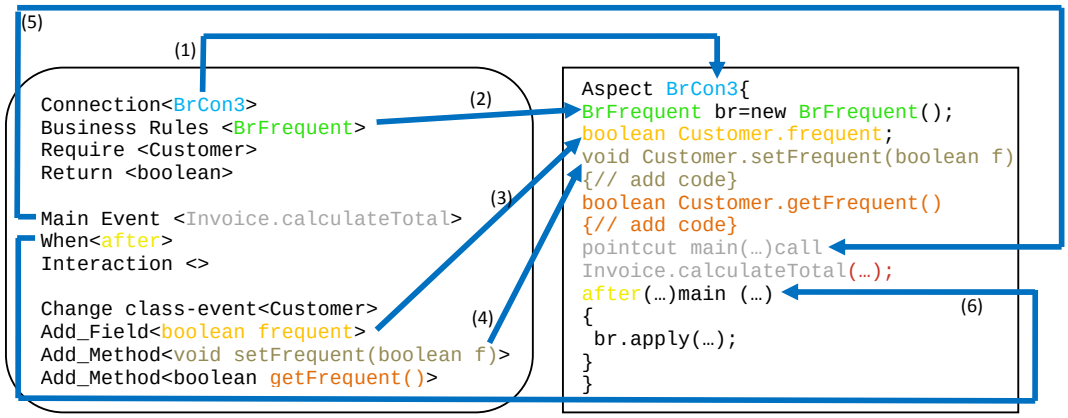


Figura 5. Generación de código del aspecto desde PCA para una conexión de cambio.

6.4 CONEXIÓN COMPLEJA

Respecto de las reglas para generar el código de los aspectos que corresponden a las conexiones complejas, estas combinan los pasos presentados para la conexión de consulta y la conexión de cambio.

7. MACS

MACS (Management of Aspectual Connections) es una herramienta que hemos desarrollado que cumple con los requerimientos funcionales y operacionales planteados en éste informe (Secciones 2 a 5). Es decir MACS, permite que a partir de la registración de las PCA, puedan aplicarse en forma automática todas las funciones y métricas presentadas en la Tabla 1.

Asimismo genera automáticamente el código en AspectJ, a partir de las PCA y la taxonomía, según las directrices presentadas en Sección 5. A continuación brevemente presentamos el diseño de MACS y unas vistas que permiten someramente conocer sus prestaciones.

7.1 DISEÑO

En la Figura 6 se presenta una versión resumida del diagrama de clases de MACS. El gestor-contenedor de conexiones aspectuales esta representado por la clase *ManagementConnections.* El comportamiento de este gestor-contenedor responde a la administración de todas las conexiones aspectuales, las cuales están encapsuladas en objetos de tipo *BasicConnection, QueryConnection, ChangeConnection y ComplexConnection*. Las características estructurales compartidas entre las conexiones las mismas se representan en distintas relaciones de herencia y jerarquía. Las conexiones a su vez, se componen de otros elementos que pueden diferenciarse, como los elementos de eventos y las reglas de negocio, a los cuales corresponden las clases *EventElement* y *BusinessRulesElement* respectivamente. Los elementos de eventos vendrían a representar a las clases del dominio, cos sus métodos y atributos. A su vez, son atributos de esta clase y dependen de su tiempo de vida. Lo mismo sucede entre *QueryConnection* y *QueryElement*; y entre *ChangeConnection* y *ChangeElement.*

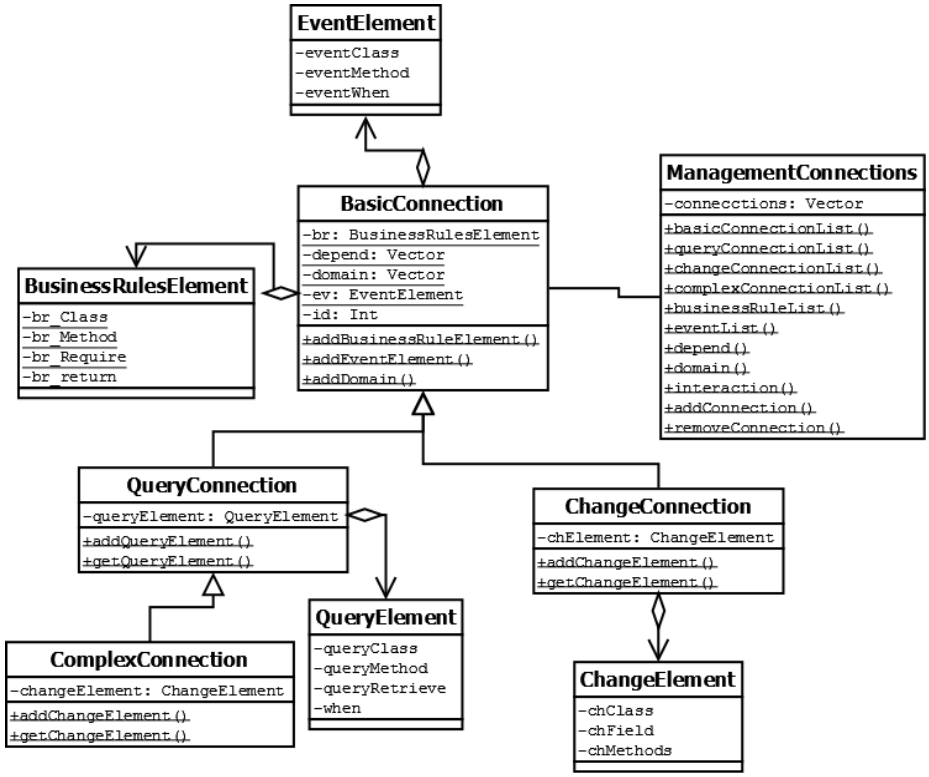


Figura 6. Diagrama de clases MACS

7.2 VISTAS DE MACS

A continuación se presentan algunas vistas del gestor de conexiones aspectuales.

La Figura 7 muestra la vista inicial de la herramienta, por medio de la cual se puede acceder al menú dispuesto en la barra superior horizontal, el cual consta de tres opciones: Domain, Operations y Mapping. “Domain” permite agregar o eliminar conexiones a nuestro gestor; “Operations” ofrece las distintas consultas y métricas que se pueden realizar con la herramienta (ver sección 4) y por último “Mapping” que permite generar automáticamente el código de un aspecto de acuerdo a la conexión seleccionada.

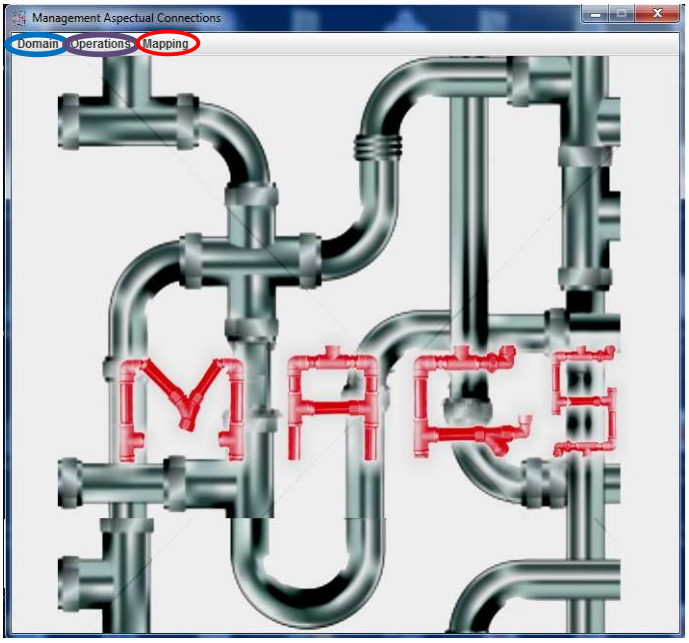


Figura 7. Vista inicial de Macs.

La Figura 8 presenta la vista que proporciona MACS para agregar una conexión, esencialmente este formulario representa a la PCA. Tal como se describe a continuación existe una coincidencia entre las secciones y elementos que la componen. Secciones: A- Identificador de la conexión. B- Elementos de la reglas de negocio. C- Datos del evento que dispara la regla de negocio. D- Elementos de consulta. E- Elementos de cambio. F- Interacción. G- Relación de conexión, dominio y dependencia. Y por último en H- Se debe seleccionar que tipo de conexión se desea crear.

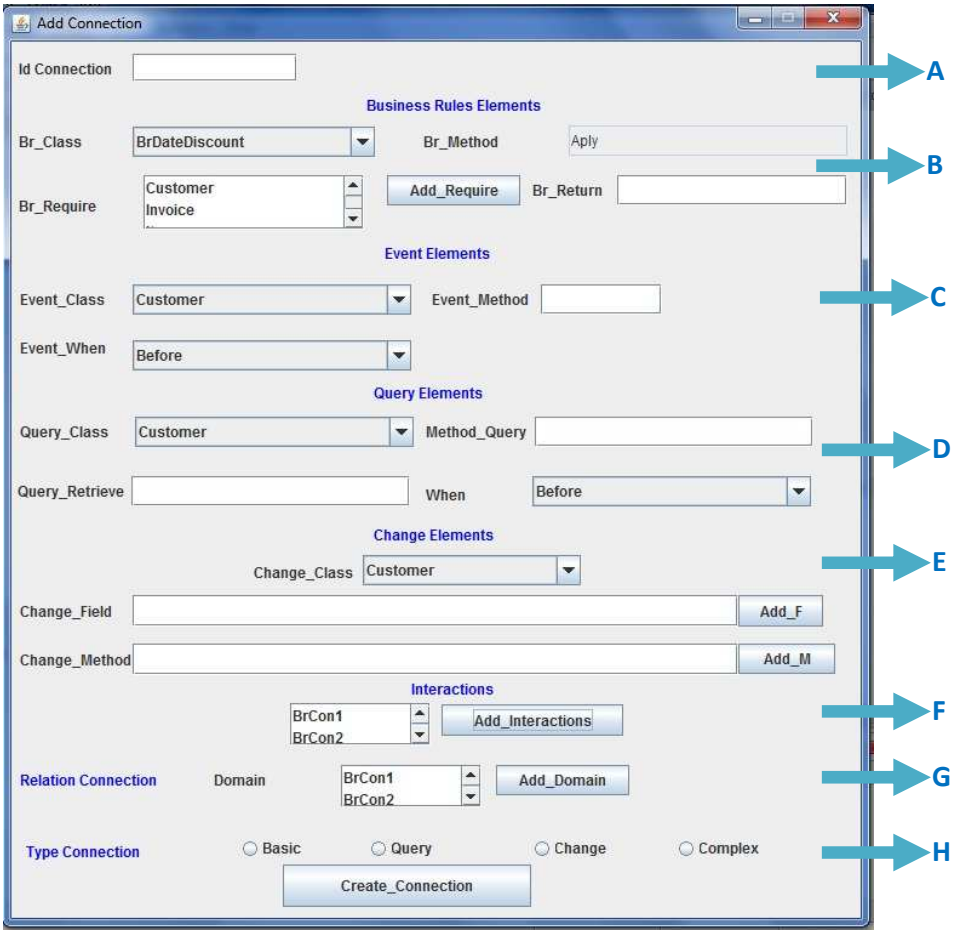


Figura 8. Formulario correspondiente a la PCA.

8. CASO DE ESTUDIO

En esta Sección se presenta un caso de estudio simple basado en un comercio, cuya lógica del negocio determina en primer lugar, que las órdenes de compra de los clientes, incluyen los datos del cliente y artículos solicitados. Más tarde (el mismo día u otro), cuando el cliente quiere retirar el pedido y pagar, la factura se lleva a cabo. En esta oportunidad el sistema calcula el subtotal, descuento y total.

En la Tabla 5 se enumera un conjunto de reglas de negocio de descuento y promoción que se han incorporado a la lógica descripta. Algunas de estas reglas establecen condiciones de descuento en la factura y promociones por las compras efectuadas por el cliente. Las reglas de negocio son clases que implementan la interfaz *condition ()*, *action ()* y *apply ()*, como sugiere el patrón Object Rule [2]. En la segunda columna esta referenciada la clase que implementa a cada regla de negocio; luego en la columna Conexión Aspectual se indica el identificador de la conexión aspectual que integra la regla a la lógica de negocio, y por último en la cuarta columna la categoría de la conexión aspectual de acuerdo a la taxonomía presentada anteriormente.

Reglas de negocio	Clase	Conexion Aspectual	Taxonomía
Si la fecha del pedido y la fecha de la factura son equivalentes *entonces,* aplicar un descuento del 3% cuando se emite la factura.	BrDateDiscount	BrCon1	Básica
S*i* la fecha de facturación es equivalente con la fecha de nacimiento del cliente entonces, aplicar un descuento del 2%	BrBirthdateDiscount	BrCon2	Consulta
S*i* el número de compras del cliente es mayor que 20, *entonces* el cliente es frecuente.	BrFrequent	BrCon3	Cambio
S*i* el cliente es frecuente entonces aplicar un descuento del 0,5% cuando se emite la factura.	BrFrequentDiscount	BrCon4	Básica
S*i* la fecha de la factura es de 30 días después de la fecha de la *orden, entonces* el sistema actualiza el precio de los artículos antes de emitir la factura.	BrPriceUpdate	BrCon5	Básica
S*i* la ultima factura de un cliente frecuente, fue 180 días *antes*, el cliente no es frecuente.	BrRemoveFrequent	BrCon6	Básica
Si el stock de artículos es menor que el stock *mínimo*, el artículo esta en promoción.	BrPromotion	BrCon7	Cambio
S*i* un artículo está en *promoción, entonces* su precio se reduce en un 25%.	BrPromotionDiscount	BrCon8	Básica
Cada $50 se le acredita un punto a la tarjeta de Puntos del cliente	BrCreditRewardsCard	BrCon9	Consulta
Sorteo especial en el mes aniversario del comercio, cada $75 se le otorga al cliente dos bonos obsequio.	BrPrintRaffleTicket	BrCon10	Básica

Tabla 5. Reglas de negocio y Conexiones Aspectuales del caso de estudio

En el sistema las órdenes de compra están representadas por la clase Order, los clientes por la clase Customer, los artículos por la clase Item y las facturas por la clase Invoice. Invoice conserva una copia de los datos del cliente, clase CustomDetails, para la impresión. Después de que la factura se crea, la cantidad de compras del cliente se incrementa (campo inv_count de la clase Customer), dependiendo del monto de la compra se le acreditan puntos en la tarjeta de puntos del cliente (clase RewardsCard). En la Figura 9 se presenta el esquema completo, el cual además incorpora las reglas de negocio y las conexiones aspectuales. Las conexiones aspectuales están representadas con rombos, para identificar de manera distintiva a los aspectos.

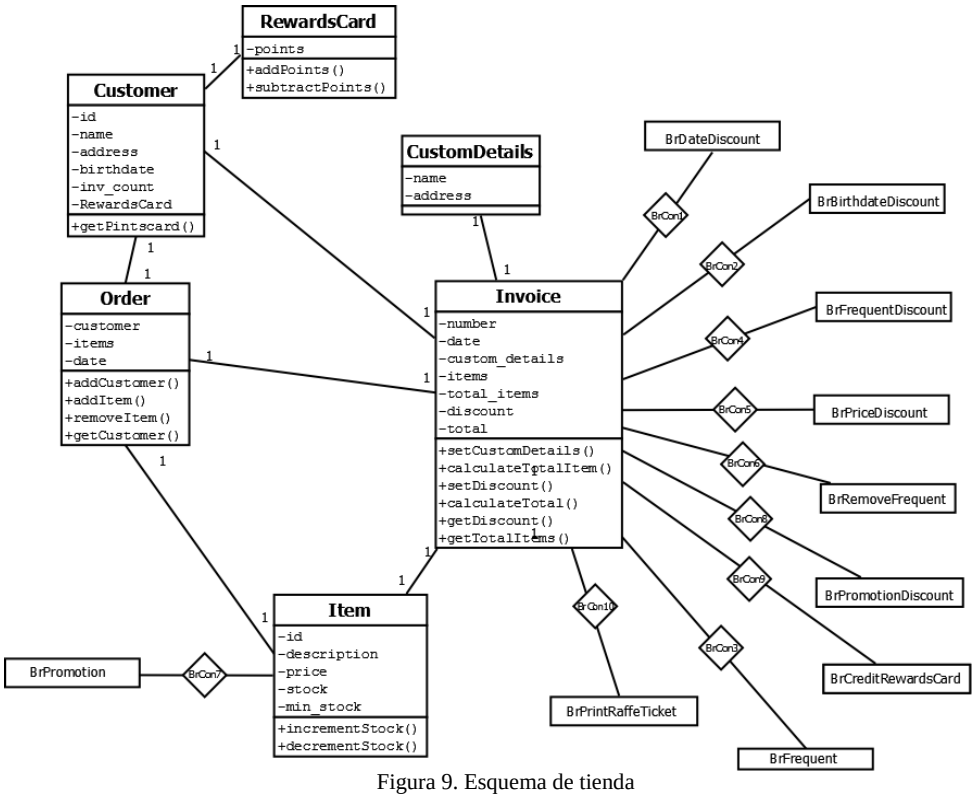
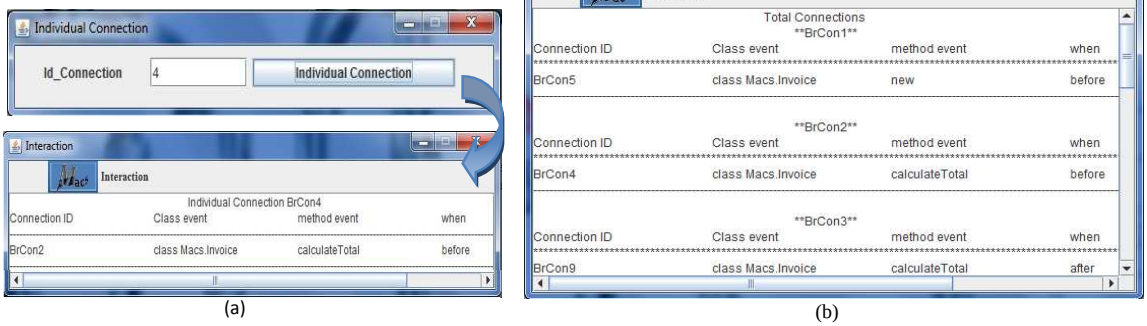


Figura 9. Esquema de tienda

8.1 CÁLCULOS CON MACS

A continuación (en la figura 10) se exhiben algunas vistas producidas al ejecutar algunas operaciones con MACS, posteriormente a la registración de las conexiones relacionadas con el caso de estudio en cuestión. En (a) se ha consultado la/s interacción en las que participa conexión X, la respuesta es que existe una interacción con la conexión Y, dado que son disparadas por el mismo evento (método “calculateToTal” de la clase Invoice) y momento (before). La vista (b) presenta todas las interacciones existentes, con similar información en cada caso (evento y momento). Todas las métricas pueden obtenerse mediante un listado y/o en forma gráfica (c). Las operaciones de análisis se presentan mediante listados que permiten conocer las relaciones de dependencia (d) y dominio (e) entre las conexiones aspectuales.

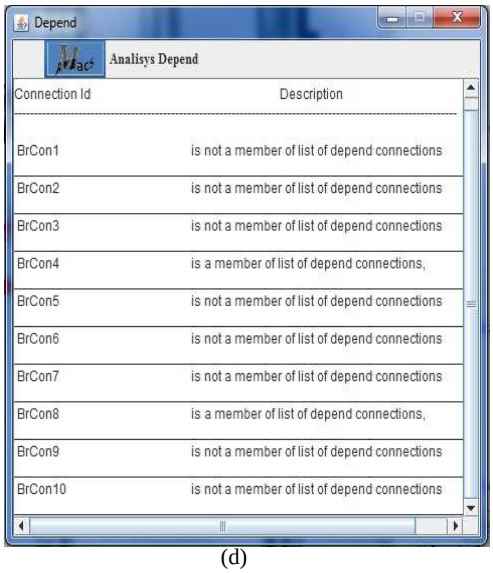


(a)

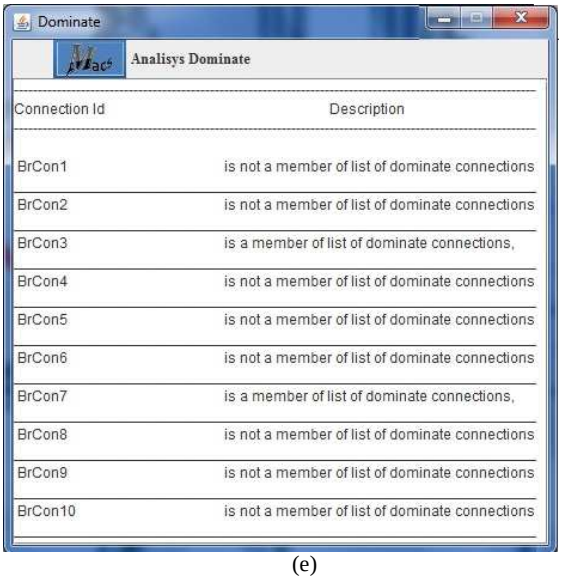
(b)

(c)

(c)



(d)



(e)

Figura 10. Vistas de MACS: Consultas y Métricas

En la Figura 11 se presentan vistas que corresponden a la generación del código de los aspectos de tres 3 conexiones. En (a) la conexión “BrCon1”, que refiere a una conexión básica, en (b) la conexión “BrCon2”, que refiere a una conexión de consulta y en (c) la conexión “BrCon3” que se trata de una conexión de cambio.

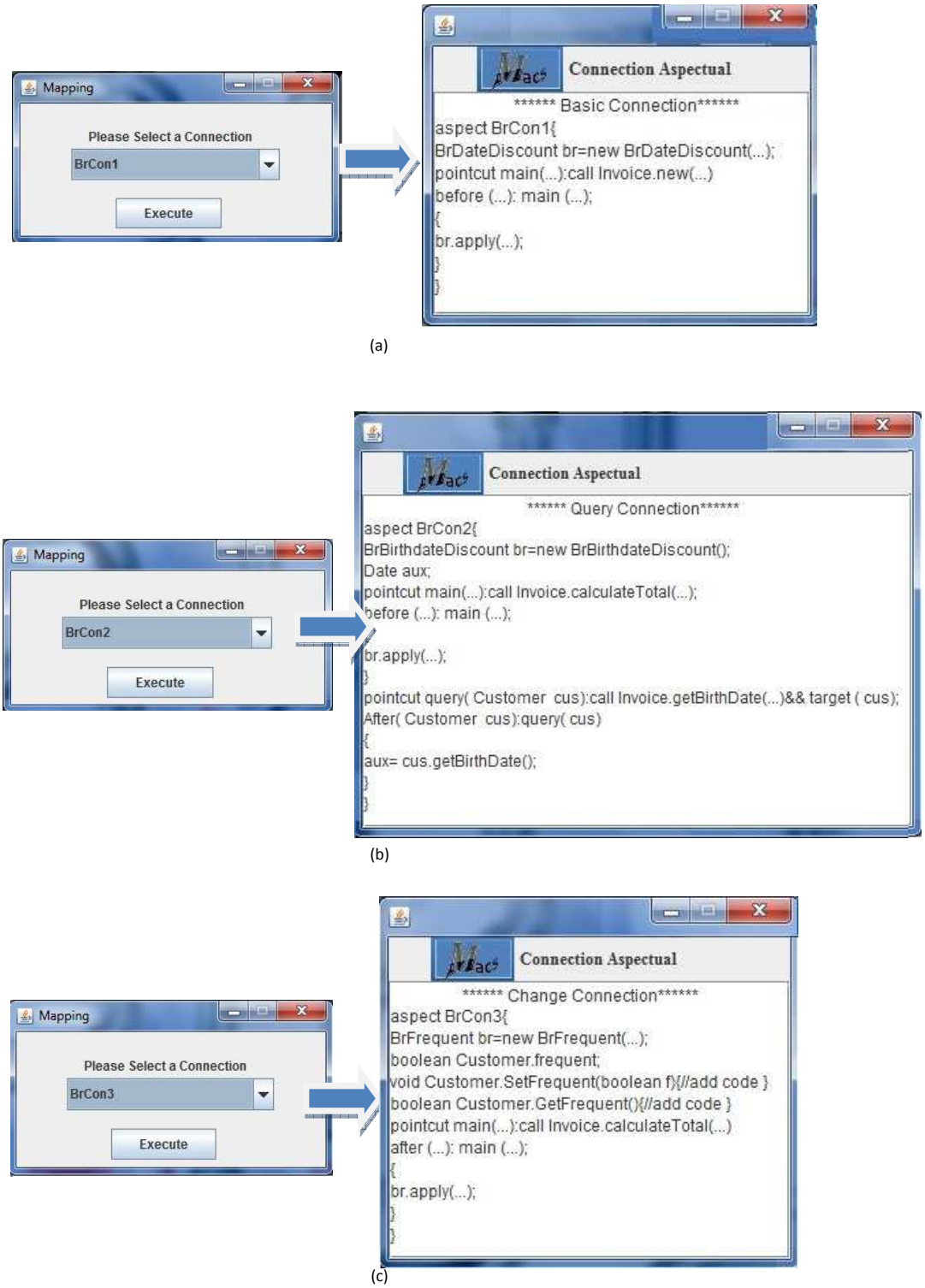


Figura 11. Generación de código de los aspectos.

9. TRABAJOS RELACIONADOS

Varios métodos fueron propuestos para describir las reglas de negocios, tales como plantillas [16][17], tablas [16], lenguaje natural [16], XML [18], OCL [19], etc. Sin embargo es muy difícil encontrar anotaciones o mecanismos específicos para describir sus conexiones o integración con la lógica de negocio. Diferentes clasificaciones de reglas de negocios han sido expuestas [20][16][21], pero no existe una clasificación de las conexiones de reglas de negocios. En este sentido la plantilla de conexiones aspectuales como la taxonomía de las conexiones aspectuales presentadas son contribuciones relevantes.

Cibrián en [22] presenta un lenguaje de alto nivel para conectar reglas de negocios. Esta notación especifica los detalles de la integración de las reglas con el núcleo de aplicación; denota el evento al cual la regla debe ser aplicada, el momento exacto en el cual la regla debe ser aplicada ante la ejecución de dicho evento, y la especificación de cómo la información de la regla requerida se hace disponible para la regla. Esta propuesta sólo utiliza este lenguaje para mapear automáticamente la conexión al lenguaje orientado a aspectos JasCo [23] .

Algunos trabajos han tratado con conexiones aspectuales para integrar reglas de negocio, pero estas se han enfocado a la implementación con diferentes herramientas AOP, como AspectJ [7], JasCo [24], y Spring AOP Framework [8]. [6] presenta una plantilla para implementar las reglas de negocios con AspectJ. [25] presenta una experiencia de refactorizar las reglas de negocios con AspectJ en una importante aplicación J2EE. Sin embargo ninguno de estos trabajos propone gestionar de manera integral las conexiones aspectuales en pos de una futura automatización.

Otros aportes consideran el manejo de requerimientos volátiles, entre los que se ubicarían las reglas de negocios, en etapas tempranas del desarrollo del software. Por ejemplo una importante contribución es [26], los autores presentan un método para manejar concerns volátiles, durante la etapa inicial de modelado del software. El método consiste en varios pasos: clasificación de concerns, refactorización de los requerimientos, instanciación del modelo y composición de modelo.

En una línea similar, un marco de trabajó es propuesto para identificar concerns volátiles y crosscutting concerns en los niveles de requerimientos [27] [28]. La identificación de tales concerns se basa en un patrón de Crosscutting y una simple matriz de operaciones. Estas técnicas mejoran el tratamiento de las reglas de negocios y sus conexiones aspectuales en las actividades de modelado, pero no alcanzan a cubrir la etapa de diseño avanzado en implementación.

10. CONCLUSIONES

Como hemos mencionado, en trabajos previos hemos propuestos diversas estrategias para facilitar el desarrollo de conexiones entre reglas de negocio y la funcionalidad central de las aplicaciones con aspectos. Con el objetivo de asistir al desarrollador de software se ha propuesto la PCA, como artefacto de documentación; la taxonomía de conexiones aspectuales, como método de clasificación; funciones y métricas que permitan el análisis particular como el contexto general en que están insertas, mecanismos de mapeo a código real, a AspectJ. La herramienta MACS ha implementado estas estrategias y mecanismos con un alto nivel de efectividad y ha constituido en un valioso aporte para la validación del trabajo previo.

Las principales limitaciones del trabajo presentado no refieren directamente a MACS. En principio una restricción del enfoque propuesto recae en el hecho de que una conexión aspectual conecta una y solo una regla de negocio con el dominio. Es decir, si una regla de negocio es disparada por distintos eventos, entonces se requieren distintas conexiones; de igual forma, si un mismo evento dispara más de una regla de negocio, cada una de estas conexiones deben ser planteadas en conexiones diferentes. Esta decisión responde fundamentalmente la facilidad de manejar la modificación del código durante la evolución y mantenimiento. Cibrán plantea el desacoplamiento de los pointcuts, de manera tal que nunca existe en un aspecto más de un pointcuts, aunque esto permite un reutilización de ciertos pointcuts, genera una proliferación de aspectos que se tornan difíciles de mantener.

El tratamiento de las interacciones, se limita a la detección, es decir identificar aquellas conexiones aspectuales que son disparadas por el mismo evento en el mismo momento; y posteriormente especificar en la PCA el orden en que dichas interacciones son resueltas. A pesar de que la mayoría de las interacciones entre conexiones aspectuales encontrará en este simple mecanismo una respuesta acorde, queda un espacio no resuelto que requiere otras estrategias. Este es el caso, de aquellas interacciones entre conexiones aspectuales cuyas reglas de negocio son excluyentes. En estos escenarios la resolución de una posible interacción no es alcanzada por los mecanismos previstos. En estos casos la resolución esta gobernada por una “super” regla de negocio que decide que regla de negocio aplicar. Esta super regla de negocio esta totalmente condicionada a las políticas del negocio. El tratamiento de estas situaciones no contempladas en el modelo conceptual subyacente propuesto forma parte del trabajo futuro, para luego ser soportado por la herramienta.

En lo que refiere a MACS, el trabajo a futuro esta dirigido esencialmente a potenciar sus capacidades: a) Posibilitar que genere código de las conexiones en otras herramientas AOP, como Spring AOP Framework, para lo cual es necesario crear las nuevas reglas de mapeo y desarrollar el nuevo generador de código; b) Generar la importación de las conexiones a XML; c) agregar mas funciones y métricas.

AGRADECIMIENTOS

A la Universidad Nacional de la Patagonia Austral que ha financiado parcialmente este trabajo.

REFERENCIAS

[1] Leite J.C.S.P, Leonardi M. C. 1998. *Business rules as organizational Policies.* IEEE IWSSD9: Ninth International Workshop on Software Specification and Design, IEEE Computer Society Press pp 68-76

[2] Arsanjani A. 2001. *Rule object 2001: A pattern language for adaptive and scalable business rule construction.* Technical Report, IBM T.J. Watson Research Centre.

[3] Jess homepage, http://www.jessrules.com/

[4] Kiczales G., Lamping L., Mendhekar A., Maeda C., Lopes C., Loingtier J., Irwin J. 1997. *Aspect-Oriented Programming*. In Proceedings ECOOP'97 – Object-Oriented Programming, 11th European Conference, Finland, Springer-Verlang.

[5] Cibrán M., D'Hondt M. and Jonckers V. 2003. *Aspect-oriented programming for connecting business rules*. In Proceedings of the 6th International Conference on Business Information Systems. Colorado, USA.

[6] Laddad R. 2003. *AspectJ in Action*. Manning Publications Co.

[7] Cibrán M. and D'Hondt M. 2003. *Composable and reusable business rules using AspectJ*. In Workshop on Software engineering Properties of Languages for Aspect Technologies (SPLAT) at the International Conference on AOSD. Boston, USA.

[8] Vidal G., Enriquez J. and Casas S. 2010. *Integración de Reglas de Negocio con Conectores Aspectuales Spring*. 11th Argentine Symposium on Software Engineering - Argentina

[9] Casas S. 2010. *Clasificación y Documentación de Conexiones Aspectuales para Reglas de Negocio*. I Encuentro Internacional de Computación e Informática del Norte de Chile. Chile.

[10] Casas S., Enriquez J. 2011. *Mapping Connection Templates to Spring Aspects to Integrate Business Rules*. Workshop of Early Aspects AOSD 2011 – Brasil.

[11] Casas S. y Herrera F. "*Toward an Automatic Management of Aspectual Connections to Compose Business Rules*" VIII WORKSHOP DE INGENIERÍA DE SOFTWARE (WIS) - XVII CACIC 2011, pp 899-910. Argentina. ISBN 978-950-34-0756-1

[12] The AspectJ Programming Guide, http://eclipse.org/aspectj

[13] Hürsch W., Lopes C. *Separation of Concerns*. Northeastern University Technical Report NU-CCS-95-03, Boston, February (1.995).

[14] Mens K., Lopes C., Tekinerdogan B., Kiczales G. *Aspect-Oriented Programming.* Workshop Report ECOOP. 11th. Finland (1.997).

[15] Piveta E., Zancanela L.: *Aspect Weaving Strategies.* Journal of Universal Computer Sciencie. Vol.9. Num.8 (2.003).

[16] Ross R. 2001. *The BRS Rule Classification Scheme*.

[17] Ross R. 2003. *Principles of the Business Rule Approach*. Addison Wesley.

[18] XRules homepages, http://www.xrules.org/

[19] Demuth B., Hußmann H., Loecher S. *OCL as a Specification Language for Business Rules in Database Applications*. Proceeding «UML» '01 Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools Springer-Verlag London, UK. 2001

[20] Business Rule Group. 2001. *Defining Business Rules: What Are They Really?*. http://www.businessrulesgroup.org/.

[21] Date C. J. 2000. *What Not How: The Business Rules Approach to Application Development. Reading*, Mass. Addison-Wesley Longman Inc.

[22] Cibrán M. 2007. *Connecting High-Level Business Rules with Object-Oriented Applications: An approach using Aspect-Oriented Programming and Model-Driven Engineering*. Tesis doctoral. Universiteit Brussel.

[23] Suvee, D., Vanderperren, W. and Jonckers, V. 2003. *JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development.* In Proceedings of the second International Conference on Aspect-Oriented Software Development. Boston, USA.

[24] Cibrán M., D'Hondt M., Suvee D., Vanderperren W., and Jonckers V. 2005. *Linking Business Rules to Object-Oriented software using JAsCo#*. Journal of Computational Methods in Sciences and Engineering, pp 13-27, IOS Press, Volume 5(1).

[25] Kellens A., De Schutter K., D´Hondt T., Jonckers V. and Doggen H. 2008. *Experiences in modularizing business rules into aspects*. ICSM 24 th. IEEE International Conference on Software Maintenance. Page(s):448 – 451. China.

[26] Moreira A., Araújo J., and Whittle J. 2006. *Modeling Volatile Concerns as Aspects*. E. Dubois and K. Pohl (Eds.): CAiSE 2006, LNCS 4001, pp. 544 – 558, 2006.Springer-Verlag Berlin Heidelberg.

[27] Conejero J.M., Hernandez J., Moreira A. and Araújo J. 2007. *Discovering Volatile and Aspectual Requirements Using a Crosscutting Pattern.* 15th IEEE International Requirements Engineering Conference.

[28] van der Berg K., Conejero J.M., and Hernández, J. 2007. *Analysis of Crosscutting in Early Software Development Phases based on Traceability*. Transactions on AOSD, Special Issue on Early Aspects.