Knowledge Sharing in a Collaborative Business Environment*

Seema Degwekar
Database Systems R&D Center
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, Florida, USA
1-352-846-3422
spd@cise.ufl.edu

Stanley Y. W. Su
Database Systems R&D Center
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, Florida, USA
1-352-392-2693
su@cise.ufl.edu

Abstract

Recent efforts focus on representing human/organizational knowledge in a format suitable for machine processing to enable systematic knowledge sharing among collaborating business organizations. Different types of business rules can be used to capture multi-faceted business policies, strategies, regulations and constraints. A structure of such rules can model business procedures and processes. Interoperation among distributed, heterogeneous rules and rule structures is needed to support collaborative business decision-making and problem solving. In this paper, we introduce a distributed event-trigger-rule-based system for processing distributed events, triggers, heterogeneous business rules and rule structures. Our approach converts heterogeneous business rules and rule structures into code automatically and wraps them as web services for their registration, discovery, invocation, interoperation and reuse in an enhanced web service infrastructure.

Keywords: Business knowledge representation and sharing, rule language, web services, event and rule processing, decision support

1. Introduction

In order for business organizations to compete in the global economy and to tackle complex problems in supply chain management, product design, planning and manufacturing, business negotiation, etc., these organizations must collaborate and share, not only their data resources, but also business knowledge expressed in their policies, constraints, regulations, processes and procedures. To allow organizations in a *collaboration federation* to have full access to one another's data and knowledge resources may not be feasible due to security and privacy reasons and the autonomous nature of these organizations. Instead, facilities should be provided for such organizations to publish/export only those data and business knowledge specifications that they are willing to share. Also, at any given point in time, an organization is usually not interested in getting all the shareable data of other collaborating organizations, but only in those data that are associated with the occurrence of some event (e.g., a delay of the delivery date of a product, a slip in a production schedule, a withdrawal of a product order, or a machine malfunction on the manufacturing floor) as well as those data that are generated by the execution of relevant knowledge rules triggered by the event occurrence. In this paper, we shall focus on this *event-triggered* data and knowledge sharing among collaborating business organizations.

* The research reported in this paper is supported by the National Science Foundation under Grant No. IIS-0534065

Knowledge can be broadly classified as tacit or explicit (Nonaka and Takeuchi, 1995). In this work, we are interested in capturing and representing the multi-faceted business knowledge expressed in an organization's policies, constraints, regulations, procedures and processes. Such knowledge can be specified in the following three popular types of *rules* (Loucopoulos and Katsouli, 1992; Sowa, 2000) used in existing rule-based systems (Business Rules Group, 2000; Rouvellou et al., 2000). First, condition-action or event-condition-action (ECA) rules (Widom and Ceri, 1996) are the most common type of rules used in event–based systems (Buchmann et al., 2004; Carzaniga et al., 2000) and production rule systems (Riley, 2006; Brownston et al., 1985). Second, logic rules are commonly used in expert systems for decision-support (Ullman, 1988). In addition, constraint rules are used to enforce data integrity in most database systems (Ullman, 1982). Rules of the above three types can be *structured* (hereafter referred to as *rule structure*) to model an organizational or inter-organizational process or procedure. Expressing organizational policies, regulations, processes etc. as high-level declarative rule and rule structure specifications is more desirable than implementing them in agent code or application code because they are easier to understand by members of collaborating organizations and also easier to modify to reflect the changes in policy, regulations, processes, etc. These high-level rules and rule structures can be translated into code automatically for their processing.

An event is anything of interest to collaborating organizations that occurs at a particular point of time. These organizations should be able to define events of common interest, subscribe to selected events and receive notifications when the subscribed events occur. They should also be able to define business rules and rule structures as a way to publish their shareable policies, constraints, processes, etc. These rules and rule structures defined at various sites should be automatically processed if the data associated with event occurrences (i.e., *event data*) can be used as input for processing these rules and rule structures to generate new data or to update the event data; thus, producing a new version of event data to be sent to those organizations that contain applicable rules. This process of event data transmission and rule processing should continue until no rules and rule structures are applicable to the last version of event data. At this time, all collaborating organizations would have processed all the applicable rules and rule structures and received all the data that are relevant to the event occurrence. In order to achieve the above distributed, event-triggered data sharing and rule processing, we will need to have, among other facilities like user interfaces, 1) a rule specification language for specifying different types of rules and rule structures, 2) a mechanism to achieve the interoperation of heterogeneous, distributed rules and rule structures, and 3) an infrastructure for managing and processing distributed events, transmitting and merging event data, and triggering the processing of rules and rule structures in a uniform manner. They form the focus of this paper.

To achieve the interoperability of heterogeneous business rules, one could have used three types of rule engines to process the three different types of rules and find some way to make the engines themselves interoperable. We believe this will result in a very complex and inefficient system. Another approach taken in (Bassiliades et al., 2000; Rosenberg and Dustdar, 2005) is to choose one knowledge representation (e.g., event-condition-action rule) and convert the other two types of rules into the chosen one so that they can all be processed by a single rule engine. One problem with this approach is that the semantics of these rule types are quite different. It is not always possible to convert one type of rule into another without some loss of meaning. Also, most existing rule engines *interpret* rules at runtime resulting in an inefficient, unscalable,

centralized rule processing system. In our work, we use a *compilation* approach by translating different types of rules and rule structures at the rule definition time into code and wrapping these code as web services for their uniform registration, discovery and invocation in a web service infrastructure. Instead of rule translation, we make them interoperable; i.e., the result of a rule execution can be used as input to another rule or rule structure.

The intended contributions of this paper are:

1. To introduce an XML-based rule specification language for the specification and exchange of multi-faceted business knowledge. It has more expressive power than existing rule markup languages because it can be used to specify different types of rules and rule structures.
2. To present an approach of processing heterogeneous rules and rule structures in a distributed fashion without the use of different types of rule engines,
3. To present an enhanced web service infrastructure and a distributed event-trigger-rule-based system used for processing dynamic event data and multi-faceted knowledge in a collaborative business environment. The rule base introduced by the Business Rules Group (Business Rules Group, 2000) is used to demonstrate the utility of the rule language, processing strategy, infrastructure and system.

We acknowledge some existing systems and approaches that are related to our work in three areas: rule markup languages, event-and-rule based systems, and rule interoperability. Several recent efforts like SRML (Cover, 2001), BRML (Cover, 2002) and RuleML (The Rule Markup Initiative, 2000) are concerned with developing a rule markup language for business applications. Of these, SRML and BRML address only condition-action and derivation rules, respectively. RuleML is an ongoing effort that aims to include all three types of rules. However, the language has not been finalized. Event-and-rule based systems presented in (Buchmann et al., 2004; Krishnamurthy and Rosenblum, 1995) couple event notification with condition-action rules alone. There are also many so-called active database systems, which use only condition-action rules as surveyed in (Widom and Ceri, 1996). E-DEVICE (Bassiliades et al., 2000) proposes an active knowledge based system to support the processing of all three rule types in an active OODB system by mapping derivation rules and integrity constraints into condition-action rules. However, the system has no support for integrity constraints yet. The system presented in (Rosenberg and Dustdar, 2005) provides a web service interface to heterogeneous rule engines, thereby providing a uniform API to access each engine. Rule execution is carried out by individual engines interpretively. Support for rule structures is lacking and it is also not clear how one rule engine can make use of the results generated by another.

The rest of this paper is organized as follows. Section 2 explains the three types of rules and rule structure. Section 3 presents the architecture of our implemented event-trigger-rule-based system and the strategy used to decompose and process a rule structure that contains different types of rules in a web service infrastructure. Section 4 summarizes the paper.

2. Business Rule, Rule Structure and Trigger

Business rules and rule structures specified in an XML format are used for two purposes: translation to the corresponding web services for rule processing, and transmission among collaborating organizations for knowledge exchange. We have developed an XML-based rule

specification language but we cannot present the details here due to space limitations. We shall describe the features of the three rule types and the rule structure. Interested readers can refer to our website (Rule Specification Language Schemas, 2005) for the complete XML schema.

2.1 Integrity Constraints

An *integrity constraint* specified in a database protects it from entering into an inconsistent state. In the context of this paper, it is used to verify the consistency of event data. For constraint specification, we adopt some syntactic constructs of our earlier work on a Constraint Specification Language (Su et al., 2001), which was patterned after the Object Constraint Language (Warmer and Kleppe, 1998). Since event data uses an object-oriented data model, constraints can be specified on an object or on one or more of its *attributes*. Constraints can thus be classified into two general types: *attribute* constraint and *inter-attribute* constraint. An attribute constraint specifies the list of allowable values for a particular attribute (e.g., *a >= 50*). An inter-attribute constraint specifies the relationship between two or more attributes. This relationship can be mathematical (e.g., $a + b = c$) yielding a so-called *formula constraint*, or conditional (*if a>10 then b=50*) yielding a so-called *conditional constraint*.

2.2 Logic-Based Derivation Rule

Logic-based *derivation rules* (Ullman, 1988), also known as *inference rules* or *deductive rules*, assess a given premise to come to some conclusion. Both the premise as well as the conclusion can be complex Boolean expressions linked with logical operators *AND* or *OR*. For clearer interpretation of the conclusion, we restrict the logical operator to be *AND*. If *OR* semantics is desired, the rule *r* can be broken up into multiple rules $r_1, r_2, \ldots r_m$ where *m* is the number of OR predicates in the expression. Each of these new rules has the same premise as the original and the corresponding OR predicate as the conclusion.

2.3 Action-oriented Rule

Action-oriented rules are typically found in active database systems (Widom and Ceri, 1996). They are known as event-condition-action (ECA) rules or just event-action rules (Krishnamurthy and Rosenblum, 1995). When an event specified by the event clause occurs, the ECA rule checks for the truth value of the condition clause, and executes the action clause if the condition expression is true. Since we allow events and rules to be defined and published by different organizations, we separate the event (E) part from the condition-action (CA) part. Also, an alternative action may need to be executed when the condition evaluates to false. We use a guarded condition expression, which has an optional sequence of guard expressions followed by a main expression. If any one of the guard expressions is evaluated to false, the entire rule is skipped (i.e., it becomes inapplicable). Otherwise, the main expression is evaluated to determine if either the action clause or the alternative action clause should be processed.

2.4 Rule Structure

When a specific event occurs, an organization may have different types of rules that need to be executed in a specific order to carry out a procedure or process. It is very natural to model such a procedure or process by specifying the structural relationships between individual rules. We capture these relationships in a rule structure introduced in our earlier work (Lee et al., 2001).

In a rule structure, a rule *r* may be required to be executed before another rule *s*. Typically, rule *r* generates data that can be used by rule *s*, thus establishing a direct *link* between *r* and *s*. Similarly, a rule *r* may be required to be executed before the execution of multiple rules $s_1, s_2, \ldots s_m$, $m>1$. In this case, rule *r* may generate data that can be used by rules $s_1, s_2, \ldots s_m$ and thus rule *r* and rules s_i ($1 \leq i \leq m$) are connected in a *split* construct. A rule *s* may be required to wait for all of a given set of rules $r_1, r_2, \ldots r_n$, $n>1$ to finish before it can start its own execution. In this case, $r_1, r_2, \ldots r_n$ are connected to *s* in an *and-join* construct, and the data generated by rules r_i ($1 \leq i \leq n$) can be used by rule *s*. Finally, *s* may be required to wait for, either all or a subset of rules $r_1, r_2, \ldots r_n$, $n>1$ to finish execution. This establishes an *or-join* relationship between $r_1, r_2, \ldots r_n$ and *s*. In each type of relationship, the rule(s) that govern(s) the execution of other rule(s) is called a predecessor(s), and the rule(s) that execute(s) after the predecessor(s) is called a successor(s). *A rule structure can now be defined as a directed graph with different types of rules as nodes, which are connected by link, split, and-join, and or-join constructs*.

2.5 Trigger

A trigger specifies a number of alternative events that will trigger the processing of a single rule or a structure of rules. Just like events and rules, triggers can be explicitly defined by collaborating organizations that are different from those that defined events and rules. In this case, the organization that contains the rule or rule structure becomes an explicit subscriber of the event. Triggers can also be automatically generated by the system to link a distributed event to a distributed rule or rule structure, if the event data (or part of it) can provide the input data needed for processing the rule or rule structure. In this case, the rule or rule structure is said to be applicable to the event. The site that contains the applicable rule or rule structure will automatically become an *implicit* subscriber of the event.

We stress the importance of allowing collaborating organizations to independently define events, rules, rule structures and triggers in a distributed, collaborative environment because it gives the flexibility and power to express and enforce policies, regulations, constraints, procedures and processes that are important to achieve inter-organizational sharing and collaboration.

A rule of any one of the above three types is a high-level specification of a granule of control and logic that operates on some data entities associated with an event(s). When different types of rules are translated into corresponding program code, these programs become the uniform representation of the semantics of these heterogeneous rules. If we wrap these generated programs as web services, they can be processed in a web service infrastructure over the Internet. Similarly, if we also translate rule structures into programs and wrap them as composite web services, which will invoke their component services (i.e. the rules that form the rule structures), then we can also achieve the interoperation of these rule structures. Our approach thus achieves the interoperation of heterogeneous business rules and rule structures at the program or code level and avoids the need to either use different types of rule engines to interpret the different types of rules and rule structures at runtime, or to convert all types of rules into one rule language with the risk of losing some semantics.

Due to space limitations, we cannot fully discuss the implemented algorithms for converting different types of rules and rule structures into web services. We outline the general process here. From the rule specification in our XML-based rule language, we generate two files: an *interface*

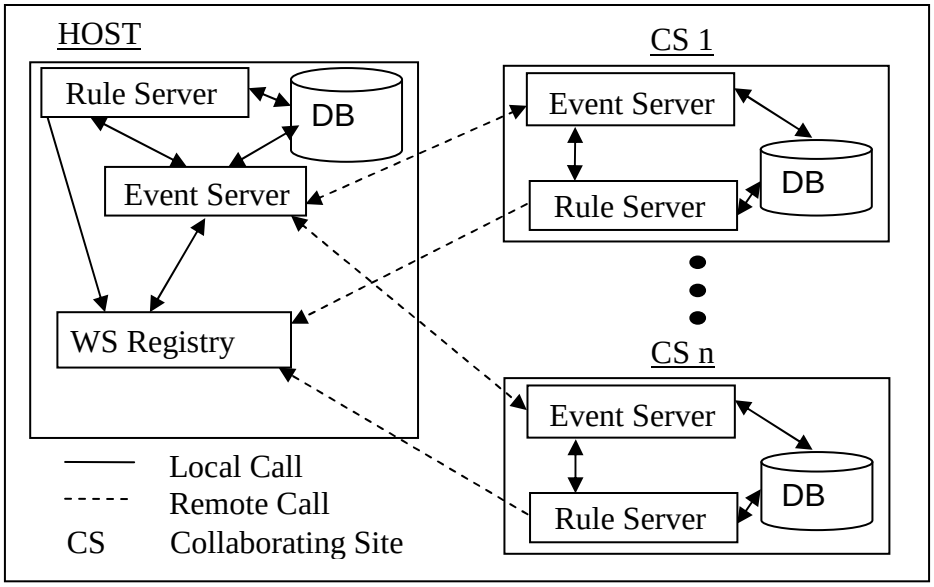

Figure 1. System Architecture

or *header* file and an *implementation* or *source* file. Each rule is represented as a web service with one *operation*, the characteristics of which depend on the rule type. A web service operation translates to a *method* in a programming language. The specification of input and output parameters of the operation is represented by the method's *signature*. The interface file contains this signature information, which is then used to create the corresponding WSDL document. The implementation file contains the actual program code for the rule. A WSDL description for the web service is also generated. This is used to publish the web service in a UDDI registry, accessible only to the collaborating organizations.

3. System Architecture and Rule Processing

3.1 System Architecture

The distributed event-and-rule-based system has a peer-to-peer server architecture shown in Figure 1. All organizations have identical subsystems installed at their sites. Each collaborating site creates and manages its own events, rules, rules structures and triggers, but their specifications are registered/published at the host site of a collaboration federation. The host maintains a repository of these specifications. The *rule server* component at each site stores and manages the web services generated for the rules and rule structures defined at that site. These web services are registered at the web service registry (*WSRegistry*) of the host site. In this work, we make use of the web service registry reported in (Degwekar et al., 2004). Different from other existing registries, our registry has constraint registration and processing capabilities. The *event server* component is responsible for storing information about events defined at that particular site and the information about event subscribers. A collaborating site can specify a trigger linking a distributed event to a rule or rule structure, thus becoming an *explicit subscriber* of that event. This information is stored by the event server in a local database. Triggers can be automatically and dynamically generated by the system if the event data schema associated with an event occurrence is a superset of the input data schema of a rule or rule structure. In this case, the site that has the rule or rule structure becomes an *implicit subscriber* of the event. Both explicit and implicit subscribers will be notified upon an occurrence of the event. Distributed rules and rule structures are invoked and processed by replicas of the rule server. An event server

at any site can serve as the coordinator for a particular knowledge sharing session initiated by an event occurrence at that site. It handles the aggregation of the dynamic event data sets associated with the event occurrence.

We have implemented the algorithms for converting knowledge rules to web services in Java, with the Sun Java System Application Server Platform Edition 8.1 2005 Q1 as our application server. The event and rule servers have been implemented using the Enterprise JavaBeans 2.1 framework. To facilitate easy and efficient lookup, we publish the deployed web service to our private UDDI registry using the UDDI4J API. This registry makes use of the Apache jUDDI project to communicate with a MySQL database that stores the web service information. The MySQL database is also used by the event server to store the event and trigger information. We use a private registry instead of a publicly available UDDI-based registry for two reasons. By eliminating clutter typically found in a business registry, we speed-up registry look up. Also, a private registry provides some level of security, as it is available only to the organizations participating in a collaboration federation.

3.2 Event-triggered Processing of Rules

We now use an example scenario to describe the event-triggered processing of distributed rules and rule structures, and explain how a rule structure is decomposed and processed.

3.2.1 Distributed Rule and Rule Structure Processing

In their first white paper (Business Rules Group, 2000), the Business Rules Group has provided a set of business rules for the operation of a fictitious car rental company named *EU-Rent*, which has 1000 branches in towns in different countries. Each branch may wish to share some data and knowledge specifications with the other branches to achieve better management of the company as a whole. Thus, each branch is a collaborating site in the collaboration federation of the EU-Rent company. Although this is not a real-world inter-organizational setting, it suffices to serve our purpose due to the following reasons. First, the rule set has been independently constructed and published for academic use by a well-known group. Second, each of the rules in the rule set belongs to one of the three different rule types processed by our system. Third, as can be seen from the rule set, managing the activities of multiple branches requires the interoperation of different rule types captured in rule structures.

We have used our rule specification language to define EU-Rent's rules and rule structures, and convert and register them as web services. There are a total of 46 rules, of which 29 are CAA rules, 13 are integrity constraint rules and 4 are derivation rules. There are 9 rule structures discernible from the rule set. Each of these rules and rule structures took about 6-7 seconds to be converted, compiled, deployed and published as a web service on a Windows XP machine with an Intel Pentium 4 processor and 1 GB RAM. The major component of the total time required for web service creation is in the compilation, deployment, and publication activities.

To demonstrate the event-triggered processing of distributed rules and rule structures, we use the scenario depicted in Figure 2. A customer approaches a local branch *Branch1* with the request for a car rental. *Branch1* is unable to satisfy his/her request. It posts an event *RemoteRentalRequest* and the event data in XML format (*ED1*) is sent to all subscribing branches (denoted as step 1 in the figure). The event data contains the branch identifier (*bID*), the

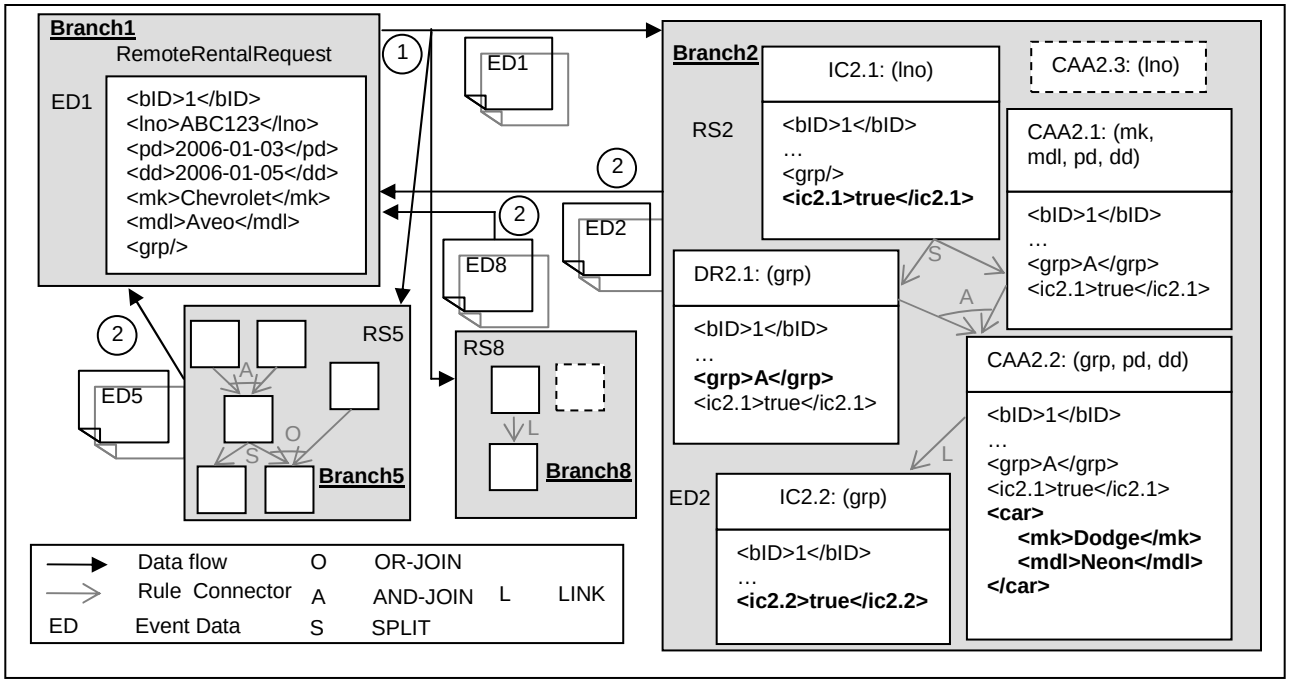

Figure 2. Event-triggered processing of distributed rules and rule structures in a collaboration federation

customer license number (*lno*), the pickup date (*pd*) and dropoff date (*dd*), the make (*mk*), the model (*mdl*) and the group or class (*grp*) of the car requested. Assume that *Branch2*, *Branch5* and *Branch8* are branches that have applicable rules. Each branch has a local set of rules that need to be processed when the *RemoteRentalRequest* event occurs. Let us consider in some detail the rules to be executed at *Branch2*, which has defined a trigger linking the event to the rule structure *RS2*, but no trigger that links the event to the applicable rule *CAA2.3*.

CAA2.3 is a rule that determines if a customer is eligible for a loyalty incentive scheme at the branch. If the customer has made four or more rental reservations at that branch, he/she is eligible and can receive free rentals or upgrades. This rule is invoked (shown by a dotted line in the figure) by the rule server because the rule input data is a subset of the event data and a trigger is automatically generated by the system to link the event to the rule. The capability to invoke applicable rules even though no triggers are explicitly specified by users is a very important feature of the system because some collaborating organization may publish only shareable rules and rule structures without specifying any trigger.

Now let us consider the rule structure *RS2*. A customer may be blacklisted due to earlier problems with payment and returns. The branch must not serve such a customer, so the first rule in *RS2* checks that the customer is not blacklisted by means of the integrity constraint rule *IC2.1*. This constraint rule checks if the customer license number is not in the list of blacklisted customers. The result of this check is written to the event data file. If the constraint is not satisfied, the rule server stops the execution of the rule structure. If it is, the derivation rule *DR2.1* and condition-action-alternative_action rule *CAA2.1* are executed. If no group is specified for the car requested, *DR2.1* assigns the default group *A* to the request. If the make and the model have been specified for the car request, *CAA2.1* checks if *Branch2* has an available car with the same make and model. Once both these rules have been executed, rule *CAA2.2* is invoked. If

CAA2.1 cannot find an available car, *CAA2.2* checks if there is an available car in the specified group, which could be the group in the event data or the default group assigned by *DR2.1*. After *CAA2.2* has finished execution, the integrity constraint rule *IC2.2* is invoked to check to make sure that the quota for the group has not been exceeded. At each stage, new data added to the event data file are shown in bold font in the figure.

From the above scenario, we see that an event occurrence can lead to the processing of individual rules as well as rule structures. The data generated by rule *CAA2.3* and the rule structure *RS2* are added to the event data file. Thus, the event data file now contains new data resulting from the application of the local rules at *Branch2*. This data is returned to *Branch1*. *Branch1* receives the updated event data from *Branch2* (*ED2*), *Branch5* (*ED5*), and *Branch8* (*ED8*) (denoted as step 2 in the figure). It then merges this data and sends out the new version to all branches (not shown in the figure) that have rules and rules structures applicable to the new version. Thus a second round of event data distribution is initiated. The process will continue until no rules and rule structures are applicable to the last version of event data. At this time, all involved branches would have received the final version of the event data, which can be used for their local decision-support and problem solving. The implemented prototype system runs on multiple computers. The above scenario takes a total of 2 seconds for the requesting branch to send out its event data file to remote branches and for the remote branches to invoke the applicable rules and send the updated event data file back to the requesting branch.

3.2.2 Decomposition of Rule Structure

We use Figure 2 to present the technique used to decompose a rule structure into substructures for processing. A rule structure is a directed *graph*, in which rules (nodes) can be interconnected by *link*, *split*, *and-join* and *or-join* constructs (edges). An XML document however, organizes its constituent elements in a *tree* structure. Each of the four structural constructs, when considered independently, is a tree, but the combination of these constructs that forms a particular rule structure may not always be a tree, as can be seen from rule structures *RS2* and *RS5* in Figure 2. It is worth noting that if we break a rule structure into substructures, where each represents a single structural relationship between two or more rules, each substructure is a tree, and can now be represented using XML.

For a given rule structure, substructures are generated as we move from the top to the bottom of the graph. At each level, we follow a left-to-right order to generate substructures. Taking *RS2* of Figure 2 as an example, the first substructure generated would be the *split* construct with *IC2.1* as the predecessor rule and *DR2.1*, *CAA2.1* as the successor rules. The second substructure would be the *and-join* with *DR2.1* and *CAA2.1* as the predecessor rules and *CAA2.2* as the successor rule. The third substructure would be the *link* with *CAA2.2* as the predecessor rule and *IC2.2* as the successor rule.

Decomposing a complicated rule structure into simpler substructures also facilitates its maintenance. Inserting a new relationship into an existing rule structure document requires only creating the appropriate substructure and inserting it into the correct position in the document. Similarly, deleting or modifying a structural relationship needs to address only the substructure that represents that relationship, without affecting other parts of the rule structure document.

3.3 Discussion

In this section, we consider some issues related to our system and highlight the approaches we have taken or are taking to resolve them. Some future tasks are also identified.

Distributed rules may form a cycle that causes a non-termination problem. Also, they may be inconsistent or conflicting. One possible approach is to analyze the published rules for contradictions, inconsistencies, and cyclic conditions (Baralis et al, 1998). However, due to the dynamic and independent nature of the rules in a collaboration federation, this strategy does not seem to be suitable. We have investigated both cycle avoidance and cycle detection and resolution strategies. So far, the most promising approach is the following. Each rule server sends certain characteristics of the rule it executes back to the coordinator. These characteristics include the data items that form the rule input, the data items that form the rule output, as well as the data characteristics that indicate when the rule condition is satisfied. At the end of every round of event data processing, the event coordinator determines if starting the next round of processing will result in a rule cycle to be initiated. If this is the case, the rule, which will cause the cycle will not be executed, but the remainder of the rule processing is carried out normally.

Inconsistencies and conflicts during rule execution in a collaboration federation may reflect differing expert opinions. However, there may be some rules to resolve these inconsistencies and conflicts. When an event coordinator determines that a particular data item has two or more values given by different sites, it checks with the host site to determine if there is a global resolution policy in place for that data item. If so, this policy is applied and the resolved value of the data item is used (e.g., by taking the average or minimum or maximum value). If not, all of the values are tagged with their site identifiers and are sent to all sites that contain applicable rules. Each site may have a local resolution policy in place to determine the resolved value. This resolved value is then used by any applicable global or local rules.

In a collaboration federation, since events, rules, and triggers can be defined by different collaborating organizations, it is very likely that there will be discrepancies in the terminologies used. People searching for registered events and web services need some form of common ontology to resolve these discrepancies. We are investigating the use of the OWL language to define an ontology for a particular application domain in a collaboration federation. Thus, the host site in the system architecture will also incorporate an ontology database to map the terms used in event, rule and trigger specifications to concepts and concept associations defined in the domain ontology. The site will also use an ontology manager to resolve discrepancies and identify similarities between the specified terms, and to facilitate search.

The notion of transaction is an important research issue. Each time an event occurs, multiple rounds of event data transmission and distributed rule processing can take place. They should be treated as a single transaction. In distributed databases, the focus is on maintaining data integrity, and hence a transaction commits only if all of its sub-transactions commit. In a collaborative e-business environment, the focus is on sharing as much knowledge associated with an event as possible. It may not be desirable to stop the rule processing if one site aborts because other sites may still produce useful data. The traditional ACID properties of transaction need to be re-examined and defined for an event-triggered, knowledge-sharing system like ours. Another important issue to be addressed is the specification and enforcement of trust and security in a

collaborative e-business environment. This issue is out of the scope of this paper. In our previous work (Yang et al., 2002; Yang et al., 2005), we have introduced a trust agreement specification language and a trust-based security model for establishing inter-organizational security policies that govern the interaction, coordination, collaboration, and resource sharing of collaborating organizations. Interested readers are encouraged to look up these references.

The system architecture allows organizations to join a collaboration federation by installing the developed software and tools at their network sites. It is highly expandable and scalable because more computational power can be added as more shareable knowledge and more collaborating organizations are added to the federation. Also, since the system components are implemented as servers, multiple collaboration federations can be accommodated: i.e., an organization can be a member of multiple federations and its servers can process different event data sets concurrently in multiple threads. However, as the number of organizations in a federation increases, the number of federations grows, and the number of event occurrences of different event types increases, the performance of the entire network can deteriorate with a centralized host. For scalability, event types may have to be categorized and managed by multiple hosts.

4. Summary

In this paper, we presented our idea of managing dynamic event data and sharing multi-faceted knowledge among organizations that form a collaboration federation. Different aspects of knowledge are specified in different types of rules, rule structures, and triggers, which link events of interest to distributed rules and rule structures. The interoperation of distributed events, rules, rule structures and triggers for supporting decision-making is the main concern of this research. The approach taken for achieving their interoperation is to translate rules and rule structures into code and wrap them as web services for their discovery, invocation and interoperation in a web service infrastructure in a uniform manner. Event data are transmitted to collaborating sites that contain applicable rules and rule structures. The architecture of an event-trigger-rule-based system implementing the above ideas is described. Several R&D issues and techniques for transmitting event data and processing distributed rules are also discussed.

References

1. Baralis, E., Ceri, S., and Paraboschi, S. “Compile-Time and Runtime Analysis of Active Behaviors,” *IEEE Transactions on Knowledge and Data Engineering* (10:3), 1998, pp. 353-370.
2. Bassiliades, N., Vlahavas, I., and Elmagarmid, A. “E-DEVICE: An Extensible Active Knowledge Base System with Multiple Rule Type Support,” *IEEE Transactions on Knowledge and Data Engineering* (2:5), 2000, pp. 824-844.
3. Brownston, L., Farrell, R., Kant, E. and Martin, N. *Programming expert systems in OPS5: An introduction to rule-based programming*, Addison-Wesley, Reading, MA, 1985.
4. Buchmann, A., Bornhövd, C., Cilia, M., Fiege, L., Gartner, F., Liebig, C., Meixner, M., and Mühl, G. “DREAM: Distributed Reliable Event-Based Application Management,” In *Web Dynamics Adapting to Change in Content, Size, Topology and Us*e, M. Levene and A. Poulovassilis (eds.), Springer-Verlag, Germany, 2004, pp. 319-352.
5. Business Rules Group. “*Defining Business Rules – What Are They Really?*,” Final report, http://www.businessrulesgroup. org/first_paper/BRG-whatisBR_3ed.pdf, 2000.

6. Carzaniga A., Rosenblum D., and Wolf, A. "Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service," *Proceedings of the Nineteenth ACM Symposium on Principles of Distributed Computing (PODC '00)*, Oregon, USA, 2000, pp. 219-227.
7. Cover, R (ed.). "Simple Rule Markup Language," http://xml.coverpages.org/srml.html, 2001.
8. Cover, R (ed.). "Business Rules Markup Language," http://xml.coverpages.org/brml.html, 2002.
9. Degwekar, S., Su, S. Y. W., and Lam, H. "Constraint Specification and Processing in Web Services Publication and Discovery," *Proceedings of the IEEE International Conference on Web Services*, San Diego, USA, 2004, pp. 210-217.
10. Krishnamurthy, B., and Rosenblum, D.S. "Yeast: A general purpose Event-Action System," *IEEE Transactions on Software Engineering* (21:10), 1995, pp. 845-857.
11. Lee, M., Su, S Y. W., and Lam, H. "A Web-based Knowledge Network for Supporting Emerging Internet Applications," *WWW Journal* (4:1/2), 2001, pp. 121-140.
12. Loucopoulos, P., and Katsouli, E. "Modelling business rules in an office environment," *ACM SIGOIS Bulletin* (13: 2), 1992, pp. 28-37.
13. Nonaka, I., and Takeuchi, H. *The Knowledge Creating Company*, Oxford University Press, New York, NY, 1995.
14. Riley, G. "C Language Integrated Production System," http://www.ghg.net/clips/CLIPS.html, 2006.
15. Rosenberg, F., and Dustdar, S. "Towards a Distributed Service-Oriented Business Rules System," *Proceedings of the IEEE Third European Conference on Web Services*, Sweden, 2005, pp.14-24.
16. Rouvellou, I., Degenaro, L., Chan, H., Rasmus, K., Grosof, B.N., Ehnebuske, D., and Barbara McKee. "Combining Different Business Rules Technologies: A Rationalization," *Proceedings of the OOPSLA 2000 Workshop on Best-practices in Business Rule Design and Implementation*, Minnesota, USA, 2000.
17. The Rule Markup Initiative. http://www.ruleml.org, 2000.
18. Rule Specification Language Schemas. http://www.cise.ufl.edu/~spd/RuleBase.xsd, http://www.cise.ufl.edu/~spd/RuleStruc.xsd, 2005.
19. Sowa, J. *Knowledge Representation: Logical, Philosophical and Computational Foundations*, Brooks Cole, Pacific Grove, CA, 2000.
20. Su, S. Y. W., Huang, C., Hammer, J., Huang, Y., Li, H., Wang, L., Liu, Y., Pluempitiwiriyawej, C., Lee, M., and Lam, H. "An Internet-based Negotiation Server for E-Commerce," *VLDB Journal*, (10:1), 2001, pp. 72-90.
21. Ullman, J. *Principles of Database Systems, 2nd Ed*, Computer Science Press, Rockville, MD, 1982.
22. Ullman, J. *Principles of Database and Knowledge-Base Systems*, Vols. I and II, Computer Science Press, Rockville, MD, 1988.
23. Warmer, J., and Kleppe, A. *The object constraint language: precise modeling with UML*, Addison-Wesley Longman, Boston, MA, 1998.
24. Widom, J., and Ceri, S. *Active Database Systems, Triggers and Rules for Advanced Database Processing*, Morgan Kaufmann, San Mateo, CA, 1996.
25. Yang, S., Lam, H., and Su, S. Y. W. "Trust-Based Security Model and Enforcement Mechanism for Web Service Technology," *Proceedings of the Third VLDB Workshop on Technologies for E-Services*, Hong Kong, China, 2002, pp. 151-160.
26. Yang, S., Su, S. Y. W., Lam, H., "A Non-repudiation Message Protocol for E-commerce," *International Journal of Business Process Integration and Management* (1:1), 2005, pp.34-42.