Toward an Automatic Management of Aspectual Connections to Compose Business Rules

Sandra Casas and Franco Herrera

Universidad Nacional de la Patagonia Austral
Campus Universitario, Av P. Rivero S/N,
9400 Río Gallegos, Argentina
scasas@unpa.edu.ar

Abstract. AOP/AOSD is a convenient approach to connect business rules to the domain without altering these components. However in complex applications such B2B and B2C systems, where rules play an important role it is necessary to manage these connections to really assist the developers. The automatic mechanisms are needful too, in order to ease business rules deployment, software maintenance and evolution. Then particular specifications of these operations must be improved. In this work we provide a semi-formal description of the aspectual connections and a set of operations to manage them. The point of view to connections and operations description that we use is taxonomy. Another subject that we outline is the description of particular AOP language. The goal is to provide a more concise notation that can serve as a guideline for the implementation of tools that automatically gather these and related operations.

Keywords: Business Rules, Aspect-Oriented Programming, Aspectual Connections, Volatile Concerns.

1 Introduction

A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business [1].The volatility of the business rules is a problem for the software development. The dynamics of the organizations produces new business rules, or decides to disable the existing business rules, or modifies some aspect of the current business rules. When a current software system is in operation, it is necessary that these modifications be carried out in a fast and easy way. Then business rules implementation is an essential part of any enterprise system; they are applies to many features of business behavior that support policy and strategy changes. Business rules tend to change over time due to new policies, new business realities, and new laws and regulations that appear.

Current mechanisms to implement business rules (Object rule pattern or rule engine systems) require embedding the rule evaluation or their calls in core modules

of the system, thus causing the implementation to be scattered over multiple modules. A change in the rule specification requires changes in all modules that are involved. These modifications are invasive and time-consuming. Further, because of business rules being far more volatile than the core business logic, mixing them together causes the core system to become just as volatile.

For this reason, AOP/AOSD [2] is convenient when providing mechanisms that allow to connect or to integrate the business rules to the domain without altering these components. Just as it is stated in [3], the AOP/AOSD facilitates the constant evolution of this type of concerns. Some contributions [4][5][6][7] show that the aspect-orientation reduces the dependencies and coupling; thus, best reusing is achieved and maintenance efforts reduced. In this context, an aspectual connection links a business rule with core functionality; and thus, it refers to any implementation mechanism (code – xml – annotation) that encapsulates: the object rule invocation; the transmission of the information required by the business rule; the interaction resolution among business rules and the returned information by the rule. However, the encapsulation of connections with aspects is not a trivial task due to the following factors: AOP languages restrictions, software application design and implementation, and design and implementation decisions of aspectual connections as we discussed in [11]. With this encouragement, in the same article, we present taxonomy of aspectual connections which serves to identify the different elements of the possible aspectual connections and the situations where they can occur, in a commendable schema, and we found an additional benefit as it can be possible to identify dependence relations among aspectual connections. Even so, we have observed that it is necessary to manage connections in order to really assist the developers, mainly in complex applications such as business-to-business (B2B) and business-to-consumer (B2C) systems, where rules play an important role and aspectual connections link business rules with the core functionality. The automatic mechanisms are needful too in order to facilitate the business rules deployment, software maintenance and evolution. Then particular specifications of these operations must be improved.

In this work we provide a semi-formal description of the aspectual connections and a set of operations to manage them. The point of view to connections and operations description that we use is the taxonomy of aspectual connections. Another subject is that we outline is description, which is independent from particular AOP language. The goal is to provide a more concise notation that can serve as a guideline for the implementation of tools that automatically gather these and other related operations.

The remainder of this work continues as follows: Section 2 presents the aspectual connections taxonomy as it is a guide for next sections. In Section 3, we describe the abstract aspectual connections manager, which is composed of aspectual connections and a set of operations and functions. In Section 4, we present a simple case study in order to enlighten the previous concepts. Finally, in Section 5, we finish with conclusions, related works and future works.

2. 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 triggered by different events. Then, according to the imposed domain constrains, we can clearly can 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: i) 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. ii) 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.

Other interesting issues are the potential business rule relations; they also cause dependences between their aspectual connections. For example, BR#1: business rule sets a condition to decide if a customer is frequent. Afterwards, BR#2: business rule sets a special discount for frequent customer. In these cases, we say, there is a relation between these business rules, where BR#2 depends on BR#1 and where BR#1 governs BR#2. Our reasoning is that BR#2 would make no sense if the BR#1 is removed. And on the other hand, BR#1 is created to sets states that will be used as a

condition for other business rules. BR#1 by itself does not change anything in the business. Then, all connections description should maintain these relations. Each connection has a list of dependence connections and a list of domain connections perhaps one or both of them are empty.

This taxonomy is independent of AOP language or base language. It only depends on the domain design and implementation and on the new business rules. The taxonomy of aspectual connections was explained in deep in [11].

3. A Framework to Manage the Aspectual Connections

In complex applications where rules play an important role and aspectual connections integrate business rules with the core functionality, it is necessary to manage these connections to really assist the developers. The automatic mechanisms are needful too in order to facilitate the business rules deployment, software maintenance and evolution. The aspectual connections will be isolated in specific package, layer or container in order to assure the separation of concerns. In this context, a dedicated manager is responsible for providing the services to add, remove, search, and analyze the connections.

Next we provide a semi-formal description of aspectual connection to compose business rules. The goal is to distinguish the different requirements of any kinds of connections and, on the other hand, we want to avoid language constructs. Here we have considered that the programming and weaver model of the AOP tools are extremely different. For example AspectJ [13], Spring AOP Framework [14] and CaesarJ [17] are very different, although all these tools support AOP. In implementation terms, a connection is an aspect. But due to the fact that our description is language-independent, we have done without specific AOP linguistic expressions such as “pointcut”, “call”, “aspect”, etc.

We describe the framework using a functional programming style (similar to Haskell [12]). This notation provides a more concise description than natural language and can serve as a guideline for the implementation of tools that automatically gather these and related metrics. We start by describing the aspectual connections manager, followed by the description of the connections and functions used to define the manager operations.

```
manager :: (container, operations)
```

3.1 Aspectual Connections

Connections do not work in isolation. Their functionality is typically implemented in conjunction with a set of business rules classes and domain events. Domain event are classes and interfaces which represent the core functionality of the software applications. One restriction leads our description: a connection links one event with one business rule. That it 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.

We define a container C to be a set of connections C_i, denoted with the following list:

$C = [C_1, C_2, ..., C_n]$

Where C is of type Container and Ci is of type `connection`. Figure 1 summarizes the abstract representation of our programs in the form of a grammar.

According to the proposed taxonomy, a connection can be basic (`basic_connection`), query (`query_connection`), change (`change_connection`) or complex (`complex_connection`).

```
container :: [connection]
connection :: basic_connection query_connection|
              change_connection | complex_connection
basic_connection :: (id_connection, br_elements, event_elements,
relation_connection)
br_elements :: (br_class, br_method, [br_require], br_return)
event_elements :: (event_class, event_method, when)
query_connection :: (id_connection, br_elements, event_elements,
query_elements, relation_connection)
query elements :: (query_class,query_method, query_retrieve, when)
change_connection :: (id_connection, br_elements, event_elements,
change_elements, relation_connection)
change_elements :: (change_class, [add_method], [add_field])
complex_connection :: (id_connection, br_elements, event_elements,
query_elements,change_elements,relation_connection)
relation_connection :: ([domain],[depend])
```

Figure 1. Aspectual connection description.

A basic connection is a tuple whose elements are a connection identifier (`id_connection`), business rule elements (`br_elements`), event elements (`event_elements`) and relations connection. In turn, a `br_elements` is denoted as tuple of business rule class name (`br_class`), method that must be invoked to trigger the rule (`br_method`), the input types of input arguments (`[br_require]`) and the return type (`br_return`). An `event_elements` is denoted as tuple of event class (`event_class`) and method name (`event_method`) where the business rule must be applied, and when it can be applied (`after/before/around`). The relation connection is a tuple composed by two lists: one contains the connections that is independent of this connection (`[domain]`) and the other list contains the connections in which this is dependent (`[depend]`).

A query connection has the same elements as a basic connection and new query elements (`query_elements`). Query element is a tuple that denote a different context of the event (`query_class, query_method`) and in which connection should retrieve information (`query_retrieve`) for the business rule.

A change connection has the same elements as a basic connection and new change elements (`change_elements`). Change element is a tuple that denote the new properties that connections should add to the domain. This tuple is formed by the

class to be changed (`change_class`), and the list of new methods (`[add_method]`) and fields (`[add_field]`). A complex connection has the same elements as a basic, query and change connections.

3.2 Aspectual Connections Manager

In this section we provide a semi-formal description of the needful operations and functions to manage the aspectual connections that compose the business rules. We have classified these operations in: auxiliary functions, query operations; update operations; and measure functions.

3.2.1 Auxiliary Functions

count. This function returns the number of elements (n) in a list ([a]) of any type.

```
count :: [a] → n
```

search. This function searches an element (x) in a list ([a]) and returns a boolean value (b).

```
search :: [a] → x → b
```

times. This function counts the times an element (x) is in a list ([a]). (where [a] and x are any type)

```
times :: [a] → x → n
```

extract. This function extracts any repeated elements from a list ([a]), (where `a` is any type).

```
extract :: [a] → [a´]
```

3.2.2 Update Operations

Let `c` be a connection and [connection] a container, we define the following update operations:

add. This operation adds a new connection into the container.

```
add :: c→[connection]→ [c | connection]
```

remove. This operation removes a connection from the container.

```
remove:: c→[c|connection]→ [connection]
```

3.2.3 Query Operations

Let [connection] be a container we define the next query operations:

BCL. This operation receives as input a list of connections of the container and returns the list of basic connections.

```
BCL :: [connection] → [basic_connection]
```

QCL. This operation receives as input a list of connections of the container and returns the list of query connections.

```
QCL :: [connection] → [query_connection]
```

CHCL. This operation receives as input a list of connections of the container and returns the list of change connections.

```
CHCL:: [connection] → change_connection]
```

CCL. This operation receives as input a list of connections of the container and returns the list of complex connections.

```
CCL :: [connection]→[complex_connection]
```

BRL. This operation receives as input a list of connections of the container and returns the list of business rule elements in connections.

```
BRL :: [connection] → [br_elements]
```

EL. This operation receives as input a list of connections of the container and returns the list of domain event in connections.

```
EL :: [connection] → [event_elements]
```

CRL. This operation receives as input a list of connections of the container and returns the list of dependence relations in the list.

```
CRL::[connection]→ [relation_connection]
```

3.2.4 Analysis Functions

Let C be a container. We define the following analysis functions:

dominate. This operation receives as input a connection and a list of connections and returns true if c is a member of the list of dominate connections.

```
dominate ::  c → [connection] → boolean
dominate (c, C) = search(c (CRL ( C ))
```

depend. This operation receives as input a connection and a list of connections and returns and returns true if c is a member of list of depend connections.

```
depend :: c → [connection] → boolean
depend (c, C) = search(c (CRL ( C ))
```

3.2.5 Measure Functions

Let C be a container. We define the following metrics:

NOC. Number of connections.

```
NOC (C) = count (C)
```

NOBC. Number of basic connections

```
NOBC (C) = count (BCL (C))
```

NOQC. Number of query connections

```
NOQC (C) = count (QCL (C))
```

NOCHC. Number of change connections

```
NOCHC (C) = count (CHCL (C))
```

NOCC. Number of complex connections

```
NOCC (C) = count (CCL (C))
```

NOBR. Number of business rules in connection.

```
NOBR (C) = count (extract (BRL (C))
```

NOE. Number of events in connections.

```
NOE (C) = count (extract (EL (C))
```

CARDBR. This function computes the cardinality of a business rule (br). That is to say, how many connections there are for specific business rule.

```
CARDBR (br, C) = times (br BRL (C))
```

CARDEV. This function computes the cardinality of an event (ev). That is to say, how many connections there are for specific event.

```
CARDEV (ev, C) = times (ev EL (C))
```

Other relevant operations for the manager are the mapping functions. A set of functions which transforms the aspectual connections with the structure of Figure 1 into aspects, in specific aspect-programming language, as AspectJ or JasCo.

4. Simple Study Case

This section enlightens the approach described in the previous section by means of a simple case study based on a store, where the logic of the business dictates that firstly the costumer orders are registered (Order), and these operations include the costumer data (Costumer) and requested items (Item). Later on (the same day or another), when the costumer wants to place the order and pays for it, the invoice (Invoice) is issued, then the system calculates subtotal, discount and total. Each invoice keeps a copy of customer details (CostumDetails) for printing. After the invoice is created, costumer purchase quantity is incremented (inv_count field in Costumer class). Figure 2 shows a simplified diagram of the store.

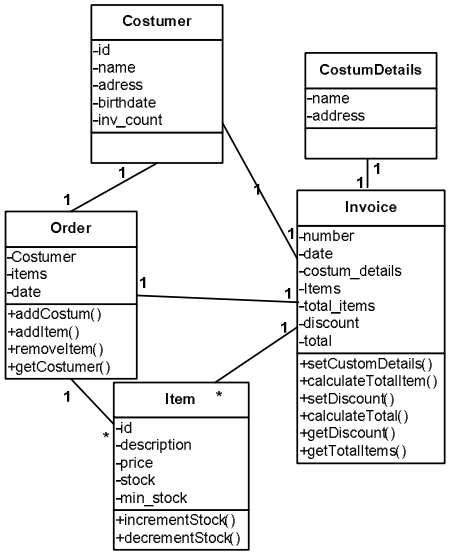

Figure 2. Summarized diagram of the Store.

In Table 1, a set of discount business rules are described. Some of these rules set discount field of Invoice according to any condition, or they set any field of Costumer or Item that should be used when issuing an Invoice. The business rules are classes that implement *condition()*, *action()* and *apply()* methods, such as suggested by the Object Rule pattern. Below, in Table 2, the set of aspectual connections are enumerated. In the second column, it is the aspectual connection description according to the notation presented in Figure 1. In the third column, the aspectual connections are classified according to the taxonomy presented in Section 2.

Table 1. New business rules

Business Rule	Class
BR#1: *if* order date and invoice date are equivalent *then* apply a discount of 3% when issuing the invoice.	BrDateDiscount
BR#2: *if* invoice date is equivalent to with costumer birth date *then* apply a	BrBirthdateDiscount

discount of 2%	
BR#3: *if* costumer purchases are grater than 20 *then* the costumer is frequent.	BrFrequent
BR#4: *if* the costumer is frequent *then* apply a discount of 0,5% when issuing the invoice.	BrFrequentDiscount
BR#5: *if* the invoice date is 30 days after the order date *then* the system updates the items price before issuing the invoice.	BrPriceUpdate
BR#6: *if* the last invoice of a frequent costumer was 180 days ago, *then* the costumer is not frequent.	BrRemoveFrequent
BR#7: *if* the stock item is less than the minimum stock *then* the item is in promotion.	BrPromotion
BR#8: *if* an item is in promotion *then* its price is reduced by 25%.	BrPromotionDiscount

Table 2. Aspectual connections description

BR#	Aspectual Connection	Taxonomy
1	(C#1 (BrDateDiscount apply [Order Invoice] none) (Invoice new before)([][]))	Basic
2	(C#2(BrBirthdateDiscount apply [Invoice Costumer] double) (Invoice calculateTotal, before) (Invoice new [Costumer] Date after) ([][]))	Query
3	(C#3 (BrFrequent, apply [Costumer] boolean) (Invoice calculateTotal after) (Costumer [setFrequentCostumer getFrequentCostumer] [frequent]) ([C#4][]))	Change
4	(C#4 (BrFrequentDiscount apply [Costumer] boolean)(Invoice calculateTotal before) ([] [C#3]))	Basic
5	(C#5 (BrPriceUpdate apply [Order] none) (Invoice new before) ([][]))	Basic
6	(C#6 (BrRemoveFrequent apply [Customer] boolean) (Invoice new after)([][]))	Basic
7	(C#7 (BrPromotion apply [Item] none) (Item decrementStock after) (Item [setPromotion getPromotion] [promotion] boolean) ([C#8][]))	Change
8	(C#8 (BrPromotionDiscount apply [Item] boolean) (Invoice calculateTotalItem before) ([][C#7]))	Basic

Lastly, if `C = [C#1 C#2 C#3 C#4 C#5 C#6 C#7 C#8]`, then in Table 3 some functions and their results are shown.

Table 3. Functions applied to study case.

Functions	Result
dominate(C#8 C)	True
dominate(C#1 C)	False
depend(C#7 C)	True
depend(C#4 C)	False
NOC (C)	8
NOBC (C)	5
NOQC (C)	1
NOCHC (C)	2
NOCC(C)	0
NOBR (C)	8
NOE (C)	4
CARDBR(C BrPriceUpdate)	1
CARDEV (C (Invoice calculateTotal before))	3

With this simple and short example, we have proved that separation and isolation

of connections in aspects enhance design, implementation and maintenance of application. This set of operations and functions allow to manage aspectual connections and to obtain system information. However, manual performance of operations and functions could be tedious and error-prone. An automatic manager of aspectual connections is needful in order to really assist to developers.

5. Conclusions

In this final section, we present related works and a summary.

5.1 Related Works.

Several methods were proposed to describe business rules, such as templates [20][22], tables[20], natural language[20], XML[23], OCL[24], etc. However, it is difficult to find notations or specific mechanisms to describe their connections.

Even several classification of business rules have been exposed [1][20][21], but it does not exist a classification of the business rule connections, then the taxonomy of aspectual connections presented is a relevant contribution.

Cibrian [15] presents a high-level business rule connections 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 aspect-oriented language [16].

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 [9] and Spring AOP Framework [10]. [6] presents a template to implement the business rules with AspectJ. [7] presents an experience of refactoring Business rule with AspectJ, in an important J2EE application. However, none of these works proposed “the management of aspectual connections”, in order to its future automation.

Other contributions consider the handling of volatile concerns in early stages of software development. For example, an interesting contribution is [3], 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. Along the same line, a framework is proposed to identify volatile and crosscutting concerns at the requirements level [18][19]. The identification of such concerns is based on a crosscutting pattern and simple matrix operations.

5.2 Summary

In this work we emphasize that the volatility of business rules is a critical factor in order to develop, maintain and evolve the software applications therefore it is advisable to isolate the management of aspectual connections and their specific operations to control them automatically. This approach assures a suitable separation of concerns. Our notation is language-independent, then, it serves to implement more sophisticated tools that automate the aspectual connections management. However, this notation also serves to map the connections to a specific AOP language. In this work we present some operations, but this set is not complete. For example, it can be completed with operations to resolve interactions among aspectual connections.

Currently, we are developing a tool in order to implement the management of aspectual connections following the descriptions mentioned in this work. This tool, must map the aspectual connections to AOP languages as well.

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

References

1. Business Rule Group. 2001. Defining Business Rules: What Are They Really?. http://www.businessrulesgroup.org/.
2. 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.
3. 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.Springer-Verlag Berlin Heidelberg 2006
4. Cibrian M., D'Hondt M., & Jonckers V. 2003. Aspect-oriented programming for connecting business rules. In Proceedings of the 6th International Conference on Business Information Systems. Colorado, USA.
5. Laddad R. 2003. AspectJ in Action. Manning Publications Co.
6. 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.
7. Cibrán A., Suvée D., D'Hondt M., Vanderperren W., Jonckers V. 2004. Integrating Rules with Object-Oriented Software Applications using Aspect-Oriented Programming. ASSE in 33th JAIIO – Argentina
8. 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.
9. 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).
10. Vidal G., Enriquez J. and Casas S. 2010. Integración de Reglas de Negocio con Conectores Aspectuales Spring. 11th Argentine Symposium on Software Engineering - Argentina – 2010

11. 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.
12. Thompson S. and R. Birds. 1999. "Haskell: The Craft of Functional Programming", 2nd edition. Addison-Wesley
13. The AspectJ Prog. Guide, http://eclipse.org/aspectj
14. Spring Framework Guide http://www.springsource.org/
15.Cibrán M. 2007. Connecting High-Level Business Rules with Object-Oriented Applications: An approach using Aspect-Oriented Programming and Model-Driven Engineering. Phd Tesis Universiteit Brussel.
16. Vanderperren, W., Suvee, D., Cibrán, M., Verheecke, B. and Jonckers, V. 2005. Adaptive Programming in JAsCo. In Proceedings of AOSD, ACM Press, Chicago, USA
17. CaesarJ homepage, http://caesarj.org
18. Conejero J., Hernández J., Moreira A., and Araújo J. 2007. Discovering Volatile and Aspectual Requirements Using a Crosscutting Pattern. 15th IEEE International Requirements Engineering Conference. India.
19. Berg, K. van den, Conejero, J. M., and Hernández, J. 2006. Analysis of Crosscutting in Early Software Development Phases based on Traceability. International workshop on Early aspects at ICSE. China.
20. Ross R. The BRS Rule Classification Scheme. 2001.
21. Date C. J. What Not How: The Business Rules Approach to Application Development. Reading, Mass. Addison-Wesley Longman Inc. 2000.
22. Ross R. Principles of the Business Rule Approach. Addison Wesley. 2003
23. XRules homepages, http://www.xrules.org/
24. 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