Event-triggered Data and Knowledge Sharing among Collaborating Government Organizations*

Seema Degwekar[1], Jeff DePree[1], Howard Beck[2], Carla S. Thomas[3], Stanley Y. W. Su[1]

[1]Database Systems R&D Center, Computer and Information Science and Engineering, University of Florida
[2]Agricultural and Biological Engineering Department, Institute of Food and Agricultural Sciences, University of Florida
Gainesville, Florida 32611
1-352-392-2693
[3]Department of Plant Pathology, University of California
Davis, California 95616
1-530-752-0300

spd@cise.ufl.edu, jdepree@cise.ufl.edu, hwb@ufl.edu, cthomas@ucdavis.edu, su@cise.ufl.edu

ABSTRACT

Solving complex global problems such as illegal immigration, border control, and terrorism requires government organizations at all levels to share not only ***data*** but, more importantly, ***knowledge*** pertinent to decision support, problem solving and activity coordination. Responding to an emergency often requires organizational and inter-organizational policies and complex operating procedures to be followed. In this work, we focus on the sharing of data associated with events of interest to collaborating organizations. Condition-action-alternative-action rules, logic/derivation rules, and constraint rules are used to define organizational and inter-organizational policies, regulations, and data and security constraints. Structures of these heterogeneous rules are used to capture organizational processes and operating procedures. A distributed event-triggered knowledge sharing system enables the interoperation of distributed, heterogeneous rules and rule structures on the data associated with each event occurrence so that all data pertinent to the event occurrence can be generated and delivered to relevant organizations. Presented in this paper are: 1) the system architecture and the distributed event and rule processing strategy, 2) algorithms used for the translation of heterogeneous rules and rule structures into web services for their uniform and efficient processing in a web service infrastructure and 3) issues and solutions related to event data aggregation, conflicting rules, and cyclic rules. The developed user interface tool and system are for deployment in the USDA's National Plant Diagnostics Network to strengthen the homeland security protection of this nation's food and agriculture.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software – d*istributed systems, information networks.*

H.4.2 [**Information Systems Applications**]: Types of Systems – *decision support.*

General Terms

Management, Documentation, Performance, Design, Reliability, Experimentation, Security, Human Factors, Standardization.

Keywords

Knowledge representation and sharing, event- and rule-based systems, decision support, collaborative federation, web services.

1. INTRODUCTION

Government agencies of today are facing complex global problems such as border control, illegal immigration, terrorism, bio-security threats, among others. Effective collaboration amongst these agencies holds the key for solving these complex problems. One important way of collaboration is for these agencies to share not only data, but also human and organizational *knowledge* useful for decision support, problem solving and activity coordination. The basic technology of sharing distributed, heterogeneous data has been extensively studied. Many recent efforts that address schema matching [12, 13], data privacy [1, 31], and schema mapping [17, 30] are important for achieving meaningful sharing of data. However, an effective way of sharing human and organizational knowledge among collaborating organizations is still lacking.

To facilitate knowledge sharing, we need to first decide what constitutes the knowledge we would like to capture and how to represent this knowledge. In this work, we are interested in capturing the knowledge embedded in an organization's policies, regulations, constraints, processes and operating procedures by using three popular types of *knowledge rules* [7, 19, 23]: integrity constraints [26], logic-based derivation rules [27], and action-oriented rules [29]. Organizational and inter-organizational processes and operating procedures are specified by *rule structures*. Thus, we can effectively capture *multifaceted knowledge* of collaborating organizations.

Secondly, we need to investigate the technique and infrastructure for sharing distributed, heterogeneous data residing in collaborating organizations' databases and knowledge rules and rule structures specified by these organizations. Typically, an organization does not want other organizations to have full access to its database due to security reasons, nor does it need to have access to the entire database of another organization.

* This project is supported by NSF under grant number IIS-0534065

Collaborating organizations are usually interested in obtaining only those data that are pertinent to the occurrence of an event of common interest (i.e., *event data*) and in processing only those knowledge rules that are applicable to the event data. An event is anything of significance to collaborating organizations (e.g. an arrest, a terrorist incidence, the detection of a disease, a special state of a database, a signal from a sensor, etc.) that occurs at a particular point in time. Event data is a dynamic data set that contains the initial data associated with an event occurrence together with the data evolved by applying relevant rules and rule structures. Thus, an event-triggered knowledge sharing system that facilitates event subscription, event notification, delivery of event data and processing and interoperation of applicable knowledge rules and rule structures would be ideal for any collaborative federation and is the focus of our research.

We define a *collaborative federation* as a number of collaborating but autonomous organizations that are connected through the Internet with the purpose of sharing event data and multifaceted knowledge. Each such organization may publish its events and rules at a designated site. Any other organization in the federation can then subscribe to the published events, publish its own rules, or use some of the published rules in its own rule structure to achieve both data and knowledge sharing.

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 [22], BRML [8] and RuleML [20] 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 [6, 14] 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 [29]. E-DEVICE [4] 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 [18] provides a web service interface to heterogeneous rule engines, thereby providing a uniform API to access each. But, 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 intended contributions of this paper are:

a) Introducing an XML-based rule specification language to capture an organization's rules and rule structures,

b) Introducing the idea and algorithms for translation of heterogeneous rules and rule structures into web services for their uniform and efficient processing in a web service infrastructure, and

c) Presenting the system architecture, the distributed event and rule processing strategy, and research issues and solutions for event data aggregation, conflicting rules and cyclic rules.

This paper is organized as follows. Section 2 describes the collaborative federation for which the developed system is to be deployed. Section 3 describes the three different types of rules and gives the algorithm for translating rules to web services. Section 4 describes the system architecture, the distributed event and rule processing strategy and some research problems and solutions. Section 5 describes the implementation framework. Section 6 summarizes the paper and describes our ongoing efforts.

2. COLLABORATIVE FEDERATION

The collaborative federation that serves as our application domain is the National Plant Diagnostic Network hereafter referred to as NPDN [16]. The U.S. Department of Agriculture's (USDA) Cooperative State Research, Education and Extension Service launched a multi-year national project in May 2002 to build NPDN to link plant diagnostic facilities across the United States. This was done to strengthen the homeland security protection of the nation's food and agriculture by facilitating quick and accurate detection of disease and pest outbreaks in crops. Such outbreaks can occur as foreign pathogens are introduced into the U.S. either through accidental importation, by wind currents that traverse continents, or by an act of bioterrorism [24, 28]. NPDN achieves its mission by creating a functional nationwide network of public agricultural institutions with a cohesive, distributed system.

The network allows land grant universities' diagnosticians and faculty, state regulatory personnel, and first detectors to communicate information, images, and methods of detection efficiently and in a timely manner. The network is organized into three tiers, with the top tier being the NPDN national repository. The next tier is the regional level. Here, lead universities have been selected and designated as Regional Hubs to represent 5 regions across the country, and they are located at Cornell University (Northeast region), Michigan State University (North Central region), Kansas State University (Great Plains region), University of Florida (Southern region), and University of California at Davis (Western region). Regional Hubs connect the systems in the states within their regions and send the collected data to the NPDN national data center at Purdue University. The national repository housed at the Center for Environmental and Regulatory Information System (CERIS) of Purdue University is the central repository for archiving this collected data.

The individual labs within each region form the third tier of the network. Information about plant samples collected by or submitted to any of the member labs are analyzed, and the lab diagnoses are sent to NPDN. Sample collection is done routinely. Member labs routinely diagnose pests/diseases observed on the sample and report to NPDN. Occasionally, all labs are on alert, looking for some particular pests/diseases. APHIS has designated 8 pests to be of particularly high concern nationally. These are called *select agents*. Each region also has a Pest-of-Concern list. A *Pest-of-Concern Standard Operating Procedure* details the steps to be taken when such a bio-security event takes place, especially for pests of regulatory concern.

As these diseases can spread rapidly from region to region, there is an overwhelming need for plant samples and diagnosis results to be transmitted to many organizations for suitable analyses. There is a need for event notification, automatic delivery of event data, and distributed processing of multifaceted knowledge and application operations among these organizations. This work builds upon a prototype system reported in [10] that processes only action-oriented rules in the NPDN environment.

3. KNOWLEDGE REPRESENTATION

3.1 Heterogeneous Knowledge Rules

We have designed a rule specification language for specifying the three different types of rules as well as rule structures. We adopt some constructs from RuleML for derivation rules. The concepts used for defining action-oriented rules and rule structures are based on our earlier work [15]. Since these rules and rule structures are to be interoperable and exchangeable, our rule language is XML-based. Due to space limitations, we cannot describe the entire language. Instead, we give a brief description of each rule type and the rule structure. Interested readers can refer to the complete schema at [21].

3.1.1 Integrity Constraints

Integrity constraints model the constraints or conditions that application data should adhere to. They are dictated by domain requirements and the effect of the violation of a constraint can range from a mild nuisance to a catastrophe. They aim to preserve the data integrity by avoiding a potentially damaging system state. There are basically two types of integrity constraints. The first type is what we call *attribute constraints*. They dictate the acceptable values that any data attribute may have at any point in time. Thus, an attribute constraint is of the form

$$x\ \theta\ n \quad \text{or} \quad x\ \gamma\ \{n_1,n_2,\ldots,n_a\}$$

where x is an attribute of an entity, n is a value from x's domain, θ is one of the six arithmetic comparison operators ($>,\geq,<,\leq,=,\neq$), $\{n_1,n_2,\ldots,n_a\}$ represents a set of enumerated values from x's domain, and γ is the set operator {in, not in}.

The second type is *inter-attribute* constraints. They model the relationship between multiple attributes. Based on this relationship, they are further divided into two sub-types. The relationship can be either mathematical (e.g., $A+B * (C-D)>=E+F$) yielding so-called *formula constraints* or conditional, (e.g. *If* $A>B$ *then* $C=50$) yielding so-called *conditional constraints*. Thus, an inter-attribute constraint is of the form

$$f_1(x_1,x_2, \ldots, x_b)\ \theta\ f_2(y_1, y_2, \ldots y_c), \text{ or}$$

$$\textit{If } (P_1\ \alpha\ P_2\ \alpha \ldots \alpha\ P_d) \textit{ then } (\ Q_1\ \alpha\ Q_2\ \alpha \ldots \alpha\ Q_e)$$

where $f_1(x_1,x_2, \ldots, x_b)$ and $f_2(y_1, y_2, \ldots y_c)$ are mathematical formulas relating the attributes $x_1,x_2, \ldots, x_b$, and $y_1, y_2, \ldots y_c$, respectively. θ is an arithmetic comparison operator. $P_1, P_2, \ldots, P_d$ and $Q_1, Q_2, \ldots, Q_e$ are predicate expressions of the form $f_1(x_1,x_2, \ldots, x_b)\ \theta\ f_2(y_1, y_2, \ldots y_c)$ connected by the logical operator α in {AND,OR}. Each of $P_1, P_2, \ldots, P_d$ and $Q_1, Q_2, \ldots, Q_e$ can be in its assertive or negated form. All the attributes of entities referenced in a constraint rule are its input data. The output of the rule is the truth value indicating whether the constraint was satisfied or not.

3.1.2 Derivation Rules

Derivation rules provide new data, if some premises on existing data are satisfied. They are generally used by expert systems to derive some new facts based on the given premises. The new data so derived can feed back into the system, till a point of convergence is reached, or till the intended results about interesting facts are obtained. Derivation rules are of the form

$$P \rightarrow Q \quad \text{or} \quad P \Rightarrow Q$$

Algorithm 1 createWebService

1. **for** each rule *r* **do**
2. *rule_interface* = *r.input* + *r.output*;
3. *rule_code* = convertToCode(*r.body*);
4. *compiled_code* = compile(*rule_code*);
5. *wsdl_doc* = generateWSDL(*rule_interface*);
6. *ws* = deploy(*compiled_code, wsdl_doc*);
7. publish(*ws*);
8. **end for**

Figure 1. Algorithm for web service synthesis

where P is the body of the implication, and Q is the head or conclusion. Both P and Q are Boolean expressions of the form

$$p_1\ \alpha\ p_2\ \alpha \ldots \alpha\ p_m,$$

where each p_i , $(1 \leq i \leq m)$ is a premise if $p_i \in P$, and a part of the conclusion if $pi \in Q$. The attributes of entities referenced in P are the rule's input data and those referenced in Q are its output data.

3.1.3 Action-Oriented Rules

Of all the three types of rules, action-oriented rules are most commonly found in event-based systems. They are usually seen as Event-Condition–Action (ECA) rules [29]. In an ECA rule, the CA portion is the actual action-oriented rule. The event E only determines when the rule is to be considered. The truth value of the condition expression C is evaluated. If it is determined to be true, the action clause A is executed. One can envision the need for the facility of executing an alternative action if the condition expression C evaluates to false. We allow organizations to define condition-action-alternative-action (CAA) rules [15, 25]. A CAA rule has the following format:

If C then A else B

where C is the condition to be evaluated, A is the action clause which is executed if C is true, and B is the alternative action clause which is executed if C is false. Each of these two action clauses can specify a number of manual and automated operations to be performed. In this rule type, attributes referenced in C as well as the input to the operations specified in A and B form the rule's input data and the result of performing the operations specified in A or B forms its output data.

3.1.4 Rule Structure

When a particular event occurs, an organization typically will have a number of rules that need to be executed in a specific order to carry out a workflow process or an operating procedure. It is very natural to model such a procedure or process by specifying the structural relationships between individual rules. A rule structure can be used to model the main constructs of a workflow process since conditional transitions and tasks specified in activities of a workflow process model can be specified by a structure of CAA rules that activate manual and automated operations. Also, constraint rules can serve as preconditions for performing activities and derivation rules can determine the appropriate input data value to an activity. The structural relationships captured by our specification are as follows.

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 ($1 \le i \le m$) 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 ($1 \le i \le n$) 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.

3.1.5 Triggers

Based on the registered information about shared events, rules and rule structures at the host site, an organization can explicitly specify a trigger to link a registered event(s) to a registered rule or rule structure. It can also be implicitly specified by the host if a registered rule or rule structure is *applicable* to the event data. A rule or rule structure is applicable to an event if the set of entities and attributes required by the rule or rule structure as its input data is a subset of the set of entities and attributes that constitute the event data. Any site that contains an applicable rule or rule structure would become an implicit subscriber of the event. When an event occurs, explicit and implicit subscribers are notified and rules and rules structures that are linked to the event by explicit and implicit triggers are activated to process the event data.

We have implemented a user interface tool for defining events, rules, rules structures and triggers and for generating rule language specifications for registration at the host site of a collaboration network. The tool as well as the prototype system will be demonstrated at the conference.

3.2 Rules and Rule Structures as Web Services

From Section 3.1, we can see that each of the rules has very different syntax and semantics. One possible approach to process these heterogeneous rules is to use multiple rule engines, each processing rules of a specific type, and build wrappers to convert the output of one rule engine to the input of the other to achieve rule interoperability. However, there are a couple of disadvantages of this approach. First, it is costly to install and maintain multiple rule systems. Second, most of the existing rule engines process rules interpretively. The resulting network system will not be very efficient. To avoid these drawbacks, we translate rules and rule structures into program code, wrap them as web services at definition time, and process them in a web service infrastructure at runtime. By doing so, rules can now interoperate programmatically without using different types of rule engines. Also, different organizations are free to use different programming languages for converting their rules and rule structures to code and deploy the rule code as web services at their own sites. The interoperation of heterogeneous rules and rule structures can thus be achieved by invoking the *rule web services*.

Figure 1 outlines the general algorithm for converting a given rule into a web service. For a rule *r* given in our rule specification language, we parse the rule to obtain the rule body (*r.body*) which is then converted to code and compiled (*compiled_code*). The WSDL document for this rule (*wsdl_doc*) is generated from the rule interface; i.e. from the rule input and the rule output (*rule_interface*). The compiled code and the WSDL document are then deployed and the resulting web service (*ws*) is published in a web service registry at the host site to make it accessible to all collaborating organizations.

The web service for a rule structure can also be generated in a similar manner. The only requirement is that all rules used in a rule structure should have been deployed earlier as web services, i.e. a rule structure can only refer to *existing* rules. Creating the rule structure then consists of composing the individual rule web services to reflect the relationships specified by the rule structure. A *link* relationship only requires the successor rule web service to be invoked after the predecessor rule web service. A *split* relationship requires the creation of multiple *threads* to process all the indicated successor rule web services in parallel, and the *and-join* and *or-join* relationships require the use of *thread synchronization* to ensure either all or the given number of predecessor rule web services have been executed.

3.3 Real-world Examples for Rules and Rule Structures

We demonstrate the use of the three rule types by modeling the Standard Operating Procedure (SOP) for a *Pest-of-Concern Scenario* in the NPDN environment. This procedure is obtained from the Plant Pathology Department at the University of California-Davis of the Western Plant Diagnostic Network. It is a national SOP used in all regions and regularly updated in collaboration with APHIS and state departments of agriculture.

Each state has a number of University Extension labs, or State Department of Agriculture labs where a sample collection or submission originates. If during diagnosis, the pest detected is on the select agent list, or is a pest of regulatory concern, the following procedure is adopted. The first detector or sample submitter sends the sample to the NPDN Triage Lab; the state facility designated to receive and examine suspect samples. The NPDN Triage Lab conducts tests on the sample. The USDA Animal Plant Health Inspection Service (APHIS) [2] is responsible for protecting and promoting U.S. agricultural health, administering the Animal Welfare Act, and carrying out wildlife damage management activities. The sample is sent to the APHIS-CDD, and the triage lab notifies the State Plant Regulatory Official (SPRO), and the State Plant Health Director (SPHD), and the regional NPDN Director that this has been done. The APHIS-CDD is the Confirming Diagnosis Designate, and makes the final decision on whether a submitted sample is confirmed positive for the suspected disease/pest. APHIS makes the confirming diagnosis and informs all other interested organizations. At each step, it is important to control the nature and the amount of information available to other relevant organizations to prevent any false alarms.

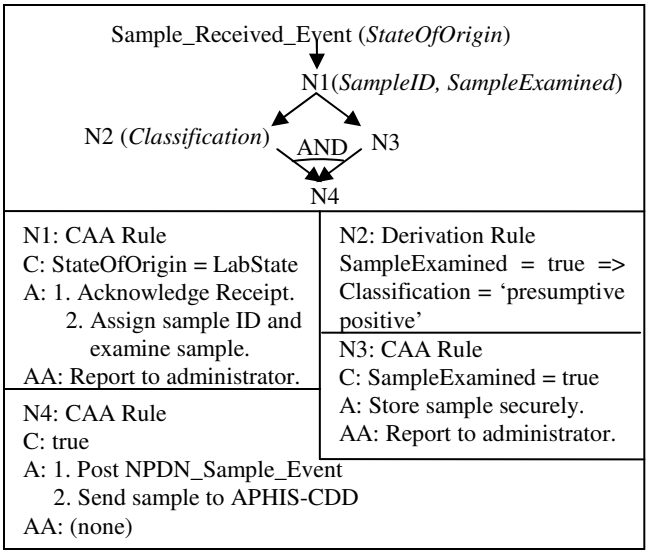

Figure 2. NPDN Triage Lab rules on sample receipt

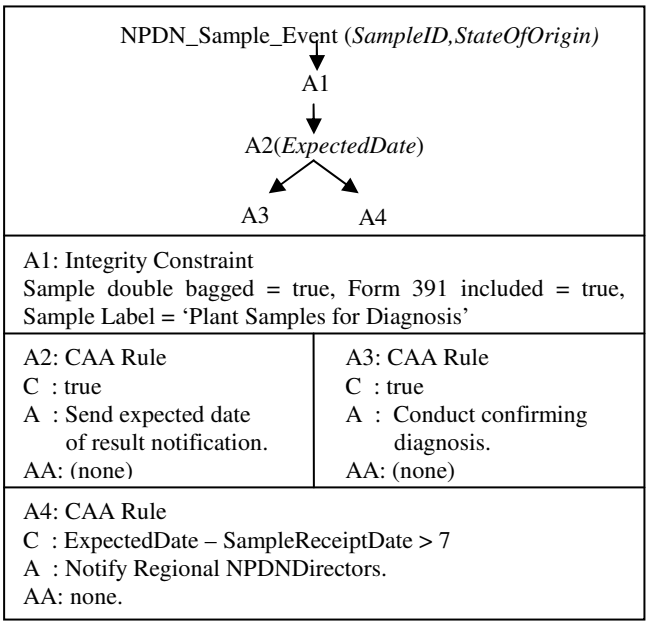

Figure 3. APHIS-CDD rules on sample receipt

The overall operation is dictated by the communication between the individual labs. A single sample entry starts the following chain of events and rules being executed. Let us start with the NPDN Triage Lab receiving a sample from a sample submitter or first detector. The rules and rule structures triggered by the event are shown in Figures 2-5. In these figures, the data introduced/modified is italicized and displayed next to the rule or event that modifies/introduces it. *C* stands for the condition expression in a CAA rule, *A* for the action clause and *AA* for the alternative action clause.

NPDN Triage Lab: The NPDN Triage Lab assigns a unique ID to the sample. The sample is then examined, and stored in a secure location with monitored access. Since the sample has been viewed by a diagnostician, it is classified as "presumptive positive". The lab staff contacts SPHD, SPRO, NPDN Regional Director and

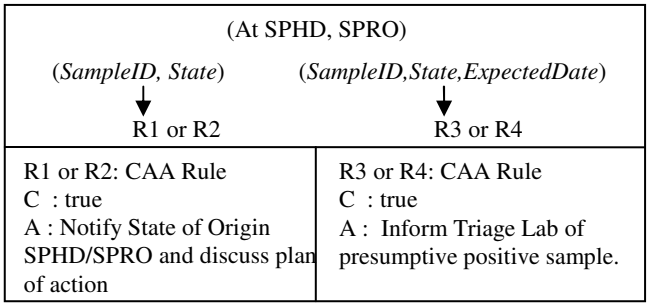

Figure 4. SPHD, SPRO rules on sample and results receipt

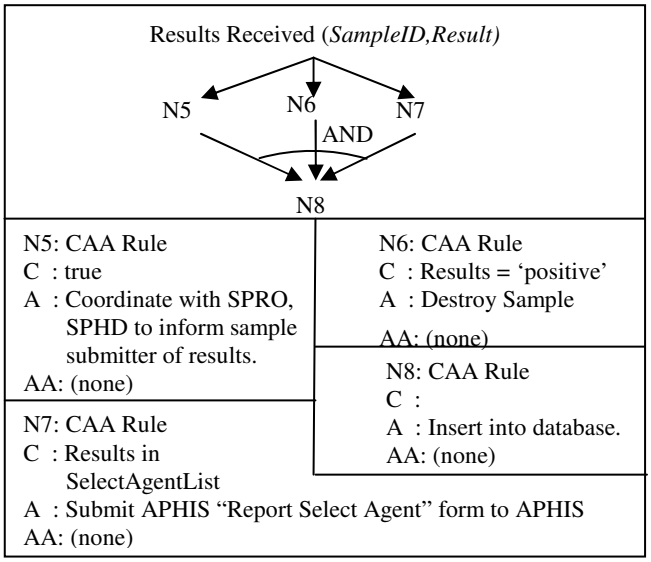

Figure 5. Triage Lab rules on receipt of sample results

APHIS-CDD by posting the *NPDN_Sample_Event* event. Then, a portion of the sample is sent to the APHIS-CDD lab. Figure 2 describes the above procedure as a rule structure.

APHIS-CDD: Once the APHIS-CDD lab receives the sample, it checks if the sample fulfilled the proper sample shipping rules. This rule is modeled as an integrity constraint. If it is violated, the CDD instructs the sender on proper packaging the next time. Otherwise, the APHIS-CDD acknowledges the receipt to NPDN along with an expected date of result notification. If this expected date is more than 7 days after the sample receipt date, the APHIS administrator contacts the NPDN Regional directors to inform them that a presumptive positive sample is in the system, and to be on alert for similar samples. The APHIS-CDD then conducts the confirming diagnosis on the sample, as illustrated in Figure 3.

SPHD, SPRO: When a "presumptive positive" sample is known to be present, SPHD and SPRO contact their counterparts in the state of origin and prepare for response procedures to follow when the results are obtained. After receiving the expected date of result notification from APHIS-CDD, they pass it on to the Triage Lab who passes it onto the NPDN Regional Directors. Figure 4 describes the above procedure as a rule structure.

NPDN Triage Lab: Once the Triage Lab receives the result from the APHIS-CDD by way of the APHIS Regional office and then the SPHD, it contacts the NPDN Regional Director and the

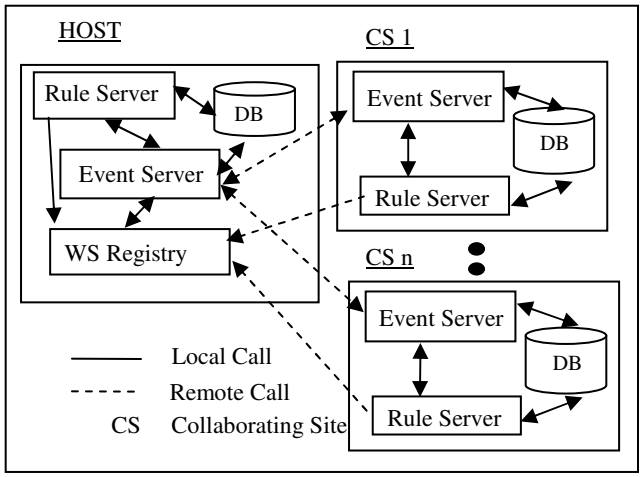

Figure 6. System Architecture

NPDN Regional Hub Lab with the confirmed diagnosis results. The Triage Lab then coordinates with State of Origin SPRO and SPHD to contact the person who initially submitted the sample with the diagnosis results. If the sample is confirmed positive, it is destroyed. Further if the confirmed positive sample was on the select agent list, the Triage Lab completes and sends an APHIS "Report of Select Agent" form to APHIS and the Center for Disease Control lab. The record is then inserted into the Triage Lab database, from where it is forwarded to the regional center, and through the regional center to the NPDN national repository. Figure 5 describes this procedure as a rule structure.

As can be seen from the examples, several manual operations are involved in the procedure. When converted to a rule web service, each manual operation is modeled by notifying the appropriate person to perform the manual operation with an instruction to let the system know when the operation has been performed so that the event and rule processing can continue.

The above real-world examples serve to demonstrate that the interoperation of different types of rules is required. In the plant diagnostics environment that we are working in, we see more CAA rules than integrity constraints or derivation rules. However, this is application domain specific. The flow of logic is mainly procedural in our example case, thus yielding more CAA rules. It is possible that other collaboration federations may have a more even mix of these three types of rules. For example, in an e-business domain [11] we have found it to be so.

4. SYSTEM ARCHITECTURE AND RULE PROCESSING

4.1 System Architecture

The event-triggered knowledge sharing system has a peer-to-peer server architecture. All participating organizations have the identical subsystem installed at their sites (see the main component servers in Figure 6). Since the existence of shared events and web services that implement knowledge rules and rule structures should be made known to collaborating organizations, there is a need to store their meta-information in a central repository. Thus, in addition to the common subsystem, one designated site that serves as the host of a federation has some additional components including a web service registry shown.

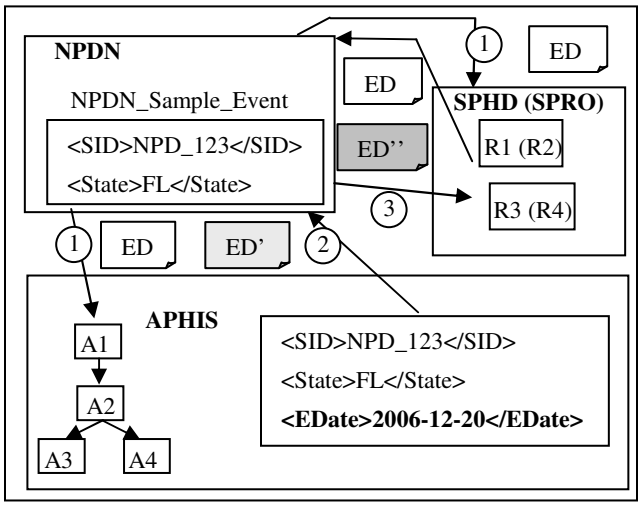

Figure 7. Event-and rule-processing

Each collaborating site creates and manages its own events, rules and triggers. When an organization defines a shared event, the Event Server at that site stores the event information in the local database as well as registers this event with the host site. The Event Server is responsible for managing information about events defined at that particular site and the information about event subscribers. 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 carries out event notification by sending event data to explicit and implicit subscribers and handles the aggregation of event data returned by them. When a rule or a rule structure is defined by an organization, the Rule Server at that site converts it into a web service, stores the rule information in the local database and registers the generated web service with the Registry at the host site. The Rule Server is responsible for processing the applicable rule web services when an event occurs.

4.2 Event and Rule Processing

To explain the event-triggered processing of distributed rules and rule structures, we use the scenario depicted in Figure 7. This scenario is a part of the rule structure explained in Section 3.3. The occurrence of the event, *NPDN_Sample_Event* generated at the NPDN site, causes the event data containing the sample id (*SID*) and the state of origin (*State*) information to be sent to APHIS, SPHD and SPRO sites in an XML document ED (step 1 shown in the figure). APHIS has an applicable rule structure and SPHD has rule *R1* and SPRO has rule *R2* as the applicable rules. Each site applies its own rules and sends back a possibly updated event data file. APHIS generates a new data item to return the expected date of result notification (*EDate*). This item is shown in bold font in the event data file. The updated file *ED'* is returned to NPDN. SPHD and SPRO use the event data *ED* to formulate their plans of action, but neither site produces any new data item nor updates any data item. This phase of processing is depicted as step 2.

NPDN then merges the event data returned from APHIS, and sends the merged document *ED''* to the applicable sites SPHD and SPRO (depicted as step 3). SPHD applies rule *R3* and SPRO applies rule *R4*, and both send the updated event data back to NPDN (not shown in the figure). Multiple rounds of event data transmission and rule processing can take place until no rule or

rule structure is applicable to the last version of event data. We note here that, once processed, a rule or rule structure will not be activated again unless its input data specification makes reference to at least one attribute whose value has been updated in the last round of event and rule processing. The updated value may cause the rule to produce a different result.

After all applicable sites have applied their applicable rules and rule structures, the final version of the event data document would contain all the data pertaining to the event occurrence. All applicable sites would receive the final document, which can be used for further decision-making and problem solving.

4.3 Research Issues

4.3.1 Event Data Aggregation

The site of an event occurrence is termed as the *coordinating site* or *coordinator* for the event occurrence. During successive rounds of event and rule processing, the event data are wrapped in an XML document (termed as the *parent* document) and sent to all applicable sites. Each collaborating site may add to or modify the event data items. These data items are returned to the coordinator as updated event data documents (*child* documents). The coordinator is responsible for aggregating the parent and child documents before starting the next round of processing.

At the end of each round of processing, the coordinator compares the contents of the parent document with the child documents in the following way. For each entity occurrence in the parent document, it creates an event data structure to store that entity instance's attributes and values. It then systematically goes through each of the child documents. For each child entity instance that has the same unique identifier value as the parent entity instance, the coordinator updates the parent instance with the updated values shown in the child instance and adds to the event data structure those new attributes and values shown in the child instance. Any new entity instance in a child document that is not in the parent document is also added to the event data structure. When all child documents have been examined, the event data structure contains the most current states of all the entities in the parent and child documents. Its contents are written into an XML document. If there are rules that refer to the updated event data or new event data in their input data specifications, a new round of event and rule processing starts by sending the XML document to the applicable sites. Otherwise, it terminates.

4.3.2 Inconsistencies and Contradictions

Rules and rule structures capture the knowledge of collaborating organizations. This knowledge reflects the opinions and experience of policy makers and experts in the organizations. In the real world, it is very possible for experts' opinions to differ. When these differing opinions are processed as knowledge rules, *inconsistencies* and/or *contradictions* may arise.

Knowledge rules (converted to web services) process the data items in the event data document and as a result may generate new data items (*additions*) or update the existing data items (*updates*). The event data documents of collaborating sites are returned to the coordinator. When the coordinator aggregates all the event data documents, it may find that inconsistent data values are given to an attribute of the same entity. A special case of inconsistency arises when the attribute is of the Boolean type and contradictory truth values are returned. From here on, we shall use the term *conflict* to mean either an inconsistency or a contradiction.

```
E_{i+1} = E_i, D_{i+1} = Φ, UC = Φ, AC = Φ
For s from 1 to n
    If U_is ≠ Φ
        For each u ∈ U_is
            If (u ∈ D_{i+1})   UC = UC + u
            Else               D_{i+1} = D_{i+1} U u
            E_{i+1} = E_{i+1} − u
        End for
    End if
    If A_is ≠ Φ
        For each a ∈ A_is
            If (a ∈ D_{i+1})  AC = AC + a
            Else              D_{i+1} = D_{i+1} U a
        End for
    End if
End for

D_{i+1} = D_{i+1} − (UC U AC)

For each u ∈ UC
    If(global_resolution_policy(u) = true)
        D_{i+1} = D_{i+1} U resolve(u)
    Else
        D_{i+1} = D_{i+1} U u^s
End for
For each a ∈ AC
    If(global_resolution_policy(a) = true)
        D_{i+1} = D_{i+1} U resolve(a)
    Else
        D_{i+1} = D_{i+1} U a^s
End for
```

Figure 8. Algorithm to detect and resolve conflicts

When a conflict is detected, we propose to resolve it in the following manner. Collaborating organizations can decide to adopt a global resolution rule to determine the value of a particular data item in case of a conflict (e.g., by taking the minimum, maximum or average of conflicting values). However, if there is no such global resolution rule for a data item, one approach is to require all sites to attach their identities with the values they produce and the coordinator to transmit event data with site ids in the next round of event and rule processing. When a collaborating site receives conflicting values tagged with site ids, it can adopt a local resolution policy to decide which site it trusts the most and adopt the value supplied by that site. In the absence of both global and local resolution policies, rules and sites that generated the conflict values can be recorded and appropriate organizations can be informed to resolve the conflict by eliminating or modifying some rule(s). The algorithm shown in Figure 8 describes the detection and the resolution mechanism employed by the coordinator.

Let ED_i ($E_i + D_i$) be the event data file sent out in round i. E_i is the portion of the event data file sent in round (i-1) that was not updated, if i >1. If i = 1, E_i is empty. D_i is the portion of the event data file which includes updates and/or additions from round (i-1), if i > 1. If i = 1, D_i is the initial event data that was made available from the event occurrence. Let U_{is} denote the updates

sent to the coordinator by site s for round i. Let A_{is} denote the additions sent to the coordinator by site s for round i. Let n be the number of sites that were applicable for round i. Let u^s and a^s denote the value of an update or addition respectively tagged with the source site s. Let Φ denote the empty set. Let UC be the set of conflicting data items due to updates and let AC be the set of conflicting data items due to additions.

The above algorithm looks at each data item sent in from a site s, and determines if there is a conflict for that data item. If so, it applies the global resolution policy if one exists. If not, it tags a particular value of the data item with its source and includes it in the event data document to be sent out.

4.3.3 Termination of Rules

Distributed knowledge rules are independently defined by collaborating organizations. During event and rule processing, each applicable collaborating site processes the relevant rules and returns the data items produced by these rules, which may trigger some other rules in the next round of processing. It is very much possible that a set of distributed rules may get locked into a cycle.

There has been a lot of work on the analysis of a static set of rules to determine if rule execution will terminate [3, 9]. In our system, knowledge rules are highly dynamic. Multiple rules can be defined, updated, suspended or reactivated. Static analysis is a conservative approach, which is based on the syntactical properties of rules and is carried out after all rules have been defined. It can correctly determine that a set of rules either will not form a cycle or there is a possibility of forming a cycle. Since it does not consider the runtime values of data items, it can not be sure what rules will be actually executed and that a cycle will definitely occur at runtime. For the above reason, we resort to a run-time approach to guarantee the termination of rule processing.

We derive some concepts of rule termination from the theory on deadlock detection and deadlock avoidance in modern operating systems. Let us consider the concept of detection and recovery first. This approach allows a cycle to occur, detect it and deactivate some rule(s) to break the cycle. In an operating system, it is not disastrous or irreparable to allow a deadlock to occur, since deadlocked processes are stalled till the deadlock is broken. For distributed rule processing, however, it can be irreparable for a cycle to continue since the cyclic rules are constantly being processed. The data values they produce may activate other rules, causing some non-idempotent operations to occur. Recovery from such a scenario is not always possible. As there is no guarantee that every cycle is self-contained and does not affect other rules, this approach is not desirable for rule termination.

Rule termination can best be guaranteed by avoiding cycles altogether. We have identified three strategies for the same. The first one is pre-computing the rule cycles that can occur for every event. The second strategy involves the use of a rule's data characteristics that are sent from collaborating sites back to the event coordinator at the end of every round of processing. The third strategy combines the first two.

- Pre-computing Possible Rule Cycles:

All shared distributed knowledge rules are registered at the host site. Thus, the host site has full knowledge of the input and output specifications of each rule. When an event is registered at the host site, it determines the applicable rules based on the initial event data specification and each applicable rule's input data specifications (i.e., only attributes of entities referenced but not their data value conditions). It then simulates the processing of that event. Starting with the set of rules that are applicable to the event data, the host site examines the output of these rules to determine those rules that will become applicable to the new version of event data. It records this information in a data structure to keep track of all the execution paths in each round of event and rule processing. For example, let us assume that during the processing of a particular event *E*, rules *R1*, *R2*, *R3* will be executed in the first round, and rules *R4* and *R5* in the second round. *R4* will acquire the necessary input data items, if either both *R1* and *R2* execute, or *R3* alone executes, whereas *R5* requires input from *R1* alone. The host site then records the following two execution paths:

$$R1\&R2 \mid R3 \rightarrow R4, \text{ and } R1 \rightarrow R5$$

where the '→' symbol indicates the beginning of a new round of processing, the '&' symbol indicates that both these rules need to execute for a rule in the next round to be applicable, and the 'I' symbol indicates that at least one of these rules needs to execute for the rule to be applicable in the next round.

The host also maintains a list of rules that have "executed" so far. Whenever a rule is set to be "executed" for a second time, the host site treats this as the beginning of a cycle. For each cycle, the host site traces the path of rule execution back to the very first round of event processing. This path of execution is stored as one estimated cycle for the event. For example, if in the third round of processing the same event *E*, *R1* is made applicable again due to the fact that some of its input data items have been modified by the execution of both *R4* and *R5*. The cyclic path

$$(R1\&R2 \mid R3) \;\&\; R1 \rightarrow R4 \;\&\; R5 \rightarrow R1 \qquad \ldots. \qquad (1)$$

will be recorded. The host stores information about all the rule execution paths until it determines that no new rules can be executed in the next round of processing. This rule termination information is store for every registered event and is incrementally updated whenever a rule is registered or updated.

When an event occurs at a collaborating site, the site downloads the rule termination structure information from the host site. Every applicable collaborating site needs to return the rules that were executed in a given round of event and rule processing back to the coordinating site. The coordinator then examines the rules that were actually executed and the rule termination structure to determine if the next round of event and rule processing will lead to a rule cycle. If so, the site where the cyclic rule will be executed is asked to deactivate the rule. For example, assume that the above event *E* occurred and *R1* and *R3* were executed in round 1. With this information, the coordinator matches the cyclic path (1) and determines that the expression for the first round *((R1&R2 | R3) & R1)* is satisfied. If in the next round, only *R4* was executed, the coordinator again checks the expression for the next round *(R4&R5)* and determines that it is not satisfied, and hence no cycle can result. If, on the other hand, during the processing of round 2, rule *R5* was also executed, the coordinator would know that *R1* will be executed in the next round causing a cycle. *R1* should be deactivated to avoid the cycle.

The above approach is better than static termination analysis methods because it assumes that rules are dynamically introduced and updated, and takes runtime information to avoid rule cycles. However, it is still conservative as it determines the existence of a possible cycle based on the syntactic properties of rule input and output specifications without considering the runtime values of data items in the event data. Only the runtime data values can determine if the input data condition of a rule is satisfied and if the rule is executed to produce its output.

- Using Rule's Data Characteristics:

In this method, the coordinating site of an event occurrence receives the data characteristics of those rules that are executed in each round of event and rule processing from every applicable site. These data characteristics enable the coordinator to determine whether the next round of event processing will cause a rule cycle to be initiated. Specifically, each site returns the condition that will invoke the rule and the equality/inequality relationship between the input and output of the rule. For example, let us consider the following two rules,

$$R1: (If\ A > 0)\ B = A+1, \text{ and}$$

$$R2: (If\ B > 0),\ A = B+1$$

being executed at a collaborating site in two different rounds of event and rule processing. This site will return the following data characteristics to the coordinator:

$$R1: Condition: A>0,\ I/O\ Relationship\ B>A,$$

$$R2: Condition: B>0,\ I/O\ relationship: A>B$$

The coordinator keeps track of all data items linked with the same inequality sign, to check if there is any cyclic relationship between data items. If such a cyclic relationship is found, it applies the rule conditions for each rule producing the data items under consideration to make the educated conclusion of whether or not a true rule execution cycle will result. For example, with rules *R1*, and *R2* above, the coordinator has the following relationship between *A* and *B*,

$$B>A>B$$

which is cyclic in nature. The coordinator now looks at the values of *A* and *B*, and applies the conditions of *R1* and *R2*. If the coordinator can conclude that the conditions will always be satisfied (since the values of *A* and *B* will always be greater than those in the previous rounds, and the conditions are $A>0$ and $B>0$ and both *R1* and *R2* have been processed once), then *R1* and *R2* will be processed indefinitely. It can then make the correct decision of not executing either *R1* or *R2* in the next round of processing. If, on the other hand, rule *R2* is as follows:

$$R2: (If\ B<15),\ A = B+1$$

The coordinator can determine that sooner or later *R2*'s input condition is not going to be satisfied since *B* always increases in value after processing *R1*, and there is a definite upper bound on the value of *B* in *R2*'s condition. Thus, it can determine that this is not a real cycle and will not suppress the processing of *R1* and *R2*.

5. IMPLEMENTATION

We use Java, Sun Application Server 9.0, Enterprise JavaBeans 3.0, the Apache jUDDI project, MySQL 5.0, and AJAX technologies to implement our prototype system. The system includes the user interface for specifying shared events, rules, rule structures and triggers as well as the event and rule servers for distributed event and rule processing. The standard operating procedure that we have implemented and used to demonstrate the utility of our technology has a total of 42 rules, most of which are CAA rules. The system will be demonstrated at the conference if our proposal for demonstration is accepted.

The work reported here is an ongoing research project. A limitation of our current implementation is that we assume the event data contains data of a single entity type. Also, for conflict resolution, we assume that either a global or a local resolution policy is available for resolving conflicting values. The implementation of the third strategy for rule cycle avoidance is an ongoing effort. In this work, we assume that organizational and inter-organizational knowledge can be manually specified in the form of knowledge rules and rule structures by using the user interface we provide. Automatic extraction of policies, regulations, constraints and procedures that have been implemented in program code is a non-trivial task. It is out of the scope of this research.

Our plan to evaluate the system is to deploy the system at several key NPDN sites. These would include the NPDN site itself, a few of the regional centers and a sizeable number of state systems. Feedback obtained will be used to improve the functionality and performance of the system. We have used our system to process the Business Rules Group's EU-rent rule set [7] in the e-business domain. Interested readers are referred to [11] for a performance evaluation discussion.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented our idea of capturing multi-faceted human and organizational knowledge by using three popular types of knowledge rules and rule structures. We also introduced the technique of managing dynamic event data and processing distributed and heterogeneous rules to achieve knowledge sharing. The occurrence of an event may trigger multiple rounds of processing and interoperation of distributed, heterogeneous rules and rule structures to derive all the data that are pertinent to the event occurrence. The approach for achieving the interoperation of different types of rules is to translate them into code at rule definition time and wrap them as web services for their uniform discovery, invocation and interoperation in a web service infrastructure. We also presented the architecture of our system, and approaches to deal with event data aggregation, conflicting rules and rule cycles.

There are several issues yet to be investigated, one of which is transaction management. Generally speaking, multiple rounds of rule processing that are triggered by an event occurrence and application operations activated by rules should be treated as a transaction. If a collaborating site aborts for any reason, its rules will not be processed, thus affecting the contents of the event data document. However, data generated by a subset of applicable rules may still contain valuable information to collaborating organizations. The enforcement of ACID properties in database systems may not be applicable to event-triggered knowledge sharing. These properties need to be examined. The second issue is regarding trust and security policies. Collaborating organizations need to negotiate and establish the policies to be

enforced by the knowledge sharing system. We are interested in specifying these policies by different types of knowledge rules and rules structures so that they can be processed uniformly with other knowledge rules. The third issue is about ontology. Since events, rules and triggers are defined by different organizations, the terms used to name data entities and attributes may have semantic discrepancies. Manually mapping terms used in event specifications to those in rule specifications would be rather tedious and error-prone. We are investigating the use of a domain ontology managed by an ontology manager [5] to either automatically or semi-automatically deal with ontological mappings by reasoning on the underlying concepts of terms.

7. REFERENCES

[1] Agrawal, R., Evfimievski, A., and Srikant, R. Information Sharing across Private Databases. *SIGMOD*, USA, 2003, 86-97.

[2] Animal Plant and Health Inspection Service, http://www.aphis.usda.gov.

[3] Baralis, E., Ceri, S., and Paraboschi, S. Compile-Time and Runtime Analysis of Active Behaviors. *IEEE Transactions on Knowledge and Data Engg.*, 10, 3 (May 1998), 353-370.

[4] 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, 824-844.

[5] Beck, H.W. On-line Content Development Tools. http://orb.at.ufl.edu/ObjectEditor.

[6] Buchmann, A., et al. DREAM: Distributed Reliable Event-Based Application Management. *Web Dynamics Adapting to Change in Content, Size, Topology and Us*e, M. Levene and A. Poulovassilis (eds.), Springer-Verlag, Germany, 2004, 319-352.

[7] Business Rules Group. *Defining Business Rules – What Are They Really?* Final report, http://www.businessrulesgroup .org/first_paper/BRG-whatisBR_3ed.pdf, 2000.

[8] Business Rules Markup Language, http://xml.coverpages.org/brml.html, 2002.

[9] Couchot, A. Termination analysis of active rules modular sets. *International Conference on Information and Knowledge Management*, Atlanta, 2001, 326-333.

[10] Degwekar, S. et al. Application of An Event-Trigger-Rule System to Agricultural Homeland Security. *International Conference on Knowledge Sharing and Collaborative Engineering*, St. Thomas, US Virgin Islands, Nov. 22-24, 2004, 50-56

[11] Degwekar, S., and Su, S. Knowledge Sharing in a Collaborative Business Environment *Workshop on e-Business*, USA, 2006, abstract (pp. 60), paper on CD, 12 pgs.

[12] Dhamankar, R. et al. iMAP: Discovering Complex Semantic Matches between Database Schemas. *SIGMOD*, France, 2004, 383-394.

[13] He, B., and Chang, K. A holistic paradigm for large scale schema matching. *SIGMOD*, France, 2004, 20-25.

[14] Krishnamurthy, B., and Rosenblum, D.S. Yeast: A general purpose Event-Action System. *IEEE Transactions on Software Engineering*, 21, 10, 1995, 845-857.

[15] Lee, M., Su, S., and Lam, H. A Web-based Knowledge Network for Supporting Emerging Internet Applications. *WWW Journal*, 4, 1/2 (March 2001), 121-140.

[16] National Plant Diagnostic Network, http://www.npdn.org.

[17] Pantel, P., Philipot, A., and Hovy, E. Aligning Database Columns using Mutual Information. *National conference on Digital Government Research,* Georgia, 2005, 205-210.

[18] Rosenberg, F., and Dustdar, S. Towards a Distributed Service-Oriented Business Rules System. *IEEE European Conference on Web Services*, Sweden, 2005, 14-24.

[19] Rouvellou, I., et al. Combining Different Business Rules Technologies: A Rationalization. O*OPSLA 2000 Workshop on Best-practices in Business Rule Design and Implementation*, Minnesota, USA, Oct. 15, 2000.

[20] Rule Markup Initiative. http://www.ruleml.org, 2000.

[21] The Rule and Rule Structure Definition Schemas, http://www.cise.ufl.edu/~spd/RuleBase.xsd, http://www.cise.ufl.edu/~spd/RuleStruc.xsd.

[22] Simple Rule Markup Language, http://xml.coverpages.org/srml.html, 2001.

[23] Sowa, J. *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks Cole, Pacific Grove, CA, 2000.

[24] Stack, J. et al. The National Plant Diagnostic Network. *Plant Dis*ease, 90, 2, 2006, 128-136.

[25] Su, S.Y.W., et al. Transnational Information Sharing, Event Notification, Rule Enforcement and Process Coordination. *International Journal of Electronic Government Research*, 1, 2 (Apr-Jun 2005), 1-26.

[26] Ullman, J. *Principles of Database Systems, 2nd Ed,* Computer Science Press, Rockville, MD, 1982.

[27] Ullman, J. *Principles of Database and Knowledge-Base Systems,* Computer Science Press, Rockville, MD, 1988.

[28] U.S. Congress. Office of Technology Assessment, Harmful Non-Indigenous Species in the United States, OTA-F-565 Washington, DC, U.S. Government Printing Office,1993,3-5.

[29] Widom, J., and Ceri, S. *Active Database Systems, Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Mateo, CA, 1996.

[30] Yu, C., and Popa, L. Semantic Adaptation of Schema Mappings when Schemas Evolve. *VLDB*, Norway, 2005, 1006-1017.

[31] Zhang, N., and Zhao, W. Distributed Privacy Preserving Information Sharing. *VLDB*, Norway, 2005, 889-900.