A GENERIC API FOR THE INTEGRATION OF RBS IN AN ESB

David Haase, Karl-Heinz Krempels and Christoph Terwelp
Informatik 5, Information Systems and Databases, RWTH Aachen University, Aachen, Germany

Keywords: Rule-based system, Application integration, JSR 94, RuleML, Web services, Enterprise service bus, Service oriented architecture.

Abstract: The implementation of business processes using Rule-based Systems requires efficient deployment of Rule-based Applications. This requires a system control interface for a Rule-based System to configure the Rule-based System and its input and output interfaces for a given application. Todays only proprietary interface are provided by Rule-based Systems based on their specific knowledge representation and implementation language.
This paper presents a generic interface for the integration of a Rule-based System in an Enterprise Service Bus. The new Rule-based System interface extends the interoperability and flexibility of Rule-based Applications based on a proposed description standard for Rule-based Applications and their service interfaces.

1 INTRODUCTION

Rule-based Systems (RBSs) become more and more interesting for the inspection of large data streams, pattern recognition in event streams, or conditional information processing. The integration of RBSs in business processes requires suitable input and output interfaces, and a system control interface. Existing systems only provide proprietary interfaces based on specific knowledge representation and control mechanisms.

A first standardization initiative defined a generic interface specification for RBSs developed in Java (JSR 94 WG Group, 2004). But as the Java Specification Request (JSR) 94 is underspecified and limited to the programming language Java, a more comprehensive approach is needed. So in this paper we design and discuss a new interface based on well defined and already widely used standards.

This paper is organized as follows: In Section 2 we introduce RBSs and in Section 3 all the concepts required to describe a Rule-based Application (RBA). Section 4 discusses the JSR 94 interface definition. Section 5 presents a new approach for a generic Application Programming Interface (API) for RBS followed by the description of a suitable configuration interface in Section 6. In Section 8 we discuss implementation tools and methods for the new interface. Section 9 comprises the conclusion.

2 RULE-BASED SYSTEMS

A RBS can be used by an user to automatically reason about an existing knowledge base with the help of rules in order to deduce new knowledge.

A RBS consists in general of:

Working Memory which stores the facts (ontological concepts) that are asserted by the system. Each fact is a data structure that represents the information about one entity or relation in the domain of consideration.

Rule Base which stores the rules that are used to deduce new knowledge. Each rule consists of a condition and an action-list that is processed every time when the condition becomes true.

Inference Engine which applies all rules from the rule base to the working memory to deduce new knowledge. This is called the reasoning process. Whenever the assignment of a new fact satisfies the condition of a rule in the rule base, the action-list of the rule will be executed. The action-list can change the working memory and the rule base to produce new knowledge.

The inference engine consists of a pattern matcher that identifies all the rules with fulfilled conditions and marks them as activated. Activated rules form together a conflict set, since the execution of a rule can deactivate another activated rule. These conflicts are solved with a conflict solving strategy leading to a

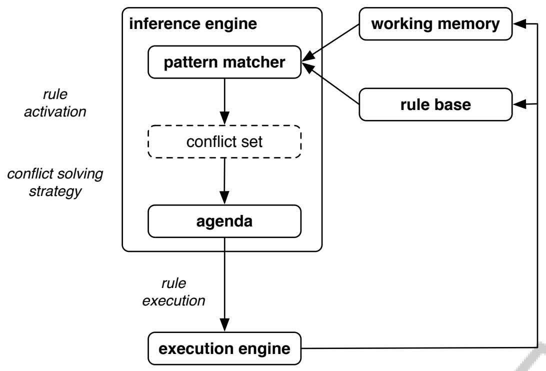

Figure 1: RBS Architecture.

ordered list of rules that are executed by the execution engine. Fig. 3 shows the components of a RBS and the interaction cycle among them .

3 DESCRIPTION OF RULE-BASED APPLICATIONS

A RBA consists of a large set of rules that are applied to data sets or data streams in real time. Therefore, a representation of rules that describe the applications logic, of data to which the rules have to be applied, and of an application interface is required. An Application Data Definition Language (ADDL) is used to describe the concepts required to represent the applications data. An Application Data Definition Document (ADDD) describes the applications ontology and consists of the description of data types using the ADDL. An Application Data Language (ADL) is used to describe the applications data. An Application Data Document (ADD) contains a reference to an ADDD and the definition of data (based on the ontology defined by the ADDD) using the ADL. Furthermore, the ADDD is used for the succeeding rule modeling task.

An Application Rule Definition Language (ARDL) is used to define the rules describing the business logic of a RBA in an Application Rule Definition Document (ARDD). An Application Service Interface (ASI) is required to provide the functionality of an RBA as a service which can be used by other applications. An ASI is defined in an Application Service Interface Definition Document (ASIDD) using an Application Service Interface Definition Language (ASIDL). All methods of the ASI defined in the ASIDD have to use only the data types defined in the ADDD. So incoming data from method invocations can directly be transferred into the RBS and the result can be queried from the RBS and returned directly to the invoker.

A Rule-based Application Description Document (RBADD) defines a RBA using a Rule-based Application Defintion Language (RBADL). A RBADD contains an ADDD, an ARDD, an ASIDD, and an ADD. The relationship among the introduced definition languages and documents required for the description of a RBA is shown in Fig. 2.

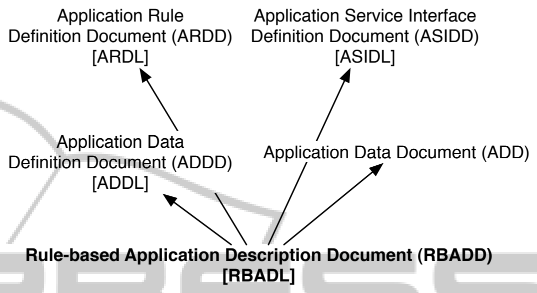
Application Rule Definition Document (ARDD) [ARDL]
Application Service Interface Definition Document (ASIDD) [ASIDL]
Application Data Definition Document (ADDD) [ADDL]
Application Data Document (ADD)
Rule-based Application Description Document (RBADD) [RBADL]

Figure 2: Rule-based Application Description Components.

4 EXISTING APPROACH: JSR 94

The JSR 94 (JSR 94 WG Group, 2004) was an early attempt to establish a common interface for RBS. While succeeding in creating a simple API for RBS, it failed in making RBS exchangeable. This is caused by underspecified data types for several defined methods. So, several interface methods just act on objects of the Java class Object. A definition of their contents is missing, making every implementation depending on their specific Java classes. In this regard JSR 94 is similar to Java Database Connectivity (JDBC) (JSR 221 Working Group, 2011), as both just define a simple Java API with underspecified data types. So JDBC requires a developer to provide Structured Query Language (SQL) (ISO/IEC 9075-1:2008, 2011) statements in the SQL dialect specific to the used Relational Database Management System (RDBMS). JDBC uses Java-String as the main datatype, JSR 94 uses plain Java-Objects, exposing the users to the implementation details of the used backend.

5 A GENERIC API ARCHITECTURE

In this section a new architecture for the integration of a RBS into a Service Oriented Architectures (SOA) based on Web Services (WSs) (WSA Working Group, 2011) is presented and the automated configuration of a RBA in an Enterprise Service Bus (ESB) is discussed.

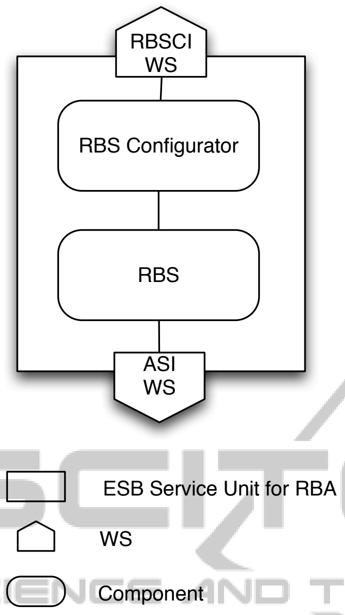

Figure 3: Proposal for a generic API Architecture.

A RBADD is used for the configuration of an instance of a RBS in order to create an instance of the corresponding RBA. Therefore, a RBS Configuration Interface (RBSCI) to import the RBADD and a RBS Configurator (RBSC) to process them are required. At first the RBSC initializes the RBS with the concepts defined in the ADDD, the rules specified in the ARDD and the initialization data defined in ADD. Finally, the RBSCI creates the WS specified in ASIDD, connects it to the RBS, and registers the WS with a suitable Universal Description, Discovery, and Integration (UDDI) directory service. Now the RBA is ready to handle method invocations through the ASI.

6 DESCRIPTION OF THE RBS CONTROL INTERFACE

In this section we discuss the methods of the RBSCI and their operation mode. The methods are invoked with a reference to overwrite a predefined value.

The following methods control the operation mode of a RBS:

load ontology [ADDD] process the predefined or the referenced ADDD.

load rules [ARDD] process the predefined or the referenced ARDD.

load instances [ADD] process the predefined or the referenced ADD.

clear ontology remove all the ontology definitions from the RBS. Implicitly all depending rules and instances are removed.

clear rules remove all the rule definitions from the RBS.

clear instances remove all existing instances of any ontological definition from the RBS.

create [ASIDD] create an ASI from the predefined or the referenced ASIDD.

destroy asi destroy the existing ASI.

attach asi attach the ASI to the RBS.

detach asi dettach the ASI from the RBS.

register asi [WSDD] register the ASI WS with the predefined or the referenced UDDI by the Web Service Description Document (WSDD).

unregister asi [WSDD] unregister the ASI WS from predefined or the referenced UDDI registry.

init [RBADD] invoke the following methods for the predefined or the referenced RBADD: reset, load ontology, load rules, load instances, create asi, attach asi, register asi, fire. In case of a referenced RBADD the methods are invoked with a reference to a corresponding (sub)document.

reset invoke the following methods: clear ontology, clear rules, clear instances, unregister asi, detach asi, destroy asi.

fire apply the rules from the RBS rule memory to the data from the RBS working memory one time.

get strategies return a list of rule processing strategies of the RBS.

set strategy [aStrategy] set the rule processing strategy of the RBS to aStrategy.

get status return a list of system operation parameters of the RBS as (parameter name, parameter value) pairs.

get settings return a list of system configuration parameters of the RBS as (parameter name, parameter value) pairs..

set settings [aConfiguration] set the system configuration parameters of the RBS to the (parameter name, parameter value) pairs from the referenced configuration aConfiguration.

The RBSCI is deployed as a WS and must be registered with an UDDI registry to be used later for the configuration and deployment of an RBA in an ESB (WS-BPEL Working Group, 2011).

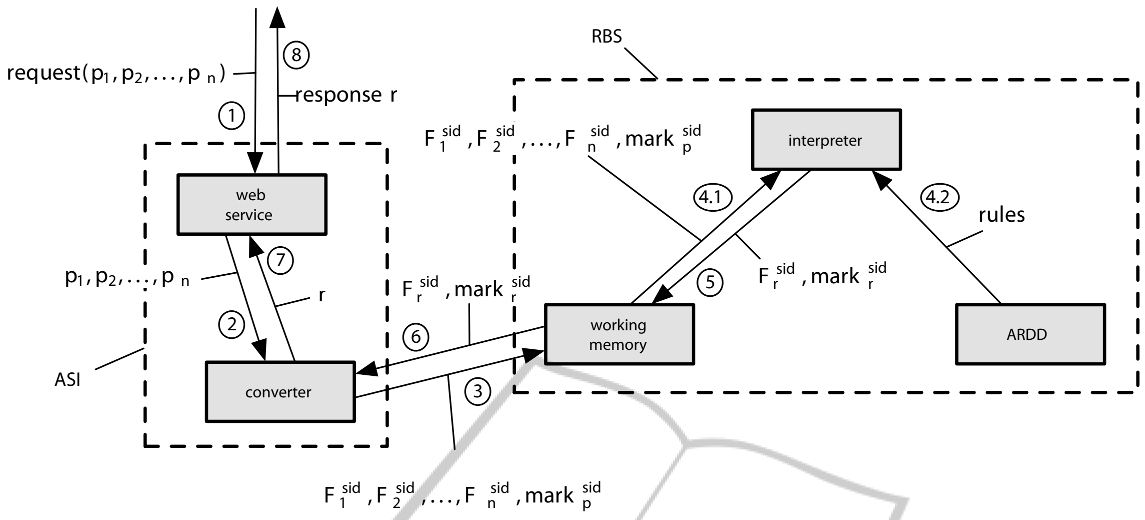
request($p_1, p_2, \dots, p_n$)
8
response r
1
RBS
$F_1^{sid}, F_2^{sid}, \dots, F_n^{sid}$, mark $_p^{sid}$
interpreter
web service
4.1
4.2
rules
7
$p_1, p_2, \dots, p_n$
r
F_r^{sid}, mark $_r^{sid}$
5
F_r^{sid}, mark $_r^{sid}$
working memory
ARDD
2
6
ASI
converter
3
$F_1^{sid}, F_2^{sid}, \dots, F_n^{sid}$, mark $_p^{sid}$

Figure 4: Translation of queries to the ASI WS.

The implementation of the RBSCI requires the definition of a set of rule selection strategies supported by RBSs. The rule selection strategies implemented by a RBS can be requested with the `get strategies` method. A suitable strategy can be configured with the `set strategy` method. If no strategy is configured the default one should be used. However, the common strategies supported by the RBSCI description have to be defined and added to the specification of RBS API discussed in this paper. This task has to be done for the common sets of settings and status.

7 QUERY TRANSLATION

The methods declared by the ASIDD of the ASI have to be defined in the RBADD and executed by the RBS. To handle ASI method invocations in the RBS the invocation must be translated into a representation understandable by the RBS. Both ASI and RBS are using the same ADDD but rendered in different ADLs. Therefore, it is required to translate the parameters for every method invocation from the ASIs ADL to the RBSs ADL (e.g., using automatically generated Extensible Stylesheet Language Transformation (XSLT)) and to synchronize this translation with RBS operation. Synchronization is realized using a marker which is created after all parameters representing a method invocation have been translated. Figure 4 shows how an ASI method invocation is processed. An incoming *request* (1) of the ASI with parameters $p_1, p_2, \dots, p_n$ is received by the WS in the ASI specific ADL. These parameters are translated by a converter (2, 3) into the RBS specific ADL $F_1^{sid}, F_2^{sid}, \dots, F_n^{sid}$. Each parameter is marked with an invocation id *sid* to identify the corresponding method invocation. The translated parameter set is extended by a synchronization marker $mark_p^{sid}$ and processed by the RBS (4) using *rules* from the ARDD. The RBS produces the result F_r^{sid} and the synchronization marker $mark_r^{sid}$ (5). The marker triggers the converter to translate the RBSs result from the ADL of the RBS into a return value r in the ADL of the ASI (6, 7). Finally the WS returns r in a *response* message (8).

8 IMPLEMENTATION TOOLS AND METHODS

The implementation of the proposed API for RBA requires the design or selection of suitable languages to describe an RBA as it is specified in Section 3. Designed ADDLs are eXtensible Markup Language (XML)-based description languages like Ressource Description Framework (RDF) (RDF Working Group, 2011) and Ontoloy Web Language (OWL) (OWL Working Group, 2011). An ARDL like Rule Markup Language (RuleML) (RuleML Working Group, 2011) or Semantic Web Rule Language (SWRL) can be used for the definition of the RBA rules based on a defined ontology. However, using SWRL for the definition of rules requires the specification of the ontology in Web Ontology Language (OWL).

Both RuleML and SWRL can represent RBS data and can be used as ADLs. The ASI is defined as WS based on the Web Service Description Language (WSDL) (WSDL Working Group, 2011). XML is used as ADL for the ASI because of the usage of WS.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="rbads"
  targetNamespace="rbads"
  elementFormDefault="qualified">
 <xs:element name="Application">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="ADDD"
    type="document"/>
    <xs:element name="ARDD"
    type="document"/>
    <xs:element name="ADD"
    type="document"/>
    <xs:element name="ASIDD"
    type="document"/>
   </xs:sequence>
   <xs:attribute name="name"
   type="xs:string"/>
   <xs:attribute name="version"
   type="xs:string"/>
  </xs:complexType>
 </xs:element>
 <xs:complexType
  name="document">
  <xs:attribute name="name"
  type="xs:string"/>
  <xs:attribute name="ref"
  type="xs:anyURI"/>
 </xs:complexType>
</xs:schema>
```

Figure 5: RBADD Schema.

Finally, a RBADD is an XML document based on the XML schema given in Fig. 5. The schema defines an `Application` element and a complex type `document`. The `Application` element contains two attributes `name` and `version`, and a set of four elements of type `document`, describing the documents of a RBADD. The complex type `document` has two attributes: `name` and `ref`. In Fig. 6 a sample RBADD is shown. Such a document is passed to the RBSC through the RBSCI in an ESB.

```
<?xml version="1.0" encoding="UTF-8"?>
<Application xmlns="rbads"
  name="Preference Based Scheduling"
  version="1.0">
 <ADDD name="Scheduler"
   ref="http://jamocha.org/RBA/pbs/ADDD.owl" />
 <ARDD name="Scheduler-Rules"
   ref="http://jamocha.org/RBA/pbs/ARDD.rml" />
 <ADD name="Scheduler-Data"
   ref="http://jamocha.org/RBA/pbs/ADD.rml" />
 <ASIDD name="PBS-Interface"
   ref="http://jamocha.org/RBA/pbs/pbs.wsdl" />
</Application>
```

Figure 6: RBADD Example.

9 CONCLUSIONS AND OUTLOOK

The presented approach for a generic API for RBA would increase the interoperability of a RBA based on its automated deployment and orchestration. The main benefits are the automated deployment of RBAs and runtime compatibility of RBA. This allows the exchange of one compliant RBS by another one. So, process integration designers became able to use a higher level of automated RBA deployment in SOA.

The rules centralization pattern discussed in (Erl, 2009)[page 216pp] assumes the use of a business rules management to access and maintain rules in a centralized manner. The new RBS API allows the extension of this pattern to cover the deployment management for RBA also.

REFERENCES

Erl, T. (2009). SOA design patterns. *The Prentice Hall Service-Oriented Computing Series From Thomas Erl*, page 800.

ISO/IEC 9075-1:2008 (2011). Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45498.

JSR 221 Working Group (2011). JSR 221: JDBC 4.0 API Specification. http://jcp.org/en/jsr/detail?id=221.

JSR 94 WG Group (2004). JSR 94: Java Rule Engine API. http://jcp.org/en/jsr/detail?id=94.

OWL Working Group (2011). OWL Web Ontology Language Overview. http://www.w3.org/2007/OWL/wiki/OWL_Working_Group.

RDF Working Group (2011). Ressource Description Framework. http://www.w3.org/RDF/.

RuleML Working Group (2011). The Rule Markup Initiative. http://ruleml.org/.

WS-BPEL Working Group (2011). Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

WSA Working Group (2011). Web Service Architecture Specification. http://www.w3.org/TR/ws-arch/.

WSDL Working Group (2011). Web Services Description Language. http://www.w3.org/TR/wsdl20/.