Towards a Distributed Service-Oriented Business Rules System

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

Abstract

Businesses are changing rapidly and organizations tend to act worldwide and are increasingly becoming distributed over the continents. As a consequence, distributed software systems have to keep track with rapidly changing markets. Business rules provide support for capturing some knowledge that changes frequently. Current business rule systems manage and execute business rules, however, typically lack support for increasingly distributed software systems, in particular, with respect to flexibility and reuse of business rules across distributed rule engines. In this paper we propose a service-oriented distributed business rules system that manages and deploys business rules to various business rule engines. Furthermore, we present the design and some implementation aspects of a service-oriented business rules system based on WS-Coordination. The system supports management and deployment of business rules to various business rules engines. Furthermore, we present a framework that unifies the access to several heterogeneous business rules engines, and we propose a solution that automatically generates and provisions Web services for executing business rules managed by a business rule engine.

Keywords: *Business Rules, Web Services, Automated Web Service Generation, Business Rules Deployment*

1. Introduction

Rapidly increasing globalization and the resulting distribution of companies poses several new challenges regarding architecture and implementation of those distributed systems. One of the key factors of successfully building and running a distributed software system is the ease of making changes to the system without recompiling and redeploying it. The market changes quite quickly, therefore, software systems need to keep track with these changes. Business rules [21, 22] can provide a much more flexible solution for making changes to the business applications by adapting the external rules. Externalizing the rules from the existing business logic is a complex task [24], but it offers the flexibility that is needed for rapidly making such changes. In the area of business rules, there is currently no standard technology for representing and executing rules stored in different knowledge bases. Moreover, no protocols or technologies exist which allow a remote execution, deployment or monitoring of business rules. These facts pose some additional challenges for using business rules in distributed environments.

In recent years, the service-oriented architectures (SOA) and especially Web services [1], as an established technology for implementing SOA, gained a lot of attraction. Standard interfaces (WSDL) and protocols (SOAP) for describing and invoking Web services, and the loose-coupling of these services are important characteristics that lead to more interoperable distributed systems. Therefore, we have investigated the use of SOA for building a distributed business rules system.

The contribution of this paper is threefold: (1) We developed a distributed architecture for managing and deploying business rules at several rule engines by using a distributed coordination model based on WS-Coordination [2]. (2) We allow to use heterogeneous business rules engines by unifying the access to the rules by introducing a *Business Rules Broker*. (3) We propose a solution, on how to expose business rules implemented in a business rules engine, to be invoked as Web services to allow loosely-coupled integration into business applications.

The paper is organized as follows: In Section 2, the basic concepts of business rules and rule engines, including an example, are presented. Section 3 motivates our approach by depicting an example of the telecommunication sector to show possible scenarios of the proposed approach. The design of our distributed approach is presented in Section 4. In Section 5, we present our coordination protocol to successfully manage the distribution and deployment of busi-

Proceedings of the Third European Conference on Web Services (ECOWS'05)
0-7695-2484-2/05 $20.00 © 2005 IEEE

ness rules. Section 6 discusses some of the related work and Section 7 concludes this work and presents some future work.

2. Background

Business rules have proven to be a valuable technology when modeling rules as separate entities that govern ones business. Such rules can then be (re)used throughout different enterprise applications by querying a business rule engine to execute a set of rules. The Business Rules Group [21] 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." Business rules are typically executed by so-called business rules engines, software components implementing algorithms (e.g., RETE [9]) for executing declarative rules.

In literature, different types of business rules have emerged. A good classification is given in [22] or [23]. Mainly, these types include *integrity rules or constraint rules*, *derivation rules* and *reaction rules*. Integrity rules specify assertions or conditions that have to be satisfied in all stages of a system. Derivation or deduction rules specify knowledge that is derived from other knowledge by using inference of mathematical calculation. Reaction or action rules specify event-condition-action rules or production rules (`if-then` statements).

Typically, existing business rules engines do not support all these different rule types as described above. The support for different rule types depends exclusively on the rule engine. Most rules engines support only reaction rules (in form of `if-then` rules). Derivation and integrity rules have to be modeled appropriately in form of reaction rules.

In Listing 1, we present an example of possible rules for the telecommunication sector:

Listing 1. Customer Registration Rules for a Cell Phone Service

```
<ruleset name="CellPhoneRegistration">
  <parameter name="cust" type="Customer"/>
  <parameter name="service" type="ServiceDescription"/>
  <rule>
    <condition>cust.regDate >= 1.8.2005</condition>
    <condition>cust.regDate <= 1.9.2005</condition>
    <action>
      service.rate = rate * 0.90
      service.freeSMS += 100
      service.addOn += "GPRS"
    </action>
  </rule>
  <rule>
    <condition>cust.sex == female</condition>
    <condition>
      hasBirthDayInNextToWeeksAfterRegistration(cust)
    </condition>
    <action>sendFlowersToCustomer(cust);</action>
  </rule>
</ruleset>
```

In this example, the rules are specified in a high level domain-specific language (which is supported, for example, by using Drools [8]). These rules can be invoked by any business component (similar to database calls) via a special API. In this example, the rules are executed by the business application when a new customer is registered. The first rule implements a typical rule for a marketing campaign. When the customer registers between August, 1^{st} and September, 1^{st}, she gets 10% discount, 100 free SMS and GPRS as an add-on. The second rule states, if the customer is female and has birthday in the next two weeks, flowers should be sent to her home.

The usefulness of business rules becomes even more evident when considering the following scenario, which happens daily in the telecommunication sector: A customer wants to buy a new mobile phone with a bundle of services (e.g., GPRS, UMTS, WLAN access, etc). The price of each bundle varies, depending on the services ordered by the customers. All the possible bundle configurations, the calculation of prices and the resolution of conflicts in case two or more services cannot be in the same bundle, can easily be modeled by using condition-action rules.

Based on these examples, we can see the flexibility and the ease of making changes to such rules or adding new or existing ones without requiring changes to the business application, thus reducing and easing software maintenance.

3. Motivating Example

We motivate our distributed approach by examining a distributed software architecture of a telecommunication company. Typically, telecommunication companies operate in multiple countries, often due to buy-outs of competitors or mergers of multiple companies. The allocation of rights and duties among different locations can become a challenge, especially when the software infrastructure does not support the administration and management of the rights and duties.

In Figure 1, a possible (abstracted) scenario of such a distributed enterprise and its computing infrastructure is presented. Let us assume, we have a telecommunication company operating at three different locations. The headquarter of the company is in location A, location B and C are in different countries. Each location basically has the same software service infrastructure, including different rule engines, to support the separation and externalization of business rules from the business logic. The separation of business logic and rules has direct impact on the flexibility of the software system regarding the time needed for changes. In most cases, only the rules need to be changed, so there is

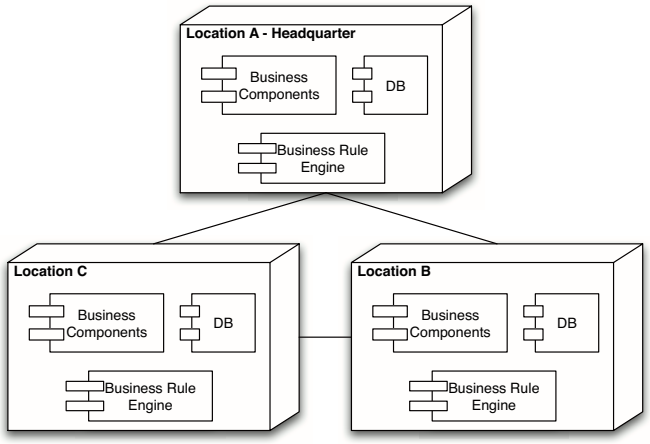

Figure 1. Distributed Business Rules Scenario

no need to recompile and redeploy the software in contrast to "implicit" rules embedded directly in the source code of the business logic.

In this scenario, we assume, based on company regulations, that the administration of the company's universally valid business rules is only performed by the headquarter and all other locations have to use them. All other locations may additionally need different rules (e.g., current location specific marketing campaigns), therefore, each location may additionally specify their customized rules different from the other locations. Due to the fact that business rules change quite often, the mixture of custom business rules for each location (authored by local staff) and common business rules (authored by staff in the headquarter) is crucial. It is a question of scalability, if, for example, all locations access the business rules engine at location A for evaluating the rules common to all locations. Typically, business applications frequently execute rules (at least as often as database calls) to perform their tasks, as a consequence, the performance of the business rules execution and, hence, the execution of the application itself would decrease rapidly. A better solution would be the deployment of business rules to the rule engine that is running at the location where the business applications need them. All business rules engines we analyzed, do not have the possibility to be queried or accessed remotely, neither do they support a remote deployment of rules.

This leads us to the question, how to deploy such rules to all the locations that need these rules for their daily operations? Furthermore, how to achieve such a deployment without any downtime of the system or the business applications? The architecture, we propose in the following paragraphs provides a solution framework to the aforementioned issues.

4. Design of a Distributed Business Rules Approach

In this section, we introduce the main design issues for a distributed service-oriented business rule management system. Firstly, we briefly introduce — for the sake of completeness — an integral part of the architecture, the *Business Rules Broker* developed in [18]. Secondly, the main components of the overall architecture, as depicted in Figure 4, are outlined, followed by a detailed discussion of these components.

4.1. Business Rules Broker Architecture

Based on the paradigm shift from object and component-based computing to service-oriented computing, new challenges arise also for business rules and their execution in various rule engines, tackled by offering business rules as loosely-coupled Web services. Previously [17, 18], we have developed a framework to encapsulate the heterogeneity of different business rules systems and representation languages. We have designed and implemented some additional software layers to abstract from specific rule engine features and provide a common interface for executing business rules. The architecture of our approach is depicted in Figure 2.

Our approach is to integrate several different heterogeneous business rules engines in one common API, the *Business Rules Broker Interface*, by using a plugin-based approach. Each rule engine that is connected to our broker can have numerous rule sets where each is uniquely identified by a URI (Uniform Resource Identifier). Typically, different domain-specific applications have their own rule (or knowledge) bases representing a collection of rule sets.

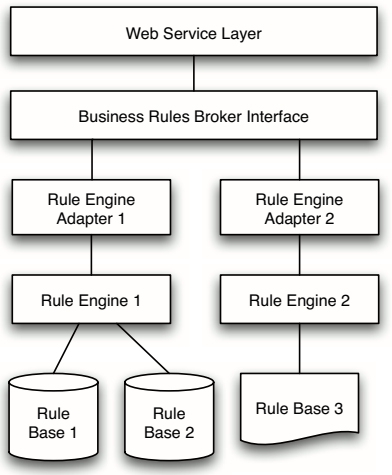

Figure 2. Business Rules Broker Architecture

A rule set is a grouping of rules that is needed to achieve a task (cf. the rules in Listing 1). An application typically consists of a large number of rules sets, which itself contains various atomic rules. Querying the broker is performed by invoking the `executeRules(in uri, in parameters)` method, where the `uri` identifies the rule set to execute and the `parameters`, containing the business objects needed by the rule engine to execute the rules on this data (c.f., the `ServiceDescription` and `Customer` object from Listing 1). These parameters are then returned back as a result, depending on the concrete rule implementation whether the business objects where changed during execution of the rules. When using the broker to execute business rules, dependency constraints between the rule set and its executing rule engine are resolved due to the use of a URI to identify a certain rule set. On execution of rules, the broker internally maps the rule engine (provided as a plugin to the broker) to the URIs identifying the correct rule set that has to be executed.

The second major part of the *Business Rules Broker* concerns dynamic provisioning of Web services for invoking business rules by using the aforementioned Business Rules Broker. We refer to such Web services as *Rule Web services*. Thus, we need a mechanism to automatically generate and deploy such Rule Web services. Each business rules broker can have several heterogeneous business rules engines plugged-in, where each engine uses a different XML-based representation for storing the rules. Therefore, we introduced a *Web Service Rule Interface Descriptor* (WSRID) [18], which is used to describe the Rule Web services that need to be generated. The WSRID encodes all the data needed by the Web service code generator. This includes: service names, service operations, operation parameters and the URIs identifying the rule sets that contain the rules. The process of generating the Rule Web services is shown in Figure 3. An XSLT transformator is used to transform the WSRID into a special XML representation for the `WebServiceCodeGenerator`. This code generator then generates the Rule Web services as specified by the WSRID. After that, the source code is ready for compilation and deployment to various application servers which support Web services. This is done by the `ServiceDeployer` component.

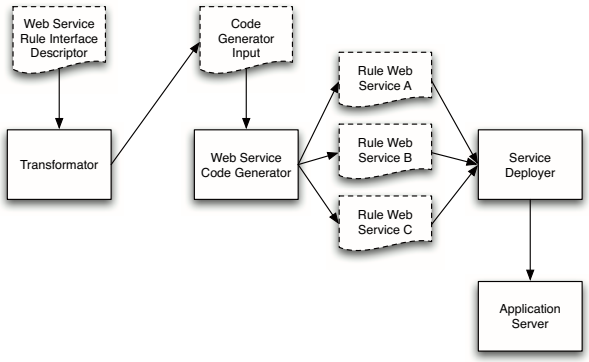

Figure 3. Web Service Generation Process

When applying our approach to the rules from our example as depicted in Listing 1, a Web service with the name `CustomerRegistrationService` (specified in the WSRID), with one operation named `cellPhoneRegistration(Customer c, ServiceDecription s)` is generated. It is also possible to generate a service with multiple operations where each operation is using a different rule engine — connected to the broker — to execute the rules. The generated Web services can then easily be integrated in every application which needs to invoke these rules.

4.2. Distribution Approach

The aforementioned *Business Rules Broker* architecture is only one major part for realizing a distributed business rules approach. The distributed architecture is completely based on Web service technologies [1]. The design is based on a service-oriented architecture (SOA), thus, it leverages loosely-coupling of the different components in the system.

Our distributed system consists of several nodes (machines), each offering a set of software services. In our configuration, we assume that every component is running on its own node, but it does not necessarily have to be the case.

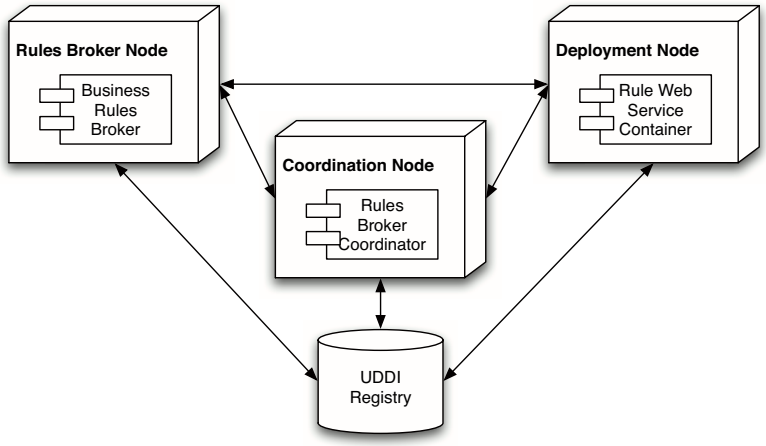

Figure 4. Architectural Approach

A number of components can also be hosted on the same node.

Figure 4 depicts the architecture of our system with the minimal setup. This means that each node is only available once. In larger environments each node is typically available multiple times to achieve high availability and maximum scalability. The minimal setup consists of a *Rules Broker Node*, a *Coordination Node*, a *Deployment Node* and a UDDI registry. We discuss a more sophisticated setup after going through each component of the architecture.

Rules Broker Node. Each *Rules Broker Node* hosts 1...n of the aforementioned *Business Rules Brokers*, referred to as RB_i (where i is the number identifying the node in case more than one are available). As described in Section 4.1, each broker can have a certain number of rule engines plugged-in. Each rule engine manages its local set of business rules, where each set is identified by an URI. The *Business Rules Broker* itself has a Web service layer which is used by the automatically generated and provisioned Rule Web services to execute rules via the available `executeRules` operation.

Besides executing business rules, an RB_i has to manage its deployed rule sets. In a typical development process, the business rule author creates and tests the business rules on a testing system. Then, the rules are deployed on a production system, which is in this case one of the RB_i nodes in the distributed system. Rules are deployed by using the `DeployRules` operation, where the XML-based rules are encoded in the input message of the operation. The received business rules are, first of all, deployed locally to a concrete rule engine connected to the broker as a plugin. Secondly, on deployment of new rules, it is possible to specify a deployment policy, which regulates the distribution of the rules to other RB_i. The RB_i which initially receives the rules to be deployed acts as a "master" and forwards the rules to other RB_i according to the deployment policy. For instance, in our telecommunication example from Figure 1 it is possible that the author of the rules in the headquarter specifies a policy which allows to deploy a set of rules from RB_i in the headquarter only to location B and not to location C.

After the deployment of the rules to certain RB_i (regulated by the aforementioned policy), each RB_i initiates the coordinated generation and deployment of the Rule Web services for executing the recently deployed business rules as Web services. The nodes involved in the coordinated deployment of the Rule Web services are explained in the next paragraphs.

Deployment Node. A deployment node hosts 1...n *Rules Web Service Containers* $RWSC_i$, which listen for deployment messages received by one or more RB_i. A deployment message contains the information which Web services have to be generated and deployed. All the information is available in the WSRID as described above. For the generation of the Web service code we use the `WebServiceCodeGenerator` service, already depicted in Figure 3. Furthermore, we have adopted a conventional Web service environment (Tomcat + Axis in our prototype) by adding a `ServiceDeployer` service. This service that takes the generated Rule Web service code, compiles it and creates a deployment package to successfully provision the automatically created Rule Web services. After the deployment of the newly created services, they are registered at the UDDI registry, thus, external applications

which want to use business rules to facilitate their tasks can immediately use the created services.

Coordination Node. The third component is the *Rules Broker Coordinator* RBC_i hosted at a coordination node. The RBC_i is responsible for coordinating the deployment among a set of rules broker nodes RB_i and one or more $RWSC_i$ to handle the correct deployment of new rules followed by the deployment of the Web services for executing the former deployed rules. Coordinating the deployment of business rules is required when multiple RB_i are available in a deployment scenario to guarantee that all rules and the resulting Rule Web services are generated successfully. For example, the headquarter specifies new business rules for all locations, therefore, the deployment to location B and C needs to be coordinated among the rules brokers and the Web service containers, as described below. Furthermore, handling priorities and conflicts of different rules (in case the local ones are newer than the received ones) and managing consistency or failures in the deployment process are also the task of the coordinator. Since we use a distributed coordination model [1], typically multiple RBC_i with various RB_i exist. We use the WS-Coordination [2] as basic framework to handle the coordination activities among the different components. Therefore, we developed a *HotDeployment* protocol on top of WS-Coordination to generate a reliable mechanism for the deployment of the business rules. We describe this protocol in Section 5.

UDDI Registry. For publishing the Rule Web services, a central UDDI registry is used. Our approach uses the registry to store the endpoints of the available RBC_i and $RWSC_i$. Furthermore, when a new Rule Web service is generated and deployed at a specific $RWSC_i$ by using the `ServiceDeployer`, the endpoint of the dynamically created Rule Web service is added to the UDDI registry to allow other business applications or services to find it.

4.3. Motivating Example Revisited – Service Interactions and Deployment

After the introduction of the distributed architecture and the different components, we use a more sophisticated setup of our example introduced in Section 3 to explain the interactions of the nodes and the deployment of new rules and their corresponding Rule Web services to various business rules engines.

In Figure 5, we have again depicted our telecommunication example setup but now consisting only of two locations. Location A, the headquarter, acts as a "master", which means that the rules are initially deployed to this location by using a deployment tool, and based on the deployment policies, RB_1 is then responsible for the deployment of the rules to the other nodes. Location A consists of one rule broker RB_1 and two Web service containers, $RWSC_{1\_1}$ and $RWSC_{1\_2}$ to achieve a better load balancing of the Rule Web services. Although it is not strictly enforced by our approach, we use a distributed coordination model for managing the deployment, thus, we have more the one coordinator, in this case RBC_{1_1} and RBC_{1_2}. In location B, we have one rules broker RB_2, one Web service container $RWSC_2$ and two coordinators, RBC_{2_1} and RBC_{2_2}.

The deployment process starts when the rule author creates a set of new business rules by using a rule editor. In this scenario, a new set of *customer registration rules* for cell phones, as depicted in Section 2 and 3, should be deployed. After testing the new rules, the deployment tool is used to deploy the newly created rules to RB_1 by invoking the `DeployRules` operation of RB_1 (message (1) in Figure 5). The input message of RB_1 consists of the XML encoded business rules including some metadata (URI of the rule engine, name of the rule set, description, version, etc.). It is also possible to send more than one rule set in the deployment message. Additionally, the deployment message contains a deployment policy specifying the endpoint of further RB_i which should deploy these rules. In our example, according to the received deployment policy, RB_1 has to forward the deployment message to RB_2 (message (2) in Figure 5).

When receiving the `DeployRules` message, RB_1 and subsequently RB_2 perform a lookup in the UDDI registry (not shown in Figure 5) to find RBC_{1_1} and RBC_{2_1} for coordinating the deployment. When the business rules itself are successfully deployed at RB_1 and RB_2, according to the exchanged protocol messages (dashed lines), the process can proceed. After the successful deployment of the business rules in the rule base of the target business rule engine, the first deployment step is done.

The second step is the deployment of the Rule Web services for executing the business rules deployed in the first step in a service-oriented way. Therefore, each RB_i performs a lookup in the UDDI registry to find the endpoint of a $RWSC_i$ in its current location. Concurrently, RB_1 and RB_2 perform a lookup the find a corresponding RBC_i to establish a deployment coordination among RB_1, RB_2, $RWSC_{1\_1}$, $RWSC_{1\_2}$ and $RWSC_2$.

After binding to each $RWSC_i$, RB_i sends a `DeployRuleService` message to each $RWSC_i$, containing the WSRID of the recently deployed customer registration rules to generate a Rule Web service, called `CustomerRegistrationService`. On reception of the `DeployRuleService` message, the $RWSC_i$ uses the `WebServiceCodeGenerator` to generate the source code of the Rule Web service. The `ServiceDeployer` uses the generated source code to compile and deploy the service to the Web service container. During the whole

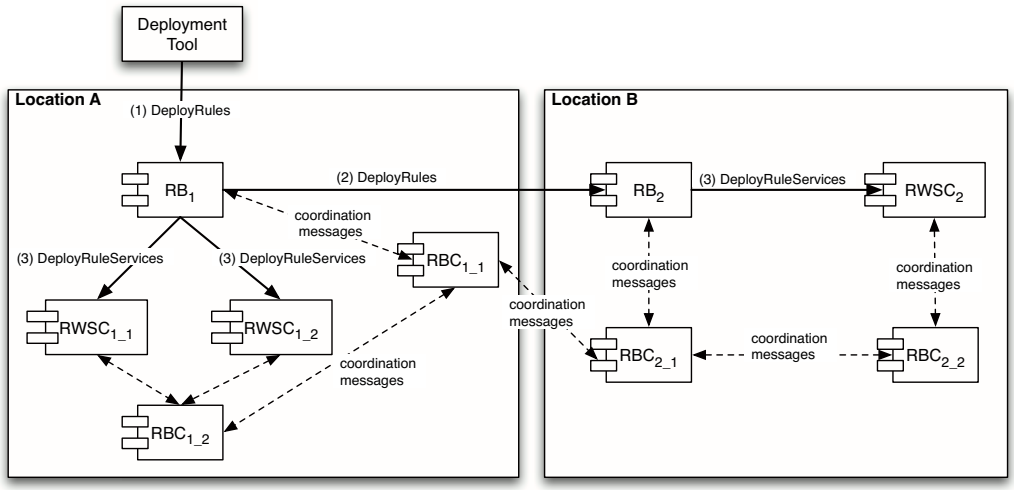

Figure 5. Distributed Business Rules Deployment Scenario

deployment process, each RBC_i exchanges coordination message to report possible errors or successfully committing the deployment. The protocol is responsible for managing a successful deployment of both, the rules in the rule base and the Rule Web service for executing the former deployed rules. If an error occurs, for example, in the first step of the deployment, the protocol initiates a rollback; otherwise it might lead to inconsistencies in different rule bases. If the deployment of the business rules has been successful, and the deployment of the Rule Web services leads to an error, then all the Rule Web services need to be rolled back to the state before the deployment. The deployment of the Rule Web services can then, e.g., be re-executed after fixing the problem.

5. Deployment Coordination Protocol

Coordinating the deployment of new business rules is one of the most important aspects in our distributed business rules environment. The successful deployment of rules to one or more specific rules brokers RB_i, e.g., from the headquarter location A, is the prerequisite for the generation and the deployment of the Rule Web services for executing the business rules. Considering a large number of different locations, a distributed coordination mechanism is necessary to manage the deployment and to ensure that the system and their provisioned Rule Web service are in consistent state.

5.1. WS-Coordination

We have build our solution on top of the WS-Coordination [2] specification, which is a general-purpose specification for achieving coordination among several participants in the Web service area. WS-Coordination is, in general, protocol independent, meaning that every proprietary protocol can be implemented for the use with WS-Coordination specification. Every coordinator has to implement the `ActivationCoordinatorPortType`, with the `CreateCoordinationContext` operation used to initiate a coordination session by creating a common context, shared among all participants. Furthermore, the `RegistrationCoordinatorPortType` has to be implemented (with the `RegisterOperation`) to register a participant with a specific protocol in the coordination. Each participant in the coordination has to implement the `ActivationRequesterPortType` with its only operation `CreateCoordinationContextResponseOperation` to send back the created coordination context to the participant. Furthermore, the `RegistrationRequestorPortType`, with the operation `RegisterResponseOperation` has to be implemented, to receive a reference to the protocol specific port type from the coordinator.

Based on this general port types needed for coordination, we depict our specific protocol for the "hot-deployment" of business rules and their corresponding Rule Web services as a state diagram in Figure 6.

Our protocol, called *HotDeployment* protocol, is used to coordinate the deployment among different RB_i and $RWSC_i$, and it fits into the WS-Coordination specification.

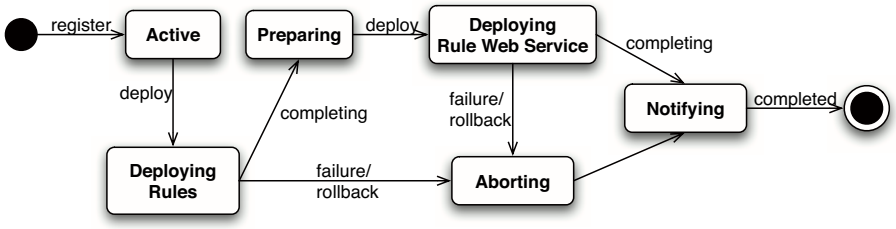

Figure 6. HotDeployment Protocol State Diagram

Hot deployment in this context means that the system is available (serving client requests), while the deployment business rules and their corresponding Rule Web services for invoking the business rules takes place. It guarantees that no downtime of the system is required to generate and deploy new Web services. The protocol is used by our system to coordinate the deployment of business rules to different RB_i followed by deployment of the Rule Web services to various $RWSC_i$ nodes. A state diagram of the protocol is presented in Figure 6.

After a participant registers to take part in the coordination, the participant is in state active. The protocol is initiated when the `DeployRules` operation at a certain rule engine RB_i is executed by using an external deployment tool. Based on the deployment policy, which specifies all the RB_i that should additionally deploy the business rules, the protocol coordinates the deployment among multiple RB_i to successfully deploy the rules. After each RB_i has deployed its business rules, the first phase is completed and the protocol reaches the `Preparing` state, where each $RWSC_i$ is notified to prepare for the generation and deployment of Rule Web services. If an $RWSC_i$ is prepared for deployment, the `DeployRuleServices` operation of each $RWSC_i$ is executed by the corresponding RB_i. When successfully finished, each $RWSC_i$ notifies the initiating RB_i about the outcome of the deployment. When an error occurs, the initiating RB_i, gets informed to react appropriately (e.g., redeploy later or restart the server, etc). Furthermore, the protocol supports a rollback of business rules, which is necessary if one RB_i cannot deploy its rules without errors or an $RWSC_i$ cannot generate and deploy the Rule Web services. In general, it is a prerequisite that a RB_i has successfully deployed its business rules before the corresponding $RWSC_i$ can deploy its Rule Web services. The deployment of the Rule Web services is initiated by the RB_i. In case of an error in the Rule Web service deployment, RB_i can decide, based on the policy, whether to retry the deployment of the services or to abort and rollback the deployment.

As required by WS-Coordination, each participant and coordinator has to implement the following port types summarized in Table 1. The provided operations specify the initiation of the deployment, with the `Prepare` operation, the successful completion of a deployment is issued with the `Complete` operation. The operations `Error` and `Unknown` are used to signal an error during the deployment or to report an unknown error (e.g., an unexpected server crash). The `Rollback` operation is used to rollback a currently running deployment process in case of unexpected error which would lead to inconsistencies.

PortType	Service Operations
DeploymentCoordinator-PortType	Prepared, Completed, Error, Unknown
DeploymentParticipant-PortType	Prepare, Complete, Rollback, Error, Unknown

Table 1. Hot Deployment Coordination PortTypes

5.2. Rule Deployment

In this section, we depict an example to show the deployment of business rules with a focus on the involved coordination messages on an abstract level by using the sequence diagram in Figure 7.

In this example, we use only one rule broker RB_1, one $RWSC_1$ and two coordinators, due to space reasons in the sequence diagram. Therefore, the deployment of business rules to RB_1 does not need to be coordinated with other rules brokers.

The interaction starts by invoking the `DeployRules` operation at RB_1, which is done by a deployment tool. After receiving the `DeployRules` message from the deployment tool, RB_1 creates a coordination context with Coordinator 1 (message 2) which in turn sends back the coordination context (message 3). Then RB_1 register for the *HotDeployment* protocol (message 4+5). After handling the registration, RB_1 queries the UDDI registry to find an $RWSC_i$ where it can deploy the Rule Web service (message 6) and

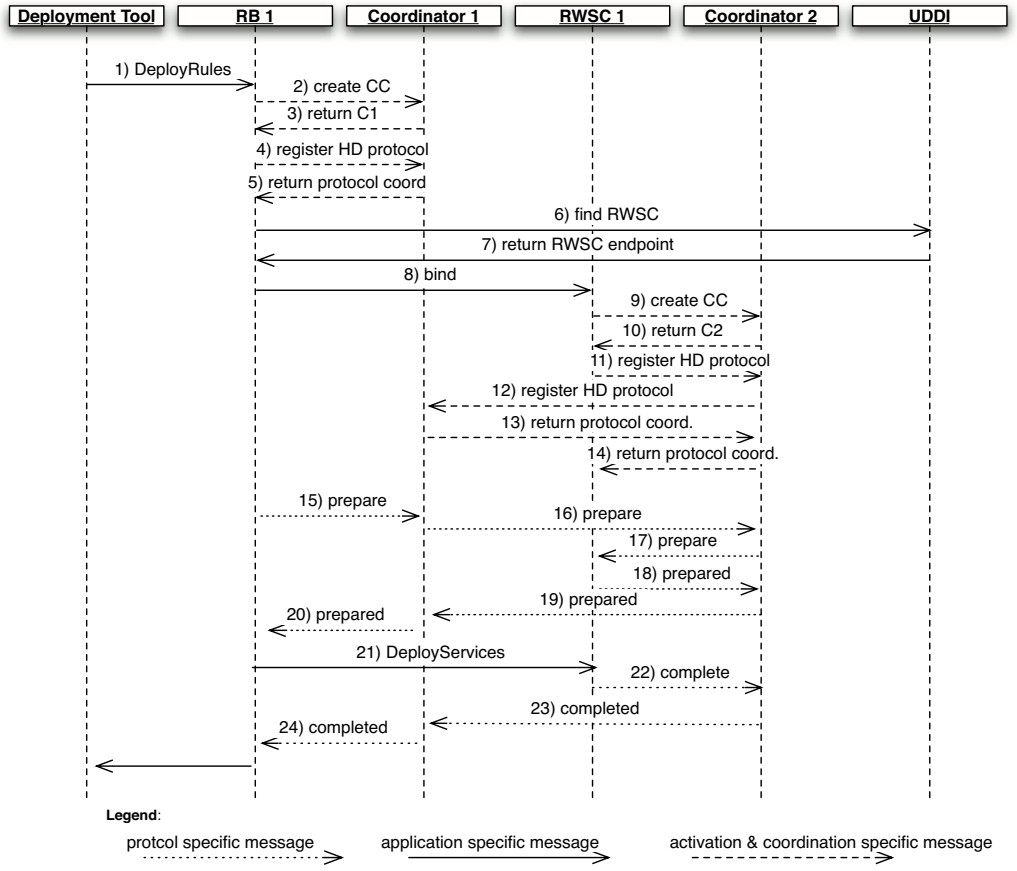

Figure 7. Rule Deployment Scenario

it dynamically binds to the returned $RWSC_1$ (message 8). Then the coordinator RBC_1 and RBC_2 agree on the coordination and register also for the *HotDeployment* protocol (message 9-14). The RB_1 sends out a prepare protocol message to notify $RWSC_1$ that it should prepare for deployment (message 15-20). When RB_1 receives that $RWSC_1$ is ready for deployment, the `DeployServices` operation is executed at $RWSC_1$. After successfully generating and deploying the Rule Web services to $RWSC_1$, a complete message is sent to notify RB_1 that the deployment was successful (message 22-24). In case of an error, the error is returned and the RB_1 can handle it appropriately (e.g., schedule for later redeployment).

6. Related Work

In recent years business rules gained a lot of interest for building enterprise applications. Much research has been dedicated to the area of rule representation and different rule engines. Nevertheless, only little work has been done in the area of architectural approaches for distributed business rule systems and, in particular, service-oriented solutions. To the best of our knowledge, there is currently no existing approach focusing on distributed business rules systems including an automated transformation of business rules knowledge into Web services that access and execute these rules via a unified API.

We categorize our related work in two different parts: Firstly, we describe some existing business rule systems, existing projects and standardization efforts. Secondly, we focus on various research approaches that leverage business rules technology.

6.1. Systems, Standards and Projects

Various commercial business rule systems exist, offering integrated business rule management suites. Such suites typically consist of a rule engine, a rule repository, a rule editor, debugger and monitoring system. We refer to ILOG

[11] as one well-known commercial representative. Furthermore, various open-source systems have emerged. In our prototypical implementation, we use Drools [8] and Jess [14] as two examples. Both engines use the well-known RETE algorithm [9] for matching the facts against the rules and both are tailored for the Java programming language. It is worth noting that the aforementioned rule engines do not provide such an integrated business management solution such as ILOG, but they provide a small software package suitable for integration in different business applications.

Most recently, the Java Community Process finished the final version of their Java Rule Engine API. The goal of the JSR-094 (Java Specification Request) is 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) [13]. In contrast to the Java Rule Engine API, we focus on the integration of heterogeneous rule engines, not limiting them to the Java platform. Furthermore, we have a rather high degree of decoupling between the rule engine and our *Business Rules Broker Interface* through a plugin-based architecture.

In December 2004, the Object Management Group (OMG)[15] issued the Business Semantics for Business Rules (BSBR) Request for Proposal (RFP)[16], 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.

In an earlier project called *Business Rules for Electronic Commerce*, carried out by IBM Research, a framework for representing business rules [10] was developed. One of the results of this project was a Java library called *CommonRules* using declarative logic as knowledge representation language.

In addition to the aforementioned approaches different rule representation languages have emerged, currently lacking a standard language. RuleML [19] is currently the most promising rule representation initiative. Other languages include SRML (Simple Rule Markup Language) [12] from ILOG, BRML (Business Rule Markup Language) as one result of the above mentioned IBM project and other proprietary languages developed by different vendors.

6.2. Research Approaches

In [17], we presented, how BPEL can be enriched with business rules evaluated during the execution of the process orchestration. The concept is similar to the concepts proposed by the aspect oriented programming (AOP) paradigm, where we apply the rules via interceptor, either before or after the execution of a BPEL activity. We implemented a prototype, which uses the Business Rules Broker, an Enterprise Service Bus and a transformation engine.

Business rules facilitate other interesting approaches as published in [3, 4]. The authors depicted an approach on how to separate and externalize business rules from the composition process, thus reducing the complexity of the composition. Furthermore, they presented a system called AOP4BPEL, an aspect-aware orchestration engine, which allows modeling business rules as aspects and integrating them into the composition process.

Cibran et. at. [7] presented different example categories of business rules that are applicable in Web service composition processes. They implemented these business rules by using AOP techniques to model the dynamic rules as stateful aspects, allowing a seamless integration into the composition process by using AOP weaving techniques.

In [5, 6], the authors try to encapsulate the reusable business rules from the core application logic. In object-oriented systems, the business rules are encapsulated from the core application logic but the connectors — code that connects the rules to the core application — are not. Since business rules evolve over time, their approach is to fully decouple the connectors from the core application logic by using aspect-oriented programming.

Distributed execution of business rules by facilitating Web service technologies was developed in [20]. The author argues that business rules are inherently distributed, therefore, the execution of the rules should also be distributed. The proposed approach uses SOAP intermediaries and business rules encoded in the header of the SOAP message. The execution sequence is specified by the order of the rules and each intermediary processes one entry along the execution path. In contrast to our approach, the author focuses more on the distributed execution aspect of business rules, whereas we focus on distributing business rules engines, managing the distributed rules engines and the provisioning of Rule Web services.

7. Conclusions

In this paper we have addressed the issue of designing a distributed business rule management system based entirely on Web service technologies. In the first part, we have motivated our work by depicting an example from the telecommunication industry. Based on our previous work, we have identified the main components for the distributed approach. We have presented the design of the each component and have shown some aspects of our current deployment strategy.

A *HotDeployment* protocol fitting into the existing WS-Coordination framework has been developed, which is suit-

able for so-called hot deployment of business rules to various business rule engines and the corresponding Rule Web services. Based on the distributed coordination architecture, the number of deployment nodes and rules engines to be interconnected is not limited to a certain amount, therefore, having high flexibility and achieving scalability in case of performance bottlenecks at certain rule engines.

7.1. Future Work

Our current prototype implementation lacks conflict handling abilities which might occur when deploying business rules and their Rule Web services. Conflicts might also occur whenever newer rules are available at certain business rule engines, or newly deployed rules conflict with other rules.

Furthermore, we have not investigated security and encryption issues, which will be required when dealing with the deployment of business rules, which represent the knowledge of an enterprise.

One of the most important aspects is to find suitable larger scale demonstration applications where we can apply our approach. Ideally, existing applications, embedding rules in source code should be reengineered (at least some parts) to evaluate the usefulness of our architecture and do some performance studies.

References

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

[2] BEA, IBM, Microsoft. Web Services Coordination (WS-Coordination). `ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf`, November 2004.

[3] 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.

[4] 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.

[5] M. Cibran and M. D'Hondt. Composable and reusable business rules using AspectJ. In *Proceedings of the Workshop on Software Engineering Properties of Languages for Aspect Technologies (SPLAT) at the International Conference on Aspect-Oriented Software Development*, 2003.

[6] M. Cibran, M. D'Hondt, and V. Jonckers. Aspect-oriented programming for connecting business rules. In *Proceedings of the 6th International Conference on Business Information Systems (BIS)*, 2003.

[7] M. A. Cibrán and B. Verheecke. Dynamic business rules for web service composition. In *Proceedings of the Second Dynamic Aspects Workshop (DAW)*, 2005.

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

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

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

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

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

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

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

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

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

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

[20] R. Schmidt. Web services based execution of business rules. In *Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

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

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

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

[24] 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.