Journal of Software Engineering and Applications, **2012, 5, 76-84**
http://dx.doi.org/10.4236/jsea.2012.52012 Published Online February 2012 (http://www.SciRP.org/journal/jsea)

Lightweight Approach to Connect Business Rules with Aspects

Sandra Casas[1], Claudia Marcos[2], Cecilia Fuentes[1], Juan Enriquez[1], Graciela Vidal[1]

[1]UARG, Universidad Nacional de la Patagonia Austral, Rio Gallegos, Argentina; [2]ISISTAN, Universidad Nacional del Centro de la Pcia de Buenos Aires, Tandil, Argentina.
Email: scasas@unpa.edu.ar

Received November 28th, 2011; revised December 31st, 2011; accepted January 11th, 2012

ABSTRACT

Aspectual connection that compose business rule, refers to the code in charge of not only triggering the application of the rules at certain events, but also gathering the necessary information for their application and incorporating their results in the rest of the core application functionality. We propose an approach to connect business rule with aspect systematically. This approach is agile, programming language-independent, business rules type independent and it can be used in different stages of development. This approach considers the identification, analysis and construction of connections with aspects, (aspectual connection). We use our experience and previous works to define the initial steps, and complete them with the analysis of interactions and propose methods to implement aspectual connections code. We have developed a tool that supports this approach.

Keywords: Business Rules; Connection; Aspectual Connections; Interactions; AOP; Volatile Concerns; AspectJ

1. Introduction

Business rules are statements about the enterprise's way of doing business. They reflect business policies. Organizations have policies in order to: satisfy the business objectives, satisfy customers, make good use of resources, and conform to laws or general business conventions [1]. Business rules become requirements, that is, they may be implemented in a software system as means of requirements of this software system [2]. But business rules tend to change over time due to new policies, new business realities, and new laws and regulations, for these reasons several approaches and technologies are created to separate business rules implementation from core modules. In all these approaches and technologies it is necessary explicitly connect business rules with core functionality.

The code that connects or links the business rules with core functionality is spread across core modules. We have named this code as connection code. A change in the rule specification requires changes in all modules where connection is present. These modifications are invasive and time-consuming. Further, because business rules are a lot more volatile compared to core business logic, mixing them together causes the core system to become just as volatile. For these reasons, AOP [3] is suitable when providing mechanisms that allow to connect or to integrate the business rules to the core modules without altering these components. Just as it is stated in [4], AOP facilitates the constant evolution of this type of concerns.

Aspectual connection that compose business rule, refers to the code in charge of not only triggering the application of the rules at certain events, but also gathering the necessary information for their application and incorporating their results in the rest of the core application functionality. The aspectual connection must meet some requirements for the business rule to be triggered. We have identified in [5] that the development (design and implementation) of connections with aspects is not a trivial work, and they are needful strategies to manage some shortcomings, as when the same business rule could require different connections in different domains; or when the business rules require the connections to adapt or change the domain; or when an interaction between excluded business rules is detected. These situations and other require specific strategy to analysis aspectual connections in previous steps of implementation stage. Some contributions show how connections could be implemented with aspects and with different AOP-tools [6-9]. The authors analyses the scheme of design and implementation with different goals but they only cover common and ordinary scenarios.

Therefore, this work outlines an approach to connect business rule with aspect. This approach considers the identification, analysis and construction of connections with aspects, (aspectual connection). We use our experience

and previous works to define the initial steps, and complete them with the analysis of interactions and propose methods to implement aspectual connections code. Unlike to other works in this line, we address the interactions of aspectual connections and propose a method that it is thought to accept different design and implementation strategies with different aspect-oriented languages. Also we have developed a tool that gives complete support to the approach.

The outline of this paper is as follows: Section 2 will expose Business Rules main concepts; Section 3 will present the approach to connect business rule with aspects, with a brief description of main concepts of the approach; Section 4 will unfold the handling of interactions proposed; Section 5 will present the method to construct aspect connection code; Section 6 will present the MACS tool that supports this approach; Section 7 related works are presented and Section 8 we will give our conclusions.

2. Business Rules

Business rules are statements about the enterprises way of doing Business [10]. Organizations have policies in order to: satisfy the business objectives, satisfy customers, make good use of resources, and conform to laws or general business conventions. The business rule model distinguishes between functional rules and non-Functional rules. Functional rules are general policies regarding the organization functionality. Macrosystem rules describe policies that constrain the behavior and structure of the organization. Quality rules are demands of the organization on the characteristics of its processes or products. They usually reflect general policies related to quality standards or expectations of the organization. Other classical classifications of business rules are proposed [11-13]. Two approaches and technologies to implement business rules explicitly and separately from core functionality of applications are: Object Oriented Patterns, using patterns, such as the Rule Object Pattern [14] and Hybrid Systems, which support explicit and separate representation of rules in a rule-based language. An example is JESS [15].

However, the code which connects core functionality with business rules is tangled and scattered in core modules, for this reason is considered a crosscutting concern. Therefore, conventional approaches, as Object-Oriented Paradigm, are not able or enough to avoid this kind of problem. On the other hand, the business rules are very volatile as they change very frequently, and thus the volatility is spread to the core, where the connections are explicit.

3. An Approach to Connect Business Rule with Aspects

As aforementioned, in our previous works we have presented some preview of this approach: taxonomy to classify the aspectual connections, a template to document aspectual connections (ACT) [5] and a mapping process to generate automatically aspects [16]. After these experiences we integrate these contributions to propose an approach to connect business rule with aspects, systematically. The main features and strengths of this approach are:

1) It can be applied in different stages of development. For example it can be applied during advanced design when domain has been modeled or after the installation of application, during maintenance stage.

2) It is valid for any type of business rules. For example it can be used it to derivation, structural assertion or action assertion [17] business rules.

3) It is independent of base-programming language and aspect-oriented language.

4) It is lithe as it does not require another additional o specific artifact (for example diagrams of UML), it is oriented to obtaining real code instead of obtaining documentation and it consists of few steps. The approach is addressed to software applications were core modules and business rules are represented with traditional unit as classes, from Object Oriented paradigm. We mention this because of we have not had experiences with other techniques, such as hybrid systems.

The approach consists of three steps: definition, analysis and construction.

1) Definition of connections is the activity which specifies the required elements of connections in ACT and classifies them according the taxonomy of aspectual connections.

2) Analysis is the activity that detects interactions among connections and tried to find resolutions of them.

3) Construction is the activity that generates the aspect connection code. This task is the only one which is dependent of AOP-tool. However it could be adapted to different AOP-tools.

The approach is based on two main pillars: taxonomy of connection and a template to register and document elements of connections.

3.1. Taxonomy of Aspectual Connections

Aspectual connection that compose business rule, refers to the code in charge of not only triggering the application of the rules at certain events, but also gathering the necessary information for their application and incorporating their results in the rest of the core application functionality. The aspectual connection must meet some requirements for the business rule to be triggered. Here it is important to stress that a particular business rule could require different aspectual connections in different applications or even in the same application if it must be trig-

gered by different events. Then, according to the imposed domain constrains, we can clearly identify four categories of aspectual connections: basic, query, change and complex.

Basic aspectual connection: the connection triggers the business rule in a specific point of the core functionality (event) the required information by the business rule is either available in the event context or it is global system information. The basic connection description needs the following elements: 1) Business rule elements, such as the class and method that encapsulates the business rule, the required information by the business rule and the business rule return. 2) Event elements, such as the domain class and method that represent the event that triggers the business rule, an indicator of when (before/ after/around) the business rule should be applied regarding the event execution.

Query aspectual connection: the connection triggers the business rule in a specific point of the core functionality but the information required by the business rule is not available in the event context. Then connection must first retrieve the information in order for it to be available when the business rule is applied. In this case, the aspectual connection should manage two events (pointcuts) and two advices, each one with different purposes. The query connection description needs the same elements of basic connections, and also the event (class and method name) where no contextual information should be retrieved plus the data type.

Change aspectual connection: the connection should add new properties (fields/methods) to the core functionality components in order for the business rule to be triggered. It means that the new business rule requires adapting the domain vocabulary. Then, the connection must support the domain adaptation such as the addition of new fields and methods in existing classes. The change connection description includes the same elements as basic connections and the description of the properties that should be added, such as new methods and fields.

Complex aspectual connection: this connection has the same characteristics of query and change connections. The connection has to update the domain for new business rules to be applied, but the needed information for the business rule condition is not available in the event context that triggers the business rule. This connection has the same elements that basic, query and change connections.

This taxonomy is independent of AOP language or base language. It only depends on the domain design and implementation and the requirements of new business rules.

3.2. Aspectual Connection Templates (ACT)

ACT is a simple artifact to identify, register and document the complete connection. In other words, all elements required by the connection could be detailed in ACT. ACT is composed of 7 sections (**Figure 1**). Section A identifies the connection. Section B identifies the business rule that would be triggered by the connection, class and method that encapsulates it. Section C identifies the event that triggers the business rule and when the business rule should be triggered (after/before/around). Section D must be completed when the information required by the business rule is not available in the event context that triggers the business rule, then in this section identifies the event where information must be retrieved. Section E must be completed when business rule requires domain adaptations. Section F identifies the interactions with other connections and the order in which they must be executed. And the last section classifies the connection according to taxonomy decrypted previously.

ACT category is basic if the only completed sections are A, B and C; it is query if Section D is also completed; it is change if Section E is also completed; and it is complex when all sections are completed. Hence Sections A, B and C are mandatory. Section F is completed during step 2 of the approach.

4. Interactions among Aspectual Connections

Aspect interactions, also known as conflicts or interferences, have been the subjects of extensive research with in AO community over the last years. An interaction arises when several aspects need to be executed at the same join point. This problem specifically, occurs when "the behavior of one aspect is affected by the behavior of another aspect", so some interactions are negative and other interactions are inoffensive. The problem can be boarded from early development stages, but the real solution is absolutely constrained by the capacities and mechanism

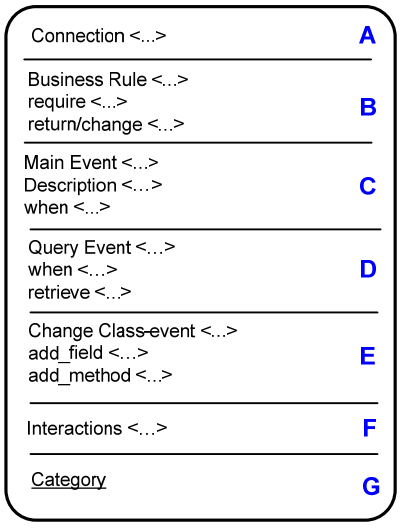

Figure 1. Aspectual connection template (ACT).

of aspect oriented language used in the implementation of connection with aspects. Even, most approaches provide primitive support by letting the programmer specify the order of aspect execution. AspectJ [18] defines the *declare precedence* construct to specify the order of aspects. Strategies more flexible still are not supported by the popular tools. Even so, it is an advantage to know the potential interactions in aspect-oriented application in advance. Particularly in this sense, we have noted that, semantically, it is not the same an interaction between logging and profile aspects, than an interaction between aspects that connects two business rules of discount.

In the context of business rules, where connections are implemented with aspects, an interaction could happen when two or more business rules are triggered by the same event.

4.1. Identification of Shared Events

The taxonomy of aspectual connections and ACT can serve to analyses the interaction among connections. Actually it is possible to anticipate which type of interactions can happen and what events are affected by them. Suppose that C#1 and C#2 connections are documented in two ACT, the potential interactions are described in **Table 1**. The third column indicates what events (of ACT) should present an interaction.

In this way, the interaction could be detected from ACT. Whichever digital format ACTs are registered (such as XML), a simple process could be executed in order to detect automatically the interactions.

4.2. Risk Assessment and Recommendation of Resolutions

After the interaction is detected it is possible to asses its risk and reason about the resolution method required. If the business rules are not mutually excluding, the critical point is posed on the information used by each business rules. The information required by business rules is in the core modules, then connections transfer the information from core to business rules. The aspect mechanism which enables this operation is known as “Exposing Context”. For example, in AspectJ-like languages it is possible to pass the intercepted object and/or method arguments (by means of this, target and args primitives). Another point to consider is how event context is used by business rules whose connections are superimposed. The information can be used in three ways: to read, to write or to read/write.

The risk of interaction could be categorized in four levels: null, low, medium and high.

1) Null Risk: this level of risk indicates the interaction is not dangerous and then is not necessary an explicit resolution. This level of risk occurs when the information used by two connections is not the same.

2) Low Risk: this level of risk denotes that although aspects execution requires a sequence, this sequence is not mandatory, then the order in execution is not crucial. This level of risk occurs when the connection uses the same information only for reading.

3) Medium Risk: this level of risk denotes that aspects execution requires a sequence, this sequence is mandatory, then an explicit resolution is required. This level of risk occurs when both connections uses the same information, one of them writes it, and the another reads it.

4) High Risk: this level of risk denotes that interaction must be managed by developer because of both connections update the same information.

A simple example (**Figure 2**) is used to show how an interaction can be analyzed. Suppose a store where an invoice (Invoice) is issued when a customer (Customer) buys any product (Item). An Invoice consists of some fields such as number of invoice, date of invoice, Customer name, an array of items, subtotal, discount and total. By default, discount field is initialized with cero and the name of Customer is initialized when invoice is instanttiated. Customers have a rewards card in order to add up points. Each time they make a purchase a business rule (BR#1) calculates the points to credit to the card.

Table 1. Points of potential interaction.

C#1	C#2	Point of Interaction
Basic	Basic	main event of C#1 = main event of C#2
	Query	main event of C#1 = main event of C#2
	Query	main event of C#1 = query event of C#2
	Change	main event of C#1 = main event of C#2
Query	Query	main event of C#1 = main event of C#2
	Query	main event of C#1 = query event of C#2
	Query	query event of C#1 = query event of C#2
	Change	main event of C#1 = main event of C#2
Change	Change	main event of C#1 = main event of C#2
	Change	query event of C#1 = main event of C#2

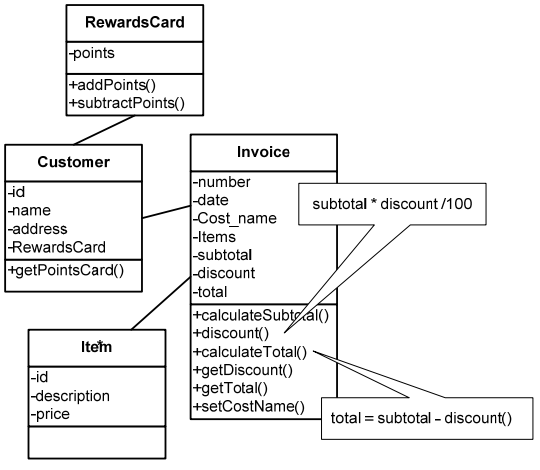

Figure 2. Diagram of simple store.

BR#1: After the system calculates the total of the invoice, every $10 a point must be credited to the rewards card. BR#1 should be implemented by CreditRewardsCard class and RCConnection is the aspectual connection (**Figure 3**).

RCConnection is "query" connection, because of information required by business rule, RewardsCard is not available in the main event. RewardsCard object should be retrieved previously, when the name of customer is initialized.

BR#2: The anniversary of the store is next month, so a special raffle would be organized. The customers participate with their purchases. After the system computes the total of the invoice, every $15, 2 raffle tickets are given. BR#2 is implemented by PrintRaffleTicket class and PRTConnection is the aspectual connection (**Figure 4**). PRTConnection is a "basic" connection.

Figure 5 shows how analysis can be performed. The aspectual connections that are interacting are identified, as well as the business rules that these connections trigger. The shared event and the moment when the interaction occurs are specified. Afterwards the event context used by each business rule is analyzed. In this case, both business rules read the same information (Invoice.total field), but they write different information (RewardsCard and Ticket objects). Then according to the indicated risk proposed, the level of risk of this interaction is low.

Connection: RCConnection

Bussines Rule: CreditRewardsCard
Require: Invoice.Total
Return/change: RewardCard

Main Event: Invoice.calculateTotal()
description
When: after

Query Event: Invoice.setCostName()
When: after
Retrieve: Customer.RewardsCards

ChangeEvent:
Add-Field
Add-Methods

Interactions:

Category: query

Figure 3. ACT of RCConnection.

Connection: PRTConnection

Bussines Rule: PrintRaffleTicket
Require: Invoice.Total
Return/change: Ticket

Main Event: Invoice.calculateTotal()
description
When: after

Query Event:
When:
Retrieve:

ChangeEvent:
Add-Field
Add-Methods

Interactions:

Category: basic

Figure 4. ACT of PRTConnection.

4.3. Interactions between Excluded Business Rules

A more critical scenario is when two o more business rules could be triggered by the same event, but only one of them, could be applied. A special and superior business rule governs and decides what business rule must be executed. Using the same case of **Figure 2**, an example can be:

BR#1: if date of purchase equals Sunday then apply a discount of 5%.

BR#2: if the payment of purchase is with cash then apply a discount of 7%.

Several special business (SBR) rules may be resolving this interaction, for example:

SBR#1: all business rules must always be applied.

SBR#2: if the conditions of BR#1 and BR#2 are true, then apply BR#2.

SBR#3: if subtotal is greater than $1000, then apply BR#1, else apply BR#2.

As we stated before, the most popular and used AOP-tools are not equipped with mechanisms to resolve this type of situations suitably. Hence, the design and implementation of special business rule is more complex, any module class or aspect should evaluate the condition and trigger the correct business rule. Two alternatives could be considered: 1) as any other business rule, the special business rule should be encapsulated in a class and one aspect connects it with the core module; 2) an aspect encapsulates the special business rule.

If we take the second option, the **Figure 6** presents a generic template in AspectJ. In this design it is clear and evident that the aspect is resolving the interaction.

Interaction: *RCConnection, PRTConnection*
Business Rules: *CreditRewardsCard, PrintRaffleTicket*
Event: *Invoice.calculateTotal*
When: *after*
Use of Information Context
CreditRewardsCard **reads** *Invoice.Total and Invoice.Customer*
CreditRewardsCard **writes** *RewardsCard.point*
PrintRaffleTicket **reads** *Invoice.Total*
PrintRaffleTicket **writes** *Ticket*
Risk: *low*
Order: *RCConnection, PRTConnection*

Figure 5. Analysis of interaction.

```
aspect MergeConnections {
   BR b1;
   BR b2;
 pointcut p() : ….
 before () : p()
 { if (condition)
         return b1.apply(…);
   else
         return b2.apply(…); }
}
```

Figure 6. Resolving interaction using merge aspect.

Another choice to resolve this interaction is based on the use of "if" primitive of AspectJ. **Figure 7** presents generic templates of this possibility. In this design it is not evident that pointcuts are designed to resolve an interaction, there are not traces left of the presence of interaction. Also, several AOP-tools do not dispose of "if" primitive.

In spite of the drawbacks of the first option, where we merge both connections in one aspect, other approaches could support these methods with specific mechanisms. A best solution, than AspectJ, it is to use another approach, for example, model of interactions [19].

5. Implementation of Aspect Connection Code

In general, the implementation-level is considered as one of the hardest in software development. Perhaps for this reason, mapping strategies are always recommended. The mapping strategies adopt different forms such as a template, as a set of steps, as a guideline. Mapping strategy could be applied in manually or automatically. In all these cases, mapping strategies are welcomed.

The implementation of connections with aspects is strongly influenced by two factors, 1) the AOP-tool used; 2) the adopted strategy to design and implement the connections with aspects.

Aspect-language programming destination: AOP supports are very different, when we consider the composition mechanisms, programming structures, syntax and semantic constructors, etc. Only to provide a simple idea of this universe of AOP-tools, we mention AspectJ [18], Caesar [20] and Spring [21], where all support AOP for Java programs, but they are completely different. Then mapping strategy is specific of AOP-tool.

Strategies to design and to implement code of aspect connection: Another decision to be taken is how the connection would be implemented. In this sense, we have observed two main strategies. 1) With the goal of achieving the reusability of aspects (pointcuts), Cibrán [8] proposes the next guidelines: there will be an aspect expressing the event that determines the application time of the business rule; an optional aspect exposing unavailable or introducing unanticipated business object to the event, and a last aspect that puts the previous ones together and actually triggers the application of the business rule; 2) Our strategy [5,16] proposes the implementation of each connection in one aspect, then it is possible that one aspect may contain more than one pointcut and advice. The pointcuts are less reusable but the system is easier to maintain. Each strategy has advantage and shortcoming. In this work it is not our intention to analyze the schemes of modularization, our purpose is to prove that ACT and Taxonomy could be used to map aspectual connections to real code. In [16] we have mapped ACT to Spring AOP Framework.

```
aspect ConnectB1 {
   BR b1;
 pointcut p() : ….&& if (condition)
 before (. .) : p(. .)
 {    return b1.apply(…); }
}
aspect ConnectB2 {
   BR b2;
 pointcut q(..) : …. && if (!condition)
 before (..) : q(. .)
 {     return b2.apply(…); }
}
```

Figure 7. Resolving interaction using if primitive.

In this case, using the previous example, a query connection is mapped to AspectJ aspects, applying the different strategies. **Table 2** enumerates and summarizes steps to map ACT to AspectJ using the guidelines of Cibrán and an instance with ACT and aspect code is given in **Figure 8**.

Next, **Table 3** enumerates and summarizes steps to map ACT to AspectJ using our guidelines and an instance with ACT and aspect code is given in **Figure 9**.

Both examples are briefly presented. Afterwards the AOP-tool and strategy of design and implementation are selected; ACT should be registered in an actionable format (as XML) and equipped with more technical details in order to perform automatically the mapping process, with a set of specific mapping rules, as [16].

6. MACS

We have developed MACS, a tool to manage aspectual connections, which follows the approach described in previous sections and implements the presented strategies. The developers only must to enter ACT in simple form. Then MACS automatically performs: 1) application of taxonomy for each aspectual connection; 2) execution of several metrics and queries; 3) detection of interactions among aspectual connections; 4) generation of aspects code in AspectJ or Spring. We have probe the functionality of MACS with different cases.

Table 2. Steps to map ACT to AspectJ.

Step	ACT Section	AspectJ Code
1	C	Create an aspect expressing the pointcut (event) that determines the application time of the business rule (PCMain aspect).
2	D	Create an aspect exposing unavailable business object to the main event (PCQuery aspect).
3	B-C	Create an aspect that puts the previous ones together and actually triggers the application of the business rule (PCTrigger aspect).
4	F	Add declare precedence sentence in PCTrigger aspect.

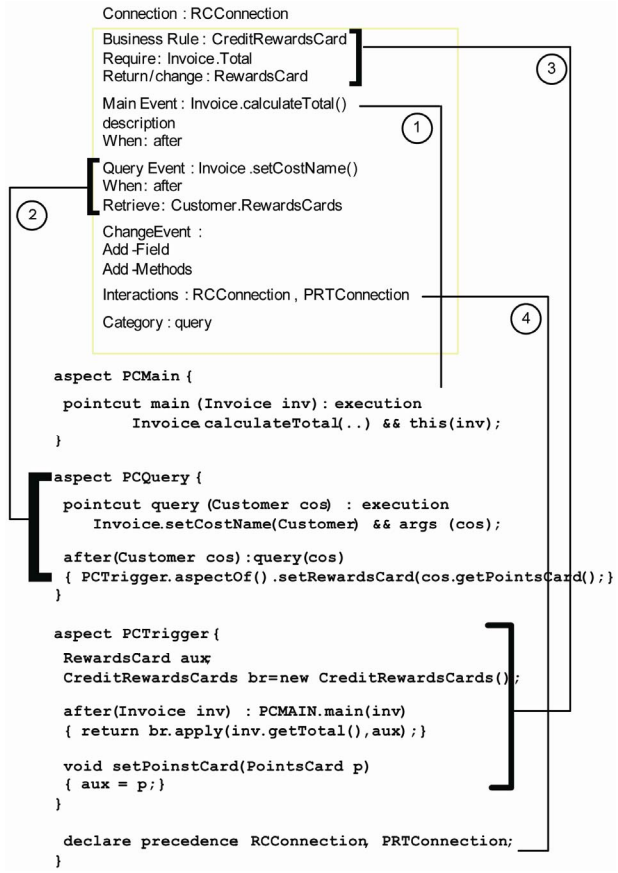

Figure 8. Instance of mapping aspectual connection using guidelines of Cibrán.

Table 3. Steps to map ACT to AspectJ.

Step	ACT Section	AspectJ Code
1	A	Declare aspect.
2	B	Declare and create a field for business rule in aspect (br).
3	D	Declare a field for a value to retrieve in aspect (aux) Declare query pointcut to intercept the query event Create advise to query pointcut.
4	C	Declare main pointcut to intercept the main event Create advise to main pointcut, where a sentence inv the business rule using retain value.
5	F	Add declare precedence sentence.

Figure 10 presents the kernel of MACS. Each aspectual connection is represented as an object according to its classification (BasicAC, QueryAC, ChangeAC or CompleAC). ACManager is responsible of manipulating all aspectual connections. MetricsEngine class and Query-

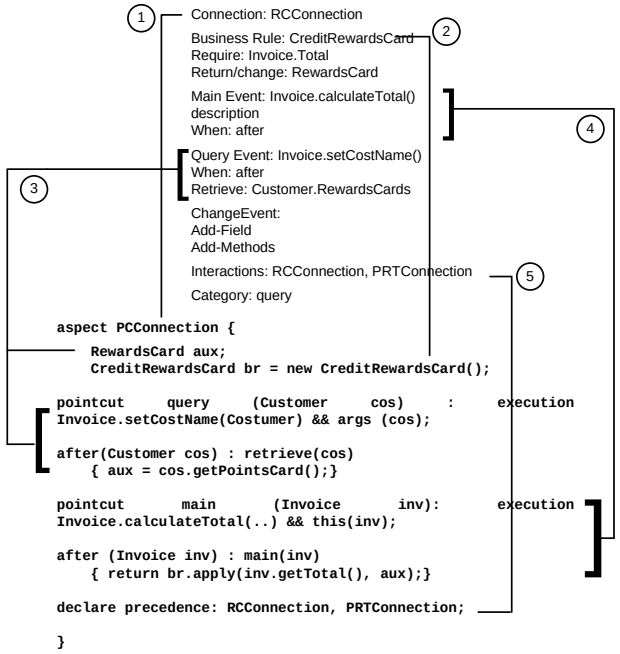

Figure 9. Instance of mapping aspectual connection in only one aspect.

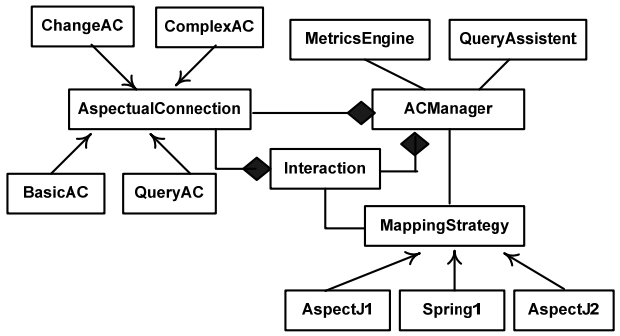

Figure 10. Kernel of MACS.

Assistant class interact with the ACManager, because of they require the aspectual connections container to apply specific operations and calculus. In order to generate aspects code automatically in different AOP-languages, we use Strategy pattern which is represented in MappingStrategy, AspectJ classes, using the approach proposed by Cibrán (AspectJ2 class) or our approach (Spring1 and AspectJ1 classes). The system uses the category of aspectual connection to apply specific steps that map the elements of ACT to code.

Interactions are objects that encapsulate information showed in **Figure 5**. In these objects, a boolean field indicates if the business rules are excluded. When this field is true, the mapping is different; the system generates only one aspect that merges the aspectual connections, such as it is presented in **Figure 6**. This generated code is not complete, the developer should write the condition of if sentence.

7. Related Works

Several methods were proposed to describe business rules, such as templates [11,13], tables [11], natural language [11], XML [22], OCL [23], etc. However, it is difficult to find notations or specific mechanisms to describe their connections. Even several classification of business rules have been exposed [11,12,17], but it does not exist a classification of the business rule connections. In this sense ACT and the taxonomy of aspectual connections presented are relevant contributions.

Cibrán [24] presents a high-level business rule connetions language, this notation specifies the details of the rules integration with the core application and typically denotes an event at which the rules need to be applied, the exact moment when the rule needs to be applied at that event, and the specification of how the required rule information is made available to the rule. She only uses this language to map automatically the connections to JasCo [25] using her guidelines. Treatment of interactions is superficially addressed, where actions are limited to using AOP-tools mechanisms, and interactions between excluded business rules is not contemplated.

Some works have dealt with aspectual connections to compose business rules, but they have been addressed as implementation with different AOP tool, such as AspectJ [8], JasCo [26] and Spring AOP Framework [9]. [7] presents a template to implement the business rules with AspectJ. [27] presents an experience of refactoring Business rule with AspectJ, in an important J2EE application. However, none of these works propose an approach or method to manage aspectual connections in order to its analysis, classification and automatic generation of code.

Other contributions consider the handling of volatile concerns in early stages of software development. For example, an interesting contribution is [4], the authors present a method for handling volatile concerns during early lifecycle software modeling. The method consists of several steps: concern classification, requirements refactoring, model instantiation and model composition. These techniques improve the business rules ant their aspectual connection in modeling activities but not their implementation directly and the interaction problem is lightly analyzed. Along the same line, a framework is proposed to identify volatile and crosscutting concerns at the requirements level [28,29]. The identification of such concerns is based on a crosscutting pattern and simple matrix operations. The approach analyzes the dependence of concerns and crosscutting concerns, but the interactions between them are not directly considered.

8. Conclusions

In this work we outline an approach to connect business rules with aspects. Our approach covers the next objectives: identification of elements of aspectual connections, documentation and registration, classification, interactions treatment and design and implementation of code of aspects. We have divided the activities into three main steps: identification, analysis and construction. In other words, this approach provides answers from beginning (when the need of adding a new business rule arise) to end (when a real code is generated).

The detection and analysis of interactions is done using ACT, it is not required the code. Our exam proposes solution when the interactions include excluded business rules and not excluded business rules. However in these cases there appears evidence of weakness of AOP-tools. These tools are not equipped with powerful and flexible mechanisms to support complex relations and compositions among aspects. Thereby, the options selected are in agreement with current solutions.

The approach is independent of AOP-languages. ACT and Taxonomy are used to generate aspectual connection code in AspectJ or Spring AOP Framework. Also, we have applied different strategies of design of aspects code.

The approach and specifics strategy as ACT and taxonomy, have been implemented in MACS Tools, where we could increase offered services with several queries and metrics.

Notwithstanding, one restriction leads our approach: an aspectual connection links one event with one business rule. That is to say, if a business rule should be linked with two or more events, then two or more connections are required respectively. In the same way, if two or more business rules are applied to the same event, then, two or more connections are required respectively. Only when an interaction between excluded business rules is detected, we solved it using one aspect, but it is not our philosophy, it is a possible solution in AspectJ.

9. Acknowledgements

This work was partially supported by the Universidad Nacional de la Patagonia Austral, Santa Cruz, Argentina.

REFERENCES

[1] M. Jackson, “Software Requirements and Specifications,” ACM Press & Addison Wesley, New York, 1995.

[2] J. Leite J and A. Padua, “A Client Oriented Requirements Baseline,” *Proceedings of the* 2*nd IEEE International Symposium on Requirements Engineering*, York, 27-29 March 1995, pp. 108-115.

[3] G. Kiczales, L. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin, “Aspect-Oriented Programming,” *Proceedings Object-Oriented Programming*, 11*th European Conference*, Jyväskylä, 9-13 June 1997.

[4] J. Araújo and J. Whittle, “Modeling Volatile Concerns as Aspects,” In: E. Dubois and K. Pohl, Eds., *CAiSE* 2006,

LNCS 4001, Springer-Verlag, Berlin, 2006, pp. 544-558.

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

[6] M. Cibrán, M. D'Hondt and V. Jonckers, "Aspect-Oriented Programming for Connecting Business Rules," *Proceedings of the* 6*th International Conference on Business Information Systems*, Colorado, 5 June 2003.

[7] R. Laddad, "AspectJ in Action," Manning Publications Co., Greenwich, 2003.

[8] M. Cibrán and M. D'Hondt, "Composable and Reusable Business Rules Using AspectJ," *Workshop on Software engineering Properties of Languages for Aspect Technologies* (*SPLAT*) *at the International Conference on AOSD*, Boston, 17-21 March 2003.

[9] G. Vidal, J. Enriquez and S. Casas, "Integración de Reglas de Negocio con Conectores Aspectuales Spring," 11*th Argentine Symposium on Software Engineering*, Buenos Aires, 2-3 September 2010.

[10] J. Leite and M. Leonardi, "Business Rules as Organizational Policies," *IEEE IWSSD*9: 9*th International Workshop on Software Specification and Design*, Ise-shima, 16-18 April 1998, pp. 68-76. doi:10.1109/IWSSD.1998.667921

[11] R. Ross, "The BRS Rule Classification Scheme," 2001.

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

[13] R. Ross, "Principles of the Business Rule Approach," Addison Wesley, Reading, 2003.

[14] Arsanjani, "Rule Object 2001: A Pattern Language for Adaptive and Scalable Business Rule Construction," *Technical Report*, IBM T.J. Watson Research Centre, Yorktown Heights, 2001.

[15] Jess homepage, 2011. http://www.jessrules.com/

[16] S. Casas and J. Enriquez, "Mapping Connection Templates to Spring Aspects to Integrate Business Rules," *Workshop of Early Aspects AOSD*, Pernambuco, 21 March 2011.

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

[18] The AspectJ Programming Guide, 2011. http://eclipse.org/aspectj

[19] C. M. Riveill, M. Blay-Fornarino and A. Pinna-Dery, "Transparent and Dynamic Aspect Composition," *Workshop on Software Engineering Properties of Languages and Aspects Technologies*, VII AOSD, Bonn, 20 March 2006.

[20] CaesarJ Homepage, 2011. http://caesarj.org

[21] Spring Framework Guide, 2011. http://www.springsource.org/

[22] XRules Homepages, 2011. http://www.xrules.org/

[23] B. Demuth, H. Hubmann and S. Loecher, "OCL as a Specification Language for Business Rules in Database Applications," *Proceedings of the* 4*th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools*, Springer-Verlag, London, 2001. doi:10.1007/3-540-45441-1_9

[24] M. Cibrán, "Connecting High-Level Business Rules with Object-Oriented Applications: An Approach Using Aspect-Oriented Programming and Model-Driven Engineering," Ph.D. Dissertation, Universiteit Brussel, Brussel, 2007.

[25] D. Suvee, W. Vanderperren and V. Jonckers, "JAsCo: An Aspect-Oriented Approach Tailored for Component Based Software Development," *Proceedings of the* 2*nd International Conference on Aspect-Oriented Software Development*, Boston, 17-21 March 2003.

[26] M. Cibrán, M. D'Hondt, D. Suvee, W. Vanderperren and V. Jonckers, "Linking Business Rules to Object-Oriented Software Using JAsCo#," *Journal of Computational Methods in Sciences and Engineering*, Vol. 5, No. 1, 2005, pp. 13-27.

[27] K. De Schutter, T. D'Hondt, V. Jonckers and H. Doggen, "Experiences in Modularizing Business Rules into Aspects," *ICSM* 24*th IEEE International Conference on Software Maintenance*, Beijing, 28 September-4 October 2008, pp. 448-451.

[28] J. Conejero, J. Hernandez, A. Moreira and J. Araújo, "Discovering Volatile and Aspectual Requirements Using a Crosscutting Pattern," 15*th IEEE International Requirements Engineering Conference*, New Delhi, 15-19 October 2007. doi:10.1109/RE.2007.33

[29] K. van der Berg, J. Conejero and J. Hernández, "Analysis of Crosscutting in Early Software Development Phases based on Traceability," *Transactions on AOSD, Special Issue on Early Aspects*, Springer-Verlag, Berlin, 2007.

