Design and Implementation of a Service-Oriented Business Rules Broker

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 define or constrain some business [24]. Many different business rules engines exist to capture and manage the rules that comprise ones business. The lack of standards in the business rule area leads to different engine implementations and various rule representation formats. Furthermore, accessing rule-based knowledge in a uniform way from different heterogeneous applications in an enterprise computing environment poses many integration challenges. In this paper, we propose the design and implementation of a service-oriented approach on how to integrate different heterogeneous business rule engines by using our Business Rules Broker architecture. Furthermore, we present a method to automatically expose the rule-based business knowledge as Web services by generating the Web service code at design-time.

Keywords: Business Rules, Service-oriented Architecture, Web Services

1. Introduction

Managing and automating business processes are key principles for successful enterprise computing systems. Service-oriented computing is a core technology for realizing automated business processes based on Web services. A vast number of different technologies for modeling and representing business processes exist, starting from workflow management systems to Web service composition (cf. [22]) and orchestration systems (e.g. BPEL [4] or BPML [2]). The service-oriented architecture (SOA) can help to implement business processes and business applications faster by separating infrastructure logic from application logic. Challenges arise, when business rules are directly imbed into the application logic [19], which is a quite common practice. This fact has a highly negative impact on maintainability of the business logic, because business rules tend to change quite frequently. Different research efforts exist which try to extract business rules from existing applications [28].

The usage of a business rules engine using different knowledge bases can greatly improve the maintainability in such cases. Such an engine handles the execution and management of business rules. The administration of rules should ideally be performed by domain experts or business analysts (depending on the rule language used for authoring). The necessity of business rules, especially in the electronic commerce domain, is stated in [1] by the following two factors: Firstly, an increased competition requires frequent changes in cooperate policies and strategies to remain competitive. This means that changes have to be made fast to react to market changes (e.g., changing insurance policies). Secondly, rules can be applied when making decisions without the involvement of humans in business-to-consumer activities. It is unrealistic that knowledgeable staff will always react in real-time to make decisions. Both factors underline the need for business rules as external entities, thus managed by a business rules management system.

Besides this, there are several other areas where business rule-based systems are a valuable technology, e.g., in Web service composition (cf. our BPEL integration approach proposed in [18] or the business rule driven Web service composition approach proposed in [17]).

In this paper, we contribute an approach on how to integrate several heterogeneous production- and rule-based systems, accessible through a so-called Web service based *Business Rules Broker*. All the rule knowledge, stored in different knowledge bases and managed by several rules engines, is accessible by customized Web services. The distinctive feature of our approach is the automated generation of the aforementioned customized Web services for the business rules based on a specific *Web Service Rule Interface Description*, specifying the mapping of the Web services to different rule sets at design-time level.

This paper is structured as follows: In the next section, business rules and rule-based knowledge are discussed

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)

0-7695-2384-6/05 $20.00 © 2005 **IEEE**

IEEE COMPUTER SOCIETY

together with some examples. In Section 3, the related work done so far is summarized ranging from business rule markup languages to existing business rule approaches. In Section 4, we introduce the design and the rationale behind our *Business Rules Broker* system and sketch some implementational aspects. Section 5 concludes this work by summarizing the major points and presenting some future work.

2. Business Rules Technologies

The Business Rules Group [24] defines a business rule as “a statement that defines and constraints some aspects of 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.”

A so-called business rule engine, some kind of rule-based system, typically manages and executes business rules. In [25], a business rules system is defined as follows: “An automated system in which the rules are separated, logically and perhaps physically, from other aspects of the system and shared across data stores, user interfaces, and perhaps applications.”

Separating rules from the core business logic possesses a lot of advantages. The following, not exhaustive, list depicts some of them:

- Business rules can be reused across other business processes and applications.
- A better decoupling of business rules from application and process logic.
- Documentation of business decisions through rules.
- Fast and easy maintenance of business rules.
- Manageability of business rules also by non-technical staff.
- Lower application maintenance costs.

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* [23] but the semantics of the rules is domain specific.

2.1. Rule Types

Currently, there is no existing standard classification scheme for the different types of business rules. The Object Management Group (OMG) is working on *Business Rules Semantics* and nears completion [8]. Nevertheless, several classifications of different rules types have emerged (cf. [24, 25, 27]).

In [27], business rules are classified into four different types:

Integrity Rules. This type of rules (also referred to as “integrity constraints”) specifies an assertion that has to be valid in all stages of a system. A distinction is drawn between *state* constraints and *process* constraints. An example of a state constraint could be, *“an authorized user needs a username and a password”*. A process constraint specifies the valid state transitions in a system (dynamic integrity), e.g., *“the valid state transitions of a* `PurchaseOrder` *are* $received \rightarrow processed \rightarrow acknowledged$*”*.

Derivation Rules. Such rules (also called “deduction rules” or “Horn clauses”) are statements of knowledge derived from other knowledge by using an inference or a mathematical calculation. Inference rules can only be evaluated with logic-based reasoning (see [21] for details). Consider the following example rule: *a customer is a gold customer if she is booking regularly*. This rule can be evaluated by using the following knowledge, 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.*

Reaction Rules. Also known as “stimulus response rules”, “action rules” or “ECA – event-condition-action rules. Such rules specify an action, invoked in response to an event only if a certain condition is true. There are various forms of reaction rules: The general form is the Event-Condition-Action-Effect (ECAE) rule, having an effect (or a post-condition) after the action part. Other rules are the commonly known ECA rule and the condition-action (or production) rule. Business rules are often represented as production rules. A production rule example could be, e.g., *“if a customer confirms the booking of a flight, the hotel booking service starts searching for appropriate rooms for this trip.”* In [25], reaction rules are often referred to as *action enablers* meaning that they enable a certain action if a condition evaluated to true.

Deontic Assignments. This rule type is only partially identified. Such rules assign powers, rights and duties to (types of) internal agents define the deontic structure of an organization, guiding and constraining the actions of internal agents. It mainly considers authorizations as business rules, e.g., *only the bank manager is allowed to lower the rate of interest.*

2.2. Business Rule Engines

Business rules often implement the aforementioned production rules (condition-action rules) in the form: `IF` c_1 `AND` c_2 ... `AND` c_n `THEN` *action* where $\{c_i | 1 \leq i \leq n\}$ are certain conditions.

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

A business rules engine typically consists of the following components:

Rule or Knowledge Base: The rules encapsulate the domain knowledge, in form of condition-action rules. Most business rules engines available do not support all different rule types depicted in Section 2.1. Production systems implement the aforementioned condition-action rules; therefore, it is necessary to model a constraint rule as a special reaction rule, where the action is used to signal "inconsistency". The same applies for derivation rules, they also need to be represented as special reaction rules [3].

Working Memory: It represents a storage containing objects (also referred to as fact base); these objects represent facts. Actions, which are executed when rules are fired, can cause the working memory to change its state. A pattern matcher is used as part of the inference engine to match facts in the fact base to rules in the rule base.

Inference Engine: The inference process starts by selecting the rules from the rule base that match the content of the working memory. Each inference engine maintains an agenda, where the activated rules are stored for firing. Typically, when knowledge is manipulated there is more than one rule on the agenda. Therefore, a conflict resolution strategy is needed which orders the activations to be fired. Its main task is to prioritize the activations on the agenda. Then, the actions associated with the matched rules are fired (executed). A well-known algorithm for matching the rule conditions is RETE [6].

3. Related Work

In this section we mainly discuss two different types of related work. First, we describe standardization efforts, rule markup languages, and formats which can be used to represent and implement business rules. In the second part, we present business rule approaches, tools and APIs used for implementing different business rule applications.

To the best of our knowledge, there is currently no existing approach which enables the use of several heterogeneous business rule systems through a service-oriented business rules broker. Furthermore, we could not find any approach applying automatic transformation of rule knowledge to meaningful Web services, which is a distinctive feature of our approach.

3.1. Rule Markup Languages

An effort by IBM was the development of the Business Rule Markup Language (BRML) within the scope of the *Business Rules for Electronic Commerce Project* [9]. The primary goal of BRML is to be an XML Interlingua for translation from and to the ANSI-draft Knowledge Interchange Format (KIF). BRML is an XML encoding which represents a broad subset of KIF.

The RuleML Initiative is currently working on the Rule Markup Language (RuleML) [20]. In their statement on the Web page they state: "The goal of the Rule Markup Initiative is to develop RuleML as the canonical Web language for rules using XML markup, formal semantics, and efficient implementations." RuleML is able to represent all different rule types, presented in Section 2.1, except of deontic assignments, by using a hierarchy of rules. Its main approach is to become the standard rule markup with translators in and out along with further tools [3].

The Simple Rule Markup Language (SRML) [11] is a generic rule language consisting of a subset of language constructs common to the popular forward-chaining rule engines. It does not use vendor-specific proprietary languages; therefore, rules can easily be translated to any other rule engine language. This makes it useful as an Interlingua for rule exchange between Java rule engines.

Another rule markup approach is the Semantic Web Rule Language (SWRL), a member submission to the W3C. It is a combination of OWL DL an OWL Lite sublanguages of the OWL Web Ontology language [26]. SWRL includes an abstract syntax for Horn-like rules in both of its sublanguages.

3.2. Business Rules Approaches

Most recently, the Java Community Process finished the final version of their Java Rule Engine API. The JSR-094 (Java Specification Request) started in November 2000 to define a runtime API for different rule engines for the Java platform. The API prescribes a set of fundamental rule engine operations based on the assumption that clients need to be able to execute a basic multiple-step rule engine cycle (parsing the rules, adding objects to an engine, firing rules and getting the results) [12]. It does not describe the content representation of the rules. The Java Rule API is already supported (at least partially) by a couple of rule engine vendors (cf. Drools [5], ILOG [10] or JESS [13]) to support interoperability.

The Object Management Group (OMG) [15] issued the Business Semantics for Business Rules (BSBR) Request for Proposal (RFP) [16] in December 2004, with the goal to define an approach for modeling business rules. The proposal solicits for a meta-model for the specification of business rules (MOF specification), a meta-model for the representation of vocabularies and definition of terms and an XML representation for business rules based on XMI.

The *Business Rules for Electronic Commerce* project

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

carried out by IBM Research, developed a framework for representing business rules [9]. One of the results of this project was a Java library called *CommonRules* using declarative logic as knowledge representation language.

Also many commercial business rule products are available, with ILOG [10] as one well-known representative. ILOG offers several rule engine technologies for different platforms, e.g., JRules for the Java platform. Furthermore, a number of open-source business rule engines are available. A collection of links can be found at [14].

4. Business Rules Broker Approach

4.1. Motivation

The lack of standardization in the area of business rules systems arouses interest in integrating several heterogeneous business rules engines through a unified interface. Furthermore, the shift from the object-oriented computing paradigm to the service-oriented paradigm raises the issue of accessing rule-based knowledge in a service-oriented way. Business rules play one key role in enterprise computing. Our approach allows to access rules in a unified way as Web services, making integration in business applications quite straightforward.

The distinctive features of our approach can be subsumed as follows:

- Connecting heterogeneous business rules engines by using a plug-in based architecture.
- Unifying access to the different rule bases (managed by different rule engines) by using a homogenous interface.
- Automatic generation of meaningful Web services for accessing and executing business rules in a service-oriented way. For example, a set of rules for discount calculation is offered as a service called `calculateDiscount(in ShoppingCart)`. The transformation is based on a *Web Service Rule Interface Description* (described in Section 4.4.1).

4.2. System Architecture

The general architecture of our system is depicted in Figure 1. The different components of the architecture are now briefly described: A *business rules engine*, including its rule bases (we use the term rule base or knowledge base interchangeably), forms the foundation technology of our approach. The architecture supports various rules engines to be plugged-in to the broker architecture.

The *Business Rules Broker Interface* implements the broker for plugging in the different engines by using the adapter

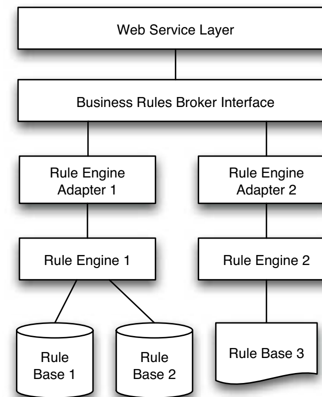

Figure 1. System Architecture

pattern [7]. Each rule base is either persisted on the file system or stored in a database system. The supported persistence mechanism of the rule knowledge is depending on the specific rule engine plugged-in to our system. Each rule engine is able to manage multiple knowledge bases to store different rule sets. Every rule base is assigned a unique identifier (in form of an URI), which is independent from a specific rule engine. This gives us great flexibility when invoking rules managed by a specific engine, since we only need the URI of the rule set to identify it without knowing the concrete engine our broker is using to execute the rules.

On top, the *Web Service Layer* implements access to the rule-based knowledge in form of Web services. This layer is generated automatically by using a so-called *Web Service Rule Interface Description*. We go into details of the proposed architecture in the next sections.

4.3. Business Rules Broker Design

The class diagram of main parts of the broker architecture is shown in Figure 2. The `IRuleEngine` interface is the basic type which has to be implemented by every business rule engine adapter. The method `executeRules(String uri, Map parameters)` is used to execute the rules using a given knowledge base, identified by the `uri` parameter. The second parameter is a map of key/value pairs identifying the parameters needed by the rule engine to execute the rules.

The system is implemented in Java and makes use of the Java Rule Engine API (JSR-094) [12]. Therefore, all JSR-094-based engines can be connected by using the `JavaRuleEngineAdapterImpl`. The functionality of the Java Rule API, used by the adapter, is encapsulated by the `JavaRuleEngineFacade` in a facade pattern [7]. All other engines written in Java has to be

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

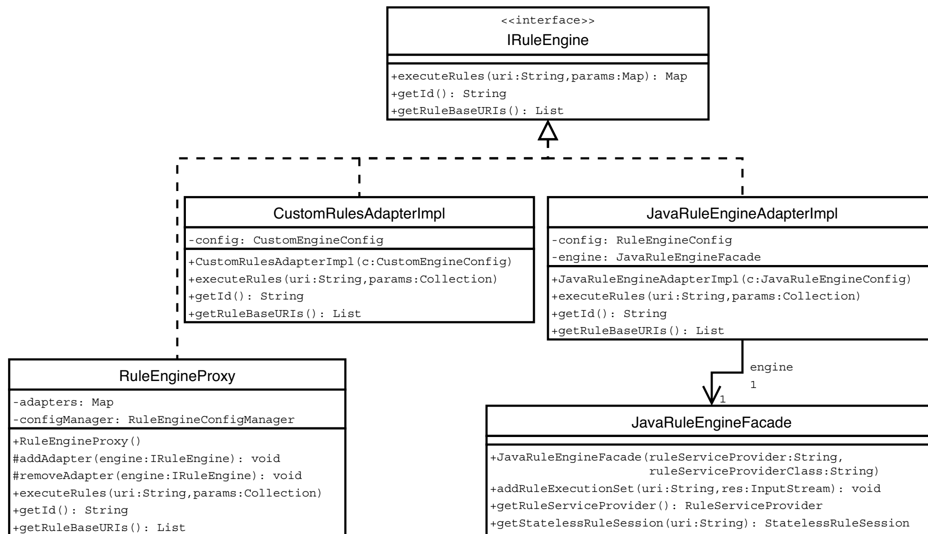

Figure 2. Business Rules Broker Class Diagram

plugged-in by implementing an custom adapter (such as the `CustomRulesAdapterImpl` class in Figure 2).

It is also possible to connect business rules engines written in any other programming language or platform. This will require the implementation of a Web service proxy which is able to call any given engine. This yields the prerequisite that the business rule engine to be integrated needs to have a WSDL interface to make integration possible.

Our *Business Rules Broker* implements a plug-in mechanism for connecting the different business rules engines dynamically at runtime. Each engine needs to provide four parts to be a "valid plug-in":

1. An implementation of the `IRuleEngine` interface, allowing the broker to execute the different rules at a specific business rule engine connected through this interface.

2. An implementation of the `IRuleEngineConfig` interface, for configuration of the rules engine. A default configuration implementation is provided but some business rule engines may require additional or special configuration which can be handled by implementing this interface.

3. A factory class for creating a specific `IRuleEngineConfig` instance. This is only needed when creating a custom rule engine configuration, otherwise the default can be used.

4. An XML configuration file for the adapter, specifying the classes above, the rule bases and necessary configuration parameters specific for the engine.

The `RuleEngineProxy` is the main class for communicating with the broker. It uses a `PluginManager` for loading all the installed adapters and holds a reference to every installed adapter. If the proxy is queried with a given URI, the proxy performs a lookup to find the rule engine adapter which is mapped to the given URI and forwards it to the appropriate rule engine which actually executes the query and returns the result of the business rule engine execution to the client. The sequence diagram of a rule execution, initiated by a rule service consumer calling a JSR-094 based engine, is depicted in Figure 3.

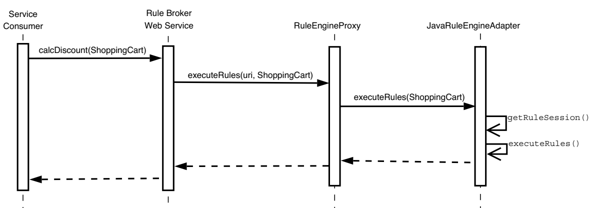

Figure 3. Rule Execution Sequence Diagram

The execution starts when the service consumer queries our Rules Broker Web service. The Web service offers different business rules services methods. When the users calls a method, e.g., `calcDiscount(in ShoppingCart)`, the service implementation forwards this query to the `RuleEngineProxy`, calling it with the `uri` of the knowledge base where this rule is declared. The Web service code is generated dynamically (see next section), therefore each Web service method implementation knows which URI to use. The `RuleEngineProxy` locates the

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

correct adapter implementation and forwards the query. The concrete rule execution is adapter specific and can be performed via the JSR-094 interface if supported by the rule engine or a custom adapter is provided by the plugin.

4.4. Web Service Generation

The *Business Rules Broker* is able to manage several different Java-based business rules engines. Typically, each engine has several knowledge bases with different rule set for each application. Due to the dynamic nature of business rules, we need a flexible way of generating Web services from declarative rule descriptions. In fact, there is currently no standardized rule markup language (cf. Section 3) supported by every rule engine vendor. Moreover, nearly every vendor has a custom format for representing the rules. For example, the Drools project [5], which is one engine we use in our prototype implementation, offers different semantics for encoding rules, based on the programming language used for expressing conditions and actions (e.g., Groovy, Python, Java), referred to as semantic modules. In contrast, Jess [13] uses a LISP-like syntax enriched with functions and statements to interact with Java. In Listing 1, a simple example of checking the credit limit of a customer in Drools syntax is depicted. These rules perform the following: If the credit limit is greater than the amount of the invoice and the status is "unpaid", the credit limit is decremented by this amount and the status is set to "paid".

```
<rule-set name="checkCreditLimit" ... >
  <rule name="rule1">
    <parameter identifier="customer">
      <java:class>
        org.example.Customer
      </java:class>
    </parameter>
    <parameter identifier="invoice">
      <java:class>
        org.example.Invoice
      </java:class>
    </parameter>
    <java:condition>
      customer.getCreditLimit() >= invoice.getAmount()
    </java:condition>
    <java:condition>
      invoice.getStatus().equals("unpaid")
    </java:condition>
    <java:consequence>
      customer.setCreditLimit(
        customer.getCreditLimit() -
        invoice.getAmount());
      invoice.setStatus("paid");
    </java:consequence>
  </rule>
</rule-set>
```

Listing 1. Drools Rules

Based on the heterogeneity of the different representations, a uniform description consisting of the all the data needed to generated Web services out of business rules, has to be specified. We have developed such a uniform description, called the *Web Service Rule Interface Description*, which specifies for the *Web Service Generation* component the services that need to be generated including their names, their input and return parameters in a rule engine independent representation.

4.4.1 Web Service Rule Interface Description (WSRID)

Creating such a description from the different business rules representations (as depicted in Listing 1) is done by extracting the necessary parts of each rule description and transforming them to the appropriate rule interface description. For creating the Web services to access the broker infrastructure in a platform and language independent way, the following information is needed:

- A unique identifier for the rule set (we refer to it as URI).
- The name of the Web service and its operation names.
- The input parameters of each Web service operation (a $< name, datatype >$ tuple).
- The output parameters of the Web service (The parameters are the same as the input parameters).

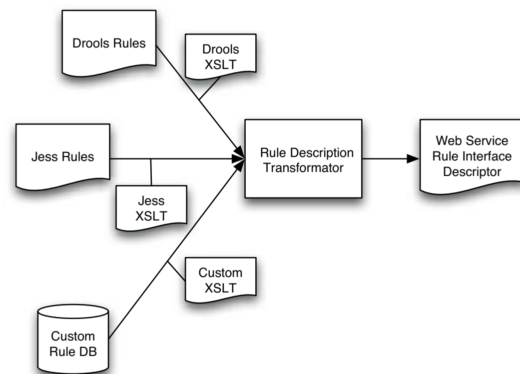

Figure 4. Web Service Rule Interface Transformation

The process of transforming the different rule formats to the rule interface description is depicted in Figure 4. This process is performed at design-time and it requires a custom stylesheet for each rule engine format, used by the *Web Service Rule Interface Transformator* to transform the rules into WSRID format. The transformator processes all the knowledge bases with their stylesheets and merges them to

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

create the rule description. When considering the example rules from Listing 1, the transformation from Drools to WSRID can be performed by using the name of the rules file as a service name (e.g., CreditCardService) and the rule set name as a service operation. An URI can be constructed with the name of the engine as prefix, followed by the service name (e.g., urn:jess-creditcardservice). Furthermore, each parameter tag is transformed to an input parameter of the new Web service. An example of such an WSRID description is depicted in Listing 2. In contrast to the simple example depicted in Listing 1, a rule set typically consists of multiple rules (cf. a `<rule>` tag in Drools syntax) whereas not each rule set has to constitute a rule Web service although it may be used for execution of other rules by the business rule engine. For example, not each derivation rule (cf. Section 2) constitutes a new rule Web service, whereas the rule engine may need all of these rules to evaluate another rule exposed and callable as rule Web service.

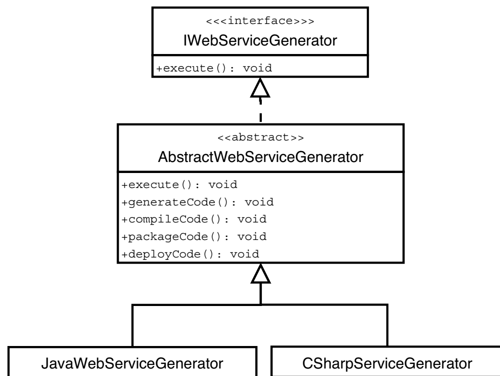

Figure 5. Web Service Generator

The WSRID as specified above, is the input for the *Web Service Generator*. In this process the Web service layer, on top of the *Business Rules Broker Interface* is generated by using a code generator component build upon a template engine. The Web service code is generated by using an XSLT stylesheet to transform the WSRID to the appropriate code generator input.

The different processing steps from the heterogeneous rule bases to the deployed and useable Web services can be summarized as follows:

1. Generate the WSRID with the *Web Service Rule Interface Transformator* by specifying which rule sets should be exposed as Web services.

2. Generate the Web service source code by using the *Web Service Generator*.

3. Compile the dynamically generated Web service source code.

4. Package and deploy the generated Web services to an application server.

Currently, the limitation of our design-time approach is the re-run of the four step cycle as described above, every time new rules are added to one of the knowledge bases. What is needed here is a central rule repository (or a rule repository coordinator when using heterogeneous rules engines with different knowledge bases as we do) and rule administration API, that notifies the business rules broker when new rules are inserted into the knowledge base in order to redeploy the new Web services automatically.

4.5. Architectural Levels

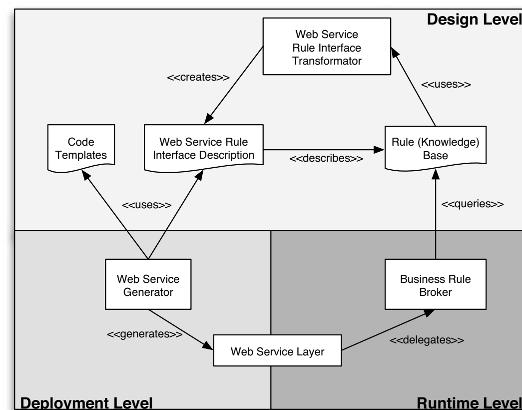

Figure 6. Architectural Levels

We differentiate three architectural levels in our business rules approach. The *design level* covers the process of authoring and managing the rules as well as the extraction of the rule interface description from the different rule representations as depicted in Figure 4. The *Web Service Rule Interface Transformator* iterates through all the knowledge bases and extracts the necessary information by using XSLT and some required post-processing.

In the deployment level, the main task is the Web service code generation, based on the WSRID and the source code templates from the design level. The four step-cycle described in Section 4.4.1 is executed. The concrete packaging and deployment of the generated Web services is application server specific.

The Web service layer is the bridge between both levels. It is generated at deployment and queried by other applications at runtime. The Web service layer itself forwards its requests to the *Business Rules Broker* which queries the knowledge base to get the results of the rule execution.

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 **IEEE**
IEEE COMPUTER SOCIETY

4.6. Implementational Aspects

The Business Rules Broker interface (cf. Figure 1) and the plug-in concept for the adapters are implemented with Java. Each business rules engine is encapsulated as a plug-in, consisting of a its libraries (as JAR archives) and a configuration file. This concept allows users to add and remove new rule engines dynamically to the broker, simply by placing the libraries and the configuration under a given `plugin` directory where it is detected automatically by using `FileSystemWatcher` component. Currently, we use Jess and Drools as two example rule engines for the prototype implementation.

The generation of the Web Service code is performed by a `JavaWebServiceCodeGenerator` component which implements the core logic for generating, compiling and deploying Java Web service code. The Web service generator uses a traditional source code generator based on an internal object model (IOM). Furthermore, we have implemented a platform specific model (PSM) for the Java platform. The code generator takes an input XML file, specifying the interfaces and classes with their methods to be generated. Furthermore, the method implementation can also be specified in XML. The process of generating a Web service is performed by transforming the aforementioned WSRID (cf. Listing 2) to the appropriate input for the code generator by using an XSLT stylesheet. Such a two-stage generation approach (transform WSRID to the XML input of code generator followed by the actual code generation) offers great flexibility. Firstly, the code generator can be easily changed without changing the Web service code generation procedure. Secondly, the target platform of the Web services, in our case Java, can be changed without much effort. For example, to generate the Web services for the .NET/C# platform it is necessary to implement a PSM and the associated source code templates for C#. Furthermore, it may be necessary to adapt the stylesheet for generating the input for the code generator.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleDescriptor>
  <RuleService name="CreditCardService">
    <ServiceOperation name="validateCreditCardPayment">
      <Description>
        Checks before credit card payment.
      </Description>
      <Uri>urn:jess-creditcardrules</Uri>
      <Parameter name="consumer">
        at.ac.tuwien.infosys.iowaf.examples.Customer
      </Parameter>
      <Parameter name="invoice">
        at.ac.tuwien.infosys.iowaf.examples.Invoice
      </Parameter>
    </ServiceOperation>
  </RuleService>
</RuleDescriptor>
```

Listing 2. Web Service Rule Interface Descriptor

In Listing 2, a simple WSRID is depicted, which transforms a credit card check rule to a Web service called `CreditCardService` with one operation called `validateCreditCardPayment` taking two parameters, the customer and the invoice. Using this WSRID the Web service generator can generate the code for various platforms. The concrete classes and interfaces to be generated and needed to successfully deploy and run a Web service are platform specific. For example, one of our target platforms, JBoss 4.0, needs a least one interface, an implementation of this interface and various deployment descriptors for successfully deploying a Web service. Therefore, a JBoss specific XSLT stylesheet is used to transform the WSRID from Listing 2 into to the format acceptable for the Java code generator. In the first step, the code generator creates an interface called `ICreditCardService` and an class called `CreditCardServiceImpl`, respectively. Then, the code is compiled, packaged and deployed by using a pre-defined parameterized Ant (`http://ant.apache.org`) build file. The generated code is depicted in Listing 3 (omitting import declarations).

```
public class CreditCardServiceImpl
  implements ICreditCardService {
  private RuleEngineProxy _engine = new RuleEngineProxy();

  public java.util.Map validateCreditCardPayment(
       Customer consumer, Invoice invoice)
       throws java.rmi.RemoteException,
                    RuleEngineExecutionException {
    Map params = new HashMap();
    params.put("consumer", consumer);
    params.put("invoice", invoice);
    return _engine.executeRules(
       "urn: jess-creditcardrules", params);
  }
}
```

Listing 3. Generated CreditCardService Code

When using a different Java based target application server for deploying the Web services, e.g., Tomcat, only the XLST stylesheet for transforming the WSRID to code generator input has to be changed to generate Tomcat specific code. In case of Tomcat, the interface generation can be omitted. Furthermore, the Ant build file needs to be replaced by a Tomcat specific one.

5. Conclusions

Using business rules in enterprise applications can greatly improve the quality and the maintainability of software. Our service-oriented approach – based on Web service technologies – of executing business rules eases the development and elevates the acceptance of using the presented system in heterogeneous computing environments.

In this paper we proposed an approach on how to implement business rules, managed by different rules engines,

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

by a so-called *Business Rules Broker*, allowing to hide the heterogeneity of different rules engines and providing a service-oriented interface to access and execute the business rules from different knowledge bases. After presenting the foundations of business rules and rules engines, we described our approach which implements a plug-in architecture where each business rules engine can be connected via a plug-in. We implemented a general plug-in for all JSR-094, a recent standardization effort by the Java Community Process, compliant rule engines. All other engines can be connected via a custom plug-in. We presented the detailed design of the system and sketched some implementational aspects.

5.1. Future Work

The current prototype implementation of the *Business Rules Broker* system is quite stable. The results are very promising but there are still some issues that need to be resolved, e.g., the WSRID Transformator is currently not implemented, therefore the WSRID is created by hand. Furthermore, a real-world scenario of using different business rule systems is on our future work list, to demonstrate the performance of using our rule-based approach.

References

[1] G. Antoniou and M. Arief. Executable declarative business rules and their use in electronic commerce. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 6–10. ACM Press, 2002.

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

[3] 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.

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

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

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

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] S. Hendryx. OMG Business Rules Proposal Nears Completion. *Business Rules Journal*, 6(2), February 2005. `http://www.BRCommunity.com/a2005/b226.html`.

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

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

[11] ILOG. Simple Rule Markup Language (SRML). `http://xml.coverpages.org/srml.html`, 2001.

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

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

[14] Managebility Website. Open Source Rule Engines Written In Java. `http://www.manageability.org/blog/stuff/rule_engines`.

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

[16] OMG. Business Semantics of Business Rules – Request for Proposal. `http://www.omg.org/cgi-bin/doc?br/03-06-03`, 2003.

[17] 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.

[18] F. Rosenberg and S. Dustdar. Business Rule Integration in BPEL – A Service-Oriented Apporach. In *Proceedings of the 7th International IEEE Conference on E-Commerce Technology (CEC 2005)*, 2005.

[19] 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.

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

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

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

[23] 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.

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

[25] B. von Halle. *Business Rules Applied*. Wiley, 1 edition, 2001.

[26] W3C. OWL Web Ontology Language Overview. `http://www.w3.org/TR/owl-features/`. W3C Recommendation 10 February 2004.

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

[28] X. Wang, J. Sun, X. Yang, Z. He, and S. Maddineni. Business rules extraction from large legacy systems. In *Proceedings of the Eighth European Conference on Software Maintenance and Reengineering*, pages 249–258, 2004.

Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)
0-7695-2384-6/05 $20.00 © 2005 **IEEE**

IEEE COMPUTER SOCIETY