To my husband and best friend, Nikhil
ACKNOWLEDGMENTS

During my years of graduate study here, I have received support from many people in many different ways. First of all, I would like to thank my advisor, Dr. Stanley Y. W. Su, who has been a perfect mentor. I feel extremely fortunate to be advised by him. Through his association, I have not only picked up valuable technical expertise, but also important personality traits such as determination, perseverance, and focus on the task-at-hand. He has shown infinite patience with me, for which I will be eternally grateful.

I would also like to thank Dr. Herman Lam, Dr. Stephen Thebaut, Dr. Christopher Jermaine, and Dr. Howard Beck for being a part of my PhD committee.

On the personal front, I thank my husband, Nikhil, without whose support, I would have never completed this very important step in my life. He has always been there, whenever and wherever I needed him. I am very grateful for my parents, Lata and Pradeep, and my brother, Sachin who have supported me in all possible ways. I would also like to thank my mother-in-law, Smita, and my sister-in-law, Swati for their encouragement and appreciation.

Big thanks go to all my colleagues in the Database Center. Special thanks to Subi, Shantanu, and Abhijit for the many lunches, coffees, stimulating conversations, and great advice! I would also like to acknowledge Gilliean, Laukik, Padma, Xuelian, Alexandra, Jayendra, Jungmin, Sanghyun, Florin, Alejandro, Mark, Jeff, Mingxi, Oguzhan, Reasey, Brian, Manas, and Manna for making the lab a great place to work.

I would like to thank Alina, Andres, Raazia, Padma, and Sameep for their enthusiasm and efforts with ASCIE. I would especially like to thank Pedro, James, and Alok for helping me kick-start the organization. I would also like to thank the CISE administration and staff for all the help they have given to me as well as to all students. Very special and heartfelt thanks go to Amy
for her constant support, and great involvement in student affairs. We are very lucky to have such a wonderful graduate advisor.

I would also like to acknowledge Debra Anderson, Maud, and all the other great folks at the International Center for helping us international students with all sorts of things. Finally, I would like to thank all of my friends and family who have always cheered me on.
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
</tr>
<tr>
<td>ABSTRACT</td>
</tr>
<tr>
<td>CHAPTER</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
</tr>
<tr>
<td>2 RELATED WORK</td>
</tr>
<tr>
<td>Rule Markup Languages</td>
</tr>
<tr>
<td>Simple Rule Markup Language (SRML)</td>
</tr>
<tr>
<td>Business Rules Markup Language (BRML)</td>
</tr>
<tr>
<td>Rule Markup Language (RuleML)</td>
</tr>
<tr>
<td>Extensible Rule Markup Language (XRML)</td>
</tr>
<tr>
<td>Other Markup Languages</td>
</tr>
<tr>
<td>Event- and Rule-Based Systems</td>
</tr>
<tr>
<td>Event-based Systems</td>
</tr>
<tr>
<td>Rule-based Systems</td>
</tr>
<tr>
<td>Rule Interoperability</td>
</tr>
<tr>
<td>E-DEVICE</td>
</tr>
<tr>
<td>Service-Oriented Business Rules System</td>
</tr>
<tr>
<td>Rule Interchange Format</td>
</tr>
<tr>
<td>3 KNOWLEDGE SPECIFICATION LANGUAGE</td>
</tr>
<tr>
<td>Integrity Constraints</td>
</tr>
<tr>
<td>Derivation Rules</td>
</tr>
<tr>
<td>Action-oriented Rules</td>
</tr>
<tr>
<td>Rule Structure</td>
</tr>
<tr>
<td>Examples of Rules and Rule Structures from Two Application Domains</td>
</tr>
<tr>
<td>EU-Rent Car Rental Company</td>
</tr>
<tr>
<td>Example Rules and Rule Structure from the EU-Rent Rule Set</td>
</tr>
<tr>
<td>Example Rule 1</td>
</tr>
<tr>
<td>Example Rule 2</td>
</tr>
<tr>
<td>Example Rule 3</td>
</tr>
<tr>
<td>Example Rule Structure</td>
</tr>
<tr>
<td>National Plant Diagnostic Network</td>
</tr>
<tr>
<td>Example Rules and Rule Structure from NPDN Plant Diagnostic Laboratories</td>
</tr>
<tr>
<td>Example Rule 1</td>
</tr>
</tbody>
</table>
Example Rule 2 ........................................................................................................44
Example Rule 3 ........................................................................................................45
Example Rule Structure ...........................................................................................46

4 HETEROGENEOUS RULES AS WEB SERVICES.......................................................49

Creating a Web Service ..........................................................................................49
Algorithms for Different Rule Types .......................................................................50
Integrity Constraints ...............................................................................................50
Derivation Rules .....................................................................................................51
CAA Rules ..............................................................................................................52
Rule Structure .........................................................................................................53

5 SYSTEM ARCHITECTURE AND RULE PROCESSING.............................................63

System Architecture..............................................................................................63
Event-triggered Processing of Rules and Rule Structures in the EU-Rent Domain.....68
Decomposition of Rule Structure ..........................................................................70
Event-triggered Processing of Rules and Rule Structure in the NPDN Domain......72
Distributed Rule and Rule Structure Processing in the NPDN domain ..................73

6 RESEARCH ISSUES ..................................................................................................78

Event Data Aggregation .........................................................................................78
Event Data Inconsistencies and Contradictions ......................................................80
Avoiding Cyclic Rule Execution ............................................................................82

7 SUMMARY, CONTRIBUTIONS, AND FURTHER RESEARCH ..................................94

APPENDIX

A XML Schema documents .....................................................................................97

RuleBase.xsd ..........................................................................................................97
RuleStruc.xsd .........................................................................................................106
EventData.xsd .......................................................................................................108
Events.xsd .............................................................................................................109
Operations.xsd ......................................................................................................110

B EVENTS AND RULES IN THE EU-RENT DOMAIN ...........................................113

Description of entities used ..................................................................................113
EU-Rent Rules ........................................................................................................114
Events and Triggers ...............................................................................................128

C EVENTS AND RULES IN THE NPDN DOMAIN ................................................134

LIST OF REFERENCES ..............................................................................................151
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-1</td>
<td>Build time and run time overhead of the cyclic rule avoidance strategy</td>
<td>90</td>
</tr>
<tr>
<td>B-1</td>
<td>Average time to create and execute rules and rule structures in the EU-Rent domain</td>
<td>132</td>
</tr>
<tr>
<td>C-1</td>
<td>Average time to create and execute rules and rule structures in the NPDN domain</td>
<td>150</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Example rule structures</td>
<td>48</td>
</tr>
<tr>
<td>4-1</td>
<td>Algorithm for web service synthesis</td>
<td>55</td>
</tr>
<tr>
<td>4-2</td>
<td>Algorithm for integrity constraint rules</td>
<td>56</td>
</tr>
<tr>
<td>4-3</td>
<td>Algorithm for derivation rules</td>
<td>57</td>
</tr>
<tr>
<td>4-4</td>
<td>Algorithm for condition-action-alternative_action rules</td>
<td>58</td>
</tr>
<tr>
<td>4-5</td>
<td>Algorithm for rule structures</td>
<td>60</td>
</tr>
<tr>
<td>5-1</td>
<td>ETKnet system architecture</td>
<td>75</td>
</tr>
<tr>
<td>5-2</td>
<td>Event and rule processing in the EU-Rent domain</td>
<td>76</td>
</tr>
<tr>
<td>5-3</td>
<td>Event and rule processing in the NPDN domain</td>
<td>77</td>
</tr>
<tr>
<td>6-1</td>
<td>Algorithm to aggregate event data and identify conflicts</td>
<td>91</td>
</tr>
<tr>
<td>6-2</td>
<td>Algorithm to find possible cyclic paths on registration of an event</td>
<td>92</td>
</tr>
<tr>
<td>6-3</td>
<td>Algorithm to compute the rule expression for a possibly cyclic rule</td>
<td>93</td>
</tr>
<tr>
<td>6-4</td>
<td>Algorithm to update possible cyclic paths on addition of a global rule</td>
<td>93</td>
</tr>
</tbody>
</table>
ETKnet: A DISTRIBUTED EVENT- AND RULE-BASED SYSTEM FOR KNOWLEDGE SHARING IN A COLLABORATIVE FEDERATION

By

Seema Degwekar

August 2007

Chair: Stanley Y. W. Su
Cochair: Herman Lam
Major: Computer Engineering

Enabling government organizations to solve complex global problems such as illegal immigration, border control, and terrorism or business organizations to maintain a globally competitive edge, requires these organizations to collaborate and share not only data and application system resources, but also knowledge that captures organizational and inter-organizational policies, constraints, regulations, processes and procedures. In our study, we focus on the research and development of a knowledge specification language and the techniques, algorithms, system and infrastructure for the sharing of knowledge among collaborating organizations.

We represent knowledge using three types of rules: condition-action-alternative-action rules, logic-based derivation rules, or integrity constraints, and structures of these rules. We introduce an XML-based language that enables the specification and sharing of multi-faceted knowledge. Collaborating organizations can specify events of importance, data associated with those events, and rules and rule structures that should be triggered upon the occurrences of those events. We develop a distributed event- and rule-based system called ETKnet for event notification, event data transmission and aggregation, and processing of distributed rules, rule
structures, triggers, automated application system operations and manual operations. Data associated with an event occurrence and generated by triggered rules, rule structures and automated/manual operations can thus be shared and used by collaborating organizations for their decision-making and problem-solving. Our approach to achieve the interoperation of heterogeneous rules is to translate rules and rule structures into program code and wrap them as web services for their uniform representation, discovery and invocation in a web service infrastructure.

This dissertation presents the system architecture, the distributed event and rule processing strategy, algorithms used for the translation of heterogeneous rules and rule structures into web services, implementation details, and issues and solutions related to event data aggregation, conflicting rules, and cyclic rules.
CHAPTER 1
INTRODUCTION

Recent breakthroughs in technology have served to bring the world closer together. While easy communication and accessibility solve many fundamental problems for the world community, they also give rise to new complex ones. Some of these problems have potentially serious consequences. For example, government organizations face complex problems such as drug trafficking, terrorism, illegal immigration, and disease detection and control. Other problems likely hamper global economic progress. For example, most business organizations are facing the challenges of increasing productivity and profit while reducing costs, developing synergy among different departments of an organization or among collaborating organizations, and recognizing and planning for future assets in an ever changing world. In order for business organizations to compete in the global economy and for government organizations to tackle complex problems, they must collaborate and share their data, application system operations (automated or manual), as well as knowledge specifications that express their policies, constraints, regulations, processes and procedures [18, 32, 65]. Research has shown that knowledge acquisition and learning can greatly benefit from interaction and collaboration with others that share similar interests [76].

The basic technology of sharing distributed, heterogeneous data has been extensively studied. Many recent efforts that address schema matching [22, 27], data privacy [1, 80], and schema mapping [52, 79] are important for achieving meaningful sharing of data. However, an effective means of sharing human and organizational knowledge among collaborating organizations is still lacking.

Organizations that share data, knowledge and application operations form a collaborative federation. Each organization in such a federation typically has its own file system or database
management system to manage its data. It may also have a knowledge-based system to store and process its organizational knowledge. One approach to achieve data, knowledge and operation sharing is to build a distributed system on top of the existing systems of the collaborating organizations for accessing distributed data, knowledge and operation resources. However, direct access to an organization’s systems and data or knowledge resources may not be possible due to security reasons and the autonomous nature of these organizations. Instead, we propose that each collaborating organization publish/export only the data, knowledge specifications and application operations that it is willing to share. We provide a distributed system capable of processing these shareable resources.

At any given point in time, an organization will not be interested in getting all the shareable data of other collaborating organizations. Usually, it will be interested in getting only the specific data associated with the occurrence of an event (e.g., a delay of the delivery date of a product, a police arrest of an illegal immigrant, a terrorist attack, etc.) and the data that are generated by the execution of relevant knowledge specifications that are triggered by the event occurrence. In this work, we focus on event-triggered data, knowledge and operation sharing among collaborating organizations.

To achieve effective event-triggered data and knowledge sharing, we need to answer three important questions: “How do we represent knowledge?”, “What mechanism can we use to achieve sharing?” and “How do we achieve the interoperability of heterogeneous knowledge specifications?”.

Knowledge can be broadly classified as tacit or explicit [51]. Explicit knowledge can be expressed in language and can therefore be codified and recorded, whereas tacit knowledge cannot be expressed in language and is usually transmitted through socialization processes [47].
A community’s knowledge is obtainable from multiple sources like documents, processes, conceptual models, etc. [9]. In this work, we are interested in representing the explicit knowledge embedded in an organization’s policies, constraints, regulations, procedures and processes. This knowledge can be specified by different types of rules [44, 62]. As such, this knowledge is multi-faceted. There are three common types of knowledge rules used in existing rule-based systems [38, 58]. Condition-action or event-condition-action (ECA) rules [78] are the most common type of rules used in event–based systems [13, 16] and production rule systems [55, 12]. Experts’ opinions expressed in forms of deductive rules are employed for decision-support [71]. In addition, constraint rules are used to enforce data integrity in most database systems [70]. The above three types of rules may be structured (hereafter referred to as rule structure) to express a rule execution order that models an organizational or inter-organizational process or operational procedure. By developing a high-level, declarative knowledge specification language for describing these three popular types of knowledge rules and rule structures, we can provide a powerful means to capture the different facets of knowledge to be shared by collaborating organizations.

We now address the type of system needed to process distributed, heterogeneous rules and rule structures. As we mentioned earlier, 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 any incident 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. Each organization of a federation can define and share its events of interest, along with data generated by these events. It can also specify rules
that should be processed upon the occurrence of an event defined by either the organization itself or by another collaborating organization. When an event occurs at an organization’s site, data associated with the event occurrence (i.e., event data) are sent to organizations that have applicable knowledge rules and rule structures. A rule or rule structure is applicable to an event if the set of data items to be processed by the rule or rule structure is a subset of the event data. The processing of these rules may add to or modify the initial event data, and/or activate shareable application operations that add to or modify the event data. The results of rule processing are sent to the site that posted the event occurrence, which then merges the data collected from different organizations to produce a new version of event data and sends it out again to sites that have applicable rules. This process of event data transmission and rule processing continues until all relevant rules have been applied. The last version of the event data contains all the data that are relevant to the event occurrence. It can be used by collaborating organizations for additional problem-solving and decision-support. Thus, an event- and rule-based 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.

Now we come to the question of how to process heterogeneous rules and make them interoperable. One approach is to use three types of rule engines to process the corresponding three different types of rules and find some way to make these engines interoperable. We believe this will result in a very complex and inefficient system. Another approach taken in [8] 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. Since the semantics of these rule types are different, one problem with this approach is that it is
not always possible to convert one type of rule into another type without some loss of meaning. Also, most existing rule engines interpret rules at runtime resulting in an inefficient and unscalable rule system. A third more promising approach taken in [43, 57] is to provide a web service interface to heterogeneous rule engines, thereby providing a uniform access API. In [57], 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. In our work, we use a compilation approach by translating different types of rules and rule structures at rule definition time into code and wrapping this code as web services for their uniform registration, discovery and invocation in a web service infrastructure.

The contributions of this research are to:

1. Develop an XML-based knowledge specification language for organizations to specify their knowledge in terms of three types of rules and rule structures,

2. Develop algorithms for converting different types of rules and rule structures to web services and a strategy for processing rules and rule structures to demonstrate the interoperation of heterogeneous knowledge rules,

3. Research on techniques for managing dynamic event data and processing distributed knowledge rules, merging event data documents, handling event data inconsistencies and preventing cyclic execution of rules, and

4. Develop a distributed event- and rule-based system to achieve inter-organizational sharing of event data, knowledge and application operations.

As mentioned before, the proposed technology and system has applications in e-business, homeland security and other such collaborative efforts. Specifically, we will present two applications to demonstrate the need and use of the proposed work: one in the e-business domain by using the EU-rent car rental company example published by the Business Rules Group [38], and the other in the homeland security domain by using the National Plant Diagnostic Network (NPDN) System [49].
The rest of this dissertation is organized as follows. Chapter 2 presents earlier efforts in this area. Chapter 3 describes the syntax of the three types of rules and rule structure, and also presents our knowledge specification language, along with examples of rules and rule structures from the abovementioned applications. Chapter 4 presents algorithms for converting heterogeneous rules and rule structure to web services. Chapter 5 presents the architecture of our event- and rule-based system, the implementation framework and technologies used. Chapter 6 describes research issues pertaining to event data aggregation, event data inconsistencies, and cyclic rule execution. Finally, Chapter 7 summarizes our contributions and presents ideas for further research.
CHAPTER 2
RELATED WORK

Rule Markup Languages

Several recent efforts are concerned with developing a rule markup language for business applications. The main aim of these languages is to capture rule syntax in an XML form. Some common approaches are outlined below:

Simple Rule Markup Language (SRML)

SRML [21], proposed by ILOG in 2001, addresses the problem of sharing rules across applications with different Java rule engines. They realize that there are several commercial Java rule engines on the market such as ILOG JRules [33], JESS [24], OPS/J [54], etc. each with their own feature sets. Each of these rule engines has their own set of Java APIs, and their own proprietary rule language. This prohibits applications from sharing rules across rule engines, and as a result, ties such applications to a specific platform.

Their proposed solution is to provide an XML representation for Java rule engines. They mainly provide the syntax to describe forward-chaining condition-action rules. Constraints and deductive rules can potentially be specified by converting them to condition-action rules. However, the language does not provide explicit mechanisms to represent these other types of rules.

Business Rules Markup Language (BRML)

BRML [20] was developed in connection with IBM’s Business Rules for E-Commerce Project. It aims to provide a common representation for exchanging rules between heterogeneous rule systems. It concentrates on heterogeneous expert systems only, and thus has support only for the representation of deductive rules.
**Rule Markup Language (RuleML)**

By far the most popular and advanced markup language available today, RuleML [30] aims to capture all three types of business rules in a standardized format. The RuleML initiative began in 2000. Since then, the language is continuously evolving. The aim is to represent both forward-chaining and backward-chaining rules for deduction and other transformations. The syntax for deductive rules has been pretty much finalized, and we adopt some syntactic constructs in our own knowledge specification language. However, the effort to extend RuleML to include constraint and event-condition-action rule specifications is still in an early developmental stage.

**Extensible Rule Markup Language (XRML)**

The eXtensible Rule Markup Language (XRML) [40] aims to represent and process rules implicitly embedded in web pages. Towards that end, it defines a Rule Identification Language which allows a user to specify rules, a Rule Structure Language which converts these rules into a formal structure usable by an expert system, and a Rule Triggering Language, which defines the conditions under which these rules will be triggered. These conditions relate to events in our terminology. The user can thus make use of special tags to augment unstructured information. This can then be processed by a software agent to derive rules embedded in the web page. The aim here is to support knowledge sharing between a human being and a software agent. Currently, XRML supports deductive rules only. Also, they focus on knowledge sharing within a single organization.

**Other Markup Languages**

Besides the ones mentioned above, there are many other efforts such as

- the Semantic Web Rule Language (SWRL) [31], which aims to extend the set of the Web Ontology Language (OWL) [45] axioms to include Horn-like (deductive) rules,
• the Rule Language in OWL (ROWL) [25], which allows the specification of deductive rules in OWL and provides the facility for translating these rules into Java Expert System Shell (JESS) rules, and

• the Agent-Object-Relationship Markup Language (AORML) [74], which represents the behavior of business entities (business processes, events, agents, claims, etc.) by means of reaction (or ECA) rules.

All of the above languages and systems support only one or two of the rule types we are interested in. As far as we know, RuleML is the only effort that aims to represent all three types of rules. However, the language is yet to be finalized.

**Event- and Rule-Based Systems**

**Event-based Systems**

Several distributed event notification/service systems such as DREAM [13], Siena [16], Herald [14], Scribe [59], Yeast [39] and the work reported in [23] provide features similar to our distributed event infrastructure. These systems focus on scalability issues of event data, event subscription, efficient event notification, event filtering, and event history processing. However, most of them do not couple event notification with rule processing. DREAM and Yeast achieve this, but only use the event-condition-action (ECA) rules.

**Rule-based Systems**

There are many rule-based systems that process rules specified in a single rule representation. For example, the Web-based knowledge management system reported in [15] uses derivation rules to assist product designers in conceptual designs of products. Each rule specifies the condition(s) under which a design attribute is satisfied. In [77], a general-purpose knowledge acquisition tool called COCKATOO uses formal grammar augmented with constraint specifications to specify structured and declarative knowledge. There are many so-called *active database systems*, which use ECA rules as surveyed in [78].
With the exception of a few rule processing systems designed for processing rules in a distributed environment, most of the knowledge-based management systems are centralized: i.e., they use a single rule processing engine to process specific type of rules. Some examples of distributed rule management systems are given below.

In [35], a distributed rule processing mechanism for multi-database systems is presented. The work deals with the processing of simple and composite event expressions and the decomposition of event expressions to create sub-rules for processing on multi-databases. In [28], the authors present a distributed active database system that has a central shared knowledge repository and an ECA rule base. They also present the technique for processing ECA rules in a distributed environment. ECA rules are decomposed into sub-rules and then these sub-rules are distributed to different sites for processing against distributed databases. The authors address the issue of how to evaluate rules correctly considering the order of distributed rule executions. In [3], the authors discuss the behavior of active rules in multi-database systems. An event service generates event managers to detect, produce and notify distributed events, while a rule service generates rule managers with specific rule engines for executing reactive rules. They propose five execution policies for processing multiple rules. In [36], the concept and the technique for coordinating peer databases using ECA rules are presented. Peer databases are stand-alone, independently developed databases that are linked to each other through acquaintances. They each contain local data, a set of mapping tables and expressions, and a set of ECA rules, which are used to exchange data among them. The set of acquaintances and peers constitutes a dynamic peer-to-peer network in which acquaintances are continuously established and abolished. The paper describes techniques for specifying data exchange policies on-the-fly based on constraints imposed on the way peers exchange and share data. It provides on-the-fly specification of data
exchange policies by building coordination ECA rules at acquaintance time. In [53], a system called KRAFT deals with the access of heterogeneous data from multiple, distributed, and heterogeneous data sources. An integration schema provides a common representation across the distributed system. Distributed database queries are processed to retrieve data instances. However, to make these instances meaningful, KRAFT attaches context information with these instances. The context information is expressed in the form of constraints, which describe how each data instance should be interpreted and used.

There are three main differences between our system and those mentioned above. First, in these systems, event notification is not linked with the processing of heterogeneous, distributed rules. Second, they deal with the processing of only one type of rules, whereas, our system deals with the processing of multiple types of rules and rule structures. Third, they decompose rules into sub-rules and transmit them to multiple sites for processing against distributed data, whereas we transmit event data, which is limited in quantity as compared with stored databases, to those sites that contain applicable rules.

**Rule Interoperability**

The following three systems aim to achieve heterogeneous rule interoperability in different ways. The first one designates one of the three types of rules as the standard, and converts the other two into the standard representation. The second and the third propose providing a standardized interface to three types of rule engines.

**E-DEVICE**

E-DEVICE [8] proposes an active knowledge based system to support the processing of all three types of rules. The core system is an active OODB system that has support for events and ECA rules. Derivation rules and integrity constraints are mapped into ECA rules and processed by a single rule engine. The events monitor the insertion, update, or deletion of an object, an
object attribute, or an intra-object pattern. The condition clause is an inter-object pattern, which consists of the conjunction of one or more intra-object patterns. The action clauses are either insertion, deletion, or update of an object or object attribute. The implemented system (at the time of writing) does not support the translation of integrity constraints into ECA rules. The operations allowed in action clauses are limited to database operations instead of application system operations.

**Service-Oriented Business Rules System**

The system presented in [57] addresses the lack of flexibility and reusability of business rules across distributed rule engines. The authors propose a service-oriented business rules system that manages and deploys business rules to different business rule engines. Their distributed system consists of several nodes, which can belong to one or more of the following three types: Rules Broker Node, Deployment Node, and Coordination Node. Each Rules Broker node hosts one or more Business Rules Brokers, which can have several different rule engines plugged in. The broker is used to direct the rule processing to the appropriate engine and uses the web service interface as a communication mechanism between different types of engines. A rule is first implemented in the appropriate rule engine. The interface to this rule is generated as a web service and is deployed at one or more deployment nodes. The coordination node is responsible for coordinating the deployment of a set of rules among multiple available rules broker nodes. This node uses a two-phase commit protocol to ensure that all participating rules broker nodes have received the correct set of rules. This same system has also been used to demonstrate rule integration in BPEL [56]. Here, the authors focus on the integration of business rules with web service composition. A process workflow is augmented with pre- and post-activity rules that encapsulate business logic. Different from this work, rule execution in our system is governed not only by explicit triggers that link events to rules and rule structures, but
also by implicit triggers determined by examining whether the input data to a rule or rule structure are a subset of the event data. Implicit triggers give our system the forward-chaining like behavior which is not present in the referenced system. Furthermore, rule execution in the referenced system is carried out by individual engines interpretively, whereas we use a compilation approach. 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.

**Rule Interchange Format**

W3C has established a Rule Interchange Format working group [10] to produce a framework or language to translate rules between different systems. The purpose of this group is to enable rule interoperability by allowing rules specified in one format to be processed by a different rule engine. At the time of this writing, the RIF Core has been developed. From the RIF Core Design document [10], “RIF Core corresponds to the language of definite Horn rules with equality (and with a standard first-order semantics). Syntactically, however, RIF Core has a number of extensions to support features such as objects and frames, URIs as identifiers for concepts, and XML Schema data types. These features make RIF a *Web language*. However, RIF is designed to enable interoperability among rule languages in general, and its uses are not limited to the Web. The semantics of RIF has provisions for future extensions towards dialects that support pure FOL, dialects that support negation as failure (NAF), business (or production) rules, reactive rules, and other features. Eventually, it is hoped that RIF dialects will cover a number of important paradigms in rule-based specification and programming. Our main target paradigms include production rules, logic programming, FOL-based rules, reactive rules, and normative rules (integrity constraints).”

It is worthwhile to point out some key differences between our system and those mentioned above. E-DEVICE is an event-based system, but it does not use a compilation
approach for rule processing. Instead, it converts different types of rules into ECA. Also, its application domain is a single organization with an OODB instead of a collaborative federation.

The service-oriented business rules system is not an event-based system. It assumes the existence of a knowledge base, against which user queries are issued, and aims to answer such queries by applying all relevant business rules. It provides a web service interface to communicate with different engines; however the actual processing of the rules is done by a rule engine, which uses an interpretive approach. There is no rule structure or workflow-like construct for modeling business processes. The Rule Interchange Format effort is a fairly new one. At the time of this writing, the core design of the language known as “Condition language” has been identified. This can be extended to derivation rules, reactive rules, as well as integrity constraints, which are future deliverables.

After surveying the existing efforts and systems that are related to our work, we have arrived at the following two conclusions.

- To our knowledge, there is no system that uses a combination of different rule types to specify individuals/organizations’ knowledge in a structured manner.
- The interoperation among distributed, heterogeneous rules in a collaborative federation has not been investigated.
CHAPTER 3
KNOWLEDGE SPECIFICATION LANGUAGE

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. Each organization uses a GUI tool to define a set of rules and rule structures that form a local rulebase to be shared with other organizations. Each rule has the following characteristics: a name, a description, an optional state (which can be either active or suspended, the default being active), and a sharing attribute to indicate whether the rule is local to the defining organization or global, i.e. shared across the federation.

The root element RuleBase represents the local rulebase. RuleBase has one or more child elements, Rule, which represent individual rules. The Rule constructs that define a RuleBase are shown in the Backus-Naur Form (BNF) syntax below. The elements IntCons, Implies, and CAARule represent an integrity constraint rule, a derivation rule and a condition-action-alternative_action rule respectively. They are described in detail in the following sections.

We adopt BNF syntax notations to describe the language for clarity and conciseness. The actual XML schema is included in Appendix A. The schema for describing the rule structure, events, event data as well as operations is also included in Appendix A. The notation A ::= B, where both A and B are non-terminal symbols, is used to denote that the XML element B is a direct descendant of the XML element A; i.e., A is described by the following schema <A><B> … </B></A>. The notation A ::= t, where A is a non-terminal symbol and t is a terminal symbol, is used to denote that the value of A is restricted to the terminal t. Square brackets denote an optional element; i.e., an element which can appear either zero or one time. Curly brackets denote an element that can appear zero or more times. Boldface font is used to distinguish
terminal symbols from non-terminal symbols. The terminal symbol letter is used to represent uppercase (A-Z) and lowercase (a-z) letters of the alphabet, digit is used to represent decimal digits 0-9, and the symbol space is used to indicate any whitespace (a single space, a tab, a line feed, etc.).

```
RuleBase ::= Rule {Rule}
Rule ::= RuleName RuleDescription [RuleState] Sharing IntCons |
   RuleName RuleDescription [RuleState] Sharing Implies |
   RuleName RuleDescription [RuleState] Sharing CAARule
RuleName ::= letter {letter | digit | space}
RuleDescription ::= letter {letter | digit | space}
Sharing ::= local | global
RuleState ::= active | suspended
```

### Integrity Constraints

An integrity constraint [70] ensures that changes made to the database do not result in a loss of data consistency. In the context of this effort, 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 [64], which was patterned after the Object Constraint Language [75]. Since we use an object-oriented data model to represent event data, constraints can be specified on an object or on one or more of its attributes. Constraints can be classified into two types: attribute constraint and inter-attribute constraint. Thus, the IntCons element representing the integrity constraint has the following syntax. The element AttCons represents an attribute constraint and InterAttCons represents an inter-attribute constraint.

```
IntCons ::= AttCons | InterAttCons
```

An attribute constraint is of the form

\[ x \theta n, \text{ or } x \gamma \{n_1,n_2,...,n_a\} \]

where \( x \) is an object or object attribute, \( n \) is a value from \( x \)'s domain, \( \theta \) is one of the six arithmetic comparison operators \{>,>=,<,<=,=,\#\}, \( \gamma \) is one of the enumeration operators \{in, not
in}, and \( \{n_1, n_2, \ldots, n_a \} \) represents a set of enumerated values from \( x \)'s domain. Thus, an attribute constraint specifies the allowed or valid constant values of an object or object attribute. Examples of attribute constraints are: \( a > 10 \), \( b \) in \( \{2, 5, 9\} \), where \( a \) and \( b \) are object attributes.

The BNF syntax of an attribute constraint is shown below.

\[
\text{AttCons ::= DataItem RelOp Value | DataItem EnumOp Value {Value}}
\]

\[
\text{Name ::= letter | digit | space}
\]

\[
\]

\[
\text{Value ::= \{letter | digit\} | digit \{digit\} \{digit\}}
\]

\[
\text{RelOp ::= gt | lt | ge | le | eq | ne}
\]

\[
\text{EnumOp ::= in | not in}
\]

The object or object attribute is represented by the element \( \text{DataItem} \), which consists of a name and a type. Since we generate web services from such constraint specification programmatically, the precise name and type information are needed by the programming language used to implement the web service. Also, the type information is specific to the programming language Java in our implementation, but it can be easily changed or extended for any other programming language. Furthermore, any additional types introduced by the application domain also need to be specified here. In the above BNF, we have included the type specifications for the NPDN and the EU-rent domains. Element \( \text{RelOp} \) stands for arithmetic comparison operators, element \( \text{EnumOp} \) for the enumeration operator, and \( \text{Value} \) for constant values.

An inter-attribute constraint is of the form

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

\[
\text{If } P_1 \alpha P_2 \alpha \ldots \alpha P_d \text{ then } Q_1 \alpha Q_2 \alpha \ldots \alpha Q_e
\]
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 objects or object attributes \( x_1, x_2, \ldots, x_b \) and \( y_1, y_2, \ldots, y_c \), respectively. \( P_1, P_2, \ldots, P_d \) and \( Q_1, Q_2, \ldots, Q_e \) are predicate expressions of the form \( f_1(x_1, x_2, \ldots, x_b) \ θ f_2(y_1, y_2, \ldots, y_c) \) connected by the logical operator \( α \), which is a member of the set \{\text{AND, OR}\}. Examples of inter-attribute constraints are: \( a+b\geq c \), \( If \ (a>10 \text{ and } b>15) \ then \ (c=5) \), where \( a, b \) and \( c \) are object attributes.

Based on the above definition, we categorize the inter-attribute constraints into two sub-types. Constraints specified using the first form described above are called \textit{formula constraints} and those specified using the second form are called \textit{condition constraints}.

Their BNF syntaxes are shown below.

\[
\text{InterAttCons} ::= \text{FormulaCons} \mid \text{CondCons}
\]
\[
\text{FormulaCons} ::= \text{Expr} \ \text{RelOp} \ \text{Expr}
\]
\[
\text{Expr} ::= \text{Term} \ \{\text{MathOp} \ \text{Term}\} \mid
\]
\[
\text{Expr} \ \text{MathOp} \ \text{Expr} \ \{\text{MathOp} \ \text{Expr}\}
\]
\[
\text{Term} ::= \text{DataItem} \mid \text{Value} \mid \text{Operation}
\]
\[
\text{MathOp} ::= + \mid - \mid / \mid *
\]
\[
\text{Operation} ::= \text{OperationName} \ \{\text{DataItem}\} \ [\text{Returns}]\n\]
\[
\text{OperationName} ::= \text{letter} \ \{\text{letter} \mid \text{digit} \mid \text{space}\}
\]
\[
\text{Returns} ::= \text{DataItem} \ \{\text{DataItem}\}
\]
\[
\text{CondCons} ::= \text{IfExpr} \ \text{ThenExpr}
\]
\[
\text{IfExpr} ::= \ [\text{Not}] \ \text{BooleanExpr} \ \{\text{LogicalOp} \ \text{BooleanExpr}\} \mid
\]
\[
\ [\text{Not}] \ \text{IfExpr} \ \text{LogicalOp} \ \text{IfExpr} \ \{\text{LogicalOp} \ \text{IfExpr}\}
\]
\[
\text{ThenExpr} ::= \ [\text{Not}] \ \text{BooleanExpr} \ \{\text{LogicalOp} \ \text{BooleanExpr}\} \mid
\]
\[
\ [\text{Not}] \ \text{ThenExpr} \ \text{LogicalOp} \ \text{ThenExpr} \ \{\text{LogicalOp} \ \text{ThenExpr}\}
\]
\[
\text{BooleanExpr} ::= \ [\text{Not}] \ \text{PredicateExpr} \mid \ [\text{Not}] \ \text{Term}
\]
\[
\text{PredicateExpr} ::= \text{Expr} \ \text{RelOp} \ \text{Expr}
\]
\[
\text{LogicalOp} ::= \text{AND} \mid \text{OR}
\]

The formula constraints are represented by the element \textit{FormulaCons}, whereas the condition constraints are represented by the element \textit{CondCons}. Each of these XML elements makes use of building blocks like the elements \textit{Expr} for an expression, \textit{IfExpr}, for the \textit{if} construct, and \textit{ThenExpr} for the \textit{then} construct. The \textit{Expr} construct is composed of one or more \textit{Term} elements linked by mathematical operators. A \textit{Term} can be a single data item, a constant.
value, or an operation. An operation represents a function that operates on zero or more parameters yielding zero or more output data items. An *IfExpr* element is composed of one or more Boolean expressions, represented by *BooleanExpr*, which in turn can be either a predicate expression (an expression that links two *Expr* elements by a comparison operator), or a single term.

**Derivation Rules**

Logic-based derivation rules [71], also known as inference rules or deductive rules, assess a given premise to come to some conclusion. They can be expressed as

\[ P \rightarrow Q, \text{ or } P \Rightarrow Q \]

the semantics of which are: If \( P \) is evaluated to true based on the contents of the event data, then \( Q \) should be made true (i.e., data expressed by \( Q \) should be added to the event data). \( P \) is the *body* of the implication, and \( Q \) is the *head* or conclusion. Both \( P \) and \( Q \) are Boolean expressions of the form

\[ r_1 \alpha r_2 \alpha \ldots \alpha r_n, \]

where each \( r_i \) (\( 1 \leq i \leq n \)) is a premise if \( r_i \in P \), and is a part of the conclusion if \( r_i \in Q \). \( \alpha \) is the logical operator, *AND* or *OR*. We allow arbitrary nesting of the logical operators in \( P \) and as such do not restrict \( P \) to conform to either conjunctive or disjunctive normal form, instead we rely on *grouping* or *nesting* constructs (parentheses), which allow the creator of the rule to specify his/her semantics precisely. We restrict the logical operator in \( Q \) to be *and*, for clear interpretation of the conclusion. If *or* semantics are desired as shown in the following rule \( R \)

\[ R: \ p \Rightarrow q_1 \lor q_2 \lor \ldots \lor q_n, \]
The single rule $R$ can be rewritten as multiple rules $r_1...r_n$, with each $r_i \ (1 \leq i \leq n)$ having the same premise $p$ as the original rule, and $q_i$ as the conclusion. The derivation rule syntax is shown below:

\[
\text{Implies ::= Body Head}
\]
\[
\text{Body ::= IfExpr}
\]
\[
\text{Head ::= Atom \{AndOp Atom\}}
\]
\[
\text{Atom ::= DataItem RelOp DataItem \mid DataItem RelOp Value}
\]
\[
\text{AndOp ::= AND}
\]

It is derived in part from RuleML [30]. The implication is represented by the element \text{Implies}. \text{Implies} consists of the \text{Head} and \text{Body} elements. The \text{Head} element contains one or more \text{Atom} elements linked by the \text{AND} operator. Each of these atoms specifies a new or derived value for the indicated object or object attribute. The \text{Body} element is represented by the element \text{IfExpr}. Premises in our language can contain executable functions in addition to variables.

**Action-oriented Rules**

Action-oriented rules are typically found in active database systems [78]. They are known as event-condition-action (ECA) rules or just event-action rules [44]. 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. The general format of the rule is thus

\[
\text{On E if C then execute A}
\]

If we separate the event (E) from the condition-action (CA) part, we see that the CA part is in fact the action-oriented rule. The event is used to indicate \textit{when} to perform the action-oriented rule. Also, an action-oriented rule may specify what actions are to be taken when the condition expression evaluates to false. We can model this concept as two complimentary if-then rules

\[
C_1A, \text{ and } C_2B
\]
where \( C_2 \) is the complement of \( C_1 \), and \( A \) and \( B \) are action clauses. Another way would be to model them as a single condition-action-alternative_action (CAA) [41] rule. Thus, the semantics of the CAA rule are as shown below. We use this form of the action-oriented rule in our work.

\[
\text{If } C \text{ then } A \text{ else } B
\]

Since we are interested in developing a distributed event- and rule-based system, the separation of the event specification and the action-oriented rule specification is important. It allows events and action-oriented rules to be defined by different organizations. Events can be flexibly linked to action-oriented rules by \textit{triggers}, which can be defined by the same or other organizations. It also allows an organization to link an event with not only a CAA rule, but any other type of rules or a rule structure. Thus an event occurrence can invoke an integrity constraint check, a derivation of new data, an execution of a CAA rule, and/or an execution of a rule structure. We show the syntax of a condition-action-alternative_action rule below.

\[
\text{CAARule ::= [Condition] Action [AlternativeAction]}
\]

\[
\text{Condition ::= [Guard] CondExpr}
\]

\[
\text{Guard ::= PredicateExpr \{PredicateExpr\}}
\]

\[
\text{CondExpr ::= IfExpr}
\]

\[
\text{Action ::= Operation \{Operation\}}
\]

\[
\text{AlternativeAction ::= Operation \{Operation\}}
\]

The condition clause is represented by the \textit{Condition} element, the action clause by the \textit{Action} element and the alternative-action clause by the \textit{AlternativeAction} element. Each of these elements uses the building blocks like \textit{IfExpr}, \textit{Expr}, \textit{Term}, etc. mentioned earlier. An action-oriented rule may want an action to be executed unconditionally. This is why the \textit{Condition} element is optional. Also, there may be no defined alternative action for a particular rule. Thus, the \textit{AlternativeAction} element is also optional.
The condition expression consists of two parts – an optional guard clause and the condition clause. If the guard evaluates to false, the execution of the rule is skipped, thereby serving as a means of conditionally deactivating the rule. If not, the condition is evaluated to determine whether to perform the action or the alternative action, which are represented by Action and AlternativeAction elements, respectively. These elements are described by one or more operations.

**Rule Structure**

When a specific event occurs, an organization may have a number of applicable rules that need to be executed in a specified order. It is very natural to model a process or an operating procedure by specifying the structural relationship between individual rules. We capture this relationship by means of a rule structure [41].

In an organizational process, a rule \( r \) may be required to execute before another rule \( s \), thus establishing a direct link between \( r \) and \( s \). Similarly, a rule \( r \) may be required to execute before multiple rules \( s_1, s_2, \ldots, s_m, m \geq 1 \), which can then be processed in parallel. In this case, \( r \) is connected to \( s_1, s_2, \ldots, s_m \) in a split construct. A rule \( s \) may be required to wait for all of a given set of rules \( r_1, r_2, \ldots, r_n, n \geq 1 \) 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. Also, \( s \) may be required to wait for not all but a subset of the rules \( r_1, r_2, \ldots, r_n, n \geq 1 \) 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 governs(governs) the execution of other rules is(are) termed predecessor(s), and the rule(s) that executes(execute) after the predecessor(s) is(are) termed successor(s).

*A rule structure can now be defined as a directed graph with different types of rules as nodes that are connected by link, split, and-join, and or-join constructs.*
Some example rule structures are shown in Figure 3-1. \( R1, R2, \ldots R8 \) are individual rules which could belong to any of the three types discussed above.

The syntax of a rule structure is given below.

\[
\begin{align*}
\text{RuleStruc} & ::= \text{Name RuleStrucDescription }[\text{RuleStrucState}] \text{ Sharing RuleSubStruc }[\text{RuleSubStruc}] \\
\text{RuleStrucDescription} & ::= \text{letter } \{\text{letter } | \text{digit } | \text{space}\} \\
\text{RuleStrucState} & ::= \text{active } | \text{suspended} \\
\text{Sharing} & ::= \text{local } | \text{global} \\
\text{RuleSubStruc} & ::= \text{Link } | \text{Split } | \text{AndJoin } | \text{OrJoin} \\
\text{Link} & ::= \text{Predecessor Successor} \\
\text{Split} & ::= \text{Predecessor Successors} \\
\text{AndJoin} & ::= \text{Predecessors Successor} \\
\text{OrJoin} & ::= \text{Num Predecessors Successor} \\
\text{Predecessor} & ::= \text{Name} \\
\text{Predecessors} & ::= \text{Name Name }\{\text{Name}\} \\
\text{Successor} & ::= \text{Name} \\
\text{Successors} & ::= \text{Name Name }\{\text{Name}\} \\
\text{Name} & ::= \text{letter } \{\text{letter } | \text{digit } | \text{space}\} \\
\text{Num} & ::= \text{digit } \{\text{digit}\}
\end{align*}
\]

It is represented by the \text{RuleStruc} element. A complicated rule structure can be broken down into \text{substructures}, each of which represents a single type of relationship between two or more rules. We represent such substructures using the \text{RuleSubStruc} element. Each type of relationship is represented by XML elements with the respective name. The predecessors and successors are represented by the following elements: \text{Predecessor} (rule \( r \) in a link or split), \text{Predecessors} (rules \( r_1, r_2, \ldots, r_n \) in a join), \text{Successor} (rule \( s \) in a link or join), and \text{Successors} (rules \( s_1, s_2, \ldots, s_m \) in a split). \text{Num} indicates the number of predecessor rules to be executed in an or-join before the successor rule can be invoked.

**Examples of Rules and Rule Structures from Two Application Domains**

We now give some selected examples of rules and rule structures defined for two application domains: e-business (EU-Rent car rental company) and Agricultural Homeland Security (the National Plant Diagnostic Network).
**EU-Rent Car Rental Company**

The GUIDE Business Rules Project [38] was organized in November 1993 to “formalize an approach for identifying and articulating the rules which define the structure and control the operation of an enterprise.” To illustrate their techniques, the project came up with a case study of a fictitious car rental company EU-Rent, which is owned by EU-Corporation. EU-Rent is one of the three businesses that EU-Corporation operates; the other two are EU-Fly airlines, and EU-Stay hotels. EU-Rent has 1000 branches in towns in several countries. At each branch cars, classified by car group, are available for rental. Each branch has a manager and booking clerks who handle rentals.

The car rental company has to handle rental reservations, rental returns, car servicing, maintain customer relations, promote customer loyalty, and deal with car purchase and sale. These tasks form the core of the company and it is therefore very essential that they be managed in a very consistent and efficient manner. EU-Rent has captured key aspects of each task in the form of business rules. Such rules are broadly similar across branches; however, each branch also has its own local set of policies that generate its own local set of business rules. 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 collaborative federation of the EU-Rent company.

We realize that the above is not a real-world inter-organizational setting, yet 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 a branch requires the interoperation of
different rule types captured in a rule structure. We now give some example rules and a rule
structure from the rule set and their conversion to our knowledge specification language.

**Example Rules and Rule Structure from the EU-Rent Rule Set**

**Example Rule 1**

Rule: Reservations may be accepted only up to the capacity of the pick-up branch on the pick-up
day.

Name: capacityCheck

Type: Integrity Constraint Rule

Representation in Knowledge Specification Language: We include only the main XML element
of interest “IntCons” in the following.

```
<IntCons>
  <InterAttCons>
    <FormulaCons>
      <Expr>
        <Term>
          <DataItem>
            <Name>Branch.numReservations</Name>
            <Type>int</Type>
          </DataItem>
        </Term>
      </Expr>
      <RelOp>le</RelOp>
      <Expr>
        <Term>
          <DataItem>
            <Name>Branch.capacity</Name>
            <Type>int</Type>
          </DataItem>
        </Term>
      </Expr>
    </FormulaCons>
  </InterAttCons>
</IntCons>
```

**Example Rule 2**

Rule: To join the loyalty incentive scheme, a customer must have made 4 rentals within a year.
Name: eligibleForLoyaltyScheme

Type: Derivation Rule

Representation in Knowledge Specification Language: We include only the main XML element of interest “Implies” in the following.

```xml
<Implies>
  <Body>
    <IfExpr>
      <BooleanExpr>
        <PredicateExpr>
          <Expr>
            <Term>
              <DataItem>
                <Name> Customer.numYearlyRentals</Name>
                <Type>int</Type>
              </DataItem>
            </Term>
          </Expr>
          <RelOp>ge</RelOp>
          <Expr>
            <Term>
              <Value valueType="int">4</Value>
            </Term>
          </Expr>
        </PredicateExpr>
      </BooleanExpr>
    </IfExpr>
  </Body>
  <Head>
    <Atom>
      <DataItem>
        <Name> Customer.eligible</Name>
        <Type>boolean</Type>
      </DataItem>
      <RelOp>eq</RelOp>
      <Value valueType="boolean">true</Value>
    </Atom>
  </Head>
</Implies>
```
Example Rule 3

Rule: After all assignments within a group have been made, 10% of the group quota for the branch (or all remaining cars in the group, whichever number is lower) must be reserved for the next day’s walk-in rentals. Surplus capacity may be used for upgrades.

Name: reserveForWalkIn

Type: Condition-Action-Alternative_Action Rule

Representation in Knowledge Specification Language: We include only the main XML element of interest “CAARule” in the following.

```
<CAARule>
  <Condition>
    <CondExpr>
      <IfExpr>
        <BooleanExpr>
          <PredicateExpr>
            <Expr>
              <Term>
                <DataItem>
                  <Name>Group.numRemainingCars</Name>
                  <Type>int</Type>
                </DataItem>
              </Term>
              <RelOp>gt</RelOp>
              <Expr>
                <Term>
                  <Value valueType = "double">0.1</Value>
                </Term>
                <MathOp>*</MathOp>
                <Term>
                  <DataItem>
                    <Name>Group.quota</Name>
                    <Type>int</Type>
                  </DataItem>
                </Term>
              </Expr>
            </PredicateExpr>
          </BooleanExpr>
        </IfExpr>
      </CondExpr>
    </Condition>
  </CAARule>
```
Example Rule Structure

Rule Structure: Each paid rental in the (loyalty incentive) scheme (including the 4 qualifying rentals) earns points that may be used to buy ‘free rentals.’ The basic rental cost of a free rental can be bought with points. Extras, such as insurance, fuel and taxes must be paid by cash or credit card. A free rental must be booked at least fourteen days before the pick-up date. Free rentals do not earn points.

Name: RS1

Representation in Knowledge Specification Language: This is a rule structure that can be used to assign “free rentals”. Descriptions of the component rules can be found in Appendix B. It first
checks eligibility using the derivation rule “eligibleForLoyaltyScheme”. Then, it makes sure that only the basic cost of the rental is bought with points and extras are paid for by using a CAA Rule “freeRental”. Next, it calculates the free rental points earned using the derivation rule “freeRentalPoints”, and verifies advance booking using the integrity constraint “checkAdvBooking”. Rule “eligibleForLoyaltyScheme” is executed first and after that rule “freeRental” is executed, followed by rules “freeRentalPoints”, and “checkAdvBooking” in parallel. We include the main XML element of interest “RuleStruc” in the following.

```xml
<RuleStruc>
  <Name>Free Rental Rule Structure</Name>
  <RuleStrucDescription>Procedure to give a free rental</RuleStrucDescription>
  <RuleStrucState>ACTIVE</RuleStrucState>
  <Sharing>global</Sharing>
  <RuleSubStruc>
    <Link>
      <Predecessor>
        <RuleName>eligibleForLoyaltyScheme</RuleName>
      </Predecessor>
      <Successor>
        <RuleName>freeRental</RuleName>
      </Successor>
    </Link>
    <Split>
      <Predecessor>
        <RuleName>freeRental</RuleName>
      </Predecessor>
      <Successors>
        <RuleName>freeRentalPoints</RuleName>
        <RuleName>checkAdvBooking</RuleName>
      </Successors>
    </Split>
  </RuleSubStruc>
</RuleStruc>

National Plant Diagnostic Network

The United States Department of Agriculture, Cooperative State Research, Education and Extension Service (USDA, CSREES) launched a multi-year national project in May 2002 to build the National Plant Diagnostic Network (NPDN) [49]. This was done with a view to
strengthen the homeland security protection for the nation’s food and agriculture by facilitating quick and accurate detection of disease and pest outbreaks in crops. Such outbreaks can occur naturally as foreign pathogens are introduced into the United States through mechanisms ranging from accidental importation to air-borne introduction by currents that traverse entire continents [63, 72]. They can also occur intentionally through deliberate introduction, as an act of bioterrorism [69]. In this nation-wide effort, the University of Florida was selected as the southern regional center of NPDN and is responsible for building a Southern Plant Diagnostic Network (SPDN) [61]. SPDN consists of a regional center system at the University of Florida and 12 state systems (counting Florida) and Puerto Rico connected to the regional center system through the Internet. The functions of the SPDN Regional Center are: to establish a network for the detection and diagnosis of plant health problems, extend and support sound public policies, implement environmentally sound prevention and management strategies, and provide leadership and training. Data collected from the region is sent to NPDN at Purdue University in Indiana. We intend to use this real-world example to demonstrate the applicability and usefulness of our system.

One of the plant diseases being monitored by USDA is Asian soybean rust [34], which is an aggressive fungal disease that can potentially reduce soybean yield by as much as 80 percent. Another disease is Sudden Oak Death, which is a disease capable of causing a range of symptoms from leaf spots to plant death on many woody hosts [68]. Plant growers and others (first detectors) may submit samples to an NPDN lab to be diagnosed. The lab runs preliminary diagnostic tests. If the results are positive, the lab staff seeks the help of a central authority designated for confirmation of the disease. We have obtained the diagnostic procedures from NPDN for the above two diseases. An analysis of these procedures reveals the existence of the
three types of knowledge rules and rule structure. We present some examples rules and rule structure from these procedures along with their representations in our knowledge specification language below. The derivation rule and the CAA rule have been obtained from [50], and the integrity constraint is obtained from the NPDN required fields list [29]. The rules obtained from [50] are described in Appendix C. The derivation rule mentioned here, is included as part of another CAA rule in the appendix to preserve the structure of the operating procedures as outlined by NPDN.

**Example Rules and Rule Structure from NPDN Plant Diagnostic Laboratories**

**Example Rule 1**

Rule: The suspect sample is reclassified from “suspect” to “presumptive positive” if it has been viewed by a diagnostician.

Name: NPDNR1

Type: Derivation Rule

Representation in Knowledge Specification Language: We include only the main XML element of interest “Implies” in the following.

```xml
<Implies>
  <Body>
    <IfExpr>
      <BooleanExpr>
        <PredicateExpr>
          <Expr>
            <Term>
              <DataItem>
                <Name>Sample.examined</Name>
                <Type>boolean</Type>
              </DataItem>
            </Term>
            <RelOp>eq</RelOp>
            <Expr>
              <Term>
                <Value valueType="boolean">true</Value>
              </Term>
            </Expr>
          </Expr>
        </PredicateExpr>
      </BooleanExpr>
    </IfExpr>
  </Body>
</Implies>
```
Example Rule 2

Rule: The valid/allowed values for the data item “Pest-Genus-Confirmation” are “Confirmed”, “Suspected”, “Inconclusive”, “Not Detected”.

Name: NPDNR2

Type: Integrity Constraint Rule

Representation in Knowledge Specification Language: We include only the main XML element of interest “IntCons” in the following.

<IntCons>
  <AttCons>
    <DataItem>
      <Name>PestGenusConfirmation</Name>
      <Type>String</Type>
    </DataItem>
    <EnumOp>in</EnumOp>
    <Value value Type="String">C</Value>
    <Value value Type="String">S</Value>
    <Value value Type="String">J</Value>
    <Value value Type="String">N</Value>
  </AttCons>
</IntCons>
Example Rule 3

Rule: Triage Lab staff, NPDN Regional Hub Lab staff and APHIS-PPQ Confirming Diagnosis

Designate may conduct live web-based distance diagnosis examination of sample and microscope mounts, if Triage Lab has this distance diagnosis capability. Or the diagnostician can take a digital image and email to the other two diagnosticians if web cam is not available

Name: NPDNR3

Type: Condition-Action-Alternative_Action Rule

Representation in Knowledge Specification Language: We include only the main XML element of interest “CAARule” in the following.

```xml
<CAARule>
  <Condition>
    <CondExpr>
      <IfExpr>
        <BooleanExpr>
          <PredicateExpr>
            <Expr>
              <Term>
                <DataItem>
                  <Name>distance_diagnosis_capability</Name>
                  <Type>boolean</Type>
                </DataItem>
              </Term>
            </Expr>
          <RelOp>eq</RelOp>
          <Expr>
            <Term>
              <Value valueType = "boolean">true</Value>
            </Term>
          </Expr>
        </PredicateExpr>
      </BooleanExpr>
    </IfExpr>
  </CondExpr>
  <Action>
    <Operation>
      <OperationName>Conduct_Web_Based_Diagnosis</OperationName>
    </Operation>
  </Action>
</CAARule>
```
Example Rule Structure

Rule Structure: If an NPDN Regional Hub or Expert Lab receives a presumptive positive sample, Expert Lab Staff acknowledges sample receipt to Triage Lab. NPDN Regional Hub or Expert Lab staff examines presumptive positive sample. NPDN Regional Hub or Expert Lab staff may contact Local expert, for additional input, but does not disclose state of origin. This is performed by the CAA rule “NHLR1”. Local Expert may examine sample. Local Expert may make preliminary diagnosis in collaboration with NPDN Regional Hub or Expert Lab staff. Local Expert contacts NPDN Regional Hub or Expert Lab Diagnostician with conclusions/results. This is performed by the CAA rule “NHLR2”. NPDN Regional Hub or Expert Lab Diagnostician contacts NPDN Regional Director and APHIS-PPQ-Confirming Diagnosis Designate with preliminary conclusions/results. NPDN Regional Hub or Expert Lab staff contacts own SPRO and APHIS-PPQ SPHD to inform them that a presumptive positive sample is housed in NPDN Regional Hub or Expert Lab until shipment to APHIS-PPQ Confirming Designate or until it is destroyed following diagnosis. The state of origin is not disclosed to NPDN Regional Hub or Expert Lab’s state SPRO or SPHD. NPDN Regional Hub or Expert Lab staff contacts its own
Campus Safety Officer to inform of presumptive positive sample in the lab. State of origin is not disclosed. This is performed by CAA rule “NHLR3”. If APHIS-PPQ Confirming Diagnosis Designate requests NPDN Regional Hub or Expert lab to send sample for confirmation, then NPDN Regional Hub or Expert Lab contacts APHIS-PPQ Confirming Diagnosis Designate and Triage lab indicating that they are sending the presumptive positive sample to APHIS-PPQ Confirming Diagnosis Designate Lab including shipment date, method, tracking number and sample number. This is performed by CAA rule “NHLR4”.

Representation in Knowledge Specification Language: Rule NHLR1 is executed first, followed by rules NHLR2 and NHLR3 in parallel. After both NHLR2 and NHLR3 finish execution, rule NHLR4 is executed. We include the main XML element of interest “RuleStruc” in the following.

```xml
<RuleStruc>
  <Name>NPDN Regional Hub Lab Procedure</Name>
  <RuleStrucDescription>Procedure followed by the NPDN Regional Hub Lab</RuleStrucDescription>
  <RuleStrucState>ACTIVE</RuleStrucState>
  <Sharing>global</Sharing>
  <RuleSubStruc>
    <Split>
      <Predecessor>
        <RuleName>NHLR1</RuleName>
      </Predecessor>
      <Successors>
        <RuleName>NHLR2</RuleName>
        <RuleName>NHLR3</RuleName>
      </Successors>
    </Split>
  </RuleSubStruc>
  <RuleSubStruc>
    <AndJoin>
      <Predecessors>
        <RuleName>NHLR2</RuleName>
        <RuleName>NHLR3</RuleName>
      </Predecessors>
      <Successor>
        <RuleName>NHLR4</RuleName>
      </Successor>
    </AndJoin>
  </RuleSubStruc>
</RuleStruc>
```
Figure 3-1. Example rule structures

A = AND-JOIN
L = LINK
O = OR-JOIN
S = SPLIT
In Chapter 1, we mentioned our idea of translating different types of rules and rule structures at rule definition time into code and wrapping this code as web services for their uniform registration, discovery and invocation in a web service infrastructure. In this chapter, we present the algorithms for automatically converting the three types of rules and rule structures to web services.

**Creating a Web Service**

The W3C group defines web services as follows.

*A web service is a software application identified by a uniform resource identifier (URI), whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols.* [60].

Since the introduction of the web services concept, the software industry has provided many tools to wrap application code as a web service. Software developers can thus focus on developing the application code instead of worrying about conformity with the web service protocols laid out by W3C [11]. Therefore, the most significant task in creating a web service is the development of application code to be exposed as a web service.

Figure 4-1 gives an overview of the algorithm for web service synthesis. It takes as input a rule specification in XML that conforms to the knowledge specification language presented in Chapter 3. The algorithm identifies the type of the rule and invokes appropriate handler methods to create the source code for the rule (steps 3-5). This code is generated as an interface (or header) file and an implementation (or source) file. The implementation file contains the translation of the rule to code. 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. In the following, we use the terms *method* and *operation* interchangeably.

For successful compilation and deployment of a web service, we need to create the appropriate WSDL document. This is done with the help of configuration files, the exact type and nature of which depend on the application framework used (steps 6-8). To facilitate the discovery of a web service using UDDI, we also publish the web service to a private UDDI registry (step 9).

**Algorithms for Different Rule Types**

In the following sections we describe the algorithms used to create the web services for each of the three rule types. The basic idea in the algorithms is to create the code (using the Java programming language) that captures the intent of the rule. Thus, we build the interface and implementation files line by line. We use the pseudo code statement *write “A”* to indicate that program statement *A* should be written out to either the interface or the implementation file, as specified. When navigating through the XML rule document specified, we use the construct *elem1 -> elem2* to refer to the XML child element *elem2* of the XML element *elem1*. We use the term *method* to refer to the web service operation generated for the specific rule type, and the term *algorithm* to refer to the algorithm for creating the method.

**Integrity Constraints**

Each integrity constraint rule is represented as a web service with the operation/method *check(...)*. The algorithm determines the type of the constraint and generates the appropriate program statements for *check*, details of which are given in Figure 4-2. The method *check(...)* examines the supplied input data and returns a Boolean value which is *true* if the specified
constraint is satisfied and \textit{false} otherwise. Specific cases to handle comparison operators, enumeration operators, formula constraints and condition constraints are shown in the algorithm. All data items referenced in the constraint rule constitute the input to \textit{check}(\ldots).

If the constraint type is an attribute constraint, the rule code checks if the relationship specified by the comparison or the enumeration operator holds. If the constraint is an inter-attribute constraint of the formula constraint subtype, the algorithm first generates code for the expressions on the left hand side and the right hand side as \textit{lExprCode} and \textit{rExprCode}, respectively. The method \textit{check}(\ldots) then determines if the \textit{lExprCode} is related to \textit{rExprCode} as specified by \textit{op}, where \textit{op} is the comparison operator used. For a condition constraint, the algorithm generates code for the if-part and the then-part as \textit{ifPartCode} and \textit{thenPartCode}, respectively. The method \textit{check}(\ldots) then determines if \textit{ifPartCode} is true. If so, it determines if \textit{thenPartCode} also holds true.

\textbf{Derivation Rules}

Each derivation rule is translated to a web service with the operation/method \textit{implies}(\ldots). This method examines the input data to determine if the body of the implication is true. If so, it returns the new data specified in the head of the implication. Figure 4-3 shows the algorithm for creating the method.

The input parameters to the method \textit{implies}(\ldots) are the data items required by the body of the implication and the data items referenced in the “value” part of the head. The body contains one or more Boolean expressions linked by logical operators \textit{and} and \textit{or}. The \textit{getBodyVal}(\ldots) function translates the implication body into program statements. Each Boolean expression in the implication body makes use of the function \textit{getExprCode}(\ldots) from Figure 4-2 to generate the code for the expression. The head of the implication contains one or more \textit{Atom} elements linked by the \textit{and} operator. Each of these elements contains a derived data item and the corresponding
value. This value can be a constant value or another data item. The function `getHeadMap(…)` constructs a hashmap named `headVal`, which contains the new data as a `(name, value)` pair indexed by `name`. If the body evaluates to true, the output contains each pair in `headVal`, otherwise it is null.

**CAA Rules**

Each CAA rule is translated to a web service with the operation/method `perform(…)`. If the guard clause evaluates to true, this method examines the input data to see if the condition is satisfied. If so, the operations specified in the action clause are executed, and the output of those operations is returned, otherwise the operations specified in the alternative_action clause are executed and the output of those operations is returned. Figure 4-4 gives the algorithm for creating the method `perform(…)`.

The input parameters to the rule are the data items referenced in the condition clause, and the data items specified as input to the operations in the action and alternative_action clauses. The output data from the operations in the action clause are added to the hashmap `actionData`, and the output data from the operations in the alternative_action clause are added to the hashmap `altActionData`. Both these maps contain `(name, value)` pairs, indexed by `name`. The function `getConditionCode(…)` translates the condition clause to program statements. Similarly, the functions `getActionCode(…)` and `getAltActionCode(…)` translate the action and alternative_action clauses to code, respectively. The output of the method `perform(…)` is a hashmap named `map`, which contains the result of the operations performed by the rule. At runtime, either the action or the alternative_action clause is executed, and `map` contains the corresponding output.

The function `getActionCode(…)` is shown in some detail in Figure 4-4B. `getAltActionCode(…)` is very similar and is not shown separately. An action (or alternative_action) clause consists of a list of operations to be executed. The function translates
each operation to program statements of the form \( \text{output} = \text{operation name} (\text{input}) \), or \( \text{operation name} (\text{input}) \) depending upon whether the operation contains return parameters or not. In case the operation returns multiple data items, we create an output class that captures all of these data items as the class’ data members. All such program statements are stored in the array \( \text{actionVal} \) (or \( \text{altActionVal} \)), which is returned.

**Rule Structure**

A rule structure links rules together in a directed acyclic graph as explained in Chapter 3. It is defined and published as a web service just like any other rule. We require the following condition to be met to guarantee that the generated web service for a rule structure will be executable.

*Each of the rules in a rule structure must have already been defined and published as a web service before the rule structure is defined.*

Each rule structure is represented as a web service with the operation/method \( \text{execute}(\ldots) \). Figure 4-5 shows the algorithm for creating \( \text{execute}(\ldots) \). This method takes in all the input data items required by all the rules in the structure, and returns the output data items produced by all these rules. The objective of the method is to invoke the rules in the order specified by the structure. To facilitate this, it creates *invoker threads* for each rule in the structure. The algorithm to create such an invoker thread (\( \text{createInvokerThread}(\ldots) \)) is also shown in the figure.

The algorithm \( \text{rsHandler}(\ldots) \) breaks down the given rule structure into substructures. Details of how a given rule structure is partitioned into substructures and how such partitioning aids rule structure execution will be presented in Chapter 5. Each substructure contains a *link*, a *split*, an *and-join*, or an *or-join* relationship between two or more rules. The algorithm generates invoker threads for every distinct rule in the substructure. Depending on the substructure type, one of the four *handler* functions is called. Each of these handler functions is responsible for
generating code that will cause the rule execution in the manner specified by the corresponding substructure type. The method `execute(...)` is then constructed by putting together code generated by each of the handler routines. The method `execute(...)` maintains an array `toBeExecuted` to keep track of those rules that have yet to begin execution.

`linkHandler(...)` generates program statements to ensure that a successor rule $r_2$ is executed only after the predecessor rule $r_1$ has finished execution as follows. First, the code creates an invoker thread object for $r_1$. If this is a newly created thread, $r_1$ is included in `toBeExecuted`, if not already put in, and starts the thread for $r_1$. Once this rule has finished execution, it copies the output items from $r_1$ to the rule structure’s member variables, and removes $r_1$ from `toBeExecuted`. It creates an invoker thread for $r_2$, and puts it in `toBeExecuted` if this is a newly created thread. Once $r_1$ finishes execution, $r_2$ is allowed to proceed. Once $r_2$ has finished execution, it is removed from `toBeExecuted`, and its output copied to member variables.

Similarly, `splitHandler(...)` generates code to invoke the successor rules only after the predecessor rule has been executed, and `JoinHandler(...)` generates code to execute the successor rule only after all the predecessor rules have been executed, and `orJoinHandler(...)` generates code to execute the successor rule only after the specified number of predecessor rules have been executed. The algorithms for `splitHandler` and `orJoinHandler` are included in Figure 4-5.
Algorithm 1 createWebService

1) \( rules = \text{getRules}(\text{ruleBase}) \);
2) \textbf{for} each rule \( r \) \textbf{do}
3) \quad \textbf{if} (\( r \) is an integrity constraint) \( \text{icHandler}(r) \); \textbf{end if}
4) \quad \textbf{else if} (\( r \) is a derivation rule) \( \text{drHandler}(r) \); \textbf{end if}
5) \quad \textbf{else if} (\( r \) is a CAA rule) \( \text{caaHandler}(r) \); \textbf{end if}
6) \quad \text{createConfigFiles}(r);
7) \quad \text{compile}(r);
8) \quad \text{deploy}(r);
9) \quad \text{publish}(r);
10) \textbf{end for}

Figure 4-1. Algorithm for web service synthesis
Algorithm 2 icHandler

1. \( \text{paramNames} = \text{null}, \text{paramTypes} = \text{null}; \text{stmts} = \text{null}; \)
2. \( \text{if} \ \text{constraint} \ c \ \text{is an \ attribute \ constraint} \ \text{do} \)
3. \( d = c \rightarrow \text{DataItem}; \)
4. \( \text{add} \ d \rightarrow \text{Name} \ \text{and} \ d \rightarrow \text{Type} \ \text{to} \ \text{paramNames} \ \text{and} \ \text{paramTypes} \ \text{respectively} \)
5. \( \text{op} = c \rightarrow \text{getNextChild()}; \)
6. \( \text{if} \ \text{op} \ \text{is a \ comparison \ operator} \ \text{do} \)
7. \( \text{add} \ \text{“if} \ d \rightarrow \text{Name} \ \text{op} \ c \rightarrow \text{Value} \ \text{return \ true;}” \ \text{to} \ \text{stmts} \)
8. \( \text{end \ if} \)
9. \( \text{if} \ \text{op} \ \text{is the \ enumeration \ operator} \ \text{in} \)
10. \( \text{values} = \text{getValues}(c); \)
11. \( \text{add} \ \text{“if} \ d \rightarrow \text{Name} \ \text{in} \ \text{values} \ \text{return \ true;}” \ \text{to} \ \text{stmts} \)
12. \( \text{end \ if} \)
13. \( \text{if} \ \text{op} \ \text{is the \ enumeration \ operator} \ \text{not \ in} \)
14. \( \text{values} = \text{getValues}(c); \)
15. \( \text{add} \ \text{“if} \neg(d \rightarrow \text{Name}) \ \text{in} \ \text{values} \ \text{return \ true;}” \ \text{to} \ \text{stmts} \)
16. \( \text{end \ if} \)
17. \( \text{else \ if} \ \text{constraint} \ c \ \text{is an \ inter-attribute \ constraint} \ \text{do} \)
18. \( \text{dataItemList} = \text{getInputData}(c); \)
19. \( \text{for} \ \text{each} \ \text{DataItem} \ d \ \text{in} \ \text{dataItemList} \ \text{do} \)
20. \( \text{add} \ d \rightarrow \text{Name} \ \text{and} \ d \rightarrow \text{Type} \ \text{to} \ \text{paramNames} \ \text{and} \ \text{paramTypes} \ \text{respectively} \)
21. \( \text{end \ for} \)
22. \( \text{if} \ c \ \text{is a \ formula \ constraint} \ \text{do} \)
23. \( \text{lExprCode} = \text{getExprCode}(c \rightarrow \text{Expr}); \)
24. \( \text{op} = c \rightarrow \text{RelOp}; \)
25. \( \text{rExprCode} = \text{getExprCode}(c \rightarrow \text{Expr}); \)
26. \( \text{add} \ \text{“if} \ \text{lExprCode} \ \text{op} \ \text{rExprCode} \ \text{return \ true;}” \ \text{to} \ \text{stmts} \)
27. \( \text{end \ if} \)
28. \( \text{else \ if} \ c \ \text{is a \ condition \ constraint} \ \text{do} \)
29. \( \text{ifPartCode} = \text{getIfExprCode}(c \rightarrow \text{IfPart}); \)
30. \( \text{thenPartCode} = \text{getThenExprCode}(c \rightarrow \text{ThenPart}); \)
31. \( \text{add} \ \text{“if} \ (\text{ifPartCode} = \text{true} \ \text{and} \ \text{thenPartCode} = \text{true}) \ \text{return \ true;}” \ \text{to} \ \text{stmts} \)
32. \( \text{end \ else} \)
33. \( \text{end \ else} \)
34. \( \text{createServiceInterfaceFile(} \text{paramNames, paramTypes} \text{);} \)
35. \( \text{createServiceImplementationFile(} \text{paramNames, paramTypes, stmts} \text{);} \)

Function: \text{createServiceImplementationFile(} \text{paramNames, paramTypes, stmts} \text{)}

1. \( \text{for} \ \text{each stmt} \ s \ \text{in} \ \text{stmts} \ \text{do} \)
2. \( \text{write} \ s \ \text{to file} \)
3. \( \text{end \ for} \)
4. \( \text{write} \ \text{“return \ true;}” \ \text{to file} \)

Figure 4-2. Algorithm for integrity constraint rules
**Algorithm 3 drHandler**

(1) \( \text{paramNames} = \text{null}, \text{paramTypes} = \text{null}; \)
(2) \( \text{body} = \text{implies} \rightarrow \text{Body}; \)
(3) \( \text{head} = \text{implies} \rightarrow \text{Head}; \)
(4) \( \text{dataItemList} = \text{getInputData(body, head)}; \)
(5) \( \text{for each DataItem } d \text{ in dataItemList do} \)
(6) \( \quad \text{add } d \rightarrow \text{Name and } d \rightarrow \text{Type to paramNames and paramTypes respectively.} \)
(7) \( \text{end for} \)
(8) \( \text{bodyVal} = \text{getBodyCode(body)}; \)
(9) \( \text{headVal} = \text{getHeadCode(head)}; \)
(10) \( \text{createServiceInterfaceFile(paramNames, paramTypes)}; \)
(11) \( \text{createServiceImplementationFile(paramNames, paramTypes, bodyVal, headVal)}; \)

**Function: getHeadCode(head)**

(1) \( \text{headVal} = \text{null}; \)
(2) \( \text{atomList} = \text{getAtoms(head)}; \)
(3) \( \text{for each Atom } a \text{ in atomList do} \)
(4) \( \quad \text{dataItem} = a \rightarrow \text{DataItem}; \)
(5) \( \quad \text{value} = a \rightarrow \text{getNextChild}(); \)
(6) \( \quad \text{add the pair (dataItem, value) to headVal} \)
(7) \( \text{end for} \)
(8) \( \text{return headVal}; \)

**Function: createServiceImplementationFile(paramNames, paramTypes, bodyVal, headVal)**

(1) \( \text{write “map = null;” to file} \)
(2) \( \text{write “if (bodyVal) then”} \)
(3) \( \text{for each pair (dataItem, value) in headVal do} \)
(4) \( \quad \text{write “map.put(dataItem, value);” to file} \)
(5) \( \text{end for} \)
(6) \( \text{write “end-if” to file} \)
(7) \( \text{write “return map;” to file} \)

Figure 4-3. Algorithm for derivation rules
Algorithm 4 caaHandler

(1) \( paramNames = \text{null}, \ paramTypes = \text{null}, \ actionMap = \text{null}, \ altActionMap = \text{null}; \)
(2) \( actionData = \text{null}, \ altActionData = \text{null}; \)
(3) \( condition = \text{caarule} \rightarrow \text{Condition}; \)
(4) \( condExpr = \text{condition} \rightarrow \text{CondExpr}; \)
(5) \( action = \text{caarule} \rightarrow \text{Action}; \)
(6) \( altAction = \text{caarule} \rightarrow \text{AlternativeAction}; \)
(7) \( dataItemList = \text{getInputData}(condition, \ action, \ altAction); \)
(8) \( \text{for each DataItem} \ d \text{ in dataItemList do} \)
(9) \( \quad \text{add} \ d \rightarrow \text{Name and} \ d \rightarrow \text{Type to paramNames and paramTypes respectively} \)
(10) \( \text{end for} \)
(11) \( dataItemList = \text{getInputData}(action, \ altAction) \)
(12) \( \text{for each DataItem} \ d \text{ in dataItemList do} \)
(13) \( \quad \text{add} \ d \rightarrow \text{Name and} \ d \rightarrow \text{Type to paramNames and paramTypes respectively}. \)
(14) \( \text{end for} \)
(15) \( dataItemList = \text{getOutputData}(action); \)
(16) \( \text{for each DataItem} \ d \text{ in dataItemList do} \)
(17) \( \quad \text{add} \ d \rightarrow \text{Name and} \ d \rightarrow \text{Type to actionData} \)
(18) \( \text{end for} \)
(19) \( dataItemList = \text{getOutputData}(altAction); \)
(20) \( \text{for each DataItem} \ d \text{ in dataItemList do} \)
(21) \( \quad \text{add} \ d \rightarrow \text{Name and} \ d \rightarrow \text{Type to altActionData} \)
(22) \( \text{end for} \)
(23) \( conditionVal = \text{getCondCode}(condition); \)
(24) \( actionVal = \text{getActionCode}(action); \)
(25) \( altActionVal = \text{getAltActionCode}(altAction); \)
(26) \( \text{createServiceInterfaceFile}(paramNames, \ paramTypes); \)
(27) \( \text{createServiceImplementationFile}(paramNames, \ paramTypes, \ conditionVal, \ actionVal, \ altActionVal, \ actionData, \ altActionData); \)

Figure 4-4. Algorithm for condition-action-alternative_action rules. A) Description of the overall algorithm
Figure 4-4. Algorithm for condition-action-alternative_action rules. B) Description of supplementary functions
Figure 4-5. Algorithm for rule structures. A) Description of the overall algorithm and the functions for creating a caller routine and handling the ‘split’ relationship

```
Algorithm 5 rsHandler

(1) ruleStrucList = getRuleSubStruc(ruleStruc);
(2) ssVal = null;
(3) distinctRules = get distinct rules referred in ruleStruc
(4) for each rule r in distinctRules do
(5)    createCallerRoutine(r)
(6) end for
(7) for each RuleSubStruc rss in ruleStrucList do
(8)    if rss is a Link ssVal += linkHandler(rss → Link); end if
(9)    if rss is a Split ssVal += splitHandler(rss → Split); end if
(10)   if rss is an ANDJoin ssVal += andJoinHandler(rss → ANDJoin); end if
(11)   if rss is an ORJoin ssVal += orJoinHandler(rss → ORJoin); end if
(12) end for
(13) createServiceInterfaceFile();
(14) createServiceImplmentationFile(ssVal);

Function: createInvokerThread (Rule r)

(1) write “call”+r.name+” extends Thread”, thus creating a caller class for the rule
(2) dataItemList = getInputAndOutput(r);
(3) for each DataItem d in dataItemList do
(4)    declare a corresponding public data member with same name and type
(5) end for
(6) write “public void run()” to file
(7) prepare for rule web service call and write it out to the file
(8) write code to call the rule web service to the file
(9) for each DataItem d generated by the web service do
(10)   copy d’s the value to the corresponding data member of the class
(11) end for

Function: splitHandler(Split s)

(1) pred = s → Predecessor;
(2) succList = s → Successors;
(3) splitVal += “callPred = invoker thread to call pred;”
(4) splitVal += “for each Rule succ in succList do”;
(5) splitVal += “callSucc = invoker thread to call succ;”;
(6) splitVal += “end for”;
```
Figure 4-5. Algorithm for rule structures. B) Description of the function for handling the OR join relationship

Function: orJoinHandler(ORJoin o)

   predList = o → Predecessors;
   succ = o → Successor;
   orJoinVal = “for each Rule pred in predList do”
   orJoinVal += “callPred = invoker thread to call pred;”
   orJoinVal = “if callPred.state == NEW and callPred in toBeExecuted”
   orJoinVal += “add callPred to toBeExecuted;”
   orJoinVal += “Start callPred;”
   orJoinVal += “end if”;  
   orJoinVal = “if callPred.state == TERMINATED and callPred in toBeExecuted”
   orJoinVal += “remove callPred from toBeExecuted;”
   orJoinVal += “copy output of rule to member variables;”
   orJoinVal += “end if”;  
   orJoinVal = “for each Rule succ in succList do”
   orJoinVal += “if callSucc.state == NEW and callSucc not in toBeExecuted”
   orJoinVal += “add callSucc to toBeExecuted;”
   orJoinVal += “Start callSucc;”
   orJoinVal += “end if”;  
   num = o → Num; count = 0;
   orJoinVal = “for each Rule pred in predList do”
   orJoinVal += “if callPred.state == TERMINATED and callPred in toBeExecuted”
   orJoinVal += “remove callPred from toBeExecuted;”
   orJoinVal += “copy output of rule to member variables;”
   orJoinVal += “end if”;  
   orJoinVal += “end for”;  
   orJoinVal += “end for”;

return orJoinVal;
Figure 4-5. Algorithm for rule structures. C) Description of the function to generate the web service implementation file

(20)  `orJoinVal += “callSucc = invoker thread for succ;”`
(21)  `orJoinVal += “if callSucc.state == NEW and callSucc not in toBeExecuted”`
(22)  `orJoinVal += “add callSucc to toBeExecuted;”`
(23)  `orJoinVal += “end if”`
(24)  `orJoinVal += “if callSucc.state == NEW and count >= num”`
(25)  `orJoinVal += “Start callSucc;”`
(26)  `orJoinVal += “end if”`
(27)  `orJoinVal += “if callSucc.state == TERMINATED and callSucc in toBeExecuted”`
(28)  `orJoinVal += “remove callSucc from toBeExecuted;”;
(29)  `orJoinVal += “copy output of rule to member variables;”`
(30)  `orJoinVal += “end if”;`
(31)  `return orJoinVal;`

Function: createServiceImplementation(ssVal)

(1)    write “toBeExecuted == null” to file;
(2)    write “do” to file
(3)    for each path execution p in ssVal
(4)    write p to file
(5)    end for
(6)    write “while toBeExecuted is not empty” to file
CHAPTER 5
SYSTEM ARCHITECTURE AND RULE PROCESSING

In this chapter, we describe our system architecture and implementation framework. We use example scenarios from the two application domains described in Chapter 3 to explain the event-triggered processing of distributed rules, rule structures and application operations.

**System Architecture**

The distributed event- and rule-based system called ETKnet has a peer-to-peer server architecture (Figure 5-1). All collaborating organizations have identical subsystems installed at their sites. Each site creates and manages its own events, rules, rules structures, triggers and operations, but their specifications are registered at the host site of a collaborative 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. The *event server* component is responsible for storing information about events defined at that particular site and information about event subscribers. A collaborating site can specify a trigger linking a distributed event to a rule or a 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 the occurrence of an event. Distributed rules and rule structures are invoked and processed by replicas of the rule server. These rules and rule structures may invoke automated system operations and manual operations of collaborating sites. 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 integration and organization of the dynamic event data associated with the event occurrence.

In a collaborative federation, since events, rules, triggers and operations can be defined by different collaborating organizations, terminology becomes an important issue. Specific terms used by an organization in its event, rule, trigger and operation specifications and in metadata descriptions can be quite different from those used by another organization. People searching for registered events and web services need some form of common ontology to resolve these discrepancies. It would be beneficial to define an ontology for a particular application domain in a collaborative federation. Thus, the host site in the system architecture also incorporates an ontology database to contain the terms used in event, rule, trigger and operation specifications, in event data and in metadata of these information and knowledge resources, and their mappings to concepts and concept associations defined in the domain ontology. The architecture also includes an ontology manager to resolve discrepancies and identify similarities between the specified terms, and to facilitate search. These components are included to give the reader a complete picture of the collaborative system. It should be noted that the design and implementation of such an ontology database manager is beyond the scope of this work.

The user interface tools at the host and the collaborating sites aid users in the definition and maintenance of events, rules, rule structures, triggers and operations. These tools use the local security and access control policies to ensure that only authorized users get access to the system. These tools also provide the facility for creating and maintaining the domain ontology. However, their design and implementation is not one of this author’s contributions. They have
been implemented by one of the members in the group working on application of the event and rule processing ideas in the NPDN domain.

We have implemented the algorithms presented in Chapter 4 for converting knowledge rules and rule structures to web services in Java, with the Sun Java System Application Server Platform Edition 9.0 [66] as our application server. The event and rule servers have been implemented using the Enterprise JavaBeans 2.1 framework [46]. To facilitate easy and efficient lookup, we publish the deployed web services to our private UDDI registry. This registry makes use of the Apache jUDDI project [73] to communicate with MySQL Server 5.0 [48] 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 security, as it is available only to the organizations participating in a collaborative federation.

The Sun Application Server provides many tools which facilitate the easy development of a web service. The bulk of the work is in generating the rule-specific source code. In addition, the following configuration files are needed to deploy the web service: config.xml, web.xml, sun-web.xml, and jaxrpc-ri.xml. The createConfigFiles function (Figure 4-1) uses the information from the service interface file to generate these configuration files. The web service is then compiled using the wscompile tool of the application server. This generates the WSDL description file and the service is then deployed using the wsdeploy tool of the application server. Once the service is deployed, it is published at the host site using the UDDI4J [67] API.

The general event and rule processing strategy is as follows. When an event occurs at a collaborating site, the event data are first captured in an XML document. The site of occurrence
now becomes the coordinating site (or coordinator) for this particular event processing scenario. The coordinator needs to determine the sites that contain applicable rules and/or rule structures (i.e., applicable sites) so that event data can be sent to them. Applicable sites are of two types. The first is those sites that have defined explicit triggers linking the event to one or more rules and/or rule structures. The second type is those sites that do not define explicit triggers, but do have rules and/or rule structures whose input data items are a subset of the event data items (i.e., applicable rules and/or rule structures). Such rules and/or rule structures can contribute to the event data and thus should be processed. Our system generates implicit triggers for the second type of applicable sites, details of which are described later.

To determine the applicable sites, the coordinator queries the host event server passing the event data as the query input. The host site first examines the explicit triggers to determine the first type of applicable sites. For the remaining sites in the federation, the host site examines the published rule information to determine if any site contains applicable rules and/or rule structures. Once the coordinator receives this site information, it sends the XML document containing the event data to these sites’ event servers. These event servers pass the event data to their corresponding local rule servers. Each local rule server carries out a three-step process. In the first step, it examines the explicit triggers and records the rule and rule structures that need to be processed. In the next step, it examines the event data and determines the rules and/or rule structures whose input is a subset of the event data. These rules and/or rule structures are also recorded. In the last step, it examines the recorded rules to see if any of them will be executed as part of a rule structure. A rule structure is specified by a decision-maker, and captures the required order of execution for the rules that participate in the structure. Thus, any applicable rule that is already part of an applicable rule structure, if processed individually, will result in the
same rule being processed twice. Also, the processing of individual rules may not honor the order of execution. Thus, applicable rules that will be processed as part of an applicable rule structure are removed, and the remaining applicable rules and/or rule structures are processed.

The processing of these applicable rules and rule structures may add data to or update the data in the event data document. They may also activate automated or manual operations that add or modify the event data document. Each applicable site returns its possibly updated event data document to the coordinator. The coordinator then merges all the event data documents to create a new version of the event data document to be sent out to applicable sites in the next round. During the data merging process, the coordinator identifies and separates event data information into old and new. It also detects inconsistencies and determines if any cyclic rules will be processed in the next round. Details of event data aggregation, detecting inconsistencies and avoiding cyclic rule execution will be presented in Chapter 6. At the beginning of every round of event data transmission and rule processing, the coordinator asks the host site for the applicable sites and sends the new version of event data document to them. There is one difference, however, in determining applicable sites as well as applicable rules and rule structures in the second and subsequent rounds. To determine whether a particular rule or rule structure is applicable in a round other than the first one, we also employ the condition that at least one of the rule’s or rule structure’s input data items must have been added/modified in the previous round. This is to prevent the processing of the same set of rules and rule structures on the same event data.

Once no more applicable sites can be determined, the process of event data transmission and rule processing terminates. All sites involved in this process would have received the final
event data document that contains all the data that are relevant to an event occurrence. The data can be used by these organizations for further decision-support and problem-solving.

**Event-triggered Processing of Rules and Rule Structures in the EU-Rent Domain**

In this section, we use an example scenario of the EU-Rent application domain to explain how the general event and rule processing strategy described above can be applied. We shall also explain how a rule structure is decomposed and processed. The complete set of events, rules, and rule structures derived from the EU-Rent rule set is included in Appendix B.

We have used our knowledge specification language to define EU-Rent’s rules and rule structures, convert them to code, and register them as web services. The total number of rules we obtained from [38] is 46, 29 of which are CAA rules, 13 of which are integrity constraint rules and 4 of which are derivation rules. There were also 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. The time taken to generate the appropriate service interface, implementation and configuration files has little impact on the total time. As a result, the web service creation time is more or less independent of the rule type.

To demonstrate the event-triggered processing of distributed rules and rule structures, we use the following scenario (Figure 5-2). A customer approaches a local branch *Branch 1* with the request for a car rental. *Branch 1* is unable to satisfy his/her request. It posts an event *Reservation Request* and the event data in XML format is sent to all subscribing branches. The event data contains information about the customer and the type of car desired. Assume that *Branch 2*, *Branch 5* and *Branch 8* are branches that have the applicable rule structure shown in detail for
Branch5. As all the EU-Rent branches have similar rules, a similar rule structure is defined at each one. Details of the rules that are used in the rule structure are given in Appendix B.

The rule structure checks if the driver of the car satisfies some required constraints by means of the rule driverCheck. In parallel, a check is made to ensure that the branch will not exceed its capacity by approving the request through the rule capacityCheck. If no group or model is specified, a default group of ‘A’ is considered due to the derivation rule noGroupAndModelSpecified. A suitable car is determined based on the group or model through the CAA rule assignCarOnModelOrGroup. If a customer is in the company’s loyalty incentive scheme, or a suitable car cannot be found, an upgrade is given through the CAA rule giveUpgrade. If even then, a suitable car cannot be found, availability of a car from the next day’s reserved quota is determined, suitability of a bumped upgrade or a downgrade is considered, or availability of a car scheduled to be returned earlier on the pick-up day is determined.

The event data file contains new data resulting from the application of these rules at Branch2. This data is returned to Branch1 (the coordinating site of the event). Branch1 receives the updated event data from Branch2, Branch5, and Branch8 (not shown). Branch1 then merges this data and sends out the new version of the event data to all branches (not shown) because there may be rules and rules structures that are applicable to the new version of event data. In our scenario, we assume that there are no new applicable rules. Branch1 can then apply a local rule to select the branch among the three which preferably has the requested make and model available and is also the closest. Also, all other branches receive 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.
Decomposition of Rule Structure

We use the same scenario 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 rule structure constructs, if considered independently, can be represented using a tree (by means of the predecessor-successor relationships), but the combination of these constructs that forms a particular rule structure may not always be a tree (Figure 5-2). 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 by using XML. This does result in the same rule being referred at a maximum of two times, first when describing the relationship where this rule is a successor, and the next when describing the relationship where this rule is a predecessor. Rules that have no predecessors or successors still appear only once in the representation.

The directed edges of a rule structure specify the order of execution of the constituent rules. This order is reflected in how the XML elements representing each substructure are ordered in the rule structure document. For a given rule structure, substructures are generated as we move from the top to the bottom of the graph. At any level, we follow a left to right ordering, thus if substructure $S_1$ is to the left of substructure $S_2$, $S_1$ will be generated first. Taking the rule structure from Figure 5-2 as an example, the first substructure to be generated would be for the link construct with getBranch as the predecessor rule and capacityCheck as the successor rule. The second substructure generated would be for the and-join with driverCheck and capacityCheck as the predecessor rules and noGroupAndModelSpecified as the successor rule. After two link structures linking noGroupAndModelSpecified with assignCarOnModelOrGroup,
and assignCarOnModelOrGroup with giveUpgrade, the final substructure would be the split construct with giveUpgrade as the predecessor rule and allocateNextDay, bumpedUpgrade, downgrade, assignScheduledCar as the successor rules. The above scenario takes about a total of 3 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.

If we revisit the algorithm for converting a rule structure to a web service (Figure 4-6), we see that substructures are processed in the order specified in the rule structure document, which reflects the actual order of the rule structure. The program statements for each substructure are generated by the respective handler routines. Each routine checks if the toBeExecuted array contains the predecessor rule(s). This check is important since the predecessor rule(s) for a particular substructure might be the successor rule(s) of an earlier substructure, and thus may already be either executed or at least in the pipeline. This check in combination with the proper ordering of the substructures ensures that rules are always processed in the correct order during the actual execution of the rule structure.

Decomposing a complicated rule structure into simpler substructures also facilitates the maintenance of the structure. 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 and modifying a structural relationship need to address only the substructure that represents that relationship, without affecting other parts of the rule structure document.
Event-triggered Processing of Rules and Rule Structure in the NPDN Domain

In this section, we provide more details about the NPDN application domain and use an example scenario from this domain to describe how the general event and rule processing strategy is applied.

The key organizations in the NPDN environment and their functions [50] are given below:

- **NPDN Triage Lab:** The state facility designated to receive and examine suspect samples. This lab is often associated with the state Land Grant University, but in some states is part of the state department of agriculture.

- **NPDN Regional Hub Lab:** The key coordinating lab for an NPDN region. Currently, these labs are located at the California Department of Food and Agriculture (WPDN), Kansas State University Department of Plant Pathology (GPDN), Cornell University Department of Plant Pathology (NEPDN), Michigan State University Department of Plant Pathology (NCPDN), and University of Florida Department of Plant Pathology (SPDN). These labs provide coordination, training, funding, and surge capacity support to the NPDN triage labs within their region and occasionally to other regions.

- **APHIS-PPQ:** Animal and Plant Health Inspection Service, Plant Protection and Quarantine. Administered by the USDA.

- **SPRO:** State Plant Regulatory Official. Highest ranking state plant regulatory official. The SPRO is employed by the state department of agriculture.

- **SPHD:** APHIS-PPQ State Plant Health Director. Highest ranking federal plant regulatory official in a state.

- **APHIS-PPQ-NIS:** National Identification Service. The USDA-authorized lab for diagnosing plant diseases (fungal and viral).

- **APHIS-PPQ-CPHST:** Center for Plant Health, Science and Technology. The USDA-authorized lab for conducting DNA diagnosis (PCR) and bacterial diagnosis of plant diseases.

- **APHIS-PPQ Confirming Diagnosis Designate:** The person authorized to make a confirming diagnosis for a high risk pest. This diagnosis must withstand legal scrutiny if challenged in court. This lab may be one of the APHIS-PPQ labs (NIS or CPHST) in Beltsville, MD or may be one that has been approved or provisionally approved by APHIS-PPQ or APHIS-CPHST.

Our study of the procedures used in this domain for diagnosis of plant samples has revealed the need for the NPDN organizations to communicate with each other effectively and in
a timely manner. We provide the solution approach of capturing the procedures as rules and rule structures to be triggered by suitable events. The complete set of events, rules, rule structures, triggers and operations along with their descriptions, the data items they need or provide, as obtained from the NPDN SOP draft [50] are included in Appendix C.

**Distributed Rule and Rule Structure Processing in the NPDN domain**

We have used our knowledge specification language to define the NPDN domain’s rules and rule structures, convert them to code, and register them as web services. The total number of rules we obtained from [40] is 27, all of which are CAA rules. We have identified 10 different events and 5 rule structures. The time taken to publish the rules and rules structures as web services is similar to that in the EU-Rent domain.

As mentioned earlier, collaborating sites can also share automated application system operations and manual operations. System operations are defined by collaborating organizations and published as web services for use in rules and application systems. Manual operations are manual tasks performed by people of these organizations. Each manual operation is defined in terms of who should be notified and instructed to perform the operation, what means of notification should be used (e.g., by email, short message to a cell phone, and/or instruction displayed on a monitor) and what data should be provided to the system when the task has been carried out. When a manual operation is activated by a rule, the system would notify the appropriate person using the specified means of notification, and instruct the person to 1) perform the manual operation, 2) inform the system when the operation has been carried out, and 3) input the data produced by the operation. Both automated and manual operations can be referenced in and activated by rules.

We use a scenario (Figure 5-3) to demonstrate the distributed event and rule processing technique in the NPDN domain. The scenario begins with the NPDN Triage Lab receiving a
suspect sample from the State Department of Agriculture, APHIS-PPQ, or university staff. The NPDN Triage Lab staff that receives the sample posts a “Suspect Sample Received” event which has information about the sample, such as the genus of the host plant, the date the sample was collected, etc. This event data is encapsulated in an XML document and processed by the rule structure shown in the figure. The Triage Lab conducts its own preliminary diagnosis, changes the classification from “suspect” to “presumptive positive” (Rule NTLR1 in the figure). It notifies appropriate personnel such as the APHIS-PPQ CDD, the NPDN Regional Hub Lab, the state of origin SPHD and SPRO, and ships portions of the sample to the NPDN Regional Hub Lab and the APHIS-PPQ CDD (Rules NTLR2-NTLR4 in the figure). This results in the event “Presumptive Positive Sample Received,” which has applicable rule structures at the APHIS-PPQ CDD and the NPDN Regional Hub Lab. The NPDN Regional Hub Lab may also employ a local expert to help with the diagnosis process, in addition to conducting its own diagnosis (Rules NHLR1, and NHLR2 in the figure). In addition, it is also responsible for contacting some key personnel like the NPDN Regional Director (Rule NHLR3 in the figure). Optionally, it may send the sample to the APHIS-PPQ CDD (Rule NHLR4 in the figure). The APHIS-PPQ CDD conducts the confirming diagnosis and contacts the NPDN Triage Lab as well as the SPHD with these results (Rules ACDDR1-ACDDR3 in the figure). This will result in the event “Results Received” to occur at the NPDN Triage Lab (not shown in the figure). The initial event data contain only some portion of the sample information. As diagnosis proceeds along the different organization, more and more information about the sample is obtained. The final diagnosis results represent a new piece of information which contains the confirmed host and pest information about the sample.
The rule structure at the NPDN Triage Lab takes about a total of 14 seconds to execute.
The rule structures at the APHIS Lab takes about a total of 101 seconds to execute and the one at the NPDN Regional Hub Lab takes about a total of 149 seconds. This discrepancy is mainly due to the varied number of automated and manual operations. The actual time required for the manual operations was not measured, as it will vary greatly in every situation; however the time required to send the notification to the concerned individual was measured.

Figure 5-1. ETKnet system architecture
Figure 5-2. Event and rule processing in the EU-Rent domain
Figure 5-3. Event and rule processing in the NPDN domain
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 explicit and implicit subscriber 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 as follows. 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 a corresponding 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 the new attributes and values obtained from 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 explicit triggers that link an event to rules and/or rule structures, or if there are rules or rule structures that refer to the updated event data or new event data in their input data specifications and all of the rule or rule structure input is a subset of the event data, a new round of event and rule processing would start by sending the XML document to the applicable sites. This process
of event notification, event data transmission and aggregation, and rule and rule structure
processing terminates when no site has a rule or rule structure applicable to the last version of
event data.

Event data aggregation is achieved by viewing the event data document as the union of two
disjoint sets, $E_i$ and $D_i$, for round $i$. $E_i$ is the portion of the event data file sent in round $(i-1)$ that
was not updated, if $i \geq 1$. If $i = 0$, $E_i$ is empty. $D_i$ is the portion of the event data file which
includes updates and/or additions from round $(i-1)$, if $i \geq 1$. If $i = 0$, $D_i$ is the initial event data
that was made available from the event occurrence. Algorithm 1, presented in Figure 6-1
explains this process in greater detail. In Figure 6-1, $n_i$ is the number of applicable sites in round
$i$ of event processing. Below we give a correctness proof for the algorithm.

**Claim:** Algorithm 1 ensures that for every round $i$, $i \geq 0$, $D_i$ contains the data items that were
added or updated in round $i-1$ and $E_i$ contains those data items that were not (i.e. old
data).

Let $U_{is}$ denote the set of updates from site $s$ in round $i$

Let $A_{is}$ denote the set of additions from site $s$ in round $i$

Let $d_s$ denote the data item being considered from site $s$

**Proof:** (By induction)

**Basis step:** For round 0: $E_0 = \Phi$, $D_0 = $ initial event data

This is correct since all data provided as a result of the event posting is considered new. At this
point there is no old data.

**Induction step:** Assume $E_{i-1}$ and $D_{i-1}$ are correctly defined

For round $i$: $E_i = E_{i-1} U D_{i-1}$, $D_i = \Phi$

For every data item $d_s$, there are three possible cases,
(1) \( d_s \) is unchanged from the previous round, and thus is part of the old data. Since \( d_s \) was present in the previous round, \( d_s \in E_{i-1} \cup D_{i-1} \), and hence \( d_s \in E_i \).

(2) \( d_s \) was present in the previous round, but it’s value has now been updated, i.e. \( d_s \in U_is \). Algorithm 1 removes \( d_s \) from \( E_i \) and adds it to \( D_i \). This is correct since \( d_s \) now has an updated value.

(3) \( d_s \) is a new data item, not present in the previous round, i.e. \( d_s \in A_is \). Algorithm 1 adds \( d_s \) to \( D_i \), which is correct, since \( d_s \) is a data item that was added. Also, \( d_s \not\in E_i \).

Conclusion: Thus, repeating steps 1-3 for every data item \( d_s \) will ensure that all the updated/added data items are stored in \( D_i \), and those that were not are stored in \( E_i \). Hence proved.

Event Data 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.

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 6-1 describes the detection and the resolution mechanism employed by the coordinator.

Let \( (E_i + D_i) \) be the event data file sent out in round \( i \). \( U_i \) denotes the updates sent to the coordinator by site \( s \) for round \( i \), and \( A_i \) denotes the additions sent to the coordinator by site \( s \) for round \( i \). Let \( n_i \) be the number of sites that were applicable for round \( i \). Let \( u^s \) and \( d^s \) denote the value of an update or addition respectively tagged with the source site \( s \). Let \( \Phi \) 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 algorithm looks at each data item sent in from a site \( s \), and determines if there is a conflict for that data item, by checking if it was already updated by some other site \( s' \). If so, it applies the global resolution policy if one exists, and replaces the conflicting value with the resolved one. 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. As each data item for a particular site is examined once, the algorithm has a complexity of the order \( O(m_i n_m) \), where \( m_i \) is the number of applicable sites for a particular round \( i \), and \( n_m \) is the number of data items present in the event data document returned by site \( m \). So, on average, the complexity can be assumed to be \( O(m'n') \), where \( m' \), and
are the mean values for a particular event processing scenario. Below we give a correctness proof for the algorithm.

Claim: Algorithm 1 identifies a conflict iff it is present

Let \( U_{is} \) denote the set of updates from site \( s \) in round \( i \)

Let \( UC \) denote the set of conflicting data items due to updates

Let \( A_{is} \) denote the set of additions from site \( s \) in round \( i \)

Let \( AC \) denote the set of conflicting data items due to additions

Let \( d_s \) denote the data item being considered from site \( s \)

Proof: For every data item \( d_s \) considered in the aggregation phase,

(1) if \( d_s \in U_{is} \) and \( d_s \in D_i \), the algorithm compares the value of \( d_s \), denoted by \( d_v \) with the one stored in \( D_i \), denoted by \( d_{(D_i).v} \). If these values are not equal, a conflict due to update is registered, and \( d_s \) is added to \( UC \).

(2) if \( d_s \in A_{is} \) and \( d_s \in D_i \), the algorithm compares the value of \( d_s \) with the one stored in \( Di \). If these values are not equal, a conflict due to addition is registered, and \( d_s \) is added to \( AC \).

So, the algorithm identifies a conflict only in case of multiple values being present for the same data item, and thus detects a true conflict. Hence proved.

Avoiding Cyclic Rule Execution

Distributed knowledge rules and rule structures are independently and dynamically 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. This may trigger some other rules in the next round of processing. It is possible that a set of distributed rules may get locked into a cycle.

This issue, more commonly known as the “termination problem” for active rules has been widely studied [7, 19, 37, 42]. All of the work in this area addresses the issue of static determination of rule termination. The majority of these works use the concept of a triggering
graph [17]. In [2], a basic static rule termination algorithm is described. This analysis does not consider the possible deactivation of a rule’s condition. Works such as [5, 42] do take into account this consideration. Also, in most cases, the rule set is globally known or a global rule set is assumed to be available for rule termination analysis [6]. The work in [19] does not assume the knowledge of a global rule set. However, all of these methods address static termination analysis. In our event- and rule-based system (ETKnet), knowledge rules are dynamic. Thus, the “global” rule set is constantly changing. Multiple rules can be defined, updated, suspended or reactivated. Thus, the above approaches become inapplicable.

The existing approaches consider termination of active rules in the active database paradigm. In this environment, active rules update a database relation on occurrence of a triggering event. These rules are usually expressed as ECA rules. The condition expression is a query on the current database state, and is true if the answer relation is non-empty. In ETKnet, we do not directly operate on a global database, however, the current event data document can be viewed as the current “database state”. Since the above approaches aim to determine termination at build-time, they cannot consider the run-time values of data items. Thus, a build-time algorithm must always assume that the action in the triggered rule will get executed, and hence new data will be introduced as a result of the action. However, at run-time, a rule condition may not be satisfied, causing no execution of the action specified in the rule, making the above assumption false. As a result, in the ETKnet environment, use of static termination analysis can accurately determine termination, but not non-termination. For the above reasons, we resort to a run-time approach to guarantee termination of rule processing.

We derive some concepts for 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, detects it, and then deactivates some rule(s) to break the cycle. This is an acceptable approach to handle deadlocks since the deadlocked processes are stalled until the deadlock is broken. For distributed rule processing, however, if the rules locked in a cyclic processing are allowed to continue processing, 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. Our avoidance strategy combines pre-computing possible rule cycles that can occur for every event with rule monitoring at runtime by the coordinating site. Thus, the coordinating site can make a runtime decision on whether or not a cycle exists. We explain the definition time process and runtime process below.

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 possibly become applicable in the next round of event processing. It keeps track of all executed rules in each round of event 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, rules \( R4, R5, R6, R7, R8, R9 \), will be executed in the second, and rules \( R10, R11, R12 \) in the third round. The host will record the following information:
where the → symbol is used to denote the next round of rule processing. If, during a round of event processing, the host site determines that a rule \( R_c \), that was executed in the previous round is applicable again in this round, it treats this as the beginning of a possible cycle. \( R_c \) is thus marked as a “cyclic rule”. For each \( R_c \), the host site looks at all rules executed in the previous round to determine the set that can provide any item \( i_{Rc} \) (an input item to \( R_c \)) as their output. Let us call this set, \( R_p \), to denote the rules that can possibly trigger the execution of \( R_c \). Each rule in this set is recorded, and all such rules are Boolean “OR”ed together. This information thus has the following form:

\[
\bigvee_{i=1}^{n} r_i \quad \ldots \quad (2)
\]

where each \( r_i \) (\( 1 \leq i \leq n \)) \( \in \) \( R_p \), and \( \bigvee \) is the Boolean OR operator.

For example, assume that in the third round of processing the same event \( E, R1 \) (with four input items \( a, b, c \) and \( d \)) is made applicable again due to the fact that rule \( R4 \) updates data item \( a \), \( R7 \) updates data items \( a, b \), and \( c \), and \( R9 \) updates data items \( a \) and \( b \). The expression

\[
R4 \bigvee R7 \bigvee R9 \quad \ldots \quad (3)
\]

will be recorded for rule \( R1 \).

As \( R_c \) was executed once earlier, an update on any one of its input items in a round of processing will result in the rule being applicable for the next round. Using the above example, execution of any of the rules \( R4, R7, \) and \( R9 \) will cause at least one of \( R1 \)’s input data items to be updated, thus making \( R1 \) applicable for processing in the next round. The host stores information about all cyclic rules until it determines that no new rules can be executed in the next round of processing. It is possible that multiple cycles for a single rule exist. In such cases, the host will
calculate the rule expression for \( R_c \) for the current round, and use this to update any previous rule expressions computed for \( R_c \). This rule termination information is stored for every registered event. The algorithm for the above process is described in Figure 6-2. The process to compute the rule expression is described in Figure 6-3. The function in Figure 6-3 iterates over all rules over all input items \( i \) in \( r_c \). Step 4 in the function examines each output item of \( r_p \) to check if \( i \) is contained in \( r_p \).output, and also examines each rule in \( R_{expr} \) to check that \( r_p \) is not already included in \( R_{expr} \). This step takes a total of \((r+n)\) time, where \( r \) is the total number of rules in \( R_{expr} \), which is at most the total number of defined rules, and \( n \) is the number of output items in \( r_p \). Assuming each rule has \( n \) input and output items, the complexity of the function \textit{computeCyclicExpression} is of the order \( O(rn(r+n)) \). This is the most expensive calculation in Algorithm 2. The algorithm iterates over all rules in \( R_p \), which in the worst case be all of the defined global rules. This is repeated until there are applicable rules that have not been examined. In the worst case, each iteration of the while loop just adds a single rule, and all defined rules are applicable to the event, making the worst-case complexity of the order \( O(r^3n(r+n)) \).

Claim: In round \( i \), The rule expression \( expr \) computed for a possibly cyclic rule \( r_c \) using function 1 is necessary and sufficient to predict whether or not the rule will be re-executed in round \( i+1 \).

Let \( r_c \) denote the possibly cyclic rule being considered

Let \( i_{rc} \) denote a single input data item for \( r_c \)

Proof: After processing a given round \( i \), the sets \( E_i \) and \( D_i \) contain all the old and new/updated data items respectively. Event data items are never deleted, they are only added or they move between \( E_i \) and \( D_i \). When determining whether \( r_c \) will be re-executed in \( i+1 \), we know that \( r_c \) has
already been executed in some previous round $p, p < i$. Thus, all of the input data items $i_{rc}$ are already present in $(E_i U D_i)$. An update on any input data item $i_{rc}$ will make $r_c$ applicable for processing in round $i+1$. Examining only the previous round is sufficient as the rules executed in all rounds $q, q < i$ either do not contribute to $r_c$’s input, or have already derived a different rule expression for $r_c$. Function 1 examines each rule executed in round $i$ and Boolean “OR”s all those rules $R_{expr}$ that contribute to $r_c$’s input. Thus, if any rule $r, r \in R_p$ is executed, the rule expression becomes true and rule $r_c$ becomes applicable in round $i+1$. Similarly, if the rule expression is determined to be true, it implies that at least one $r, r \in R_p$ has been executed. As the rule expression captures exactly this and only this property of determining when any input item $i_{rc}$ is updated, the rule expression is necessary and sufficient to predict whether or not the rule will be re-executed in round $i+1$.

When a new global rule $R_{new}$ is added, it is registered at the host site. The host must determine what events can cause this rule to be processed, and accordingly update the rule processing path, similar to that shown in (1), and any information for cyclic rules, similar to that shown in (2). To achieve this, the host first determines if any event can directly provide the input for $R_{new}$, and records this information. Next, for each input item, $i_{R_{new}}$, the host determines the rules that can provide this input item and Boolean “OR”s them together. Let us denote the set of rules that can provide input item $i_{R_{new}}$ to be $R_i$. Finally, such expressions from each input item are Boolean “AND”ed together to get the expression, which if true, can cause rule $R_{new}$ to be processed, and has the following form:

$$\sum_{i=1}^{m} \sum_{j=1}^{n_i} V_{r_{ij}}$$

(4)
where 1 ≤ i ≤ m are the m input items of \( R_{\text{new}} \), \( r_{ij} \), 1 ≤ j ≤ n_i \( \in R_i \), are the n_i rules that provide can provide item i as input to \( R_{\text{new}} \), Λ is the Boolean AND operator, and V is the Boolean OR operator.

Once the host computes the above expression, it examines all processing paths stored for every event, and checks if this rule expression satisfies any of them. If so, the processing of this event may likely include \( R_{\text{new}} \). Once all such affected events are determined, the host goes through the algorithm shown in Figure 6-2 again to determine the processing and cyclic paths for each event. The algorithm for the above process is shown in Figure 6-4. The most expensive step in Algorithm 3 is step 22, where Algorithm 2 is invoked for every affected event. In the worst case, all of the e events can be affected by a rule, and hence the complexity of the algorithm is of the order \( O(er^3n(r+n)) \), where e is the number of registered events.

At runtime, when an event occurs at a collaborating site, this site, now called the coordinator, downloads the cyclic path information for the event 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 coordinator. The coordinator then examines the rules that were actually executed and compares it with the cyclic path information 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 event E with the possible rule processing scenario mentioned in (1) occurred and \( R1 \) and \( R3 \) were executed in round 1. In round 2, assume that only \( R5 \) was executed. With this information, the coordinator determines that the expression shown in (3) is not satisfied. If, on the other hand, during the processing of round 2, rule \( R4 \) was also executed, the coordinator would know that \( R1 \) will be
executed in the third round causing a cycle. Thus, $R_1$ is deactivated for that particular round to avoid the cycle.

Claim: Any rule processing initiated for a given event terminates.

Proof: For a given event, there are three processing scenarios:

(1) The host determines that the rules processed for this event can never get locked into a cycle. Thus, event processing eventually terminates.

(2) The host determines possible cyclic paths of execution. In every round, the coordinator for an event examines these possibly cyclic paths and determines whether or not a particular rule will be executed in the following round. If a cyclic rule is predicted to execute in the following round, it is deactivated for that round. Thus, cycles are avoided and event processing eventually terminates.

(3) During event processing, one or more new global rules that can cause cycles are introduced into the system. Whenever rules are defined at a collaborating site, they are inactive until registered with the host site. During registration, the host site computes the possible cyclic paths as explained in Algorithm 3. Thus, when the host registration is completed, possible cyclic paths for this rule have been updated. The rule now becomes active and can take part in the event processing. The coordinator requests cyclic path information at the beginning of each round of event processing. Thus, in the next round after the rule registration, the coordinator receives the updated possible cyclic paths. The new rule can be executed at most once before the coordinator becomes aware of the possible cyclic paths. From now on, this becomes similar to case 2, and event processing eventually terminates.

The above approach is better than static termination analysis methods because it takes into consideration the fact that rules are dynamically introduced and updated, and also takes runtime data values into account when determining likely rule cycles. However, the general problem of rule termination is undecidable [4], and so the best we can do is to take a conservative approach and treat every second occurrence of a rule in a given event processing scenario as the beginning of a cycle. Since correctness is more important than performance, the proposed algorithms represent a useful step towards achieving rule termination. As the performance data (Table 6-1) shows, a huge percentage of the build time process is spent in pre-computing possible cyclic paths, even as high as over 90%. However, the run time overhead is less than 1% at all times. This is acceptable since the build-time performance drop is a one-time occurrence.
It is possible to have self-terminating rule cycles, which are not handled by our approach. To understand this, consider the situation where we have the following rule cycle, $R1 \rightarrow R2 \rightarrow R3 \rightarrow R1$. It is possible that after a few rounds, the output item produced by $R3$ does not satisfy the input condition of rule $R1$, and hence it is no longer executed, i.e. the cycle has been broken. It is unreasonable to expect that the system should predict such self-terminating cycles. A better approach would be to enable operation-level sharing and powerful looping constructs, so that a desired “self-terminating cycle” or “program loop” can be described. This is, however, beyond the scope of this work.

Table 6-1. Build time and run time overhead of the cyclic rule avoidance strategy

<table>
<thead>
<tr>
<th>Number of rules (% cyclic)</th>
<th>Build time overhead (total) ms</th>
<th>Run time overhead (total) ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (10)</td>
<td>121 (155)</td>
<td>16 (4890)</td>
</tr>
<tr>
<td>10 (50)</td>
<td>153 (165)</td>
<td>15 (4828)</td>
</tr>
<tr>
<td>10 (90)</td>
<td>189 (196)</td>
<td>15 (4745)</td>
</tr>
<tr>
<td>20 (10)</td>
<td>123 (159)</td>
<td>21 (5170)</td>
</tr>
<tr>
<td>20 (50)</td>
<td>171 (212)</td>
<td>15 (5216)</td>
</tr>
<tr>
<td>20 (90)</td>
<td>212 (250)</td>
<td>18 (5240)</td>
</tr>
<tr>
<td>50 (10)</td>
<td>137 (178)</td>
<td>15 (11941)</td>
</tr>
<tr>
<td>50 (90)</td>
<td>541 (593)</td>
<td>15 (11648)</td>
</tr>
<tr>
<td>100 (10)</td>
<td>212 (237)</td>
<td>15 (22268)</td>
</tr>
<tr>
<td>100 (50)</td>
<td>674 (744)</td>
<td>18 (22030)</td>
</tr>
<tr>
<td>100 (90)</td>
<td>1514 (2229)</td>
<td>17 (22733)</td>
</tr>
<tr>
<td>250 (10)</td>
<td>125 (515)</td>
<td>15 (56371)</td>
</tr>
<tr>
<td>250 (50)</td>
<td>139 (3498)</td>
<td>29 (56804)</td>
</tr>
<tr>
<td>250 (90)</td>
<td>139 (8583)</td>
<td>40 (57300)</td>
</tr>
</tbody>
</table>
Algorithm 1 aggregateAndIdentifyConflicts

(1) \( E_0 = \Phi, D_0 = \) initial event data, \( U = \Phi, A = \Phi \)

(2) \( i = 1; \)

(3) \( \text{do} \}

(4) \( U = \Phi, A = \Phi; \)

(5) determine applicable sites and send event data for rule processing

(6) \( E_i = E_{i-1} \cup D_{i-1}, D_i = \Phi, UC = \Phi, AC = \Phi; \)

(7) \( \text{for} s \text{ from } 1 \text{ to } n_i \)

(8) \( \text{if} U_{is} \neq \Phi \)

(9) \( \text{for each } d_s \in U_{is} \)

(10) \( \text{if } (d_s \in D_i \land d_s.v \neq d_s.(D_i).v) \quad UC = UC + d_s; \)

(11) \( \text{end if} \)

(12) \( \text{if} (d_s \not\in C D_i) \quad D_i = D_i \cup d_s; \)

(13) \( \text{end if} \)

(14) \( \text{end if} \)

(15) \( \text{end for} \)

(16) \( \text{end if} \)

(17) \( \text{if} A_{is} \neq \Phi \)

(18) \( \text{for each } d_s \in A_{is} \)

(19) \( \text{if } (d_s \in D_i \land d_s.v \neq d_s.(D_i).v) \quad AC = AC + d_s; \)

(20) \( \text{end if} \)

(21) \( \text{if} (d_s \not\in C D_i) \quad D_i = D_i \cup d_s; \)

(22) \( \text{end if} \)

(23) \( \text{end for} \)

(24) \( \text{end if} \)

(25) \( \text{end for} \)

(26) \( D_i = D_i - (UC \cup AC); \)

(27) \( \text{for each } u \in UC \)

(28) \( \text{if} (\text{global_res_policy}(u) = \text{true}) \quad D_i = D_i \cup \text{resolve}(u); \)

(29) \( \text{else} \quad D_i = D_i \cup u; \)

(30) \( \text{end for} \)

(31) \( \text{for each } a \in AC \)

(32) \( \text{if} (\text{global_res_policy}(a) = \text{true}) \quad D_i = D_i \cup \text{resolve}(a); \)

(33) \( \text{else} \quad D_i = D_i \cup a; \)

(34) \( \text{end for} \)

(35) \( U = U \cup U_{is}; \)

(36) \( A = A \cup A_{is}; \)

(37) \( i++; \)

(38) \} \text{while}(U \cup A \neq \Phi); \)

Figure 6-1. Algorithm to aggregate event data and identify conflicts
Algorithm 2 findCyclicPaths

(1) $D = \text{set of event data, } ND = \Phi, R_e = \Phi, R_0 = \Phi$

(2) $R_p = \text{all rules } r \text{ s.t. } r.\text{input} \in D$

(3) initialize path to a blank string;

(4) for each $r \in R_p$

(5) $R_e = R_e \cup r$;

(6) $R_0 = R_0 \cup r$;

(7) $ND = ND \cup r.\text{output}$

(8) end for

(9) for each $r \in R_0$

(10) add $r$ to path

(11) end for

(12) add path separator (→) to path;

(13) $R_p = \Phi$;

(14) $i = 1$;

(15) do

(16) $R_i = \Phi$;

(17) $R_p = \text{all rules } r \text{ s.t. } r.\text{input} \in (D \cup ND) \text{ and } r.\text{input} \text{ not all in } D$

(18) $D = ND \cup D, ND = \Phi$

(19) for each $r \in R_p$

(20) if $r \in R_e$

(21) computeCyclicExpression($r, R_{i-1}$);

(22) else

(23) $R_e = R_e \cup r$;

(24) $R_i = R_i \cup r$;

(25) $ND = ND \cup r.\text{output}$;

(26) end else

(27) end for

(28) $i++$;

(29) for each $r \in R_i$

(30) add $r$ to path

(31) end for

(32) add path separator (→) to path;

(33) while $R_p \neq \Phi$

Figure 6-2. Algorithm to find possible cyclic paths on registration of an event
Function: computeCyclicExpression($r_c, R_{prevRound}$)

1. $R_{expr} = \emptyset$;
2. for each $i \in r_c.input$
3.   for each rule $r_p \in R_{prevRound}$
4.     if $i \in r_p.output$ and $r_p \notin R_{expr}$
5.       $R_{expr} = R_{expr} \cup r_p$
6.     end if
7.   end for
8. end for
9. initialize expr to a blank string;
10. for each rule $r \in R_{expr}$
11.   add $r$ to $expr$ using the Boolean OR operator;
12. end for
13. store $expr$ for cyclic rule $r_c$;

Algorithm 3 updateCyclicPathsForRule

1. $r_{new} =$ newly added global rule;
2. $R =$ set of defined global rules;
3. for each input item $i \in r_{new}.input$
4.   $R_i = \emptyset$;
5.   for each rule $r \in R$
6.     if $i \in r.output$ $R_i = R_i \cup r$;
7.   end if
8. end for
9. $exprR_i =$ Boolean OR expression linking rules in $R_i$;
10. end for
11. $exprR_{new} =$ Boolean AND expression linking all $exprR_i$;
12. $E =$ set of defined events;
13. $AE = \emptyset$;
14. for each event $e \in E$
15.   if $r_{new}.input \in e.data$ $AE = AE \cup e$;
16.   else
17.     path = execution path for this event;
18.     if $exprR_{new}$ is satisfied by path $AE = AE \cup e$;
19.   end else
20. end else
21. for each event $e \in AE$
22.   invoke algorithm findCyclicPaths for $e$;
23. end for

Figure 6-3. Algorithm to compute the rule expression for a possibly cyclic rule

Figure 6-4. Algorithm to update possible cyclic paths on addition of a global rule
In this dissertation, we have presented the basic problem of resource sharing among collaborating organizations. Complex problems faced by government organizations and business enterprises can be more effectively solved if organizations that form a collaborative federation are given the following facilities, system and infrastructure to effectively and efficiently define and share not only data, but also knowledge and application operations:

- A knowledge specification language and user interface tools for defining events of common interests, knowledge rules to capture organizational and inter-organizational policies, regulations and constraints, rule structures to model processes and operational procedures, triggers to link distributed events with rules and rule structures, and sharable application operations (automated and manual) to perform the needed tasks.

- A distributed, event- and rule-based system capable of delivering and aggregating data associated with event occurrences, and triggering the processing of applicable knowledge rules, rule structures, and application system operations (automated and manual).

- A web service-based infrastructure to achieve the uniform processing and interoperation of knowledge rules of different types and application (automated and manual) operations.

In this dissertation, we have presented our idea of managing dynamic event data and sharing multi-faceted knowledge and application operations among collaborating organizations. Different aspects of knowledge are specified in three different types of rules and rule structures. When an event of interest to a federation occurs, the initial event data serve as input to applicable rules and rule structures. These rules and rules structures may add to or modify the event data, and activate application operations, which may also add to or modify the event data. The new event data may make some other rules and rule structures applicable. Thus, multiple rounds of event data transmission and aggregation, and rule and rule structure processing would be carried out by the system until all the collaborating organizations have processed their applicable rules and rule structures and received all the data that are relevant to the event occurrence. Techniques and algorithms for the distributed processing and interoperation of events, rules, rule structures,
triggers and application operations for supporting decision-making and problem-solving are the main focuses 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. A knowledge specification language, and the architecture and implementation of an event- and rule-based system have been described. We have also discussed the research issues of event data aggregation, detecting event data inconsistencies, and avoiding cyclic rule execution and presented our solution approaches for them.

The specific contributions of this work are as follows:

- We have developed an XML-based knowledge specification language for organizations to specify their knowledge in terms of three types of rules and rule structures. As yet, there exists no standard markup language which allows the specification of all three types of knowledge rules. Our knowledge specification language integrates the syntax of each different rule type and is a step towards achieving a markup language capable of capturing multifaceted knowledge.

- We have developed algorithms for converting different types of rules to web services and the strategy for processing a rule structure to demonstrate the interoperation of heterogeneous rules. Instead of using three different types of rule engines to process the corresponding type of rule, we convert rules to code and wrap them as web services. This is a compilation approach which is more efficient than the interpretive rule engine approach. Also, conversion to a web service exposes each rule in a uniform manner making it more open to interoperability.

- We have researched techniques for managing dynamic event data and processing distributed knowledge rules, rule structures and application operations. Any collaborating site can be a potential event provider. Each site should thus have the capability of coordinating a particular event occurrence. This is achieved by developing a peer-to-peer server system rather than a client-server system. We have developed a distributed event- and rule-based system to achieve inter-organizational event data and knowledge and operation sharing. Business or knowledge rules exposed as web services lend themselves easily to sharing.

- We have developed and implemented algorithms for aggregating distributed event data, avoiding non-terminating processing of cyclic rules as well as techniques for handling inconsistent and contradictory data introduced by different organizations.
We have applied the knowledge specification language, the distributed, event- and rule-based system and the web service infrastructure in two application domains (e-business and agriculture homeland security) to demonstrate their utilities to achieve resource sharing among collaborating organizations.

Through this R&D work, we have laid the foundation of a distributed knowledge sharing system. Further work needs to be carried out to make the system deployable in real world environments. For example, when different organizations interact, there is always the issue of semantic heterogeneity in the language terms used to define data, events, rules and operations. Thus, there needs to be an ontology and an ontology management system to reason over the underlying concepts of these terms, and resolve their semantic discrepancies. Also, our rule language captures a very basic form of the CAA rules. The action and alternative action clauses can be made more expressive, thereby extending the rule language's capabilities to capture more complex structures of operations found in workflows. An event occurrence marks the beginning of a new transaction. It triggers the processing of rules, rule structures and application operations. The applicability of ACID properties of transaction in this event-triggered rule processing environment needs to be examined, and techniques of transaction management need to be investigated. Trust and security management in a collaborative environment is also an important topic of further research.
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:rulebase="http://www.dbcenter.cise.ufl.edu/rules"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.dbcenter.cise.ufl.edu/rules" elementFormDefault="qualified"
attributeFormDefault="unqualified">

<xs:element name="RuleBase">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Rule" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:simpleType name="DataType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="String"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="float"/>
    <xs:enumeration value="int"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="java.io.File"/>
    <xs:enumeration value="java.util.Date"/>
    <xs:enumeration value="java.util.HashMap"/>
    <xs:enumeration value="npdn.Sample"/>
    <xs:enumeration value="npdn.Shipment"/>
    <xs:enumeration value="npdn.ResponsePlan"/>
    <xs:enumeration value="npdn.Result"/>
    <xs:enumeration value="eurent.Branch"/>
    <xs:enumeration value="eurent.Car"/>
    <xs:enumeration value="eurent.Rental"/>
    <xs:enumeration value="eurent.Group"/>
    <xs:enumeration value="eurent.Customer"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="Expr">
  <xs:complexType>
    <xs:choice>
      <xs:sequence>
        <xs:element ref="rulebase:Term"/>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
</xs:element>

</xs:schema>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
  <xs:element ref="rulebase:MathOp"/>
  <xs:element ref="rulebase:Term"/>
</xs:sequence>
</xs:sequence>
<xs:sequence>
  <xs:element ref="rulebase:Expr"/>
  <xs:element ref="rulebase:MathOp"/>
  <xs:element ref="rulebase:Expr"/>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="rulebase:MathOp"/>
    <xs:element ref="rulebase:Expr"/>
  </xs:sequence>
</xs:sequence>
</xs:choice>
</xs:complexType>
</xs:element>

<xs:element name="Term">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="rulebase:DataItem"/>
      <xs:element ref="rulebase:Value"/>
      <xs:element ref="rulebase:Operation"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:simpleType name="RelOpType">
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
    <xs:maxLength value="2"/>
    <xs:enumeration value="eq"/>
    <xs:enumeration value="ne"/>
    <xs:enumeration value="le"/>
    <xs:enumeration value="lt"/>
    <xs:enumeration value="ge"/>
    <xs:enumeration value="gt"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="EnumOpType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="in"/>
    <xs:enumeration value="not in"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType>
</xs:simpleType>

<xs:element name="DataItem">
<xs:complexType>
<xs:sequence>
<xs:element name="Name" type="xs:string"/>
<xs:element name="Type" type="rulebase:DataType"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Value">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="valueType" type="xs:string" use="optional" default="String"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>

<xs:simpleType name="MathOpType">
<xs:restriction base="xs:string">
<xs:enumeration value="+"/>
<xs:enumeration value="-"/>
<xs:enumeration value="*"/>
<xs:enumeration value="/"/>
</xs:restriction>
</xs:simpleType>

<xs:element name="BooleanExpr">
<xs:complexType>
<xs:sequence>
<xs:element ref="rulebase:Not" minOccurs="0"/>
<xs:choice>
<xs:element ref="rulebase:PredicateExpr"/>
<xs:element ref="rulebase:Term"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="IfExpr">
<xs:complexType>
<xs:sequence>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="IfExpr"/>
<xs:element ref="rulebase:Not" minOccurs="0"/>
<xs:choice>
  <xs:sequence>
    <xs:element ref="rulebase:BooleanExpr"/>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="rulebase:LogicalOp"/>
      <xs:element ref="rulebase:BooleanExpr"/>
    </xs:sequence>
  </xs:sequence>
  <xs:sequence>
    <xs:element ref="rulebase:IfExpr"/>
    <xs:element ref="rulebase:LogicalOp"/>
    <xs:element ref="rulebase:IfExpr"/>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="rulebase:LogicalOp"/>
      <xs:element ref="rulebase:IfExpr"/>
    </xs:sequence>
  </xs:sequence>
</xs:choice>
</xs:complexType>

<xs:simpleType name="LogicalOpType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="AND"/>
    <xs:enumeration value="OR"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="RelOp" type="rulebase:RelOpType"/>
<xs:element name="EnumOp" type="rulebase:EnumOpType"/>
<xs:element name="LogicalOp" type="rulebase:LogicalOpType"/>
<xs:element name="MathOp" type="rulebase:MathOpType"/>
<xs:element name="ThenExpr">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Not" minOccurs="0"/>
      <xs:choice>
        <xs:sequence>
          <xs:element ref="rulebase:BooleanExpr"/>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="rulebase:LogicalOp"/>
            <xs:element ref="rulebase:BooleanExpr"/>
          </xs:sequence>
        </xs:sequence>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element ref="rulebase:LogicalOp"/>
<xs:element ref="rulebase:BooleanExpr"/>
</xs:sequence>
</xs:sequence>
<xs:sequence>
<xs:element ref="rulebase:ThenExpr"/>
<xs:element ref="rulebase:LogicalOp"/>
<xs:element ref="rulebase:ThenExpr"/>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element ref="rulebase:LogicalOp"/>
<xs:element ref="rulebase:ThenExpr"/>
</xs:sequence>
</xs:sequence>
</xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="AndOp" type="rulebase:AndOpType"/>

<xs:simpleType name="AndOpType">
<xs:restriction base="xs:string">
<xs:enumeration value="AND"/>
</xs:restriction>
</xs:simpleType>

<xs:element name="AttCons">
<xs:complexType>
<xs:sequence>
<xs:element ref="rulebase:DataItem"/>
<xs:choice>
<xs:sequence>
<xs:element ref="rulebase:RelOp"/>
<xs:element ref="rulebase:Value"/>
</xs:sequence>
<xs:sequence>
<xs:element ref="rulebase:EnumOp"/>
<xs:sequence>
<xs:element ref="rulebase:Value" maxOccurs="unbounded"/>
</xs:sequence>
</xs:sequence>
</xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="InterAttCons">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="rulebase:FormulaCons"/>
      <xs:element ref="rulebase:CondCons"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="CondCons">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:IfExpr"/>
      <xs:element ref="rulebase:ThenExpr"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="IntCons">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="rulebase:AttCons"/>
      <xs:element ref="rulebase:InterAttCons"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="Implies">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Body"/>
      <xs:element ref="rulebase:Head"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="CAARule">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Condition" minOccurs="0"/>
      <xs:element ref="rulebase:Action"/>
      <xs:element ref="rulebase:AlternativeAction" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Rule">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="RuleName" type="xs:string"/>
      <xs:element name="RuleDescription" type="xs:string"/>
      <xs:element name="RuleState" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="ACTIVE"/>
            <xs:enumeration value="SUSPENDED"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Sharing">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="local"/>
            <xs:enumeration value="global"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:choice>
        <xs:element ref="rulebase:IntCons"/>
        <xs:element ref="rulebase:Implies"/>
        <xs:element ref="rulebase:CAARule"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Not" type="xs:boolean"/>

<xs:element name="Atom">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:DataItem"/>
      <xs:element ref="rulebase:RelOp"/>
      <xs:choice>
        <xs:element ref="rulebase:DataItem"/>
        <xs:element ref="rulebase:Value"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
  <xs:element ref="rulebase:DataItem"/>
</xs:sequence>

<xs:element name="Returns" minOccurs="0">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element ref="rulebase:DataItem"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Condition">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Guard" minOccurs="0">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element ref="rulebase:PredicateExpr"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="CondExpr">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="rulebase:IfExpr"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Action">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Operation" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="AlternativeAction">
  <xs:complexType>
    <xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>
RuleStruc.xsd

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.dbcenter.cise.ufl.edu/rulestruc"
xmlns:rulestruc="http://www.dbcenter.cise.ufl.edu/rulestruc"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="RuleStruc">
<xs:complexType>
<xs:sequence>
<xs:element name="Name"/>
<xs:element name="RuleStrucDescription" type="xs:string"/>
<xs:element name="RuleStrucState" minOccurs="0">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="ACTIVE"/>
<xs:enumeration value="SUSPENDED"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Sharing">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="local"/>
<xs:enumeration value="global"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element ref="rulestruc:RuleSubStruc"/>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element ref="rulestruc:RuleSubStruc"/>
</xs:sequence>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:schema>
<xs:element name="OrJoin">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Num" type="xs:int"/>
      <xs:element ref="rulestruc:Predecessors"/>
      <xs:element ref="rulestruc:Successor"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Link">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulestruc:Predecessor"/>
      <xs:element ref="rulestruc:Successor"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Successors">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulestruc:RuleName" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="RuleName" type="xs:string"/>
</xs:schema>

EventData.xsd

<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.dbcenter.cise.ufl.edu/eventdata"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:eventdata="http://www.dbcenter.cise.ufl.edu/eventdata" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:element name="EventData">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Deactivate" type="xsd:string" minOccurs="0"/>
        <xsd:element name="RulesExecuted" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:eventspec="http://www.dbcenter.cise.ufl.edu/events"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.dbcenter.cise.ufl.edu/events" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="Events">
<xs:complexType>
<xs:sequence maxOccurs="unbounded">
<xs:element name="Event">
<xs:complexType>
<xs:sequence>
<xs:element name="Name" type="xs:string"/>
<xs:element name="Description" type="xs:string"/>
          <xs:element name="Sharing">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="global"/>
                <xs:enumeration value="local"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

Events.xsd
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:targetNamespace="http://www.dbcenter.cise.ufl.edu/operations"
xmlns:operations="http://www.dbcenter.cise.ufl.edu/operations"
xmlns:rulebase="http://www.dbcenter.cise.ufl.edu/rules"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:import namespace="http://www.dbcenter.cise.ufl.edu/rules"
schemaLocation="RuleBase.xsd"/>
<xs:element name="Operations">

Operations.xsd
<xs:complexType>
  <xs:sequence>
    <xs:element name="Operation" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="OperationName" type="xs:string"/>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="rulebase:DataItem"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Returns" minOccurs="0">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element ref="rulebase:DataItem"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:element>
    <xs:element name="Manual" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Message" type="xs:string"/>
          <xs:element name="ContactName" type="xs:string"/>
          <xs:choice>
            <xs:sequence>
              <xs:element ref="EmailAddress"/>
              <xs:sequence minOccurs="0" maxOccurs="1">
                <xs:element name="PhoneCompany" type="xs:string"/>
                <xs:element ref="PhoneNumber"/>
              </xs:sequence>
            </xs:sequence>
            <xs:sequence>
              <xs:element name="PhoneCompany" type="xs:string"/>
              <xs:element ref="PhoneNumber"/>
            </xs:sequence>
          </xs:choice>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name="PhoneNumber">
  <xs:simpleType>
    <xs:restriction base="xs:string">
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:pattern value="\d+"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="EmailAddress" type="xs:string"/>
</xs:schema>
APPENDIX B
EVENTS AND RULES IN THE EU-RENT DOMAIN

Description of entities used

Customer:
- custID of type integer
- licenseExpiry of type date
- insured of type boolean
- blacklisted of type Boolean
- age of type integer
- rented of type boolean
- points of type integer
- oneYearLicenseHeld of type Boolean
- numRented of type integer
- numReservations of type integer
- numYearlyRentals of type integer
- eligible of type Boolean
- pointsYear of type integer

Car:
- carID of type integer
- rented of type boolean
- mechanicalCondition of type string
- emissionsLevelMet of type boolean
- owningBranch of type integer
- assigned of type boolean
- scheduled of type boolean
- model of type string
- group of type string
- reserved of type boolean
- serviceDate of type date
- serviceMileage of type float
- threeMonthsServiceDate of type date
- needsService of type boolean
- scheduledDate of type date
- mileage of type float
- year of type integer
- toBeSold of type boolean

Rental:
- rentalID of type integer
- driverID of type integer
- group of type string
- model of type string
- startDate of type date
- endDate of type date
- numCars of type integer
- mode of type string
- driverName of type string
- creditCardName of type string
- driverSigned of type boolean
- addDriversSigned of type boolean
- extendedEndDate of type date
- driverOK of type boolean
- carOK of type boolean
- guarantee of type boolean
- dropOffBranch of type integer
- extension of type boolean
- mileage of type float
- oneWay of type boolean
- free of type boolean
- bookDate of type date

Group:
- groupID of type string
- quota of type integer
- numRemainingCars of type integer

Branch:
- branchID of type integer
- numReservations of type integer
- capacity of type integer

Note: messages are sent via email, text message on the receiver’s cell phone, and displayed on the screen when a user logs in.
The EU-Rent rules used to derive the ETKnet rules and operations are also included.

**EU-Rent Rules**

**EU-Rent Rule:** Each driver authorized to drive the car during a rental must be over 25 and have held a license for at least one year.

**EU-Rent Rule:** Each driver authorized to drive the car during a rental must have a valid driver’s license.

**EU-Rent Rule:** Each driver authorized to drive the car during a rental must be insured to the level required by the law of each country that may be visited during the rental.

**EU-Rent Rule:** If the customer requesting the rental has been blacklisted, the rental must be refused.

**EU-Rent Rule:** The driver who signs the rental agreement must not currently have an EU-Rent car on rental.

The rules shown above can be implemented by the following constraint rule.

**Name:** driverCheck

```java
if (Rental.driverID == Cusomter.custID)
then (Customer.licenseExpiry >= currentDate AND Customer.oneYearLicenseHeld == true AND Customer.insured == true AND Customer.blackListed == false AND Customer.age > 25 AND Customer.rented == false)
```

**Type:** Integrity Constraint

**EU-Rent Rule:** Rented cars must meet local legal requirements for mechanical condition and emissions for each country that may be visited during the rental.

**Name:** mechanicalConditionAndEmissionCheck

```java
if (Car.rented == true)
then (Car.mechanicalCondition == "Satisfactory" AND Car.emissionsLevelMet == true)
```

**Type:** Integrity Constraint

**EU-Rent Rule:** If a rental request does not specify a particular car group or model, the default is group A (the lowest-cost group).

**Name:** noGroupAndModelSpecified

```java
if (Rental.group == null AND Rental.model == null)
then (Rental.group = "A")
```

**Type:** Derivation Rule

**EU-Rent Rule:** Reservations may be accepted only up to the capacity of the pick-up branch on the pickup day.

**Name:** capacityCheck

```java
Branch.numReservations <= Branch.capacity
```

**Type:** Integrity Constraint

**EU-Rent Rule:** A customer may have multiple future reservations, but may have only one car at any time.
Name: noMultipleCars
    if(Customer.numReservations > 1)
        then (Customer.numRented == 1)
Type: Integrity Constraint

EU-Rent Rule: Only cars that are physically present in EU-Rent branches may be assigned.

The rule shown above can be implemented as the following four constraint rules

Name: checkOwningBranch
    Car.owningBranch == Branch.branchID
Type: Integrity Constraint

Name: checkNotAssigned
    Car.assigned == false
Type: Integrity Constraint

Name: checkNotRented
    Car.rented == false
Type: Integrity Constraint

Name: checkNotScheduled
    Car.scheduled == false
Type: Integrity Constraint

EU-Rent Rule: If a specific model has been requested, a car of that model should be assigned if one is available. Otherwise, a car in the same group as the requested model should be assigned.

EU-Rent Rule: If no specific model has been requested, any car in the requested group may be assigned.

The rules shown above can be implemented using the following CAA rule.

Name: assignCarOnModelOrGroup
    Condition: Car.model != null
    Action: 1. Car.group = getGroup(Car.model)
            2. Car.carID = assignCarOnModel(Car.model, Car.group)
    Alternative Action: 1. Car.carID = assignCarOnGroup(Car.group)
Type: CAA Rule

Operation getGroup definition:
Input: Car.model
Output: Car.group
Query the database for the group of the given model and return it.
**Operation assignCarOnModel definition:**
Input: Car.model, Car.group
Output: Car.carID
Query the database for a suitable car by model, if no results, query by group and return the carID.

**Operation assignCarOnModel definition:**
Input: Car.group
Output: Car.carID
Query the database for a suitable car by group and return the carID.

**EU-Rent Rule:** After all assignments within a group have been made, 10% of the group quota for the branch (or all remaining cars in the group, whichever number is lower) must be reserved for the next day’s walk-in rentals. Surplus capacity may be used for upgrades.

**Name:** reserveForWalkIn

**Type:** CAA Rule

**Operation reserveGroupByQuota definition:**
Input: Group.groupID, Group.quota
Output:
Find the first 0.1 * Group.quota cars for Group.groupID that have not been assigned, scheduled, rented, or reserved, and reserve them.

**Operation reserveGroupByQuota definition:**
Input: Group.groupID, Group.numRemainingCars
Output:
Find all remaining cars for Group.groupID that have not been assigned, scheduled, rented, or reserved, and reserve them.

**EU-Rent Rule:** If there are not sufficient cars in a group to meet demand, a one-group free upgrade may be given (i.e. a car of the next higher group may be assigned at the same rental rate) if there is capacity.

**EU-Rent Rule:** Customers in the loyalty incentive scheme have priority for free upgrades.

The two rules shown above can be implemented as the following CAA rule.

**Name:** giveUpgrade

**Condition:** Group.numRemainingCars == 0 OR Customer.points > 0

**Action:**
1. Car.carID = upgrade(Group.groupID, Group.numRemainingCars, Rental.rentalCharge, Customer.points)

**Alternative Action:** None
**Type:** CAA Rule

**Operation upgrade definition:**
Input: Group.groupID, Group.numRemainingCars, Rental.rentalCharge, Customer.points
Output: Car.carID
The system will notify the rental agent with the following message: “If the customer is in the loyalty incentive scheme, please assign an upgrade. If the number of cars remaining in a group is zero, please assign an upgrade. When you have done so, please log on to the system and indicate that operation upgrade has been performed.” The agent needs to provide the rental car ID as the output of the operation, which is entered through the user interface.

**EU-Rent Rule:** A car may be allocated from the capacity reserved for the next day’s walk-ins

**EU-Rent Rule:** A car due for return the next day may be allocated, if there is a suitable car available and there is time to transfer it to the pick-up branch.

The two rules shown above can be implemented as the following CAA rule.

**Name:** allocateNextDay
  
  **Condition:** Car.carID == -1

  **Action:**
  1. Car.carID = assignFromReservedOrNextDay(Rental.startDate)

  **Alternative Action:** None

**Type:** CAA Rule

**Operation assignFromReservedOrNextDay definition:**
Input: Rental.startDate
Output: Car.carID
Find a car that has been reserved for next day’s walk-in or if a car due to be returned the next day can be allocated, provided there is time to prepare it for servicing.

**EU-Rent Rule:** A ‘bumped upgrade’ may be made.

**Name:** bumpedUpgrade
  
  **Condition:** Car.carID == -1 && Group.groupID != “D”

  **Action:**
  1. Car.carID = findHigherGroup(Group.groupID)

  **Alternative Action:** None

**Type:** CAA Rule

**Operation findHigherGroup definition:**
Input: Group.groupID
Output: Car.carID
Find an available car from the next group and return the ID.

**EU-Rent Rule:** A downgrade (a car of a lower group) may be made.

**Name:** downgrade
  
  **Condition:** Car.carID == -1 && Group.groupID != “A”
1. Car.carID = findLowerGroup
   (Group.groupID)

Alternative Action: None

Type: CAA Rule

**Operation findLowerGroup definition:**

- **Input:** Group.groupID
- **Output:** Car.carID
- Find an available car from the next lower group and return the ID.

**EU-Rent Rule:** A car from another branch may be allocated, if there is a suitable car available and there is time to transfer it to the pick-up branch.

*(This is best expressed as an event and rule processing scenario)*

**Name:** findCarInOtherBranch

- **Condition:** Car.carID == -1
- **Action:**
  1. PostEvent("RemoteRentalRequest");
  2. Car.carID = getCarID("RemoteRentalRequest");
- **Alternative Action:** None

Type: CAA Rule

**Operation getCarID definition:**

- Gets the car ID returned as a result of posting "RemoteRentalRequest"

**EU-Rent Rule:** A car scheduled for service may be used, provided that the rental would not take the mileage more than 10% over the normal mileage for the service.

The rule shown above is implemented as assignScheduledCar for a later similar EU-Rent rule.

**EU-Rent Rule:** Pick-up may have to be delayed until a car is returned and prepared

**Name:** delayPickUp

- **Condition:** Car.carID == -1
- **Action:**
  1. Car.carID, Rental.startDate = findFirstAvailable()
- **Alternative Action:** None

Type: CAA Rule

**Operation findFirstAvailable definition:**

- **Input:**
- **Output:** Car.carID, Rental.startDate
- Find the first available car and return it. Return the date it will be available.

**EU-Rent Rule:** A car may have to be rented from a competitor

**Name:** rentFromCompetitor

- **Condition:** Car.carID == -1
- **Action:**
  1. Car.carID = findFromCompetitor(Rental.startDate, Rental.endDate)
**Operation findFromCompetitor definition:**
The system will notify the rental agent with the following message: “Please find an available car from a competitor. When you have done so, please log on to the system and indicate that operation findFromCompetitor has been performed.” The agent needs to provide the rental car ID as the output of the operation, which is entered through the user interface.

**EU-Rent Rule:** The end date of the rental must be before any scheduled booking of the assigned car for maintenance or transfer.

**Name:** checkCarNotScheduled
**Rental.endDate < Car.serviceDate**

**Type:** Integrity Constraint

**EU-Rent Rule:** If there are several available cars of the model or group requested, the one with the lowest mileage should be allocated.

**Name:** findAvailableCars
**Condition:** Rental.numCars > 0

**Action:**
1. Car.carID = allocateWithLowestMileage(Car.model, Car.group)

**Type:** CAA Rule

**Operation allocateWithLowestMileage definition:**
Input: Car.model, Car.group
Output: Car.carID
Find the number of available cars for the specified model or group and return its ID.

**EU-Rent Rule:** The credit card used to guarantee a rental must belong to one of the authorized drivers; and this driver must sign the rental contract. Other drivers must sign an ‘additional drivers authorization’ form.

**Name:** authorizedDrivers
if(Rental.mode == ‘Credit Card’)
then (Rental.driverName == Rental.creditCardName AND Rental.driverSigned = ‘true’
AND Rental.addDriversSigned = ‘true’)

**Type:** Integrity Constraint

**EU-Rent Rule:** Before releasing the car, a credit reservation equivalent to the estimated rental cost must be made against the guaranteeing credit card.

**EU-Rent Rule:** A customer may request a rental extension by phone - the extension should be granted unless the car is scheduled for maintenance.

The above two rules can be implemented as the following CAA rule.

**Name:** beforeRelease
Condition: None
Action: 1. makeCardReservation(Rental.rentalCharge)
         2. notifyExtensionProcedure(Rental.startDate, Rental.endDate)

Alternative Action: None
Type: CAA Rule

**Operation makeCardReservation definition:**
Input: Rental.rentalCharge
Output: The system will notify the rental agent with the following message: “Please charge the drivers credit card with the amount of the rental charge.”

**Operation notifyExtensionProcedure definition:**
Input: Rental.startDate, Rental.endDate
Output: Rental.extendedEndDate
The system will notify the rental agent with the following message: “Please approve the customer’s request to grant an extension by phone unless the car is scheduled for maintenance.”

**EU-Rent Rule:** The car must not be handed over to a driver who appears to be under the influence of alcohol or drugs.
**EU-Rent Rule:** The driver must be physically able to drive the car safely - must not be too tall, too short or too fat; if disabled, must be able to operate the controls.

The above two rules can be implemented as the following CAA rule.

Name: checkDriver
Condition: None
Action: 1. Rental.driverOK = checkDriverOK()
         Alternative Action: None
Type: CAA Rule

**Operation checkDriverOK definition:**
Input: 
Output: Rental.driverOK
The system will notify the rental agent with the following message: “Please check that the driver does not appear to be under the influence of alcohol or drugs, and is physically able to drive the car safely - must not be too tall, too short or too fat; if disabled, must be able to operate the controls. When you have done so, please log on to the system and indicate that operation checkDriverOK has been performed.” The agent needs to provide if the driver seems OK as the output of the operation, which is entered through the user interface.

**EU-Rent Rule:** The car must have been prepared - cleaned, full tank of fuel, oil and water topped up, tires properly inflated.
**EU-Rent Rule:** The car must have been checked for roadworthiness - tire tread depth, brake pedal and hand brake lever travel, lights, exhaust leaks, windscreen wipers.
**EU-Rent Rule:** Cars needing repairs (other than minor body scratches and dents) must not be used for rentals.

The above three rules can be implemented as the following CAA rule.

**Name:** checkCar  
**Condition:** None  
**Action:** 1. Rental.carOK = checkCarOK()  
**Alternative Action:** None  
**Type:** CAA Rule

**Operation checkCarOK definition:**  
**Input:**  
**Output:** Rental.carOK  
The system will notify the rental agent with the following message: “Please check that the car has been cleaned, has a full tank of fuel, oil and water topped up, tires properly inflated. Please check the car for roadworthiness - tire tread depth, brake pedal and hand brake lever travel, lights, exhaust leaks, windscreen wipers. Please make sure that the car does not need repairs. When you have done so, please log on to the system and indicate that operation checkCarOK has been performed.” The agent needs to provide if the car seems OK as the output of the operation, which is entered through the user interface.

**EU-Rent Rule:** If an assigned car has not been picked up 90 minutes after the scheduled pick-up time, it may be released for walk-in rental, unless the rental has been guaranteed by credit card.  
**Name:** noShowNoGuarantee  
**Condition:** ninetyMinuteDelay == true AND Rental.guarantee = false  
**Action:** 1. releaseCar (Car.carID)  
**Alternative Action:** None  
**Type:** CAA Rule

**Operation releaseCar definition:**  
**Input:** Car.carID  
**Output:**  
Set assigned, rented to false and set reserved to true

**EU-Rent Rule:** If a rental has been guaranteed by credit card and the car has not been picked up by the end of the scheduled pick-up day, one day’s rental is charged to the credit card and the car is released for use the following day.  
**Name:** noShowWithGuarantee  
**Condition:** Rental.endDate < endOfDay AND Rental.guarantee = true  
**Action:** 1. releaseCar (Car.carID)  
**Alternative Action:** None  
**Type:** CAA Rule
**EU-Rent Rule:** If a car is returned to a location other than the agreed drop-off branch, a drop-off penalty is charged.

**Name:** chargeDropOffPenalty  
**Condition:** Rental.dropOffBranch != Branch.branchID  
**Action:**  
1. Rental.rentalCharge = addDropOffPenalty(Rental.rentalCharge)  
**Alternative Action:** None

**Type:** CAA Rule

**Operation addDropOffPenalty definition:**  
**Input:** Rental.rentalCharge  
**Output:** Rental.rentalCharge  
Add the drop off penalty to the rental charge and return the new charged amount.

**EU-Rent Rule:** At the end of a rental, the customer may pay by cash, or by a credit card other than the one used to guarantee the rental.

**EU-Rent Rule:** Local tax must be collected (at the drop-off location) on the rental charge.

**EU-Rent Rule:** The car must be checked for wear (brakes, lights, tires, exhaust, wipers etc.) and damage, and repairs scheduled if necessary.

**EU-Rent Rule:** If the car has been damaged during the rental and the customer is liable, the customer’s credit card company must be notified of a pending charge.

The above rules can be implemented as the following CAA rule.

**Name:** checkCarOnReturn  
**Condition:** None  
**Action:**  
1. acceptableModeOfPayment()  
2. Rental.rentalCharge = collectLocalTax(Rental.dropOffBranch, Rental.rentalCharge)  
3. Rental.rentalCharge = checkForWear(Rental.rentalCharge)  
**Alternative Action:** None

**Type:** CAA Rule

**Operation acceptableModeOfPayment definition:**  
**Input:**  
**Output:**  
The system will notify the rental agent with the following message: “Please be informed that at the end of a rental, the customer may pay by cash, or by a credit card other than the one used to guarantee the rental.

**Operation collectLocalTax definition:**  
**Input:** Rental.dropOffBranch, Rental.rentalCharge  
**Output:** Rental.rentalCharge  
float localTax = access the database to get the local tax at the drop-off branch  
Rental.rentalCharge = localTax * Rental.rentalCharge/100 + Rental.rentalCharge;
**Operation checkForWear definition:**
Input: Rental.rentalCharge
Output: Rental.rentalCharge
The system will notify the rental agent with the following message: “Please check the car for wear (brakes, lights, tires, exhaust, wipers etc.) and damage, and schedule repairs if necessary. If the car has been damaged during the rental and the customer is liable, return the new amount charged and notify the customer’s credit card company of a pending charge. When you have done so, please log on to the system and indicate that operation checkCarOK has been performed.” The agent needs to provide the new amount charged as the output of the operation, which is entered through the user interface.

**EU-Rent Rule:** If a car is returned early, the rental charge is calculated at the rate appropriate to the actual period of rental (e.g., daily rate rather than weekly).
**Name:** adjustOnEarlyReturn
- **Condition:** Rental.endDate > currentDate
- **Action:**
  1. Rental.rentalCharge = adjustCharge(Rental.rentalCharge, Rental.startDate, Rental.endDate)
  - **Alternative Action:** None
**Type:** CAA Rule

**Operation adjustCharge definition:**
Input: Rental.rentalCharge, Rental.startDate, Rental.endDate
Output: Rental.rentalCharge
The system will notify the rental agent with the following message: “Please calculate the rental charge at the rate appropriate to the actual period of rental. When you have done so, please log on to the system and indicate that operation adjustCharge has been performed.” The agent needs to provide the new amount charged as the output of the operation, which is entered through the user interface.

**EU-Rent Rule:** If the car is returned late, an hourly charge is made up to 6 hours’ delay; after 6 hours a whole day is charged.
**Name:** calculateLateCharge
- **Condition:** Rental.endDate < currentDate
- **Action:**
  1. Rental.rentalCharge = lateCharge(Rental.rentalCharge, Rental.startDate, Rental.endDate)
  - **Alternative Action:** None
**Type:** CAA Rule

**Operation lateCharge definition:**
Input: Rental.rentalCharge, Rental.startDate, Rental.endDate
Output: Rental.rentalCharge
Determine the delay, if less than 6 hours, add hourly rate and return the new amount, else charge for the whole day.
EU-Rent Rule: If a car is not returned from rental by the end of the scheduled drop-off day and the customer has not arranged an extension, the customer should be contacted.

Name: lateNoExtension
Condition: Rental.endDate < endOfDay AND Rental.extension == False
Action: 1. contactCustomer(Rental.custID)
Alternative Action: None

Type: CAA Rule

Operation contactCustomer definition:
Input:
Output:
The system will notify the rental agent with the following message: “Please contact the customer as the rental date has passed and the customer has not asked for an extension.”

EU-Rent Rule: If a car is three days overdue and the customer has not arranged an extension, insurance cover lapses and the police must be informed.

Name: threeDaysOverdue
Condition: Rental.endDate < endOfThreeDays AND Rental.extension == false
Action: 1. contactPolice()
Alternative Action: None

Type: CAA Rule

Operation contactPolice definition:
Input:
Output:
The system will notify the rental agent with the following message: “Please contact the police as the rental is three days overdue and the customer has not asked for an extension.”

EU-Rent Rule: Each car must be serviced every three months or 10,000 kilometers, whichever occurs first.

Name: checkServiceNeeded
if(Car.serviceMileage >= 10000 OR Car.threeMonthsServiceDate <= currentDate) then
(Car.needsService == true)

Type: Integrity Constraint

EU-Rent Rule: If there is a shortage of cars for rental, routine maintenance may be delayed by up to 10% of the time or distance interval (whichever was the basis for scheduling maintenance) to meet rental demand.

Name: assignScheduledCar
Condition: Car.carID == -1
Action: 1. Car.carID = assignFromScheduled(Rental.mileage, Car.serviceMileage, Car.serviceDate, Car.scheduledDate, Rental.endDate)

Alternative Action: None
**Type:** CAA Rule

**Operation assignFromScheduled definition:**
- **Input:** Rental.mileage, Car.serviceMileage, Car.serviceDate, Rental.endDate
- **Output:** Car.carID

Find if a scheduled car can be assigned, provided the rental mileage would not take it more than 10% of approved service mileage, or the delay is no more than 10% of the interval between the service date and the scheduled date.

**EU-Rent Rule:** Only cars on the authorized list can be purchased.
- **Name:** checkAuthorizedCar
  - Car.model in {list of authorized models}
- **Type:** Integrity Constraint

**EU-Rent Rule:** Cars are to be sold when they reach one year old or 40,000 kilometers, whichever occurs first.
- **Name:** isToBeSold
  - if(Car.year < currentYear + 1 OR Car.mileage >= 40000)
  - then (Car.toBeSold = true)
- **Type:** Derivation Rule

**EU-Rent Rule:** A branch cannot refuse to accept a drop-off of a EU-Rent car, even if a one-way rental has not been authorized.
**EU-Rent Rule:** When a car is dropped off at a branch other than the pick-up branch, the car’s ownership (and, hence, responsibility for it) switches to the drop-off branch when the car is dropped off.

The above two rules can be implemented as the following CAA rule.

- **Name:** noAuthorizedOneWay
  - **Condition:** Rental.dropOffBranch != Branch.branchID AND Rental.oneWay == false
  - **Action:**
    1. adviseNoRefusal(Rental.rentalID)
    2. assignOwner(Branch.branchID, Car.carID)
  - **Type:** CAA Rule

- **Operation adviseNoRefusal definition:**
  - **Input:**
  - **Output:**
  The system will notify the rental agent with the following message: “Please do not refuse to accept a drop-off of a EU-Rent car, even if a one-way rental has not been authorized.”

- **Operation assignOwner definition:**
  - **Input:** Branch.branchID, Car.carID
  - **Output:**
Change the car’s owner to this branch.

**EU-Rent Rule:** When a transfer of a car is arranged between branches, the car’s ownership switches to the ‘receiving’ branch when the car is picked up.

**Name:** switchOwner

**Condition:** None

**Action:** 1. Car.owningBranch = assignOwner(Branch.branchID)

**Alternative Action:** None

**Type:** CAA Rule

**EU-Rent Rule:** In each car group, if a branch accumulates cars to take it more than 10% over its quota, it must reduce the number back to within 10% of quota by transferring cars to other branches or selling some cars.

**Name:** adjustQuotaDown

**Condition:** 1. Group.numRemainingCars > Group.quota + 0.1 * Group.quota

**Action:** 1. sellOrTransfer(Group.groupID)

**Alternative Action:** None

**Type:** CAA Rule

**Operation sellOrTransfer definition:**

**Input:** Group.groupID

**Output:**
The system will notify the branch manager with the following message: “Please reduce the number of cars back to within 10% of quota by selling or transferring.”

**EU-Rent Rule:** In each car group, if a branch loses cars to take it more than 10% below its quota, it must increase the number back to within 10% of quota by transferring cars from other branches or buying some cars.

**Name:** adjustQuotaUp

**Condition:** 1. Group.numRemainingCars < Group.quota - 0.1 * Group.quota

**Action:** 1. buyOrTransfer(Group.groupID)

**Alternative Action:** None

**Type:** CAA Rule

**Operation buyOrTransfer definition:**

**Input:** Group.groupID

**Output:**
The system will notify the branch manager with the following message: “Please increase the number of cars back to within 10% of quota by buying or transferring.”

**EU-Rent Rule:** To join the loyalty incentive scheme, a customer must have made 4 rentals within a year.

**Name:** eligibleForLoyaltyScheme

```
if(Customer.numYearlyRentals >= 4)
```
then (Customer.eligible = true)

**Type:** Derivation Rule

**EU-Rent Rule:** Each paid rental in the scheme (including the 4 qualifying rentals) earns points that may be used to buy ‘free rentals.’

**Name:** earnPoints

Condition: Customer.points > 0 AND Rental.free == false

Action: 1. Customer.points = addPoints(Rental.rentalID, Customer.custID)

Alternative Action: None

**Type:** CAA Rule

**Operation addPoints definition:**
Calculate the number of points the customer will earn and add that to his total. Return the new points.

**EU-Rent Rule:** Only the basic rental cost of a free rental can be bought with points. Extras, such as insurance, fuel and taxes must be paid by cash or credit card

**Name:** freeRental

Condition: Customer.points >0 AND Rental.free == true

Action: 1. Customer.points, Rental.rentalCharge = getFreeRental(Rental.rentalID, Rental.rentalCharge, Customer.custID)

Alternative Action: None

**Type:** CAA Rule

**Operation getFreeRental definition:**
Input: Rental.rentalID, Rental.rentalCharge, Customer.custID
Output: Customer.points, Rental.rentalCharge
Subtract the basic cost of rental from the rental charge, and subtract the customer’s points, return the adjusted charge and the new points.

**EU-Rent Rule:** A free rental must be booked at least fourteen days before the pick-up date.

**Name:** checkAdvBooking

if(Rental.free == true)
  then (getBookingGap(Rental.bookDate, Rental.startDate) >= 14)

**Type:** Integrity Constraint

**Operation getBookingGap definition:**
Input: Rental.bookDate, Rental.startDate
Output: bookingGap
Returns the number of days between the book and the start dates

**EU-Rent Rule:** Free rentals do not earn points

**Name:** freeRentalPoints

if(Rental.free == true)
then (Rental.points = 0)

**Type:** Derivation Rule

**EU-Rent Rule:** Unused points expire three years after the end of the year in which they were earned.

**Name:** unusedPoints

- **Condition:** getDifference(Customer.pointsYear, currentYear) >= 3
- **Action:** 1. Customer.points = expireUnusedPoints(Customer.custID, Customer.pointsYear)
  
  **Alternative Action:** None

**Type:** CAA Rule

**Operation getDifference definition:**

- **Input:** Customer,pointsYear, currentYear
- **Output:** difference

Returns the number of years between the year the customer first earned the points and the current year

**Operation expireUnusedPoints definition:**

- **Input:** Customer.custID, Customer.pointsYear
- **Output:** Customer.points

Expire unused points of the customer and return the adjusted points

**ETKnet Rule:** Get the branch information

**Name:** getBranch

- **Condition:** None
- **Action:** 1. Branch.* = getBranchInfo(Branch.branchID)
  
  **Alternative Action:** None

**Type:** CAA Rule

**Operation getBranchInfo definition:**

- **Input:** Branch.branchID
- **Output:** Branch.*

Return the information about this branch

**Events and Triggers**

On the occurrence of an event, the associated trigger will be processed

**Event:** Reservation Request (Customer.*, Car.*, Branch.branchID)

**Trigger:**

- driverCheck
- getBranch
  
  capacityCheck
  
  AND
  
  (RS1)
noGroupAndModelSpecified
  ↓
assignCarOnModelOrGroup
  ↓
giveUpgrade
    ↓ allocateNextDay    bumpedUpgrade    downgrade    assignScheduledCar
Event: No Suitable Car Found (Car.*)

Trigger:
(RS2) delayPickUp
  ↓
rentFromCompetitor

Event: Suitable Car Found (Car.*, Customer.*)

Trigger:
(RS3) checkOwningBranch
  ↓ checkNotAssigned
  ↓ checkNotRented
  ↓ checkNotScheduled
  ↓ noMultipleCars
  ↓ reserveForWalkIn

Event: Walk In Request(Customer.*, Car.*)

Trigger:
(RS4) driverCheck
  ↓
Event: Handover(Customer.*, Car.*)

Trigger:
  (RS5)
    driverCheck
      mechanicalConditionAndEmissionCheck
        authorizedDrivers
          beforeRelease
            checkDriver
            checkCar
              AND
                earnPoints

Event: Rental Return (Rental.*)

Trigger:
  (RS6)
    noAuthorizedOneWay
      switchOwner
        chargeDropOffPenalty
          adjustOnEarlyReturn
          calculateLateCharge
**Event:** No Show (Rental.*, Customer.*)

**Trigger:** Rules ‘noShowNoGuarantee’ and ‘noShowWithGuarantee’ will be processed

**Event:** Car Transfer(Car.*, Branch.*)

**Trigger:**
- switchOwner
  - adjustQuotaDown
  - adjustQuotaUp

**Event:** Free Rental Request (Customer.*, Rental.*)

**Trigger:**
- eligibleForLoyaltyScheme
  - freeRental
    - freeRentalPoints
    - checkAdvBooking

**Event:** Car Sale And Purchase(Car.*)

**Trigger:**
- isToBeSold
  - checkAuthorizedCar

**Event:** Three Month Service Check (Car.*)

**Trigger:** Rule ‘checkServiceNeeded’ will be processed

**Event:** Expire Points (Customer.*)

**Trigger:** Rule ‘unusedPoints’ will be processed
<table>
<thead>
<tr>
<th>Rule or rule structure name</th>
<th>Avg time to define (sec.)</th>
<th>Avg time to execute (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriverCheck</td>
<td>8</td>
<td>0.229</td>
</tr>
<tr>
<td>MechanicalConditionAndEmissionCheck</td>
<td>7.7</td>
<td>0.099</td>
</tr>
<tr>
<td>NoGroupAndModelSpecified</td>
<td>7.8</td>
<td>0.033</td>
</tr>
<tr>
<td>CapacityCheck</td>
<td>7.6</td>
<td>0.096</td>
</tr>
<tr>
<td>NoMultipleCars</td>
<td>7.7</td>
<td>0.102</td>
</tr>
<tr>
<td>CheckOwningBranch</td>
<td>7.6</td>
<td>0.096</td>
</tr>
<tr>
<td>CheckNotAssigned</td>
<td>7.7</td>
<td>0.075</td>
</tr>
<tr>
<td>CheckNotRented</td>
<td>7.6</td>
<td>0.077</td>
</tr>
<tr>
<td>CheckNotScheduled</td>
<td>7.4</td>
<td>0.075</td>
</tr>
<tr>
<td>AssignCarOnModelOrGroup</td>
<td>8.3</td>
<td>1.089</td>
</tr>
<tr>
<td>ReserveForWalkIn</td>
<td>8.1</td>
<td>1.058</td>
</tr>
<tr>
<td>GiveUpgrade</td>
<td>8.1</td>
<td>8.2</td>
</tr>
<tr>
<td>AllocateNextDay</td>
<td>7.7</td>
<td>1.023</td>
</tr>
<tr>
<td>BumpedUpgrade</td>
<td>7.9</td>
<td>3.5</td>
</tr>
<tr>
<td>Downgrade</td>
<td>7.8</td>
<td>3.5</td>
</tr>
<tr>
<td>DelayPickUp</td>
<td>9.5</td>
<td>0.181</td>
</tr>
<tr>
<td>RentFromCompetitor</td>
<td>8.0</td>
<td>2.7</td>
</tr>
<tr>
<td>CheckCarNotScheduled</td>
<td>7.5</td>
<td>0.054</td>
</tr>
<tr>
<td>FindAvailableCars</td>
<td>7.6</td>
<td>1.004</td>
</tr>
<tr>
<td>AuthorizedDrivers</td>
<td>7.7</td>
<td>0.100</td>
</tr>
<tr>
<td>BeforeRelease</td>
<td>7.5</td>
<td>4.1</td>
</tr>
<tr>
<td>CheckDriver</td>
<td>7.8</td>
<td>2.7</td>
</tr>
<tr>
<td>CheckCar</td>
<td>7.6</td>
<td>2.6</td>
</tr>
<tr>
<td>NoShowNoGuarantee</td>
<td>7.8</td>
<td>1.072</td>
</tr>
<tr>
<td>NoShowWithGuarantee</td>
<td>7.7</td>
<td>1.028</td>
</tr>
<tr>
<td>ChargeDropOffPenalty</td>
<td>7.7</td>
<td>1.871</td>
</tr>
<tr>
<td>CheckCarOnReturn</td>
<td>7.9</td>
<td>4.6</td>
</tr>
<tr>
<td>AdjustOnEarlyReturn</td>
<td>7.9</td>
<td>1.883</td>
</tr>
<tr>
<td>CalculateLateCharge</td>
<td>8.0</td>
<td>1.816</td>
</tr>
<tr>
<td>LateNoExtension</td>
<td>7.9</td>
<td>4.5</td>
</tr>
<tr>
<td>ThreeDaysOverdue</td>
<td>8.0</td>
<td>4.6</td>
</tr>
<tr>
<td>CheckServiceNeeded</td>
<td>7.8</td>
<td>0.096</td>
</tr>
<tr>
<td>AssignScheduledCar</td>
<td>7.9</td>
<td>2.939</td>
</tr>
<tr>
<td>CheckAuthorizedCar</td>
<td>7.6</td>
<td>0.054</td>
</tr>
<tr>
<td>IsToBeSold</td>
<td>8.4</td>
<td>0.102</td>
</tr>
<tr>
<td>NoAuthorizedOneWay</td>
<td>8.3</td>
<td>1.932</td>
</tr>
<tr>
<td>SwitchOwner</td>
<td>8.1</td>
<td>1.986</td>
</tr>
<tr>
<td>AdjustQuotaDown</td>
<td>8.4</td>
<td>0.465</td>
</tr>
<tr>
<td>AdjustQuotaUp</td>
<td>8.1</td>
<td>0.500</td>
</tr>
<tr>
<td>EligibleForLoyaltyScheme</td>
<td>8</td>
<td>0.055</td>
</tr>
<tr>
<td>EarnPoints</td>
<td>8</td>
<td>1.491</td>
</tr>
<tr>
<td>FreeRental</td>
<td>8.2</td>
<td>1.512</td>
</tr>
<tr>
<td>CheckAdvBooking</td>
<td>8</td>
<td>1.7</td>
</tr>
<tr>
<td>FreeRentalPoints</td>
<td>7.8</td>
<td>0.054</td>
</tr>
<tr>
<td>UnusedPoints</td>
<td>8</td>
<td>0.5</td>
</tr>
<tr>
<td>GetBranch</td>
<td>7.9</td>
<td>0.499</td>
</tr>
<tr>
<td>RS1</td>
<td>9.1</td>
<td>2.047</td>
</tr>
<tr>
<td>RS2</td>
<td>8.3</td>
<td>2.285</td>
</tr>
<tr>
<td>RS3</td>
<td>8.6</td>
<td>8.6</td>
</tr>
<tr>
<td>RS4</td>
<td>8.5</td>
<td>7.1</td>
</tr>
<tr>
<td>RS5</td>
<td>8.8</td>
<td>23</td>
</tr>
<tr>
<td>RS6</td>
<td>8.7</td>
<td>30</td>
</tr>
<tr>
<td>RS7</td>
<td>8</td>
<td>4.7</td>
</tr>
<tr>
<td>RS8</td>
<td>8.1</td>
<td>4.9</td>
</tr>
<tr>
<td>RS9</td>
<td>7.6</td>
<td>2.3</td>
</tr>
</tbody>
</table>
APPENDIX C
EVENTS AND RULES IN THE NPDN DOMAIN

Note: Operations in the SOP can be either automated or manual. Automated operations are performed by the system automatically. Manual operations are implemented by notifying the proper persons or organizations to perform the operations. All notifications specified in this document are sent via emails, text messages on the receivers’ cell phones, and displayed on the screen when users log in.

The steps of the SOP that the events and rules are derived from are also mentioned.

**Step 1a**
**Organization:** Grower, Pest Advisor, or other Sample Submitting Entity (SSE)
Any of the organizations that can be the sample submitting entity can post the event in case a suspect sample is observed. By posting, we mean the occurrence of the following event.

**Event SSEE1:** Suspect Sample Observed (npdn.Sample Sample.*)

**Rule SSER1:** Condition : None

Action: 1. time_of_arrival, method_of_delivery = Send_Sample_To_Diagnostician(npdn.Sample Sample.*)

Alternative Action: None

The data item *Sample* (i.e. the entity npdn.Sample) contains the following attributes, which may or may not have any data. As the diagnosis proceeds, the attributes get assigned appropriate values.

- **sampleID** of type *integer* (blank)
- **sampleDate** of type *date* (filled in by Grower, Pest Advisor or other SSE)
- **cropCode** of type *integer* (blank)
- **cropGenus** of type *string* (blank)
- **cropSpecies** of type *string* (blank)
- **pestCode** of type *string* (blank)
- **pestGenus** of type *string* (blank)
- **pestSpecies** of type *string* (blank)
- **classification** of type *string* (“Suspect”)
- **stateOfOrigin** of type *string* (filled in by Grower, Pest Advisor or other SSE)
- **notes** of type *string* (blank)

Send_Sample_To_Diagnostician() definition: The system will notify the Grower, Pest Advisor or other Sample Submitting Entity (SSE) with the following message: “Please send the sample to an NPDN triage lab. When you have done so, please log on to the system and indicate that operation Send_Sample_To_Diagnostician has been performed.” The SSE needs to provide the time_of_delivery (of type date) and the method_of_delivery (of type string) as the output of the operation, which are entered through the user interface.
Contact_Diagnostician definition: The system will notify the Grower, Pest Advisor or other Sample Submitting Entity with the following message: “Please contact the Diagnostician at the State Department of Agriculture to notify that a suspect sample is enroute with time of arrival and method of delivery. When you have done so, please log on to the system and indicate that operation Contact_Diagnostician has been performed.”

Trigger: SSEE1 → SSER1 (i.e., Event SSEE1 will trigger the processing of Rule SSER1)

Step 1b
Organization: State Department of Agriculture, APHIS-PPQ, or University Staff (i.e. “Diagnostician” mentioned above)
Any of the organizations that can be the diagnostician can post the event in case a suspect sample is received. By posting, we mean the occurrence of the following event.

Event SDAE1: Suspect Sample Received (npdn.Sample Sample.*)
The data item Sample contains the following attributes, which may or may not have any data. As the diagnosis proceeds, the attributes get assigned appropriate values.

- sampleID of type integer (blank)
- sampleDate of type date (filled in by Grower, Pest Advisor or other SSE)
- cropCode of type integer (blank)
- cropGenus of type string (blank)
- cropSpecies of type string (blank)
- pestCode of type string (blank)
- pestGenus of type string (blank)
- pestSpecies of type string (blank)
- classification of type string (“Suspect”)
- stateOfOrigin of type string (filled in by Grower, Pest Advisor or other SSE)
- notes of type string (blank)

Rule SDAR1: Condition: None
Action: 1. Send_Sample_To_NPDN(npdn.Sample Sample.*)
Alternative Action: None

Send_Sample_To_NPDN() definition: The system will notify the State Department of Agriculture, APHIS-PPQ, or University Staff with the following message: “Please send the sample to the National Plant Diagnostic Network Triage Lab. When you have done so, please log on to the system and indicate that operation Send_Sample_To_NPDN() has been performed.”

Trigger: SDAE1 → SDAR1 (i.e., Event SDAE1 will trigger the processing of Rule SDAR1)

Steps 2a-2d
Organization: NPDN Triage Lab
Event NTLE1: Suspect Sample Received (npdn.Sample Sample.*)
Rule NTLR1: Condition: None
Action: 1. Ack_Receipt(npdn.Sample Sample.*)
2. Sample.sampleID = Assign_ID (npdn.Sample Sample.*)
3. Sample.* = Examine_Sample (npdn.Sample Sample.*)
4. Sample.classification = 
   Classify_As_Presumptive_Positive()
5. Store_Sample(npdn.Sample Sample.*)
6. Request_Confirming_Diagnosis(npdn.Sample Sample.*)
7. distance_diagnosis_capability = Get_Capability()

Alternative Action: None

Ack_Receipt definition: The system will notify the NPDN Triage Lab staff with the following message: “Please acknowledge receipt of the sample to the State Department of Agriculture or University Staff. When you have done so, please log on to the system and indicate that operation Ack_Receipt has been performed.”

Assign_ID definition: An automated operation that generates a unique id for the given sample. After this operation is performed, the sample has the sampleID attribute filled in:
- sampleID of type integer (filled in by Assign_ID)
- sampleDate of type date (filled in by Grower, Pest Advisor or other SSE)
- cropCode of type integer (blank)
- cropGenus of type string (blank)
- cropSpecies of type string (blank)
- pestCode of type string (blank)
- pestGenus of type string (blank)
- pestSpecies of type string (blank)
- classification of type string (“Suspect”)
- stateOfOrigin of type string (filled in by Grower, Pest Advisor or other SSE)
- notes of type string (blank)

Examine_Sample definition: The system will notify the NPDN Triage Lab staff with the following message: “Please examine the sample. When you have done so, please log on to the system and indicate that operation Examine_Sample has been performed.” The staff needs to provide the updated sample information Sample.* (of type npdn.Sample) as the output of the operation, which is entered through the user interface.

Classify_As_Presumptive_Positive definition: An automated operation that fills in the value “Presumptive Positive” for the sample classification since the sample has been examined.

After this operation the sample may have the remaining attributes filled in:
- sampleID of type integer (filled in by Assign_ID)
- sampleDate of type date (filled in by Grower, Pest Advisor or other SSE)
- cropCode of type integer (possibly filled in by Examine_Sample)
- cropGenus of type string (possibly filled in by Examine_Sample)
- cropSpecies of type string (possibly filled in by Examine_Sample)
- pestCode of type string (possibly filled in by Examine_Sample)
- pestGenus of type string (possibly filled in by Examine_Sample)
- pestSpecies of type string (possibly filled in by Examine_Sample)
- **classification** of type string ("Presumptive Positive")
- **stateOfOrigin** of type string (filled in by Grower, Pest Advisor or other SSE)
- **notes** of type string (possibly filled in by Examine_Sample)

**Store_Sample** definition: The system will notify the NPDN Triage Lab staff with the following message: “Please store the sample securely. When you have done so, please log on to the system and indicate that operation Store_Sample has been performed.”

**Request_Confirming_Diagnosis** definition: The system will notify the NPDN Triage Lab staff with the following message: “Please contact the state of origin SPRO, state of origin SPHD, APHIS-PPQ CDD, and NPDN Hub Lab informing them that a presumptive positive sample is in the system. When you have done so, please log on to the system and indicate that operation Request_Confirming_Diagnosis has been performed.”

**Get_Capability** definition: An automated operation that finds out whether or not this lab has the distance diagnosis capability, indicated by the output distance_diagnosis_capability.

**Step 2e**

**Rule NTLR2**: Condition: distance_diagnosis_capability = true
   Action: Conduct_Web_Based_Diagnosis(npdn.Sample Sample.*)
   Alternative Action: sample_image = Email_Picture()

**Conduct_Web_Based_Diagnosis** definition: The system will notify the NPDN Triage Lab staff with the following message: “Please contact the APHIS-PPQ CDD, and NPDN Hub Lab to set up a web-based distance diagnosis session. When you have done so, please log on to the system and indicate that operation Conduct_Web_Based_Diagnosis has been performed.”

**Email_Picture** definition: The system will notify the NPDN Triage Lab staff with the following message: “Please take a picture of the sample and email it to the APHIS-PPQ CDD, and the NPDN Hub Lab. When you have done so, please log on to the system and indicate that operation Email_Picture has been performed.” The staff needs to provide the picture of the sample, sample_image (of type File) as output, which is entered through the user interface.

**Steps 2f-2h**

**Rule NTLR3**: Condition: None
   Action: 1. Shipment.*, time_of_delivery, method_of_delivery = Divide_And_Ship_Sample(npdn.Sample Sample.*)

   Alternative Action: None

**Divide_And_Ship_Sample** definition: The system will notify the NPDN Triage Lab staff with the following message: “The sample should go to an NPDN hub lab OR to the CDD. Please consult the CDD on which it should be. If routine sample or if hub lab has provisional approval to ID from CDD then it should go to the hub lab, otherwise if hub lab is not provisionally
approved for the suspect species or if it is very high consequence first occurrence, then it should go to CDD directly. Accordingly, divide the sample into portions and send a portion to the APHIS-PPQ CDD Lab or to the NPDN Regional Hub Lab. Please ensure that the sample is double bagged in zippable bags and sealed in a box with tape, sample submission form is included, diagnostician’s business card is included, and sample box is marked as ‘Plant samples for diagnosis’. When you have done so, please log on to the system and indicate that operation Divide_And_Ship_Sample has been performed.” The staff needs to provide shipment information, Shipment.* (of type npdn.Shipment), time_of_delivery (of type string), and method_of_delivery (of type string) as output, which are entered through the user interface.

The data item Shipment (i.e. the entity npdn.Shipment) contains the following attributes.
- sender of type string (filled in by NPDN Triage Lab)
- receiver of type string (filled in by NPDN Triage Lab)
- trackingNumber of type string (filled in by NPDN Triage Lab)
- doubleBagged of type Boolean (filled in by NPDN Triage Lab)
- submissionFormIncluded of type Boolean (filled in by NPDN Triage Lab)
- businessCardIncluded of type Boolean (filled in by NPDN Triage Lab)
- shippingLabel of type string (filled in by NPDN Triage Lab)

Contact_About_Shipment definition: The system will notify the NPDN Triage Lab staff with the following message: “Please contact the state of origin SPRO, APHIS-PPQ SPHD, APHIS-PPQ CDD, and NPDN Hub Lab or Expert Lab informing them of the presumptive positive sample shipment time and delivery method, including tracking number. When you have done so, please log on to the system and indicate that operation Contact_About_Shipment has been performed.”

Step 2i
Rule NTLR4: Condition: None
    Action: 1. Contact_Campus_Safety_Officer(npdn.Sample Sample.*)
    Alternative Action: None

Contact_Campus_Safety_Officer definition: The system will notify the NPDN Triage Lab staff with the following message: “Please contact the Campus Safety Officer informing them of a presumptive positive sample in the system. When you have done so, please log on to the system and indicate that operation Contact_Campus_Safety_Officer has been performed.”

Trigger: The following rule structure is identified as “NTLRS1”

```
NTLE1 →
   ↓
   NTLR1
   ↓
   NTLR2
   ↓
   NTLR3
   ↓
   NTLR4
```
(i.e. Event NTLE1 will trigger the processing of rule NTLR1, followed by rule NTLR2. Once rule NTLR2 completes execution, rules NTLR3 and NTLR4 are executed in parallel)

Steps 3a-3b
Organization: State of Origin SPRO
Event SPROE1: Presumptive Positive Sample Recorded (npdn.Sample Sample.*)
Rule SPROR1: Condition: None
   Action: 1. ResponsePlan.* = Contact_SPHD(npdn.Sample Sample.*)
   Alternative Action: None

Contact_SPHD definition: The system will notify the State of Origin SPRO with the following message: “Please contact the State of Origin SPHD to discuss response plans in the event that the presumptive positive sample is confirmed positive by the APHIS-PPQ CDD. You can communicate with other regulatory officials in neighboring states to plan and/or activating the strategy. When you have done so, please log on to the system and indicate that operation Contact_SPHD has been performed.” The SPRO needs to provide information about the response plan, ResponsePlan.* (of type npdn.ResponsePlan) as output, which is entered through the user interface.

The data item ResponsePlan (i.e. the entity npdn.ResponsePlan) contains the following attributes:
- officialsToBeNotified of type string
- pestGenus of type string
- pestSpecies of type string
- pressRelease of type string

Trigger: SPROE1 → SPROR1

(i.e. Event SPROE1 will trigger the processing of rule SPROR1)

Steps 4a-4b
Organization: NPDN Regional Hub Lab
Event NHLE1: Presumptive Positive Sample Received (npdn.Sample Sample.*)
Rule NHLR1: Condition: (Shipment.doubleBagged == true) &&
   (Shipment.submissionFormIncluded == true) &&
   (Shipment.businessCardIncluded == true) &&
   (Shipment.shippingLabel == “Plant samples for diagnosis”)
   Action: 1. Ack_Receipt (npdn.Sample Sample.*)
      2. Sample.* = Examine_Sample(npdn.Sample Sample.*)
      3. contact_local_expert = Is_OK_To_Contact_Local_Expert()
   Alternative Action: 1. Contact_NPDN_Triage_Lab(Boolean
      Shipment.doubleBagged, boolean
      Shipment.submissionFormIncluded, boolean
      Shipment.businessCardIncluded, String
      Shipment.shippingLabel, npdn.Sample Sample.*)
Ack_Receipt definition: The system will notify the NPDN Hub Lab staff with the following message: “Please acknowledge receipt of the sample to the NPDN Triage Lab. When you have done so, please log on to the system and indicate that operation Ack_Receipt has been performed.”

Examine_Sample definition: The system will notify the NPDN Regional Hub Lab staff with the following message: “Please examine the sample. When you have done so, please log on to the system and indicate that operation Examine_Sample has been performed.” The staff needs to provide updated information about the sample, Sample.* (of type npdn.Sample) as output, which is entered through the user interface.

After this operation is performed, the sample may have some changes to the values of some of its attributes:

- `sampleID` of type integer (filled in by Assign_ID)
- `sampleDate` of type date (filled in by Grower, Pest Advisor or other SSE)
- `cropCode` of type integer (possibly changed by Examine_Sample)
- `cropGenus` of type string (possibly changed by Examine_Sample)
- `cropSpecies` of type string (possibly changed by Examine_Sample)
- `pestCode` of type string (possibly changed by Examine_Sample)
- `pestGenus` of type string (possibly changed by Examine_Sample)
- `pestSpecies` of type string (possibly changed by Examine_Sample)
- `classification` of type string (“Presumptive Positive”)
- `stateOfOrigin` of type string (filled in by Grower, Pest Advisor or other SSE)
- `notes` of type string (possibly changed by Examine_Sample)

Is_OK_To_Contact_Local_Expert definition: The system will notify the NPDN Regional Hub Lab staff with the following message: “Please contact the Director to determine if it is OK to bring in a local expert. When you have done so, please log on to the system and indicate that operation Is_OK_To_Contact_Local_Expert has been performed.” The staff needs to provide `contact_local_expert` (of type boolean) as output, which is entered through the user interface.

Contact_NPDN_Triage_Lab definition: The system will notify the NPDN Regional Hub Lab with the following message: “Please contact the NPDN Triage Lab indicating that proper sample shipping procedures were not followed. When you have done so, please log on to the system and indicate that operation Contact_NPDN_Triage_Lab has been performed.”

Step 4c

Rule NHLR2: Condition: (contact_local_expert == true)
Action: 1. results_received = Contact_Local_Expert(npdn.Sample Sample.)*
Alternative Action: None

Contact_Local_Expert definition: The system will notify the NPDN Hub Lab with the following message: “Please contact the Local Expert for confirming diagnosis. When you have done so, please log on to the system and indicate that operation Contact_Local_Expert has been performed.” The staff needs to provide `results_received` (of type boolean) as output (this will indicate that the local expert provided the results), which is entered through the user interface.
Steps 4g-4i
Rule NHLR3: Condition: None
  Action: 1. Contact_NPDNDirector_And_APHIS(npdn.Sample Sample.*)
          2. Contact_SPRO_SPHD(String Sample.classification)
          3. Contact_CS_Officer(String Sample.classification)
          4. to_send_to_APHIS = Contact_APHIS(npdn.Sample Sample.*)
Alternative Action: None

Contact_NPDNDirector_And_APHIS definition: The system will notify the NPDN Hub Lab with the following message: “Please contact the NPDN Regional Director and APHIS-PPQ-Confirming Diagnosis Designate with preliminary conclusions/results. When you have done so, please log on to the system and indicate that operation Contact_NPDNDirector_And_APHIS has been performed.”

Contact_SPRO_SPHD definition: The system will notify the NPDN Hub Lab with the following message: “Please contact your own SPRO and SPHD to inform them that a presumptive positive sample is housed in NPDN Regional Hub until shipment to APHIS-PPQ Confirming Designate or until it is destroyed following diagnosis. Do not disclose the state of origin to SPRO or SPHD. When you have done so, please log on to the system and indicate that operation Contact_SPRO_SPHD has been performed.”

Contact_CS_Officer definition: The system will notify the NPDN Hub Lab staff with the following message: “Please contact the Campus Safety Officer informing them of a presumptive positive sample in the system. Do not disclose state of origin. When you have done so, please log on to the system and indicate that operation Contact_CS_Officer has been performed.”

Contact_APHIS definition: The system will notify the NPDN Hub Lab with the following message: “Please contact the APHIS-PPQ Confirming Diagnosis Designate and ask if they request sending sample for confirmation. When you have done so, please log on to the system and indicate that operation Contact_APHIS has been performed.” The staff needs to provide to_send_to_APHIS (of type boolean) as output, which is entered through the user interface.

Steps 4j-4k
Rule NHLR4: Condition: to_send_to_APHIS == true
  Action: 1. Shipment.*, time_of_delivery, method_of_delivery = Send_Sample_To_APHIS(npdn.Sample Sample.*)
          2. Contact_APHIS_With_ShipmentDetails(npdn.Shipment Shipment.*)
Alternative Action: None

Send_Sample_To_APHIS definition: The system will notify the NPDN Hub Lab with the following message: “Please send the sample to the APHIS-PPQ CDD. When you have done so, please log on to the system and indicate that operation Send_Sample_To_APHIS() has been performed.” The staff needs to provide information about the shipment, Shipment.* (of type
npdn.Shipment), time_of_delivery (of type date), and method_of_delivery (of type string) as output, which are entered through the user interface.

Contact_APHIS_With_ShipmentDetails definition: The system will notify the NPDN Hub Lab with the following message: “Please contact the APHIS-PPQ CDD to notify that a presumptive positive sample is enroute with time of arrival and method of delivery. When you have done so, please log on to the system and indicate that operation Contact_APHIS_With_ShipmentDetails has been performed.”

Trigger: The following rule structure is identified as “NHLRS1”

```
NHLE1 → NHLR1
  /   \
 /     \  
NHLR2  NHLR3 AND
   \    /
     \  
      \ NHLR4
```

(i.e. Event NHLE1 will trigger the processing of rule NHLR1. Once rule NHLR1 completes processing, rules NHLR2 and NHLR3 will be executed in parallel. Once both these rules finish processing, rule NHLR4 will be executed)

Steps 4d-4f
Organization: Local Expert
Event LEE1: Presumptive Positive Sample Received (npdn.Sample Sample.*)
Rule LER1: Condition: None
  Action: 1. Sample.* = Examine_Sample(npdn.Sample Sample.*)
           2. Contact_NPDN_With_Results(npdn.Sample Sample.*)
     Alternative Action: None

Examine_Sample definition: The system will notify the Local Expert with the following message: “Please make the preliminary diagnosis in collaboration with NPDN Regional Hub staff. When you have done so, please log on to the system and indicate that operation Examine_Sample has been performed.” The expert needs to provide information about the sample, Sample.* (of type npdn.Sample) as output, which is entered through the user interface.

Contact_NPDN_With_Results definition: The system will notify the Local Expert with the following message: “Please contact the NPDN Regional Hub Diagnostician with conclusions/results. When you have done so, please log on to the system and indicate that operation Contact_NPDN_With_Results has been performed.”

Trigger: LEE1 → LER1
(i.e. Event LEE1 will trigger the processing of rule LER1)
Step 5
Organization: NPDN Regional Director from state of origin
Event NRDE1: Presumptive Positive Sample Under Diagnosis (npdn.Sample Sample.*)
Rule NRDR1: Condition: None
   Action: 1. Contact_NPDN_Program_Leader(npdn.Sample Sample.*)
   Alternative Action: None

Contact_NPDN_Program_Leader definition: The system will notify the NPDN Regional Director with the following message: “Please contact the NPDN Program Leader indicating that the presumptive positive sample is under diagnosis. Do not disclose state of origin except to Program Leader. When you have done so, please log on to the system and indicate that operation Contact_NPDN_Program_Leader has been performed.”

Trigger: NRDE1 → NRDR1
(i.e. Event NRDE1 will trigger the processing of rule NRDR1)

Step 6a
Organization: APHIS-PPQ CDD
Event ACDDE1: Presumptive Positive Sample Received (npdn.Sample Sample.*)
Rule ACDDR1: Condition: None
   Action: 1. Ack_Receipt(npdn.Sample Sample.*)
          2. expected_date, more_than_seven_days = Date_Of_Result_Notification(npdn.Sample Sample.*)
          3. ok_to_contact_NPDN = Is_OK_To_Contact_NPDN()
   Alternative Action: None

Ack_Receipt definition: The system will notify the APHIS-PPQ CDD staff with the following message: “Please acknowledge receipt of the sample to the NPDN Triage Lab. When you have done so, please log on to the system and indicate that operation Ack_Receipt has been performed.”

Date_Of_Result_Notification definition: The system will notify the APHIS-PPQ CDD staff with the following message: “Please indicate to the NPDN Triage Lab the expected date of result notification. When you have done so, please log on to the system and indicate that operation Date_Of_Result_Notification has been performed.” The staff provides the expected_date (of type date) of confirmation information as well as whether confirmation it will be more than seven days after submission, more_than_seven_days (of type boolean) as output, which are entered through the user interface.

Is_OK_To_Contact_NPDN definition: The system will notify the APHIS-PPQ CDD staff with the following message: “Please ask the APHIS-PPQ Program Manager to consult with the NPDN Program Leader and determine if it is OK to inform other NPDN Regional Directors and their diagnosticians. When you have done so, please log on to the system and indicate that operation Is_OK_To_Contact_NPDN has been performed.” The staff needs to provide ok_to_contact_NPDN (of type boolean) as output, which is entered through the user interface.
Steps 6b-6c
Rule ACDDRR2: Condition: (more_than_seven_days == true) &&
   (ok_to_contact_NPDN == true)
Action: 1. Contact_Other_Directors()
       2. Contact_Diagnosticians()
Alternative Action: None

Contact_Other_Directors definition: The system will notify the NPDN Regional Director of
affected region with the following message: “Please contact other NPDN Regional Directors to
inform them that a presumptive positive sample is in the system. Kindly do not disclose origin
of sample. When you have done so, please log on to the system and indicate that operation
Contact_Other_Directors has been performed.”

Contact_Diagnosticians definition: The system will notify all NPDN Regional Directors with the
following message: “Please contact diagnosticians in your region at NPDN labs in each state to
inform them that a presumptive positive sample is under diagnosis in an unknown location in the
nation and to be alert for similar samples that may be appearing in other labs. When you have
done so, please log on to the system and indicate that operation Contact_Diagnosticians has
been performed.”

Steps 6d-6g, Step 7a
Rule ACDDRR3: Condition: None
Action: 1. Result.* = Conduct_Confirming_Diagnosis(npdn.Sample
       Sample.*)
       2. Contact_Administrator(npdn.Result Result.*)
       3. Contact_Regional_APHIS_Staff(npdn.Result Result.*)
       4. Contact_SPHD(npdn.Result Result.*)
       5. Contact_TriageLab(npdn.Result Result.*)
Alternative Action: None

The data item Result (i.e. the entity npdn.Result) contains the following attributes.
- sampleID of type integer
- sampleDate of type date
- cropCode of type integer
- cropGenus of type string
- cropSpecies of type string
- pestCode of type string
- pestGenus of type string
- pestSpecies of type string
- classification of type string
- confirmedDiagnosisDate of type date
- expectedDateTime of type date
- notes of type string

Conduct_Confirming_Diagnosis definition: The system will notify the APHIS-PPQ CDD Staff
with the following message: “Please conduct the confirming diagnosis. When you have done so,
please log on to the system and indicate that operation *Conduct_Confirming_Diagnosis* has been performed.” The staff needs to provide the diagnosis results, *Result.* (of type npdn.Result) as output, which is entered through the user interface.

**Contact_Administrator** definition: The system will notify the APHIS-PPQ CDD Staff with the following message: “Please contact the APHIS-PPQ Administrator to inform him of the final results. When you have done so, please log on to the system and indicate that operation *Contact_Administrator* has been performed.”

**Contact_Regional_APHIS_Staff** definition: The system will notify the APHIS-PPQ Administrator with the following message: “Please contact the Regional APHIS Staff to inform them of the final results. When you have done so, please log on to the system and indicate that operation *Contact_Regional_APHIS_Staff* has been performed.”

**Contact_SPHD** definition: The system will notify the APHIS-PPQ Administrator with the following message: “Please contact the State Plant Health Director of the state of origin to inform him of the final results. When you have done so, please log on to the system and indicate that operation *Contact_SPHD* has been performed.”

**Contact_TriageLab** definition: The system will notify the APHIS-CDD Staff with the following message: “Please contact the Triage Lab to inform them of the final results after enough time has passed that state of origin SPHD and SPRO and Triage Lab Diagnostician should have been informed of the confirmed diagnosis results by regional APHIS-PPQ office. When you have done so, please log on to the system and indicate that operation *Contact_TriageLab* has been performed.”

**Trigger:** The following rule structure is identified as “ACDDRS”

```
ACDDE1  →  ACDDR1
         ↓
         ACDDR2
         ↓
         ACDDR3
```

(i.e. Event ACDDE1 will trigger the sequential processing of rules ACDDR1, ACDDR2, and ACDDR3)

**Step 6h**

**Organization:** SPHD from State of Origin

**Event SPHDE1:** Results Received(npdn.Result Result.)*

**Rule SPHDR1:** Condition: None

```
Action:  1. Contact_SPRO(npdn.Result Result.)*
Alternative Action: None
```
**Contact_SPRO definition:** The system will notify the SPHD with the following message: “Please contact the State Plant Regulatory Official of the state of origin to inform him of the final results. When you have done so, please log on to the system and indicate that operation Contact_SPRO has been performed.”

**Trigger:** SPHDE1 → SPHDR1

(i.e. Event SPHDE1 will trigger the processing of rule SPHDR1)

**Step 6i, Step 9b**

**Organization:** SPRO from State of Origin

**Event SPROE2:** Results Received from SPRO(npdn.Result Result.*)

**Rule SPROR2:** Condition: None

Action: 1. Contact_TriageLab(npdn.Result Result.*)

2. Submit_Record(npdn.Result Result.*)

Alternative Action: None

**Contact_TriageLab definition:** The system will notify the SPRO with the following message: “Please contact the Triage Lab to inform them of the final results. When you have done so, please log on to the system and indicate that operation Contact_TriageLab has been performed.”

**Submit_Record() definition:** The system will notify the SPRO with the following message: “Please submit the record to the NAPIS CAPS database. When you have done so, please log on to the system and indicate that operation Submit_Record has been performed.”

**Trigger:** SPROE2 → SPROR2

(i.e. Event SPROE2 will trigger the processing of rule SPROR2)

**Steps 7b-7c, Steps 7g-7h**

**Organization:** NPDN Triage Lab

**Event NTLE2:** Results Received from APHIS(npdn.Result Result.*)

**Rule: NTLR5:** Condition: None

Action: 1. Contact_Relevant_Organizations(npdn.Result Result.*)

2. Contact_NPDNHub(npdn.Result Result.*)


4. Contact_Campus_Officer(npdn.Result Result.*)

Alternative Action: None

**Contact_Relevant_Organizations definition:** The system will notify the NPDN Triage Lab Staff with the following message: “Please contact the state of origin SPHD, SPRO, and the Regional NPDN Director to acknowledge receipt of results. When you have done so, please log on to the system and indicate that operation Contact_Relevant_Organizations has been performed.”

**Contact_NPDNHub definition:** The system will notify the NPDN Triage Lab Staff with the following message: “Please contact the NPDN Regional Hub Lab to notify them of the results.
When you have done so, please log on to the system and indicate that operation Contact_NPDNHub has been performed.”

Contact_SSE definition: The system will notify the NPDN Triage Lab Staff with the following message: “Please coordinate with SPRO and SPHD to contact Sample Submitting Entity with results as approved and designated by the state of origin SPRO and SPHD. When you have done so, please log on to the system and indicate that operation Contact_SSE has been performed.”

Contact_Campus_Officer definition: The system will notify the NPDN Triage Lab Staff with the following message: “Please contact the Campus Safety Officer with result and assurance of sample destruction. When you have done so, please log on to the system and indicate that operation Contact_Campus_Officer has been performed.”

Step 7i
Rule: NTLR6: Condition: Result.classification == “Positive”
Action: 1. Destroy_Sample()
2. select_agent = Is_Select_Agent()
Alternative Action: None

Destroy_Sample() definition: The system will notify the NPDN Triage Lab Staff with the following message: “Please destroy the confirmed positive sample by placing all packing material and the sample in an autoclave for at least 20 minutes. When you have done so, please log on to the system and indicate that operation Destroy_Sample has been performed.”

Is_Select_Agent definition: The system will notify the NPDN Triage Lab Staff with the following message: “Please check if the confirmed positive sample was a select agent. When you have done so, please log on to the system and indicate that operation Is_Select_Agent has been performed.” The staff needs to provide select_agent (of type boolean) as output, which is entered through the user interface.

Rule: NTLR7: Condition: select_agent == true
Action: 1. Submit_Form4()
Alternative Action: None

Submit_Form4() definition: The system will notify the NPDN Triage Lab Staff with the following message: “Please complete and submit Form 4. When you have done so, please log on to the system and indicate that operation Submit_Form4 has been performed.”

Step 9a
Rule: NTLR8: Condition: None
Action: 1. Submit_Record(npdn.Result Result.*)
Alternative Action: None

Submit_Record definition: The system will notify the NPDN Triage Lab Staff with the following message: “Please submit the record to the NPDN database. When you have done so, please log on to the system and indicate that operation Submit_Record has been performed.”
**Trigger:** The following rule structure is identified as “NTLS2”

\[
\begin{array}{c}
\text{NTLE2} \rightarrow \\
\quad \downarrow \\
\quad \text{NTLR5} \\
\quad \downarrow \\
\quad \text{NTLR6} \\
\quad \downarrow \\
\quad \text{NTLR7} \\
\quad \downarrow \\
\quad \text{NTLR8}
\end{array}
\]

(i.e. Event NTLE2 will trigger the sequential processing of rules NTLR5, NTLR6, NTLR7, and NTLR8)

**Step 7d, Steps 8a-8b**

**Organization: NPDN Regional Hub Lab**

**Event NHLE2:** Results Received from NPDN Triage Lab (npdn.Result Result.*)

**Rule: NTLR5:** Condition: None

Action: 1. Contact_Program_Leader(npdn.Result Result.*)

2. Contact_SPRO_And_SPHD (npdn.Result Result.*)

3. Contact_Campus_Officer (npdn.Result Result.*)

4. select_agent = Is_Select_Agent()

Alternative Action: None

**Contact_Program_Leader definition:** The system will notify the NPDN Hub Lab Staff with the following message: “Please contact the NPDN Program Leader to notify him/her of the results. When you have done so, please log on to the system and indicate that operation Contact_Program_Leader has been performed.”

**Contact_SPRO_And_SPHD definition:** The system will notify the NPDN Hub Lab Staff with the following message: “Please contact the SPRO and SPHD with result and assurance of sample destruction. Do not disclose state of origin. When you have done so, please log on to the system and indicate that operation Contact_SPRO_And_SPHD has been performed.”

**Contact_Campus_Officer definition:** The system will notify the NPDN Hub Lab Staff with the following message: “Please contact the Campus Officer with result and assurance of sample destruction. Do not disclose state of origin. When you have done so, please log on to the system and indicate that operation Contact_Campus_Officer has been performed.”

**Is_Select_Agent definition:** The system will notify the NPDN Hub Lab Staff with the following message: “Please check if the confirmed positive sample was a select agent. When you have done so, please log on to the system and indicate that operation Is_Select_Agent has been
performed.” The staff needs to provide select_agent (of type boolean) as output, which is entered through the user interface.

**Rule: NHLR6**: Condition: select_agent == true
- Action: 1. Submit_Form4()
- Alternative Action: None

Submit_Form4() definition: The system will notify the NPDN Hub Lab Staff with the following message: “Please complete and submit Form 4. When you have done so, please log on to the system and indicate that operation Submit_Form4 has been performed.”

**Trigger**: The following rule structure is identified as “NHLRS2”

\[
\text{NHLE2} \rightarrow \text{NHLR5} \rightarrow \text{NHLR6}
\]

(i.e. Event NHLE2 will trigger the sequential processing of rules NHLR5 and NHLR6)

**Step 7e**
**Organization**: NPDN Regional Director from state of origin
**Event NRDE2**: Results Received from NPDN Triage Lab (npdn.Result Result.*)
**Rule: NRDR2**: Condition: None
- Action: 1. Contact_Other_Dirs(npdn.Result Result.*)
- Alternative Action: None

Contact_Other_Dirs definition: The system will notify the NPDN Regional Director with the following message: “Please contact other NPDN Regional Directors, if engaged at the discretion of APHIS-PPQ in consultation with NPDN Program Leader, with the results. Do not disclose state of origin. When you have done so, please log on to the system and indicate that operation Contact_Other_Dirs has been performed.”

**Trigger**: NRDE2 → NRDR2
(i.e. Event NRDE2 will trigger the processing of rule NRDR2)

**Step 7f**
**Organization**: Other NPDN Regional Directors
**Event ONRDE1**: Results Received from NPDN Regional Director in state of origin (npdn.Result Result.*)
**Rule: ONRDR1**: Condition: None
- Action: 1. Contact_Regional_Labs (npdn.Result Result.*)
- Alternative Action: None

Contact_Regional_Labs definition: The system will notify the NPDN Regional Director with the following message: “Please contact NPDN Regional Labs, if engaged at the discretion of APHIS-PPQ in consultation with NPDN Program Leader, with the results. Do not disclose state
of origin. When you have done so, please log on to the system and indicate that operation
*Contact_Regional_Labs* has been performed.”

Trigger: ONRDE1 → ONRDR1
(i.e. Event ONRDE1 will trigger the processing of rule ONRDR1)

<table>
<thead>
<tr>
<th>Rule or rule structure name</th>
<th>Average time to define (sec.)</th>
<th>Average time to execute (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSER1</td>
<td>8.7</td>
<td>44</td>
</tr>
<tr>
<td>SDAR1</td>
<td>9.3</td>
<td>22</td>
</tr>
<tr>
<td>NTLR1</td>
<td>8.3</td>
<td>29</td>
</tr>
<tr>
<td>NTLR2</td>
<td>8.4</td>
<td>25</td>
</tr>
<tr>
<td>NTLR3</td>
<td>8.5</td>
<td>46</td>
</tr>
<tr>
<td>NTLR4</td>
<td>8.3</td>
<td>5</td>
</tr>
<tr>
<td>NTLRS1</td>
<td>8.4</td>
<td>46</td>
</tr>
<tr>
<td>SPROR1</td>
<td>8.2</td>
<td>5</td>
</tr>
<tr>
<td>NHLR1</td>
<td>8.7</td>
<td>6.7</td>
</tr>
<tr>
<td>NHLR2</td>
<td>8.2</td>
<td>2.7</td>
</tr>
<tr>
<td>NHLR3</td>
<td>8.7</td>
<td>30</td>
</tr>
<tr>
<td>NHLR4</td>
<td>8.6</td>
<td>54</td>
</tr>
<tr>
<td>NHLRS1</td>
<td>8.4</td>
<td>63</td>
</tr>
<tr>
<td>LER1</td>
<td>8.6</td>
<td>48</td>
</tr>
<tr>
<td>NRDR1</td>
<td>8.5</td>
<td>26</td>
</tr>
<tr>
<td>ACDDR1</td>
<td>8.3</td>
<td>16</td>
</tr>
<tr>
<td>ACDDR2</td>
<td>8.2</td>
<td>45</td>
</tr>
<tr>
<td>ACDDR3</td>
<td>8.4</td>
<td>89</td>
</tr>
<tr>
<td>ACDDRS</td>
<td>8.5</td>
<td>54</td>
</tr>
<tr>
<td>SPHDR1</td>
<td>8.2</td>
<td>22</td>
</tr>
<tr>
<td>SPROR2</td>
<td>8.4</td>
<td>2</td>
</tr>
<tr>
<td>NTLR5</td>
<td>8.8</td>
<td>11</td>
</tr>
<tr>
<td>NTLR6</td>
<td>8.2</td>
<td>4</td>
</tr>
<tr>
<td>NTLR7</td>
<td>8.1</td>
<td>20</td>
</tr>
<tr>
<td>NTLR8</td>
<td>8.4</td>
<td>0.3</td>
</tr>
<tr>
<td>NHLR5</td>
<td>8.7</td>
<td>16</td>
</tr>
<tr>
<td>NHLR6</td>
<td>8.3</td>
<td>4</td>
</tr>
<tr>
<td>NHLRS2</td>
<td>8.5</td>
<td>20</td>
</tr>
<tr>
<td>NTLRS2</td>
<td>8.5</td>
<td>31</td>
</tr>
<tr>
<td>NRDR2</td>
<td>8.5</td>
<td>22</td>
</tr>
<tr>
<td>ONRDR1</td>
<td>8.5</td>
<td>28</td>
</tr>
</tbody>
</table>
LIST OF REFERENCES


34. Iowa State University, Iowa Soybean Association and Iowa Soybean Promotion Board, Iowa Department of Agriculture and Land Stewardship, & United States Department of Agriculture – Animal and Plant Health Inspection Service – Plant Protection and Quarantine, Iowa Response and Action Plan for Asian Soybean Rust, Draft 1.5, 2004


153


BIOGRAPHICAL SKETCH

Seema Degwekar was born in Thane, India on November 10, 1978. She received her Bachelor of Engineering in Computer Engineering from Vivekanand Education Society’s Institute of Technology, affiliated with University of Mumbai, in June 2000. She joined the Computer and Information Science and Engineering (CISE) graduate program at the University of Florida in August 2000. She has been a part of the Database Research and Development Center from August 2001. She graduated in December 2002 with a Master of Science in Computer Engineering.

At University of Florida, she has worked as a research assistant, a teaching assistant, and has taught an undergraduate course for around two years. She is the founder of the department’s graduate student association. She is also the recipient of the Outstanding International Student award from the College of Engineering for her academic achievements, and the Alec Courtelis award in recognition of academic excellence and University and community service.

Her research areas include event and rule based systems, knowledge management and sharing, distributed computing, indexing large databases, web services, peer-to-peer systems, XML and web databases.