Dynamic Business Rules for Web Service Composition

María Agustina Cibrán
Vrije Universiteit Brussel
Pleinlaan 2
1050 Elsene
++32(2)629.29.64
mcibran@vub.ac.be

Bart Verheecke
Vrije Universiteit Brussel
Pleinlaan 2
1050 Elsene
++32(2)629.38.13
Bart.Verheecke@vub.ac.be

1. INTRODUCTION

The domains of many software applications are inherently knowledge-intensive. Examples of such domains are e-commerce, the financial industry, television and radio broadcasting, hospital management and rental business. Part of this knowledge is rule-based, typically representing knowledge about policies, preferences, decisions, advice and recommendations. The current software engineering practices result in software applications that contain implicit rule-based knowledge, which is tangled with the object-oriented core functionality. Nowadays, rule-based knowledge has become a hot topic and is also referred to as business rules [8, 12, 17].

On one hand we have conducted previous work in the field of business rules [5, 6] in which we observe the crosscutting nature of business rules connectors and the suitability of AOP for their implementation.

On the other hand, we are working on the design and development of the Web Services Management Layer (WSML) [7, 16], management middleware in between client applications and the web services. Such a management layer allows the definition of web services compositions to provide the functionality requested by client applications. Service compositions are expressed using a process-based language. An example of such a process-based language is WS-BPEL [2], a service choreography and orchestration language that allows the definition of business processes as interactions between web services in order to achieve a certain goal.

In this paper we focus on business rules present in the domain of web services compositions. We observe that composition business rules govern different aspects of the composition: *how* services need to be composed together, *how* suitable services can be discovered to fill in the roles of the composition, *which* services should be chosen at deployment time to carry out the activities of the process, and *how* the composition should adapt to the changing business environment. In this paper we only focus on the last kind of composition business rules.

We observe that support for explicitly defining business rules is inexistent or hardly supported in current business processes languages. Business rules are only implicitly expressed and their implementation is tangled with the core business process. As a consequence it is difficult to reason about and to evolve both parts independently, the core composition and the business rules. WS-BPEL, as a representative example of process-based languages, fails at providing support for explicitly representing the business rules in a clean and decoupled way.

Building on top of our previous work on business rules and AOP, we observe that composition business rules crosscut the service compositions and thus AOP can contribute to achieve their decoupling.

In the rest of the paper, we identify different categories of business rules that govern *how* the compositions should adapt to the changing business environment. In particular we focus on *dynamic business rules*, rules that take decisions on how to modify the core composition based on advanced patterns of execution history. We analyze the different kinds of AOP features needed for the realization of such rules. The JAsCo AOP language [13] is used as an example AOP technology. In particular, this paper shows how JAsCo stateful aspects [15] are suitable for realizing the identified dynamic business rules.

This paper aims to contribute by providing useful examples of dynamic aspect behavior, meaning in this context, the invocation or change of aspect behavior based on the dynamics of program execution.

2. BUSINESS RULES

The Business Rules Group defines a business rule as a statement that defines or constrains some aspect of a business. It is intended to assert business structure or to control the behavior of the business [14]. A significant characteristic of business rules is that they tend to change whenever the business policies they embody change, which is more often than the core application functionality does [1][11][17]. Examples of business rules are "*If a customer has purchased more than 20 books, then he or she becomes a frequent customer*" and "*If a customer is a frequent customer, then he or she gets a 10% discount*". Business rules are applied at events which are well-defined point in the execution of the core application functionality. Example events are "*before the price of a product is retrieved*" and "*after the customer has checked out*", at which the two examples rules are applied.

As business domains become more and more complex, it is fundamental to explicitly capture business processes and policies as business rules. The Business Rules Approach [17] states that it is crucial to implement them adhering to four objectives: separate business rules from the core application, trace business rules to business policies and decisions, externalize business rules for a business audience, and position business rules for change.

3. PROBLEM STATEMENT

Using a process-based language, a business process can be realized by specifying how different services interact to carry out a certain goal. In the definition of a business process a set of activities is identified. Each activity is associated with a role,

which is mapped to a concrete web service or web services composition at deployment time in order to perform the functionality of the business process.

Currently, WS-BPEL [2] is one of the most promising process-based languages, candidate for standardization. WS-BPEL (Business Process Execution Language) for Web services is an XML-based language designed to enable task-sharing for a distributed computing - even across multiple organizations - using a combination of web services. Using BPEL, a programmer describes a business process that will take place across the Web in such a way that any cooperating entity can perform one or more steps in the process the same way. In a supply chain process, for example, a BPEL program might describe a business protocol that formalizes what pieces of information a product order consists of, and what exceptions may have to be handled. The BPEL program would not, however, specify how a concrete web service should process a given order internally.

We observe that the problems encountered when implementing business rules in object-oriented languages [6, 10] also arise in process-oriented languages. These problems occur due to the impossibility of achieving the following objectives in an oblivious way, i.e. without having to manually change the core application: (1) connect business rules to core application events which depend on run-time properties, (2) retrieve the needed information and make it available at those events when the rules are applied, (3) reuse the rules by connecting them at different events, (4) combine, prioritize and resolve rule interferences and (5) achieve all this preferably dynamically. The reason of this impossibility is that the rules are implicitly represented as *if-then* statements, and as result they appear tangled and scattered in the core application.

A business process written in WS-BPEL is one monolithic specification. It does not support the definition of business rules in a clean, modularized and reusable way and the specification of the rules gets tangled with the main process itself. Changes in the workflow due to changes in business requirements need to be done manually and invasively. The only support WS-BPEL offers is a limited kind of rules such as alternatives between tasks and repetitions based on business logic. Only limited workarounds or no support at all is provided for certain business rules such as time or order constraints between activities.

4. OUR APPROACH

We aim at defining service composition driven by explicit business rules. It is important to explicitly represent them since they tend to change faster than the core business processes. They are very volatile since they need to adapt to business requirements and thus should be decoupled. In order to achieve highly flexible service compositions we observe the need for defining rules explicitly and decoupled from the service composition itself. In the remainder of this section, we will illustrate how aspects are useful in this regard.

As mentioned before, in this paper we focus on composition rules that govern *how* the core composition needs to be adapted according to changing business knowledge. These rules will decide whether to add, replace, change or remove activities that are present in the core composition. In particular we consider ***dynamic business rules***, rules whose triggering events and/or conditions are based on the dynamics of the execution of the core application. To illustrate this kind of composition business rules, consider the following example application. It describes the business process of buying books online. In this context, customers of the shop can send in a quote request. If the customer is a valid customer then a quote is sent back and remains valid for a two-day period in which the customer can purchase an item. On the contrary, if the customer is not registered in the shop, then an error message is returned. During the time the quote is valid, the customers can place orders for buying books which are accepted by the shop. The next step in the business process is the payment of the ordered books. Depending on the results of this activity, the workflow continues with the shipping of the goods or with the refusal of the order, if the card payment is not authorized for instance.

In the following sections we identify different categories of dynamic business rules in the domain of service compositions. Examples of these categories are provided based on the introduced scenario. These rules are triggered depending on the dynamics of core compositions execution flow. Note that we are not presenting an exhaustive categorization, but the intention is to provide significantly different example categories of service compositions business rules that serve as a basis for the identification of AOP solutions.

Examples of a possible implementation in JAsCo will be presented for each category. JAsCo is an AOP language tailored for the component based context. JAsCo builds on top of Java and introduces two additional entities: aspect beans and connectors. An aspect bean is an extended version of a regular Java bean and allows describing crosscutting behaviour by means of a special kind of inner class, called a hook. Aspect beans are specified independently of concrete component types and API's, making them highly reusable. A connector on the other hand, is used for deploying one or more aspect beans within a concrete component context. In addition, connectors are able to specify explicit precedence and combination strategies in order to manage the cooperation among several aspects that are applicable onto the same join point. In addition, the JAsCo technology provides an extensive run-time infrastructure. Using this infrastructure, aspects remain first-class entities at run-time and dynamic aspect addition and removal becomes possible.

Sections 4.1, 4.2 and 4.3 discuss different example categories of dynamic business rules.

4.1 Category 1

Condition: occurrence of behaviors in a specific sequence. Additionally, checks on properties of business objects can be specified on the identified sequence

Action: addition of extra activity to the core process

This category of rules is considered when certain activities can be executed in different orders in the core composition. For instance, consider the case where activities a, b and c are part of the workflow, and the execution sequences a→b→c, b→c→a and c→b→a can occur in the core composition. The rule checks whether a particular path is followed, for instance the path a→b→c. The condition can additionally define extra checks on properties of business objects to be done at any of the points in the identified path. As an action the rule defines the insertion of an extra activity in the core composition, which needs to be added

at an execution point posterior to the identified path on which the condition is checked.

Note that the activities a, b and c can be either consecutive (they occur one after the other) or not (other behaviors can be interleaved in between the identified activities in the path). Moreover, they can occur within the control flow of each other or outside. The path a → b means that the execution of activity b should be triggered after the execution of a is triggered, either within the control flow of a or after the execution of a is completed.

Example: suppose that the payment of the books can be done electronically by credit card or by cash. Consider a business rule that specifies that "*if not trustworthy customer and cash payment selected then check whether the payment has been received before shipping the products*". Then, before the actual shipping we first need to check whether the money was effectively received before dispatching the items. This extra check will determine whether to continue or not with the original workflow. This check is not done if the electronic payment branch was followed instead. Moreover, we only want to add this extra checking if the customer is registered in the system as *not trustworthy*.

In this example, the condition needs to check whether the sequence *a:login* → *b:cashPayment* is executed and check the trustworthiness of the customer when *a:login* is performed. The action identifies the addition of an activity that would verify the reception of the cash and act accordingly. This activity is added *before c:shipping*.

Solution: A naïve solution using current process-based languages would opt for adding variables in the core composition to keep track of whether the customer is trustful and the execution path that was taken (either the electronic payment or the cash one). Next, before allowing the shipment of the purchased products, those variables would need to be consulted in order to decide whether the extra checking for receiving the money is needed. However, this solution implies tangling the core composition with code for the implementation of the business rule. Moreover, the business rule results scattered in the composition and its identity is lost. Aspects help to avoid tangling the core composition as the variables and extra checking code would be encapsulated in a single module, the aspect, outside the main composition. However, keeping track of these variables inside the aspect code might be tedious and unclean.

```
class UnknownPaymentAspect {
  hook UnknownPaymentHook {
    UnknownPaymentHook(
      logIn(String username, String password),
      cashPayment(Order order),
      shippingOrder(Order order)) {
        start > logInCustomer;
        logInCustomer: execute(logIn) > payment;
        payment: execute(cashPayment) > shipping;
        shipping: execute(shippingOrder);
    }
    isApplicable logInCustomer(){
        return !store.trustworthyCustomer
                          (username, password);
    }
    replace shipping() {
        if (PurchaseDepartment.cashReceived(order))
          then proceed();
        else System.out.println
                ( "Shipping cannot proceed");
      }
    }
}
```

Code fragment 1 – Stateful aspect for payment check

JAsCo supports the definition of stateful aspects [15], aspects that are triggered on protocol history conditions. Stateful aspects allow achieving a cleaner implementation of this business rule, it is possible to specify the desired execution path of interest and plug in the crosscutting functionality at any time in the execution of the identified path. In this case, the path of interest is the execution of the sequence of activities *a:log-in* → *b:cashPayment* → *c:shipping*. The crosscutting code is the extra check for receiving the money that needs to be done before allowing the actual shipping of the purchased goods. The JAsCo stateful aspect is shown in code fragment 1.

Note that this aspect should be instantiated *perThread* to avoid inconsistencies when concurrent access.

This solution based on aspects that are triggered on protocol history conditions is much cleaner since the execution path of interest for the pluggability of the aspect behavior is explicitly captured. Conditions can be checked at the different transitions in the path as well as the extra/replacing behavior can be plugged at each transition. This solution is illustrated in Figure 1.

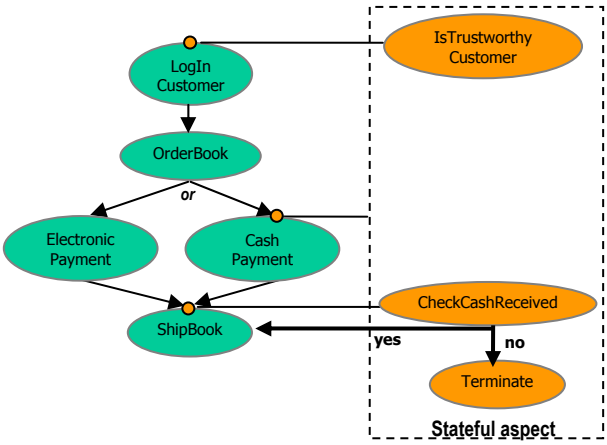

Figure 1 – Stateful aspect for adding additional payment checking to e-commerce business process

4.2 Category 2

Condition: checks on properties of business objects at a specific point in time during the process execution

Action: addition of extra activity in the process that needs to be applied on the execution of behaviors that occur in a specific sequence

At a certain point in the execution of the composition a condition that involves checking a certain property of a business object is checked. Based on the result of this checking, it is decided whether to add an extra crosscutting behavior. In order to execute, this behavior operates on different execution points of a defined

sequence of events that occur later on, when future tasks in the execution path are executed.

Example: This example represents a conditional time constraint: "*If a purchase order is received and labeled as urgent then the whole process payment-shipping-delivery should occur within a maximum of 3 days. Otherwise, the customer is not charged for the purchase*".

Solution: In this example, the condition of the rule is checked at a certain point in time, which is the event *a:orderBook*. The result of this check will determine the pluggability of the *monitoringTime* activity. This extra behavior is applied on a sequence of events, *b:payment* → *c:shipping*, since it needs to measure the execution time of these two activities.

The solution using stateful aspect in JAsCo looks as follows:

```
class ConditionalTimeConstraintAspect {
  hook ConditionalTimeConstraintHook {
    ConditionalTimeConstraintHook(
     orderBook(CustomerId customer, BookId book,
               Priority priority),
     paymentOrder(Order order),
     shippingOrder(Order order)) {
       start > placeOrder;
       placeOrder: execute(orderBook) > payment;
       payment: execute(paymentOrder) > shipping;
       shipping: execute(shippingOrder);
    }
    isApplicable orderBook(){
        return priority.isUrgent();
    }
    before payment() {
      timestampbefore=System.currentTimeMillis();
    }
    after shipping() {
      timestampafter=System.currentTimeMillis();
      if (//time difference not OK)
        store.reimbourseCustomerOfOrder(order);
    }
  }
}
```

Code fragment 2 – Stateful aspect for timing constraint

Figure 2 illustrates this example.

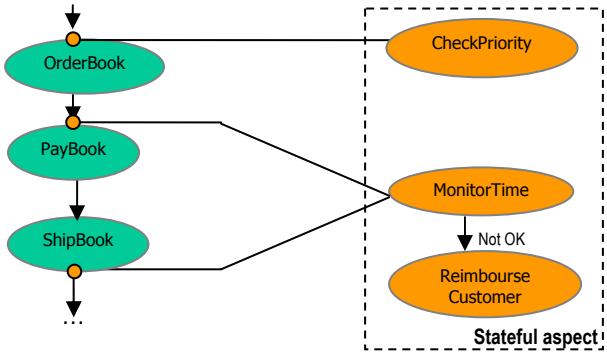

Figure 2 – Stateful aspect for adding additional time monitoring to payment-shipping activities in e-commerce workflow

4.3 Category 3

Condition: execution of behaviors in a sequence that is not allowed

Action: perform extra behavior

When parallel threads of tasks executions are possible in core composition, it might be desirable to restrict certain paths depending on a certain condition. A constraint rule restricts which paths are allowed, filtering out the not desired ones.

Example: An execution order constraint between tasks is enforced by a business rule. Imagine a scenario where the shipment and payment activities belong to different execution threads of parallel activities. Suppose a business rule that states that "*goods can only be shipped after payment has been received*". If the opposite occurs, then notify the manager of the shop.

Solution: JAsCo stateful aspects support triggering crosscutting behavior on the opposite of a protocol using the *complement* keyword. In this example such a feature is useful since the manager needs to be notified whenever the protocol is not respected. The solution looks as follows:

```
class ExecutionOrderAspect {
  hook ExecutionOrderConstraintHook {
    ExecutionOrderConstraintHook(
      paymentOrder(Order order),
      shippingOrder(Order order),
      methodsContext(..args)) {

        complement[execute(methodsContext)]:
        start > payment;
        payment: execute(paymentOrder) > shipping;
        shipping: execute(shippingOrder);
    }
    replace complement() {
       Manager.notify
         ("Shipping done before payment");
    }
  }
}
```

Code fragment 3 – Stateful aspect for payment execution order

```
static connector ExecutionOrderConnector {
  ExecutionOrderAspect.ExecutionOrderHook checker
  = new ExecutionOrderAspect.ExecutionOrderHook (
   void paymentService.pay(Order order),
   void shippingService.ship(Order order), {
      void paymentService.pay(Order order),
      void shippingService.ship(Order order)
   }
 );
}
```

Code fragment 4 – Connector for payment execution order

The `ExecutionOrderAspect` in code fragment 3 states that if the specified protocol is not followed (e.g. the shipping is done before the payment) the original behavior of the core application is replaced by a notification which is sent to the manager. By using the complement keyword, the specified advice is only executed when the protocol is *not* followed.

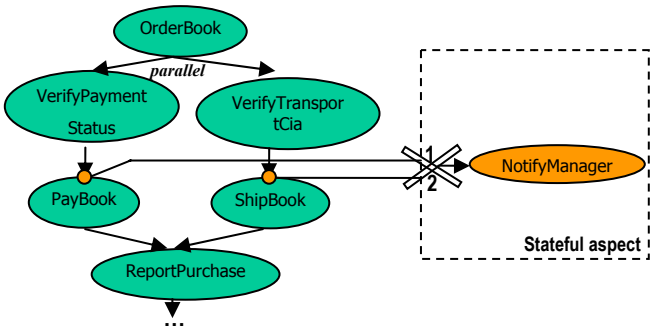

Figure 3 – Stateful aspect for triggering notification when a certain path is not respected in e-commerce workflow

5. Related work

Previous research focused on the applicability of AOP for business rules. Experiments have been done in AspectJ and JAsCo illustrating the connection of business rules with the core object-oriented application [5][6]. Other related previous research focused more deeply on the implementation level, more specific on the integration between the object-oriented paradigm and rule-based languages as a rule-based approach is a more suitable paradigm to implement business rules in [9]. Hybrid aspects were proposed to achieve an oblivious integration at the language and programming level.

However, in some situations, applications cannot afford to use a rule-based programming language for implementing their business rules since it might be too costly to incorporate support for a rule engine for the kind and amount of rules under consideration. Rule engines are proprietary, expensive products and the learning curve can be steep and unaffordable for small projects. Thus, a more lightweight approach is needed for the definition and implementation of the business rules. In this context, ongoing work is being carried out that envisions the definition of a high level business rule language for the specification of the rules, very close to natural language. This language allows the definition of business rules independently of any implementation detail. This way, business rules are defined using the concepts defined in a business model. A business model contains the different elements of the domain under consideration. We are investigating the definition of a very configurable and extensible business model in order to be extended with different terms present in different domains such as, in this case, the web service composition context. An automatic translation of the rules defined in this high level language to a possible object-oriented representation is under development as well as the transparent and automatic generation of the connectivity JAsCo aspects for their integration with the core application.

In [3], an AOP extension for BPEL is proposed. Examples of business rules written in AO4BPEL are given in [4], illustrating the use of AOP for decoupling business rules. This approach is dynamic in the sense that it is possible to plug-in and out the aspects at run-time. As it is an interpretation based approach, aspects can be plugged in at run-time and triggered when the interpreter reaches their pointcut definition. Pointcuts in AO4BPEL are based on Aspect's pointcut model. As most current mainstream AOP languages, AspectJ's pointcuts (with the exception of *cflow()*) cannot refer to the history of previously matched pointcuts in their specification. Thus, AO4BPEL does not allow the triggering of aspects depending on protocol history conditions.

6. CONCLUSION

In this paper we have presented different example categories of business rules that are applicable in the web service composition context. Business rules that are based on the dynamics of the core composition are addressed meaning that either the triggering of the rules, their conditions or actions are based on protocol history. As a consequence of their application, crosscutting behavior is plugged in which results in the addition, change or removal of activities in the base composition. We provide example categories of dynamic business rules in web services compositions. To illustrate our results, examples of a possible implementation in the JAsCo AOP language are given. JAsCo supports stateful aspects which allow the definition of stateful pointcut expressions. The use of JAsCo stateful aspects allows a more seamless integration of the identified dynamic business rules in web service compositions.

The business rules presented in this work are examples of rules that guide *how* the composition should adapt to the changing business environment. In order to fully cover the whole domain of service composition business rules this work can be continued by addressing the other kinds of composition rules already identified in these paper.

To enhance the specification and implementation of business rules, our current line of research focuses on creating a high-level business rules language. This language allows specifying rules in a very declarative way, without having to be aware of specific AOP constructs. AOP is used as an underlying layer to realize the connection of the business rules with the core applications in an oblivious way.

7. REFERENCES

[1] Arsanjani, A.. Rule object 2001: A pattern language for adaptive and scalable business rule construction.

[2] Business Process Execution Language for Web Services (WS-BPEL), Specification Version 1.1, www-128.ibm.com/developerworks/library/ws-bpel/

[3] Charfi, A., Mezini, M., Aspect-Oriented Web Service Composition with AO4BPEL, LNCS 3250 , 2004

[4] Charfi, A., Mezini, M., Hybrid Web Service Composition: Business Processes Meet Business Rules, 2nd International Conference on Service Oriented Computing, New York City, USA, November 2004

[5] Cibrán M. A., D'Hondt M., Jonckers V.: Aspect-Oriented Programming for Connecting Business Rules. In Proceedings BIS, Colorado Springs, USA (2003)

[6] Cibrán M. A., D'Hondt M., Suvée D., Vanderperren W., Jonckers V.: JAsCo for Linking Business Rules to Object-Oriented Software. In Proceedings CSITeA, Rio de Janeiro, Brazil (2003)

[7] Cibrán, M. A., Verheecke, B. and Jonckers, V. Modularizing Client-Side Web Service Management Aspects. In Proceedings of the second Nordic Conference on Web Services. Vaxjo, Sweden, November 2003.

[8] Date C.: What not How: The Business Rules Approach to Application Development. Addison-Wesley Publishing Company (2000)

[9] D'Hondt M., Jonckers V.: Hybrid Aspects for Weaving Object-Oriented Functionality and Rule-Based Knowledge. In Proceedings of AOSD, Lancaster, UK (2004)

[10] M. D'hondt: Hybrid Aspects for integrating Rule-based Knowledge and Object-Oriented Functionality, Phd Thesis, Vrije Universiteit Brussel, May 2004.

[11] Kappel, G., Rausch-Schott, S., Retschitzegger, W. and Sakkinen, M., From rules to rule patterns. In Conference on Advanced Information Systems Engineering, pages 99--115, 1996.

[12] Ross R. G.: Principles of the Business Rule Approach. Addison-Wesley (2003)

[13] Suvée, D. and Vanderperren, W. "JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development," in Proc of Second International Conference on Aspect-Oriented Software Development, Boston, USA, March 2003.

[14] The Business Rules Group. Defining Business Rules: What Are They Really?, July 2000. http://www.businessrulesgroup.org/

[15] Vanderperren, W., Suvee, D., Cibrán, M. A. and De Fraine, B. Stateful Aspects in JAsCo. To be published at the Software Composition Workshop (LNCS), ETAPS 2005, Edinburgh, Scotland, April 2005.

[16] Verheecke, B., Cibrán, M. A. and Jonckers, V. AOP for Dynamic Configuration and Management of Web services in Client-Applications. In Proceedings of 2003 International Conference on Web Services. Erfurt, Germany, September 2003.

[17] Von Halle B.: Business Rules Applied. Wiley (2001)