Technical University of Vienna
Information Systems Institute
Distributed Systems Group

Business Rules Integration in BPEL – A Service-Oriented Approach

Florian Rosenberg and
Schahram Dustdar
rosenberg@infosys.tuwien.ac.at
dustdar@infosys.tuwien.ac.at

TUV-1841-2005-30 February 10, 2005

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

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

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 [22].

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) [5] or BPML [2]. BPEL is currently the preferred standard for performing Web service composition [1] and implemented by many vendors.

When building software architectures following the object-oriented paradigm, a common practice is the *separation of concerns*. Layering software into different tiers, achieved for example through the classical three-tier architecture (presentation layer, business logic and data access layer), greatly implements this aforementioned separation of concerns. 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 [23] 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 used by the business logic to evaluate the business rules.

When applying the service-oriented paradigm to enterprise computing, the functionality is encapsulated as a service, either as a simple or a composite one [22]. Describing a composite service can be done by using BPEL, but there is no way to integrate rule-based knowledge into the composition process. In Section 2 we depict a simple example where business rules are needed during the composition process.

In this paper, we propose an approach on how to integrate rule-based knowledge, accessible through business rule engines (BRE), in a service-oriented style 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 which is used to explain different concepts used throughout this paper. In Section 3 the basic elements of BPEL are summarized. Section 4 introduces business rules, the different classified types together with some examples. In Section 5, 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 6 summarized the related work done so far and in Section 7 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. Our example uses the often presented travel agency process. A typical use case could be the booking of a trip with flight or train 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 activies, after interceptors after the BPEL activity respectively. The concepts are explained in detail in Section 5. We refer to the problems tackled by the use of business rules in Section 4.

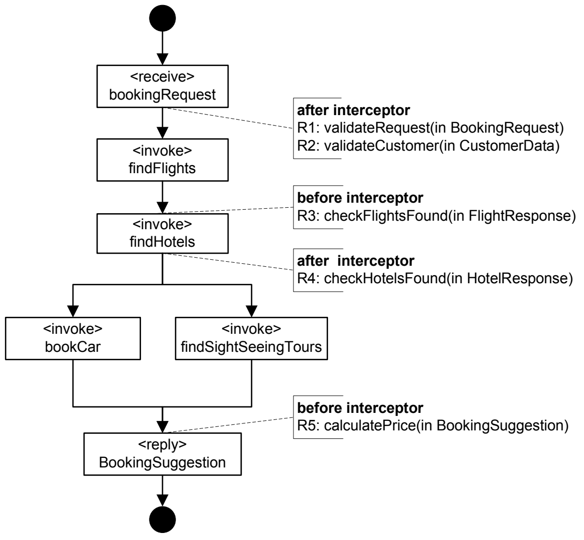
<receive> bookingRequest
after interceptor
R1: validateRequest(in BookingRequest)
R2: validateCustomer(in CustomerData)
<invoke> findFlights
before interceptor
R3: checkFlightsFound(in FlightResponse)
<invoke> findHotels
after interceptor
R4: checkHotelsFound(in HotelResponse)
<invoke> bookCar
<invoke>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

Figure 1. Travel Agency Process

The travel agency in our example is very simple, it omits a lot of functionality required to be a fully-fledged business process. Nevertheless, this simple process raises enough possibilities to show, where services which build upon business rules, can be added. How we try to achieve the integration of rule services with our approach is shown in Section 5.

3 Business Process Execution Language

BPEL is currently a frequently used technology for process-based Web service composition. BPEL is the successor of XLANG from Microsoft [19] and WSFL from IBM [13] by combining both worlds. BPEL provides a model to describe the behavior of a business process based on Web service interactions between the process and its partners [5]. Two different types of processes can be implemented with BPEL, *abstract processes* (also referred to as business protocols) or *executable processes*. An executable process describes the internal implementation of a service and an abstract process specifies the external behavior of a service [1]. The main building blocks of BPELs component model are *activities*. An activity is either a *basic* activity (such as invoke, receive and reply, or a *structured* activity (e.g. sequence, switch, flow, etc.). Furthermore, BPEL supports exception handling by specifying one or more fault handlers, compensation handlers, for semantically undo the activities of a scope and correlation sets, for identifying messages belonging to different process instances. In Listing 1, we depict an example process based on our example from the previous Section 2.

```
<process name="TravelAgency" ...>
   <!-- variable declaration -->
   <flow>
      <links>
         <link name="order-to-flight"/>
         <link name="accomodation-to-sightseeing"/>
      </links>
      <sequence>
         <receive createInstance="yes"
            name="receiveBookingRequest"
            portType="bookingPT"
            operation="sendBookingRequest"
            variable="BookingRequest">
            <source linkName="order-to-flight"/>
         </receive>
         <invoke operation="findFlights"
            inputVariable="BookingRequest"
            outputVariable="FlightData"
            portType="airlinePT"
            partner="airline">
            <target linkName="order-to-flight"/>
         </invoke>

         <!-- invoke other services -->

         <flow>
           <invoke operation="bookCar" .../>
           <invoke operation="findSightSeeingTours .../>
         </flow>
         <reply variable="BookingSuggestion"
                portType="bookingPT"
                operation="sendBookingSuggestion" />
      </sequence>
```

```
  </flow>
</process>
```

Listing 1. BPEL Travel Agency Process

4 Business Rules

According to the Business Rules Group [28], "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 in the development process of applications. 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 much 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.

In [12], different business rules from the e-commerce domain are depicted to be very valuable when describing,

- terms and conditions (e.g. price calculation rules),
- service provisions (e.g. rules for refunds),
- and surrounding business processes (e.g. rules for lead time to place an order).

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* [27].

4.1 Advantages of a Business Rule Approach

The following, not exhaustive, list considers some advantages when separating business rules from the business logic:

- Business rule reuse across other business processes and applications.
- A better understanding of application logic through externalized business rules.
- Documentation of business decisions through rules.
- Lower application maintenance costs.
- Ease of changing rules by using visual tools.

4.2 Rule Types

In [30], 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 Rule (or "integrity constraints") specifies an assertion that must be satisfied in all stages of a system. This type can be further differentiated into *state* or *process* constraints. State constraints, such as *"a customer has to be at least 18 years old"*, must be valid at any time in the system. Process constraints express the dynamic integrity of a system, thus specifying the valid state transitions in the system, such as *"the valid state transitions of a* `BookingRequest` *are reiceived → processed → acknowledged"*

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, these rules throw an exception which is handled by BPEL.

Derivation Rule (also called "deduction rules" or "Horn clauses") is a statement of knowledge derived from other knowledge by using an inference or a mathematical calculation. Consider the following rules: *a customer is a gold customer if she is booking regularly.* Such a rule can only be evaluated with logic-based reasoning. It can be resolved by using the conditions of other rules, e.g. R1: *if a customer made 3 bookings in the last 12 months, she is a regular booker.* R2: *if a regular customer made 3 recommendations she is a VIP customer.* R3: *if a VIP customer spends more than 10000 EUR she is a gold customer.*

In Figure 1, the price calculation rule R5 is a typical derivation rule. How the price is actually calculated is up to the business analyst who creates the rules (indeed it will depend on a lot of different factors, e.g., the customer status as seen in the sample rules above).

Reaction Rules (also called "stimulus response rules", "action rules" or "ECA – event-condition-action rules) specify the invocation of actions in response to an event. The action is only performed when a certain condition applies. Such rules typically consist of an event condition, a state condition (or precondition), an action term, and a state effect (or post-condition). These types of rules are referred to as Event-Condition-Action-Effect (ECAE) rules. Special cases are the commonly known ECA rule and the condition-action (or production) rule. An example could be, *"if a customer confirms the booking of a flight, the hotel booking service starts searching for appropriate rooms for this trip."*

In our example from Section 2, the rules R3 and R4 can be seen 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.

Deontic Assignments of powers, rights and duties to (types of) internal agents define the deontic structure of an organization, guiding and constraining the actions of internal agents. This type mainly considers authorizations as business rules (e.g. Only the manager of the travel agency is allowed to grant discounts higher than 5%.)

4.3 Rule-Based Systems

In [9], a good introduction to rule-based systems is presented. We will briefly summarize the most important facts about rule-based systems. For a detailed discussion on rule-based systems and knowledge representation we can refer to [25].
Typically, knowledge is represented by rules. Therefore, a rule-based system is used to control the selection and activation of rules by using an inference or rule engine. The inference engine is responsible for activating the rules when the incoming data matches activation pattern of the condition or the conclusion. Furthermore the engine is responsible for chaining the rules. Let us consider the following example of two rules: *a customer gets 3% discount if she is a frequent one and has a good credit rating*. The rule engine chains these rules together, either in forward or backward chaining mode (cf. [9]).

4.4 Rules Engines

Basically two different formalisms of expressing rules exist, production rules, used in *production systems*, and first-order predicate logic used in *logic-based systems* [9]. The term business rules engine (BRE) unifies both terms under a common name.

Production systems consist of three parts, the production rules, the working memory and the rule engine. Production rules are a special form of reaction rules, consisting of a condition and a conclusion. The working memory holds the data and rules. The data is used by the rule engine to match the conditions. Furthermore the rule engine uses different conflict resolution strategies if more than one rule matches the condition. A famous algorithm for matching the rule conditions is called RETE [11].

Logic-based systems typically use logic programming for problem solving by using inference. A well-known logic programming language is Prolog. Logic programming is out of the scope of this paper so we do not cover details here.

5 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, as discussed in previous sections. Rule-based systems have a high significance, not only for representing business rules, therefore it is reasonable to integrate them into the enterprise architecture. The importance of integrating a rule engine with an orchestration engine is also depicted in [18], where the author presents a couple of orchestration patterns, also addressing the problem of integrating rule languages with orchestration engines.

We will now focus on the integration aspects between an orchestration engine, in our case BPEL, and different rule-based systems. As presented in Section 6, a vast number of business rule engines exist, ranging from logic-based systems to production 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 rule engine through their proprietary API. Due to the fact the BPEL specification omitted the standardization of an API to access a BPEL engine, most vendors have proprietary or no interfaces communicate with a BPEL engine. These problems make it hard to tightly-couple different rule engines with a BPEL engine. Another important drawback of this approach is the lack of service-orientation. It is reasonable to offer the 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). The concepts and the design issues concerning our architectural approach are depicted in the next section.

5.1 Architecture

Our integration approach considers an *enterprise service bus* (ESB), a service 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 important design aspects. A detailed discussion of design and implementational aspects is not the intended scope of this paper.

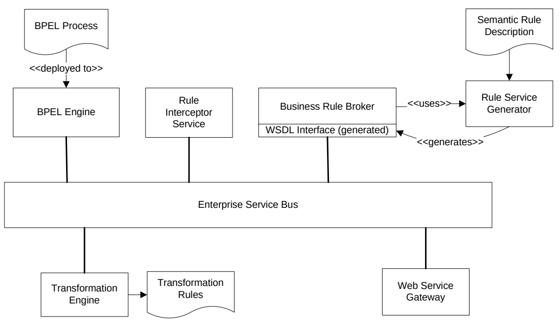

Figure 2. Service-Oriented Approach

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>`.

5.1.1 Business Rule Broker

Due to the aforementioned heterogeneity of the different rule APIs, we introduce a *Business Rule Broker* service, providing a unified access to different BREs, through a Web service interface. The architecture of the broker service is depicted in Figure 3.

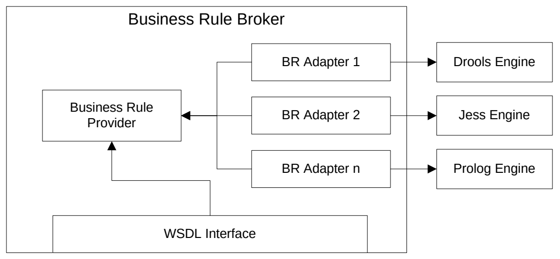

Figure 3. Business Rule Broker

The broker architecture abstracts from the specific rule engine implementations. It therefore uses a flexible architecture with following distinctive features:

- Allows to connect different rule-based engines, irrespective if they are logic-based systems or production based systems by using an *adapter*, specifically for each engine. Each connected rule engine manages its own knowledge base with its specific format.
- It offers a WSDL interface for accessing different rules or a collection of rules (referred to as rule set) in a service-oriented manner.
- The attached engines managed by the broker can be dynamically changed at runtime. Furthermore more than one rule-engine can be used at the same time using the unified WSDL interface.

At the moment, we consider only Java-based rule engines, because we use the Java Rule API [17], a recently finished Java Community Process, which offers access to different rules engines. Every other rule engine, can be plugged in by implementing an adapter.

Another distinctive feature of our approach is the generation of the WSDL interface of the Business Rules Broker based on a semantic description of the different rules. Please consider that this generation is not a dynamic runtime feature, it is done at design time. Performing such a generation needs a semantic description of a knowledge base. A knowledge base is typically organized as a collection of rules grouped into different rule sets. A semantic description of a rule or a rule set can be seen as a triple $R = (S, I, O)$ where R is the name of the rule or the rule set, I is a set of input parameter $I = < name, type >$ and O is a set of output $O = < name, type >$. Both input and output parameters are specified with a name and a valid data type (in XML Schema). This semantic description is specified in XML. Providing such a semantic description of the whole rule base can be a very time consuming task and maybe the biggest disadvantage of our approach. But it is necessary because there is currently no standard language (such as RuleML, see Section 6) in describing rules which is supported by most rule engine vendors.

5.1.2 Rule Interceptor Service

The *Rule Interceptor Service* is the bridge between the business rules and the executable BPEL process. Our approach is to intercept each incoming and outgoing BPEL Web service call to automatically apply business rules, accessible by the *Business Rule 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 is specified, later in Section 5.1.3 when explaining the *Transformation Engine*.

The interceptor concept offers two different interception times, a *before* interceptor, or an *after* interceptor, expressing that the interceptor is either executed before or after the BPEL activity. The control flow of such an execution is shown in Figure 4.

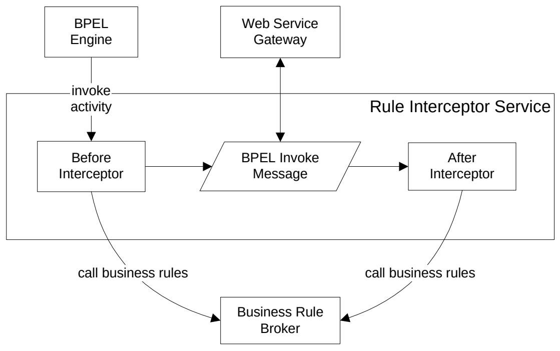

Figure 4. Rule Interceptor Concept

The BPEL engine execution processes each activity (in this example the `invoke` activity), so it calls the Web services identified by a specific partner link type in BPEL (Please note, that the binding of the partner links types is not specified directly in BPEL, is specified at deployment). Each call is intercepted by the *Rule Interceptor Service*, where the before interceptor is called. The mapping of activities to rules is done by an activity to rule mapping. If some rules are mapped to be executed in the before interceptor, then the *Business Rule Broker* is called to execute the appropriate rules. After the before interceptor the BPEL activity is executed or in case of a business rule violation an exception is thrown an propagated to the BPEL engine. After successfully executing the BPEL activity, the after interceptor is executed following the same concept as the before interceptor. After all interceptors are called, the control flow returns to the BPEL engine to further process its activities.

5.1.3 Transformation Engine

In order to make the aforementioned *Rule Interceptor Service* and the business rules it calls handle all the different message types (e.g. BPEL variables), we need a *Transformation Engine* to transform XML messages to other formats which can be understood 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.

5.1.4 Travel Agency Example Revisited

The Travel Agency process, depicted in Figure 1, is created by the BPEL engine when receiving the `BookingRequest` message at the beginning of the process as depicted in Listing 1. 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 [29]. The activity to rule mapping is shown in Listing 2. 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="bookingReq-to-customerData" />
        </parameter>
      </rule>
    </after>
  </interceptors>
</activity>
```

Listing 2. BPEL Activity to Rule Mapping

6 Related Work

In this section we discuss mainly three different types of related work. First of all, we describe alternatives for Web service composition such as BPEL. A good overview of different Web service composition approaches is presented in [26].

In the second part we present rule-based approaches and formats which can be used to implement business rules. In the last part, we discuss existing integration approaches done so far.

6.1 Web Service Composition

BPEL is currently the standard technology for specifying Web service composition. BPEL-J [3], a joint work by BEA and IBM, provides an interesting alternative to the common BPEL. Its distinctive feature is the integration of the Java programming language into BPEL. It adds an activity called *Java Snippet* for specifying conditions, join conditions, variables with Java data types and more features available because of the Java integration.

In [21], a business rule driven composition approach is presented. The authors propose a technique how to dynamically compose business processes based on business rules.

6.2 Business Rule-Based Approaches

The Java Community Process started to work on an API for Java-based rule engine in November 2000 (JSR 94). In August 2004, the final version of the rule engine API was released [17]. This Java Rule API is already supported (at least partially) by a couple of rule engine (cf. Drools [10] or JESS [16]).

The OMGs (Object Management Group) [20] Business Rules Team (BRT) recently announced to release a proposal dealing with the *Semantics of Business Vocabulary and Rules* (SBVR). It aims to define common vocabularies to express business rules in SBVR Structured English.

Also many commercial business rule products are available, with ILOG [15] as one well-known representative. ILOG offers several rule engine technologies for different platforms, e.g., JRules for the Java platform.

Another initiative, focusing on a standard representation of business rules, is RuleML [24]. It started in August 2000 and is currently the most promising initiative for representing rule markup for the *Semantic Web*. The RuleML approach encompassed all the different rule types described in Section 4.2. It main approach is to become the standard rule markup with translators in and out along with further tools [4].

The *Business Rules for Electronic Commerce* project carried by IBM Research, developed a framework for presenting business rules [14]. One of the results of this project was a Java library called *CommonRules* using declarative logic as knowledge representation language.

6.3 Integration Approaches

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 [6, 7], 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. Their approach, called AO4BPEL, is an extension to BPEL by aspect-oriented concepts which allow to model business rules as aspects and weaving them into the BPEL code by using an aspect-aware orchestration engine. The authors also note that they do not know any approaches which integrate rule-based languages with process-based Web service composition approaches.

An interesting approach on how to integrate rule-based knowledge into object-object oriented languages is presented in [9]. Moreover, the author evaluated several interesting rule-based systems and presented an approach for integrating rule-based languages with object-oriented ones by using aspect-oriented programming.

7 Conclusions

Integrating business rules in process-oriented Web service composition can greatly 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 getting 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. Our approach consists of multiple parts: A *Business Rules Broker* allowing to hide the heterogeneity of different rule engines and providing a service-oriented interface (based on WSDL) to access and evaluate the different rules from the knowledge base. The *Rule Service Interceptor* uses a special rule to activity mapping to intercept outgoing BPEL calls and automatically applied the business rules specified by the designer. The *Transformation Engine* is used by the interceptors to transform XML to other schemas acceptable by the specific rules to call. Furthermore, we presented a way to dynamically generate the WSDL interface of the *Business Rules Broker* by using a semantic rule description.

Our research work concerning business rules is in very early stage. Currently we are building a prototype system and some examples scenarios. We plan to extend the system by adding more advanced services, such as a contracting authority as proposed in [8].

References

[1] G. Alsonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications.* Springer Verlag, 2004.

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

[3] BEA Systems Inc. and IBM Corp. BPELJ: BPEL for Java. `ftp://www6.software.ibm.com/software/developer/library/ws-bpelj.pdf`, March 2004.

[4] H. Boley, S. Tabet, and G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proceedings of the 1st Semantic Web Working Symposium*, July/August 2001.

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

[6] A. Charfi and M. Mezini. Aspect-Oriented Web service Composition with AO4BPEL. In L. J. Zhang, editor, *Proc. ECOWS 2004*, volume 3250 of *LNCS*. Springer, 2004.

[7] 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.

[8] I. Chebbi, S. Tata, and S. Dustdar. The view-based approach to dynamic inter-organizational workflow cooperation. Technical Report TUV-1841-2004-23, Vienna University of Technology, 2004.

[9] M. D'Hondt. *Hybrid Aspects for Integrating Rule-Based Knowledge and Object-Oriented Functionality.* PhD thesis, Vrije Universiteit Brussel, May 2004.

[10] Drools – Java Rule Engine. `http://www.drools.org`.

[11] C. Forgy. RETE: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[12] B. N. Grosof, Y. Labrou, and H. Y. Chan. A declarative approach to business rules in contracts: courteous logic programs in XML. In *ACM Conference on Electronic Commerce*, pages 68–77, 1999.

[13] IBM Software Group. WSFL – Web Service Flow Language Specification. `http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, May 2001.

[14] IBM T.J. Watson Research. Business Rules for Electronic Commerce Project. `http://www.research.ibm.com/rules/home.html`, 1999.

[15] ILOG Website. `http://www.ilog.com`.

[16] JESS - Java Rule Engine. `http://herzberg.ca.sandia.gov/jess`.

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

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

[19] Microsoft Corporation. XLANG Specification. `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`, 2001.

[20] OMG - Object Management Group. `http://www.omg.com`.

[21] 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.

[22] 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.

[23] I. Rouvellou, L. Degenaro, K. Rasmus, D. Ehnebuske, and B. McKee. Extending business objects with business rules. In *Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages*, pages 238–249, 2000.

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

[25] S. Russell and P. Norvig. *Artificial Intelligence – A Modern Approach.* Prentice Hall, second edition, 2003.

[26] W. Schreiner and S. Dustdar. A Survey on Web services Composition. *International Journal of Web and Grid Services*, 1, 2005.

[27] K. Taveter and G. Wagner. Agent-Oriented Enterprise Modeling Based on Business Rules? In *Proceedings of 20th Int. Conf. on Conceptual Modeling (ER2001)*, LNCS, Yokohama, Japan, November 2001. Springer-Verlag.

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

[29] W3C. XSL Transformations (XSLT) - Version 1.0. `http://www.w3.org/TR/xslt`, November 1999.

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