Applying Distributed Business Rules – The VIDRE Approach

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

Abstract

Today's business processes are not static, they need to be adapted frequently to reflect changing business requirements. Several business process languages such as WS-BPEL have emerged for specifying business processes based on Web service technologies. Activities in such business processes are typically implemented as Web services by using modern programming languages. These services encapsulate the business logic in terms of application-specific code. This approach lacks flexibility in terms of capturing and executing the business rules that define how certain activities work and how decisions are made. Changing "hard-coded" business rules leads to changes in the service implementations and it cannot be done efficiently without redeploying the service which may affect running business processes. Therefore, we propose VIDRE *a distributed service-oriented business rule engine, which enables business processes or enterprise applications to access business rules as easily as a database by exposing them as Web services. Furthermore,* VIDRE *enables the definition of distributed business rules, a novel feature allowing a distributed execution of rules.*

1. Introduction

Web services attract a lot of interest by being one popular technology for implementing a service-oriented architecture. The service-oriented computing paradigm strives to solve common problems having a long tradition in software engineering, such as interoperability or dynamic invocation. Services constitute pieces of software with well-defined interfaces which are loosely-coupled and can be invoked dynamically. Thus, it is possible to use them easily within various applications. These technological advantages does not necessarily solve the problems from a business perspective.

By using Web services it is easily possible to capture parts of a business process (e.g., with WS-BPEL [15]) and implement them in terms of a service (e.g., billing or stock management). Nevertheless, it is hardly possible to capture the rules and policies that define how for example billing works without hard-coding them in source code of the service implementations. Such rules and policies are typically expressed in terms of business rules and executed by a so-called business rule engine. Combining a rule-based approach to support service development improves the quality of the implementation by clearly separating the rules from the implementation logic, thus, enabling service providers to react more quickly to changes in the requirements and as a consequence being more competitive.

In this paper, we present the VIDRE[1] approach (Vienna Distributed Rules Engine), a service-oriented and distributed rule engine, which combines rule-based techniques with the advantages of service-oriented computing to access business rules as services. VIDRE even removes the limitation of a centralized execution of business rules by introducing a distributed business rule execution. To accomplish interoperability among several heterogeneous rule engines, VIDRE uses RuleML [10], a current standardization effort for rule markup, as an interlingua for exchanging and representing business rules among several heterogeneous rule engines. Furthermore, VIDRE provides a lightweight plugin-based mechanism to allow arbitrary rule-engines to be used for the concrete execution of the rules.

This paper is organized as follows: Section 2 introduces some basic concepts of business rules and rule engines needed to understand the VIDRE approach. In Section 3 we briefly discuss the related work in this area. Section 4 depicts how typical supply chain management (SCM) processes are described and implemented in terms of business rules. Section 5 summarizes the design and implementation aspects of VIDRE. In Section 6, we demonstrate how we implemented and evaluated the VIDRE approach based on the case study from the supply-chain management domain presented in Section 4. In Section 7 we conclude this work

[1] A Web site with a demo is available at `http://www.vitalab.tuwien.ac.at/projects/vidre`

and highlight some future work.

2. Business Rules – Basic Concepts

The term "business rules" is frequently used, nevertheless it is interpreted quite differently due to the lack of a standard definition. The Business Rules Group [14] defines a business rule as a statement that defines or constrains some aspect of the business, and it is intended to assert business structure or to control or influence the behavior of the business. Several different rule types have emerged [13, 14] where the following three are the most important ones:

Derivation rules represent statements of knowledge that are derived from other knowledge by an inference or a mathematical calculation. Derivation rules enable capturing heuristic domain knowledge without storing the information explicit, hence it can be derived from existing or other derived information on demand. An example of a derivation rule is the following business rule: *If the turnover of a customer is more than 5000$ in the previous year, the customer is premium.*

Integrity rules represent assertions that must be satisfied in all evolving states. Integrity rules limit possible actions within well defined boundaries. Any crossing of such a boundary will violate a constraint and cause some corrective actions. An example of a integrity constraint applied by our computer retailing company is: *Online customers must be at least 18 years old.* Integrity rules can also be used for validation, for example to ensure that a form is filled out correctly.

Reaction rules cause constructive actions when a certain event occurs and/or when a certain condition is met. Such an action could be the manipulation of one or more business objects, the triggering of another business activity, the creation of an entry in the "to do" list of a business actor or any combination of such actions.

VIDRE supports all three rule types: Reaction rules are modeled in terms of derivation rules, where the consequence part of a rule invokes a user-defined function which executes the "reaction logic". Integrity rules are implemented in a similar way, thus, they signal an error if the integrity constraint is violated.

Rule Engine Components. A typical rule engine consists of several components: a *rule base* contains all the rules for execution. The *working memory* holds the data on which the rule engine operates. The *pattern matcher* decides which rules from the rule base apply, given the content of the working memory. An *inference engine* works in discrete cycles and is used to find out which rules should be activated in the current cycle (including the activated ones from previous cycles) by using the pattern matcher. All activated rules from the conflict set, which is ordered to form the *agenda* (the list whose right hand side will be executed). The process of ordering the agenda is called the *conflict resolution*. To complete a cycle, the first rule on the agenda is fired and the entire process is repeated.

Rule Markup Initiative. It aims to provide a standard rule language and an interoperability platform for integrating various business rule languages, inference systems, and knowledge representation paradigms. It has gained increasing momentum within the standards, academic, and industrial communities [10]. The RuleML language offers XML syntax for rules knowledge representation, interoperable among major commercial and non-commercial rules systems. A RuleML example document is shown in Listing 1. It represents an `Order` for a mainboard in our SCM example.

```
<Atom>
  <Rel>Order</Rel>
  <oid><Ind>1cala68</Ind></oid>
  <slot>
    <Ind>id</Ind>
    <Ind>785</Ind>
  </slot>
  <slot>
    <Ind>customerId</Ind>
    <Ind>154</Ind>
  </slot>
  <slot>
    <Ind>product</Ind>
    <Ind>Mainboard XY</Ind>
  </slot>
  <!-- more slots (e.g., quantity, etc) -->
</Atom>
```

Listing 1. RuleML Fact

We make extensive use of RuleML as rule exchange format among several heterogeneous business rule engines by using it similar to SOAP in the Web service domain. Every request and response to and from a rule engine at each service provider is encoded in RuleML, therefore, offering a sound interoperability for adding new rule engines to a VIDRE *Service Provider*.

3. Related Work

Different approaches that enable dynamic service composition by using business rule have been proposed (e.g., [7]). However, none of these approaches focuses on distributed business rules and exposing business rules as well-defined Web services.

In [1], an extension to the Jess rule engine to allow a distributed rule execution is presented. Their distributed inference mechanism uses a shared working memory (SWM) to store the facts and each rule engine uses this memory to access the facts. We do not use a shared memory, instead we propose to use meta-rules to handle the distribution. Furthermore, we use RuleML as interlingua and provide trans-

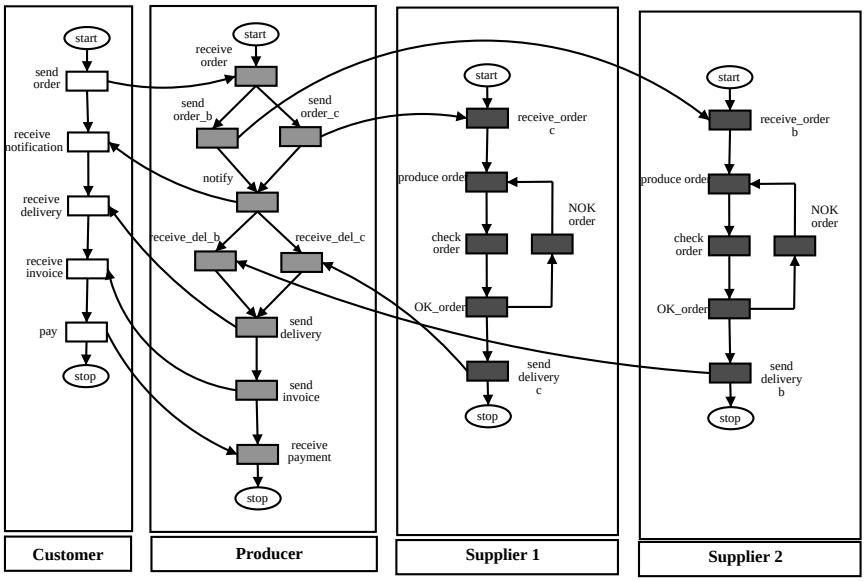

Figure 1. Supply Chain Example (from Chebbi et al. [2])

lators from RuleML to the corresponding rule engines, thus our approach is not limited to a concrete rule engine.

Schmidt [12] proposes a distributed rule execution approach by encoding RuleML rules in the SOAP header which are processed by different SOAP intermediaries. Based on some conditions in one of the business rules a different execution path can be chosen. The focus of this work is on rule execution but it is not discussed how distributed rules can be specified and executed.

In our previous work [8, 9] in this field, we presented a loosely-coupled integration of business rules into the well-known WS-BPEL [15] language, a standard for describing Web service business processes or so-called compositions. We used the interceptor concept [11] and an enterprise service bus (ESB) to apply business rules during the interception of the Web service invocations (e.g., WS-BPEL invoke activity) from a WS-BPEL engine. The WS-BPEL engine is connected to the ESB and all Web service invocation from go through a special Web service gateway which initializes the interception. In contrast to the approach presented in this paper, the work in [8, 9] focused on accessing different rule engines with the same API and generating Web service for it. In this paper we focus on using a common rule representation language and enabling a distributed rule execution by using meta rules.

A lot of efforts can be seen in the area of open source and commercial rule engine implementation. Drools [3] is a rule engine that uses an enhanced version of the Rete algorithm called the Rete-OO algorithm. Another very popular system for developing rule-based applications in Java is Jess (Java Expert System Shell) [6]. Jess has many unique features including backwards chaining and working memory queries, and can directly manipulate and reason about Java objects. We use Jess as one rule in our VIDRE implementation, the integration of others is planned.

4. Designing Applications with VIDRE

In the following, the process of assembling a build-to-order personal computer is used to demonstrate the concepts of business rules as valuable technology for combing business process management and service-oriented computing.

In Figure 1, an abstract supply chain management example is depicted involving a customer, a producer and two suppliers. For each participant, the internal process is depicted. Such a internal process is often referred to as orchestration. The lines between the participants represent the message exchanges between the different partners which is often referred to as choreography.

The process starts when the supplier receives an `order`, thus starting its internal process by issuing a `send order` to the producer. Proceeding in the producers process, the `send order_b` and `send order_c` are executed which place the orders at the suppliers respectively. The process proceeds until the order arrives at the customer and the payment is issued as a final activity.

One possibility when modeling the approach from Figure 1, is to use a service-oriented architecture where all the activities in the corresponding processes (orchestrations) are implemented in terms of Web services, which are itself composed to a higher level service with WS-BPEL resulting in the four composed services. In each of the four orches-

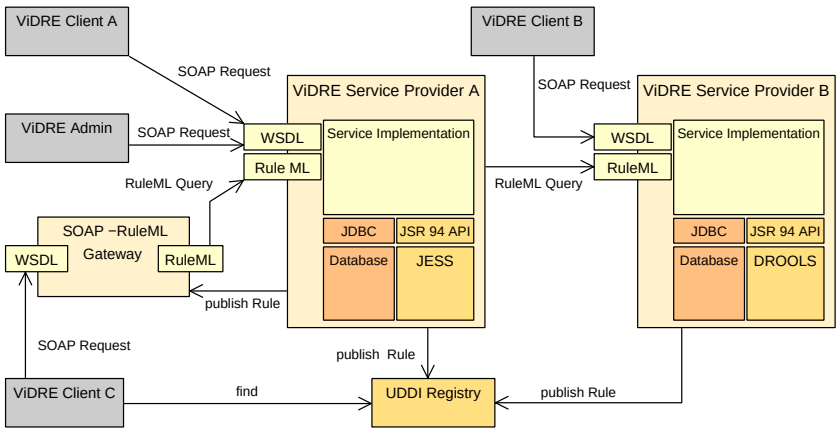

Figure 2. Architecture of VIDRE

trations, several Web service are involved to implement the business process.

In this paper we propose to use a business rule solution based on VIDRE. Thus, an activity can be implemented in terms of (distributed) business rules and then be exposed as Web services. This approach is especially powerful if a number of business rules are involved which would have to be wrapped into a service combined with other data gathered through other services. Consider the following example: In the producer orchestration, the preparation of the invoice involves several steps. Firstly, the processing of the order and data from the stock management. Secondly, billing and customer data is required to process the request. This process involves several business rules which may be physically distributed over several departments at the producers site. In such cases distributed business rules can ease the processing of such scenarios by assuming that each department runs its own business rule engine, thus managing its rules locally (e.g., billing, stock management, customer relationship). When a new order arrives at the producer the order is pre-processed and then a new `Order` fact is generated by the rule engine in the stock management. Based on the information in the `Order` fact and the information in the stock, orders at the suppliers may be necessary. For preparing the invoice two distributed rules have to be executed, one at the rule engine responsible for the customer relationship and one for the billing information of the customer to get the required data.

Besides the aforementioned intra-organizational usage of distributed business rules, they can be also applied to an inter-organizational environment. In our example this could be the case where the producer is placing new orders at the different suppliers, in case the parts are not in stock. In such cases, the producer and the supplier has to agree on a common data structure for an order. Dealing with possible data type heterogeneities is currently out of scope of our work.

Partner	Business Rule Service	Distributed Execution
Customer	Order preprocessing	no
Producer	Order processing	yes
	Invoice processing	yes
	Stock management	yes
Supplier 1&2	Order processing	no

Table 1. Business Rule Services

Table 1 summarizes activities in the SCM example which can be implemented by using business rules and then exposed as Web service to provide a more efficient and adaptive solution with respect to changes of the rules.

5. Design of VIDRE

VIDRE is a distributed infrastructure enabling clients to access rule engines in a service-oriented way to facilitate the reuse of business logic in business applications. The architecture of VIDRE is based on the JSR 94 Rule Engine API standard a Java community effort defining an API for rules engines in the Java world [5] to master the heterogeneity of the different rule engine vendors. The JSR 94 Java Rule Engine API specification standardizes a set of fundamental rule engine operations. This operations include parsing rule sets, adding objects to the inference process, firing rules and getting resultant objects from the engine. The main goal of the JSR 94 specification is to integrate different rule engines with a simple adapter into client applications without settling with one rule engine vendor.

The lack of standardization regarding the rule representation and query language makes it hard to provide interoperability with other applications and other business rule

engines. Therefore, VIDRE combines the JSR 94 standard with RuleML the emerging standard for knowledge representation. Combining these efforts gives us the ability to plugin different rule engines which support the JSR 94 API standard without changing the knowledge or rule base.

In Figure 2, the architectural approach of VIDRE is presented. The architecture depicts two distributed business rule service providers. Each business rule service provider offers a generic RuleML interface which accepts valid RuleML documents as input. These RuleML documents can contain RuleML queries and RuleML facts which will be used for the evaluation of the queries. Similar to SOAP requests, RuleML documents can be sent via any arbitrary transport channels such as HTTP, JMS or even SMTP.

Besides the generic RuleML interface each service provider publishes two WSDL interfaces for accessing the client runtime and administration interfaces as a Web service. The client interface enables the VIDRE clients to fire rules for a given business document and/or to issue queries encoded in RuleML. The administration interface is primarily designed for managing and maintaining rule sets.

On the client side the JSR 94 API is used as standard API to access the Web service interface. A client library implements the JSR 94 API and encapsulates the underlying implementation details to the programmer. Therefore, the location and the provider of the used business rule engine is transparent to the end-user. Even the rule representation language is hidden by the client libraries since the JSR 94 Runtime API supports only Java objects as input to the rule engine.

Another way to integrate business rules into a client application is to access them via the SOAP-RuleML gateway. The VIDRE implementation provides the ability to publish several rules and queries as Web service at runtime via the administration interface. These rules can be easily accessed by the client application by just using the corresponding WSDL file published on the SOAP-RuleML gateway or on a UDDI directory.

VIDRE Client A and B in Figure 2 use the WSDL interface to send a RuleML query directly to the business rule engine. The code necessary to transform the objects to RuleML and to invoke the service is hidden by the JSR 94 library implementation. This easy integration method is suitable for standalone clients. VIDRE Client C uses the UDDI registry to look up a specific business rule published by the service. Being successful, Client C uses the discovered WSDL file to access the rule engine through the SOAP-RuleML Gateway.

5.1. Service Provider Implementation

All incoming and outgoing requests to a VIDRE service provider are encoded in RuleML. VIDRE's prototype implementation provides transformers for Jess to RuleML and vice versa. Java objects are translated to RuleML with the Java reflection mechanism. These translators can be composed to achieve incremental translations. For example Jess to Drools translation can be easily achieved by composing the Jess to RuleML translator and the RuleML to Drools translator. The drawback of this approach is that we can only use a proper subset given by the least common denominator of the supported rule-engine capabilities. These constraint is induced by the different rule representation languages. Efforts, such as the RuleML initiative, to standardize the representation language are still ongoing. Changes to the vendors rule language can be easily introduced by simply writing a new translator or adopting the old ones.

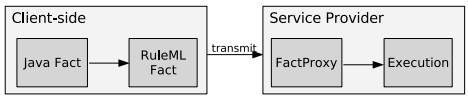

Figure 3. Adding a Java Fact

In Figure 3 the process of adding a new fact to one of the available VIDRE service providers is depicted. Whenever a client rule session is executed the objects contained in the session have to be transformed into an equivalent RuleML representation in order to transmit data to the service.

```
public class Order {
  private int id;
  private String customerId;
  private String product;
  private int quantity;
  // getter & setter methods
}
```

Listing 2. Java Fact

When adding a new fact, the first step is the transformation of a Java fact to a RuleML fact at the client-side. For example Listing 2 defines a class `Order` which represents an order in our SCAM example. VIDRE's client library transforms this class into an equivalent RuleML representation. The resulting RuleML document has been depicted in Listing 1 in Section 2.

On the server-side (see right part of Figure 3) a transient object (`FactProxy`) representation of the submitted RuleML is generated. These objects can be added to a server-side rule-session and act as input for the rule execution cycle.

Distributed Rules and Knowledge. One of the main attributes of the VIDRE approach is the ability to distribute

rules and facts. The distributed rule execution does not really differ from the normal, location bound, rules execution.

Once the rule execution cycle starts, the pattern matcher determines which rules are activated. The list of activated rules, together with any other rules activated in previous cycles, form the conflict set. The rule engine uses a conflict resolution strategy to select the next rule to fire. If no rule can be fired the rule execution cycle terminates, otherwise VIDRE's inference engine has to decide whether this rule is executed local or remotely. A description how this is implemented in VIDRE is given in the next section. For a normal location-bound rule the consequence part of the rule is executed and newly asserted facts are added to the working memory. These newly derived facts potentially activate or block rules in the conflict set, thus the execution cycle starts all over. If the activated rule is a distributed rule, so that the consequence of the rule should be executed at a remote service provider, VIDRE creates a client session and invokes the remote service provider. The result of the remote service invocation is added to the local working memory and again the rule execution cycle starts.

Distribution with Meta-Rules. Meta rules are distinguished from ordinary rules by their role which is to instrument the reasoning required to solve the problem, rather than to actually perform that reasoning [4]. Meta rules are often used in rule-based systems to encode preferences concerning the behavior of the inference process or they influence the conflict resolution strategy.

The question arises how meta rules can be used for distributed rule execution? We have to distinguish between to types of distributed rules. The first type is characterized in that the consequence part of the rule is remote. This means that the assertion of a derived fact or the enabled action should be performed on a remote machine. Here meta-rules play an important role. We have to encode that the consequence of the activated rule is a remote-rule and we have to encode the location where the action should take place.

The second type is characterized in that the premises contain remote-facts. This is the more complicated case. A remote premise is only reasonable in combination with backward chaining. In order to perform backward-chaining in Jess the remote fact has to be declared as backward-chaining reactive. With backward-chaining it is possible to incorporate data from outside the knowledge base, e.g., a relational database. Jess reasoning engine is strictly a forward chaining engine, backward chaining is simulated in terms of forward chaining rules.

6. Case Study Implementation and Evaluation

We have implemented a small supply-chain management case study from the computer manufacturing domain to demonstrate the feasibility of our approach.

Our setup consists of a customer, a computer retailer, two suppliers and one manufacturer. The computer retailer offers a Web interface to customers where computers can be customized and ordered based on the individual needs of each customer. The computer retailer itself relies on a number of suppliers (for reasons of simplicity in this case study we choose two suppliers) which itself may outsource their tasks to some manufacturers. Each participant needs to deploy the VIDRE service provider and the RuleML gateway to expose certain rules as Web services.

In Figure 4 the process of ordering a PC is depicted. The customer starts this process by ordering the PC via the Web form provided by the computer retailer. After the order is placed, the computer retailer has to place the corresponding orders of the hard disk, video card, mainboard, etc at the suppliers depending on the stock. After delivery of the various parts, the PC can be assembled and finally the customer is notified that the PC is assembled and ready for shipping.

We have modeled the processes in terms of business rules, by using a combination of local and distributed rules. For example the ordering of a mainboard at the supplier is expressed as a distributed business rule as depicted in Listing 3. The meta-fact `META_LOCATION` associates to each `OrderMainboard` object a URI. The URI encodes the target location where the consequence should be executed. The prototype implementation uses the prefix `META` for meta-information. Please note, that we use the Jess language to present the rules in this paper just for reason of simplicity. The Jess syntax is compact and easy to read whereas the rule editor does not get in contact with Jess at all (except of writing user defined functions). The definition of rules and meta-rules can be done with the VIDRE administration client.

```
(defrule OrderMainboard
  "On reception of a new computer order, order a
   mainboard remotely at the supplier"
  (OrderParts (orderId ?orderIdVar)
              (quantity ?quantityVar))
  =>
  (bind ?OrderMainboard (new OrderMainboard))
  (call ?OrderMainboard setQuantity ?quantityVar)
  (call ?OrderMainboard setOrderId ?orderIdVar)
  (definstance OrderMainboard ?OrderMainboard))

(assert (META_LOCATION (name "OrderMainboard")
  (uri "rules://madrid.vitalab.tuwien.ac.at/supplier")))

(defrule META_REMOTE_OrderMainboard
  "if a remote location is defined call defRemoteInstance"
  ?obj <- (OrderMainboard)
  (META_LOCATION (name "OrderMainboard") (uri ?uri))
  =>
  (defRemoteInstance ?obj ?uri))
```

Listing 3. Remote Supplier Rule

The next rule in Listing 4 depicts a rule which triggers the assembling of a PC at the computer retailer. This case study involves a number of business rules but due to space

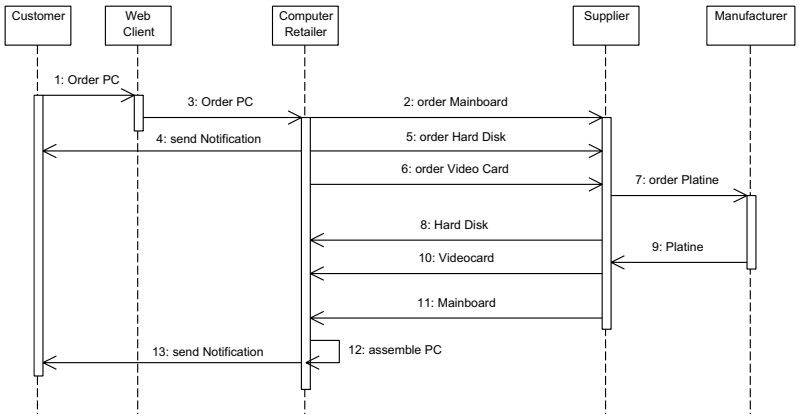

Figure 4. Sequence Diagram - Order

restrictions we do not depict all of them. Nevertheless, the structure of such rules is quite intuitive and the authoring process with our administration client is rather straightforward.

```
(defrule assembleComputer
   "If all individual parts are available,
    assemble Computer."
   (Mainboard (orderId ?orderIdVar) (qty ?quantityVar))
   (HardDisk (orderId ?orderIdVar) (qty ?quantityVar))
   (VideoCard (orderId ?orderIdVar) (qty ?quantityVar))
   =>
   (bind ?assembleComputer (new assembleComputer))
   (call ?assembleComputer setOrderId ?orderIdVar)
   (call ?assembleComputer setQuantity ?quantityVar)
   (definstance assembleComputer ?assembleComputer))
```

Listing 4. Assemble Computer Rule

6.1. Analysis

For comparing the VIDRE approach with state of the art architecture, we have implemented the same functionality in a SOA based on Web services. For the sake of simplicity, each supply chain participant is implemented as single Web service. The business rules for stock management, order processing, etc. within these Web services are "hard coded". In this section we want to analyze and compare the VIDRE approach with the "conventional" SOA solution with respect to the following well-known software quality attributes.

Maintainability. The business rules approach uses declarative programming techniques to allow separation between business data and business logic. Business rules describe what to do, not how to do it, thus making it easier to manage and maintain complex business processes. On our project Web page, we depict some demonstration videos demonstrating how we easily create and maintain rule sets with the VIDRE's administration client. For example, the customer policies of our supply chain can be edited within the administration client and can be redeployed without changing a line of source code. Also the runtime interface is accessible within VIDRE's administration client, hence changes to rule sets can be tested immediately.

Extensibility. For extensibility the same properties apply as for maintainability. Extensions can be deployed without stopping the service provider. In large production systems the number of business policies can reach up to thousands of business rules. Extensions to these systems pose though challenges to developers. An appraisal of side effects is very difficult. In rule based systems the new business policies can be added to the rule set with little concerns about side effects. The rule engine is responsible to activate and fire the rules. Conflicting rules are resolved by the conflict resolution strategy of the rule engine. In our supply chain context a new policy for inventory management can be easily added with VIDRE's administration client without considering conflicts with existing rules.

Reusability. Business rules make the business logic explicit. Conventional implementations mix up the business logic with the code necessary to execute the logic. Hence, the reuse of business logic is very hard to accomplish and is often not very efficient. Usually only components and technical assets, like logging and tracing are reused in computer science. With business rules we can address the reuse of business logic. For example, stock policies or registration validation rules can be reused within intra or even interorganizational applications.

Scalability. Scalability is not the main focus of the VIDRE approach but is mentioned here because scalability is an important issue to large scale distributed systems. The distribution of rules gives us the ability to distribute the load

on different machines. Here meta rules can be used to specify alternate service providers with the same rule set registered. The system can grow smoothly and economically as long as the entry point of the clients does not become an bottleneck.

Performance. An implementation of business process in terms of business rules induce a paradigm shift, therefore is important to analyze the timing behavior of our approach compared to our conventional implementation. Our results have shown that the performance does not differ significantly from the conventional implementation. The entry point of our application is in both implementations the order Web page. Both implementations start the order workflow by invoking the servlet of the Web application. From that point the implementation start to diverge. The VIDRE implementation creates a new rule session and initializes the session with an order fact. Behind the scenes this fact is encoded in RuleML. When the rule session is executed, the RuleML document is packaged in a SOAP envelope and is sent to VIDRE's service provider endpoint. The conventional implementation invokes the order service directly using the published WSDL files. In our case study implementation the overhead of encoding the document in RuleML does not carry authority. In a real world application this overhead can cost some performance. In the end, the whole discussion about performance boils down to the overhead of encoding and decoding RuleML documents on the client and server side. This is very similar to the SOAP performance discussion in the Web services domain.

Compared to this little trade-off in performance, the plus gained through the higher maintainability and reusability increases the overall benefit of the VIDRE based solution in case many business rules need to be managed.

7. Conclusions

In this paper, we have presented VIDRE a distributed business rule engine with several novel features: Firstly, it combines the emerging paradigm of service-oriented computing with rule-based techniques to enable the execution of rules in a service-oriented fashion. Secondly, it uses RuleML as an interlingua to represent facts and rules, with transformers to concrete business rules engines (such as Jess in our implementation). Thirdly, we introduce distributed business rules as a means to distribute knowledge among several service providers which participate in the execution. Furthermore, we applied VIDRE in the SCM domain, where we implemented a computer manufacturing case study to demonstrate the feasibility of our approach.

In future work, we will focus on implementing security mechanisms to ensure privacy and data integrity of the knowledge exchanged in a distributed execution, as well as on an improved RuleML support.

References

[1] F. Cabitza, M. Sarini, and B. D. Seno. Djess - a context-sharing middleware to deploy distributed inference systems in pervasive computing domains. In *Proceedings of the International Conference on Pervasive Services (ICPS'05)*, pages 229–238, July 2005.

[2] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data & Knowledge Engineering*, 56, 2006.

[3] Drools. Java Rule Engine, 2005. `http://drools.org/` (accessed March 2006).

[4] P. Jackson. *Introduction to Expert Systems, 3rd Edn.* Harlow, England: Addison Wesley Longman., Boston, MA, USA, 1999.

[5] Java Community Process. JSR 94 Java Rule Engine API, 2005. `http://www.jcp.org/en/jsr/detail?id=94`.

[6] JESS. Java rule engine, 2005. `http://herzberg.ca.sandia.gov/jess/` (accessed December 2005).

[7] 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.

[8] 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'05)*, 2005.

[9] 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.

[10] RuleML. The Rule Markup Initiative, 2005. `http://www.ruleml.org/` (accessed March 2006).

[11] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*. John Wiley & Sons, 1 edition, 2000.

[12] R. Schmidt. Web services based execution of business rules. In *International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, 14 June 2002, Sardinia (Italy)*, 2002.

[13] K. Taveter and G. Wagner. Agent-Oriented Enterprise Modeling Based on Business Rules. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER'01)*, pages 527–540, 2001.

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

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