Business Rules Integration in BPEL – A Service-Oriented Approach

Florian Rosenberg and Schahram Dustdar
Vienna University of Technology
Distributed Systems Group, Information Systems Institute
1040 Vienna, Argentinierstrasse 8/184-1, Austria
{*rosenberg, dustdar*}@*infosys.tuwien.ac.at*

Abstract

Business rules change quite often. These changes cannot be handled efficiently by representing business rules embedded in the source code of the business logic. Efficient handling of rules that govern ones business is one factor for success. That is where business rules engines play an important role. The service-oriented computing paradigm is becoming more and more popular. Services offered by different providers, are composed to new services by using Web service composition languages such as BPEL. Such process-based composition languages lack the ability to use business rules managed by different business rules engines in the composition process. In this paper, we propose an approach on how to use and integrate business rules in a service-oriented way into BPEL.

Keywords: Business Rules, BPEL, Service-oriented Approach

1. Introduction

Business process management is one of the core techniques to manage daily business. We are currently moving from object-orientation to service-oriented computing (SOC), considering services as fundamental elements for application development. Services are self describing, platform-agnostic computational elements that support low-cost composition of distributed applications [7].

Over the last years, different Web service composition languages have emerged such as the Business Process Execution Language for Web Services (WS-BPEL or BPEL for short) [2] or BPML [1]. BPEL is currently the preferred standard for Web service composition and implemented by many vendors.

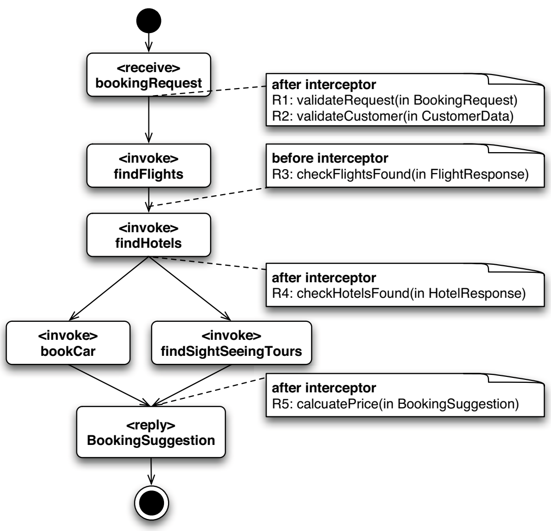
In large enterprise applications (not only legacy systems), it is a common practice that business rules are mixed with the main business logic. Changing and managing such imbed rules is hard and time-consuming and cannot be done by a business analyst, who typically does not have programming experience. Business rule knowledge should, therefore, be managed by a rule-based system, which is then queried by the business logic to evaluate the business rules. Business processes are typically orchestrated by using BPEL, but there is no way to integrate rule-based knowledge into the composition process.

In this paper, we propose an approach on how to integrate rule-based knowledge, accessible through business rules engines (BRE), in a service-oriented way in BPEL or even other Web service composition languages. We present the design of such a system using an *Enterprise Service Bus* (ESB) as middleware, where we plug in all the participating components needed for our approach.

This paper is structured as follows: In the next section, a motivating example is presented to explain different concepts used throughout this paper. Then we introduce business rules, the different classified types together with some examples. In Section 4, we present our business rule integration approach. We present the architecture of our system as well as an integration methodology by considering a simple example. Section 5 summarizes the related work done so far and in Section 6 we conclude this work by summarizing the major points.

2. Motivating Example

The following motivating example of a travel agency is used to explain the basic concepts of BPEL and the way we try to enrich a BPEL description with business rules. A typical use case could be the booking of a trip with flight tickets together with a hotel, a car for the whole stay and of course famous sightseeing trips. Modeling this process is a complex task; it requires many different steps to offer such a full service to customers. We only use a very simple example with annotations representing the business rules for the different activities, as shown in Figure 1. These annotations represent the rules executed on the data at that time. Business rules with a before interceptor are executed before the actual BPEL activities, after interceptors after the BPEL activity, respectively. The concepts are explained in detail in Section 4. We refer to the problems tackled by the use of business rules in Section 3.



<receive> bookingRequest
after interceptor
R1: validateRequest(in BookingRequest)
R2: validateCustomer(in CustomerData)
<invoke> findFlights
before interceptor
R3: checkFlightsFound(in FlightResponse)
<invoke> findHotels
after interceptor
R4: checkHotelsFound(in HotelResponse)
<invoke> bookCar
<invoke> findSightSeeingTours
after interceptor
R5: calcuatePrice(in BookingSuggestion)
<reply> BookingSuggestion

Figure 1. Travel Agency Process

3. Business Rules

According to the Business Rules Group [10], "a business rule is a statement that defines and constraints some business. It is intended to assert business structure or to control or influence the behavior of the business. The business rules which concern the project are atomic, that is, they cannot be broken down further."

Modeling business rules as separate entities offers great flexibility. Especially in the e-commerce domain, this can be a valuable advantage, since the business analyst, who ideally authors the business rules, does not need to have programming knowledge to change the rules. Typically, changing the business rules happens more often than changing the large e-commerce applications. Moreover, extracting the business rules from the business logic leads to a better decoupling of the system, which, as a consequence, increases maintainability. One of the most important facts about business rules is that they are declarative statements, they specify *what* has to be done and not *how*.

In [11], a classification of business rules into four different types is presented, whereas the fourth type (deontic assignments) is only partially identified. We will mainly focus on the first three types:

Integrity Rules (or "integrity constraints") specify assertions that must be satisfied in all stages of a system. Referring to our case study in Figure 1, the rules R1 and R2 represent such integrity rules. Rule R1 and R2 check whether the received request data is correct and consists of all the data needed for further processing. If not, the rule execution throws an exception, which is handled by BPEL.

Derivation rules (also called "deduction rules" or "Horn clauses") are statements of knowledge derived from other knowledge by using an inference or a mathematical calculation. In Figure 1, the price calculation rule R5 is a typical derivation rule. How the price is actually calculated depends on a lot of different facts, e.g., the customer status or the number of articles to buy.

Reaction Rules (also called "action rules" or "ECA") specify the invocation of actions in response to an event. The action is only performed when a certain condition applies. In Figure 1, the rules R3 and R4 can be interpreted as reaction rules in the sense of enabling an action or not. Considering R1, it checks if flights where found otherwise it skips the execution of the hotel search. The same semantic applies to rule R4.

4. Integration Approaches

Integrating rule-based systems in a service-oriented environment is a complex task, due to the fact that both worlds have their own paradigms. Rule-based systems have a high significance, therefore it is reasonable to integrate them into the enterprise architecture. The importance of integrating a BRE with an orchestration engine is also depicted in [5].

We will now focus on the integration aspects between an orchestration engine, in our case BPEL, and different rule-based systems. Therefore, we mainly differentiate between two integration approaches: (1) a tightly coupled approach and (2) a loosely coupled approach. Concerning (1), the idea is that the orchestration engine communicates directly with the BRE through its proprietary API. Due to the fact that the BPEL specification omitted the standardization of an API to access a BPEL engine, most vendors have proprietary or no interfaces to communicate with a BPEL engine, therefore, making it hard to tightly-couple different BREs with a BPEL engine. Another important drawback of this approach is the lack of service-orientation. It is reasonable to expose business rules as services, in order to allow that these services can be reused in every other inter-enterprise (or even inter-organizational) application thus ease the development of new application and the integration of other applications. Based on these drawbacks of the tightly coupled approach, we pursue (2).

4.1. Architecture

Our integration approach considers an *Enterprise Service Bus* (ESB), a middleware well-suited for the service-oriented architecture, as an integration platform. All services use the ESB as a communication platform as depicted in Figure 2. In this section we will explain the concepts of every service and discuss some important design aspects. A detailed discussion of design and implementational aspects is not the intended scope of this paper.

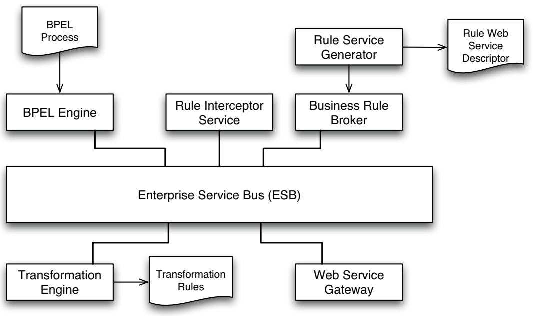


Figure 2. Service-Oriented Approach

The BPEL engine is connected to the ESB with an adapter. It uses the ESB as a messaging layer and communicates directly with the *Web Service Gateway* to call external Web services via `<invoke>` or `<reply>`, or waiting for response by using `<receive>`.

Business Rules Broker: Due to the aforementioned heterogeneity of the different rule APIs, we have introduced a *Business Rules Broker* service in [8], providing a unified access to different BREs, through a Web service interface. The broker architecture abstracts from the specific BRE implementations by providing a plugin-based mechanism to integrate the different BREs. The distinctive feature of the broker approach is the automated generation of Web services for executing business rules based on a *Web Service Rule Interface Descriptor*, describing the services to be generated and the rules to be executed by these services.

Rule Interceptor Service: The *Rule Interceptor Service* is the bridge between the business rules and the executable BPEL process. Our approach, as depicted in Figure 3, is to intercept each incoming and outgoing BPEL Web service call to automatically apply business rules, accessible through the *Business Rules Broker* service. The mapping of BPEL activities to concrete business rules is done by a mapping document, which has to be created by the BPEL designer. We present an example, how such a mapping file

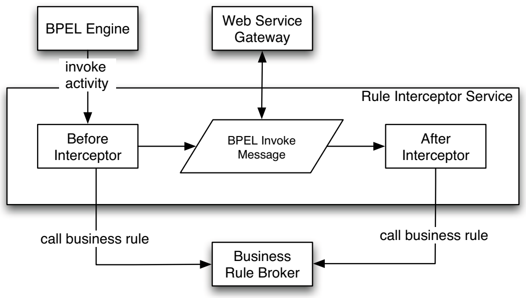


Figure 3. Rule Interceptor Service

is specified, later in Section 4.2. The interceptor concept offers two different interception types, a *before* interceptor, or an *after* interceptor, expressing that the interceptor is either executed before or after the BPEL activity. The control flow of a BPEL invoke activity interception is shown in Figure 3.

Transformation Engine: The generated business rules services have different message types (e.g., BPEL variables) as parameters. It can happen that the message type expected by the generated business rules services does not confer to a given schema but can be transformed to that type. Therefore, a *Transformation Engine* is needed to transform XML messages to other formats understandable by the business rules. The transformation is performed in the *before* or *after* interceptors based on the data types of the BPEL activity. We clarify such a transformation based on a simple example in the next section.

4.2 Travel Agency Example Revisited

The Travel Agency process is created by the BPEL engine when receiving the `BookingRequest` message at the beginning of the process. Typically, this message is generated by some other application or workflow system, so we have to ensure some data constraints on that requests. Such constraints should ideally be expressed as business rules, e.g., a valid name, flight date, a destination, and some constraints on the data are needed. We can ensure these constraints by using two business rules and an after interceptor. Rule R1, `validateRequest(in BookingRequest)` is called by simply invoking this service from the *Business Rules Broker*. What happens if we want to call the rule R2 with the signature `validateCustomer(in CustomerData)` in the after interceptor? The rule R2 only accepts a message of type `CustomerData`, but all the customer data is encapsulated in the `BookingRequest` message. That where the *Transformation Engine* is used. It transforms the necessary data, from the `BookingRequest` to the

`CustomerData` based on defined transformation rules by using XSLT. The activity to rule mapping is shown in Listing 1. It is the input configuration for the *Rule Interceptor Service*.

```
<activity name="findHotels" type="invoke">
  <interceptors>
    <before>
       <!-- no business rules needed here -->
    </before>
    <after>
      <rule name="validateRequest">
        <parameter name="BookingRequest"/>
      </rule>
      <rule name="validateCustomer">
        <parameter name="CustomerData"
          <transform rule="bookReq-to-custData"/>
        </parameter>
      </rule>
    </after>
  </interceptors>
</activity>
```

Listing 1. BPEL Activity to Rule Mapping

5. Related Work

The related work in this area is very diverse, ranging from alternative Web service composition approaches to rule-based approaches and rule representation formats as well as rule integration approaches.

The Java Community Process (JCP) released the final version of the Rule Engine API [4] in August 2004. It is already supported (at least partially) by a couple of rule engines (cf. Drools[1] or JESS[2]). Also many commercial business rule products are available, with ILOG[3] as one well-known representative. Another initiative, focusing on a standard representation of business rules, is RuleML [9], started in August 2000 and is currently the most promising initiative for representing rule markup for the Semantic Web.

To the best of our knowledge, there is no existing approach which focuses on a service-oriented integration of rule-based languages with process-based Web services composition such as BPEL. In [3], the authors present a hybrid approach for realizing the integration of business rules (modeled as aspects) with a BPEL orchestration engine by using aspect-oriented programming techniques. In [6], a business rule driven composition approach is presented. The authors propose a technique how to dynamically compose business processes based on business rules.

6. Conclusions

Integrating business rules in process-oriented Web service composition can improve the quality and ease development by using business rules authored by domain experts. But integration cannot be done if the business rules are not accessible in a unified way. This is becoming increasingly important when considering the emerging paradigm of service-oriented computing.

In this paper we proposed an approach on how to integrate business rules, managed by different rules engines, into process-oriented Web service composition languages. We use BPEL as our composition language and an ESB to integrate the presented components into the system. Furthermore, we depicted the architecture of the system and illustrated the concepts by using a travel agency scenario.

[1] `http://www.drools.org`
[2] `http://herzberg.ca.sandia.gov/jess`
[3] `http://www.ilog.com`

References

[1] A. Arkin. Business Process Modeling Language. http://www.bpmi.org/, November 2002.

[2] BPEL. Business Process Execution Language for Web Services Version 1.1. http://www.ibm.com/developerworks/library/ws-bpel/, May 2003.

[3] A. Charfi and M. Mezini. Hybrid Web Service Composition: Business Processes Meet Business Rules. In *Proceedings of the 2nd International Conference on Service Oriented Computing*, November 2004.

[4] Java Community Process. JSR 94 - Java Rule Engine API. `http://jcp.org/aboutJava/communityprocess/final/jsr094/index.html`, August 2004.

[5] D. A. Manolescu. Orchestration Patterns in Service Oriented Architectures. URL: `http://www.orchestrationpatterns.com/OrchestrationPatterns.html`, January 2005.

[6] B. Orriëns, J. Yang, and M. P. Papazoglou. A Framework for Business Rule Driven Service Composition. In *Proceedings of the Fourth International Workshop on Conceptual Modeling Approaches for e-Business Dealing with Business Volatility*, 2003.

[7] M. P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12, Dezember 2003.

[8] F. Rosenberg and S. Dustdar. Design and Implementation of a Service-oriented Business Rule Broker. In *Proceedings of the 1st IEEE International Workshop on Service-oriented Solutions for Cooperative Organizations (SoS4CO '05)*, 2005.

[9] RuleML Initiative. Website. `http://www.ruleml.org`.

[10] The Business Rules Group. Defining Business Rules – What Are They Really? `http://www.businessrulesgroup.org/first_paper/br01c0.htm`, July 2000.

[11] G. Wagner. How to design a general rule markup language? In *Workshop XML Technologien fuer das Semantic Web (XSW), Berlin*, June 2002.