META-RULE ENHANCED INTEROPERATION OF RULES AND PROCESSES FOR
ACHIEVING DYNAMIC INTER-ORGANIZATIONAL COLLABORATION

By

XUELIAN XIAO

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2012
To my family
ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. Stanley Y. W. Su for his constant guidance, support and encouragement. I am privileged to have such a distinguished advisor, who is patient, helpful and encouraging. He gave me extensive advice and insight during the course of my research work. He showed infinite patience with me, for which I will be eternally grateful.

I would also like to thank Dr. Howard Beck, Dr. Randy Chow, Dr. Shigang Chen, and Dr. Herman Lam for being a part of my PhD committee. I would like to thank my research teammates Jeff DePree, Chen Zhou and Seema Degwekar for working with me in this research project. Also special thank to Jeff DePree for helping me correct grammatical errors in my dissertation draft.

I want to express my deepest love to my husband, Ming Zhang, without whose encouragement, I would have never completed this important step in my life; also for bringing me our little angel Olivia, who is giving me the most happiness in the world. I am also very grateful for my parents, who have been supporting me in all possible ways, and for my sister, who has been taking care of our parents while I am far away from home. I would also like to thank my mother-in-law for her support. Without my families’ support, this dissertation is impossible.

Big thanks also go to all of my friends, who have always cheered me on and make my life in Gainesville so wonderful.

Finally, this R&D work is supported by NSF under grant IIS-0534065.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>CHAPTER 1 INTRODUCTION</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>CHAPTER 2 RELATED WORKS</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>2.1 Business Rule Language Design</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>2.1.1 General-Purpose Rule Specification Languages</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>2.1.2 Our Own Earlier Effort</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>2.2 Integration of Business Rules in Business Process Application</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>2.3 Dynamic Construction and Execution of Inter-organizational Processes</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>CHAPTER 3 INTEGRATED RULE AND PROCESS SPECIFICATION LANGUAGE (IRPSL)</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>3.1 Operations</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>3.2 Rules</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>3.2.1 Integrity Constraints</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>3.2.2 Derivation Rules</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>3.2.3 Action-Oriented Rules</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>3.3 Processes</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>3.3.1 Structural Constructs Defined in a Process</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>3.3.2 An Example of a Process</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>3.4 Events</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>3.5 Sharable Operations, Rules and Processes as Web Services</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>3.5.1 Operations as Web Services</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>3.5.2 Rules as Web Services</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>3.5.3 Processes as Web Services</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>CHAPTER 4 ETKNET2.0 SYSTEM ARCHITECTURE AND ITS EVENT-TRIGGERED PROCESSING TECHNIQUE</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>4.1 ETKnet2.0 System Architecture</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>4.2 Event-triggered Rule Processing and Process Enactment Technique</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>4.2.1 Event Instance and Event Data</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>4.2.2 Triggers</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>4.2.3 Applicability of Rules</td>
<td>64</td>
</tr>
<tr>
<td>Table</td>
<td>page</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>6-1</td>
<td>102</td>
<td></td>
</tr>
</tbody>
</table>

Meta-rules related to the global event “TriageLabE1”
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Examples of manual operation and automated operation</td>
</tr>
<tr>
<td>3-2</td>
<td>An example of action-oriented rule whose action part is to invoke an operation.</td>
</tr>
<tr>
<td>3-3</td>
<td>An example of action-oriented rule whose action part is to invoke a process.</td>
</tr>
<tr>
<td>3-4</td>
<td>An example process named “ProcessExampleOnTriageLabReceivingSample”</td>
</tr>
<tr>
<td>3-5</td>
<td>An example of an event named “TriageLabE1”.</td>
</tr>
<tr>
<td>3-6</td>
<td>The routine for creating OWSs, RWSs and PWSs.</td>
</tr>
<tr>
<td>3-7</td>
<td>General structure of code for an action-oriented rule.</td>
</tr>
<tr>
<td>3-8</td>
<td>General structure for generated invoker thread class code.</td>
</tr>
<tr>
<td>3-9</td>
<td>General structure for a process code.</td>
</tr>
<tr>
<td>3-10</td>
<td>The algorithm for generating code for the switched construct.</td>
</tr>
<tr>
<td>4-1</td>
<td>The architecture of ETKnet2.0 system, and its local and remote calls among distributed components</td>
</tr>
<tr>
<td>5-1</td>
<td>Algorithm to determine the applicable rules and construct a processing graph.</td>
</tr>
<tr>
<td>5-2</td>
<td>Algorithm for processing “Suppress()” meta-rule.</td>
</tr>
<tr>
<td>5-3</td>
<td>Algorithm for processing conditional suppression meta-rule.</td>
</tr>
<tr>
<td>5-4</td>
<td>Algorithm for processing “Onetime()” meta-rule.</td>
</tr>
<tr>
<td>5-5</td>
<td>Algorithm for processing “Prerequisite(and(), )” meta-rule.</td>
</tr>
<tr>
<td>5-6</td>
<td>Algorithm for processing “Prerequisite(or(), )” meta-rule.</td>
</tr>
<tr>
<td>5-7</td>
<td>Algorithm for processing “Order()” meta-rule.</td>
</tr>
<tr>
<td>6-1</td>
<td>NPDN Triage Lab rules on receiving a new sample (1)</td>
</tr>
<tr>
<td>6-2</td>
<td>NPDN Triage Lab rules on receiving a new sample (2)</td>
</tr>
<tr>
<td>6-3</td>
<td>NPDN Triage Lab rules on receiving the confirmed result.</td>
</tr>
</tbody>
</table>
6-4 SPRO rules on being informed of a presumptive positive sample in the system and the confirmed result. ................................................................. 105

6-5 SPHD rules on being informed of the confirmed result. ........................................ 105

6-6 NPDN Regional Hub Lab rules on receiving a presumptive positive sample and being informed of the confirmed result. ........................................ 106

6-7 NPDN Regional Director rules. .............................................................................. 107

6-8 APHIS-PPA CDD rules on the receipt of a presumptive positive sample, and the release of the confirmed result. ........................................ 107

6-9 NIS, EDP and RPM rules on receiving the confirmed result. ................................ 108

6-10 Part of event data transmission, rule processing and process enactment in the NPDN Domain. ........................................................................ 108

6-11 The processing graph generated by the Host in the seventh round .............. 109

7-1 The rule "FulfillOnlineOrderWithPartnersRule" defined by the retailer for fulfilling an online order with the help of partners........................................ 117

7-2 The rule "PurchasingOrderProcessingRule" defined by a supplier to fulfill the retailer’s purchase order................................................................... 118

7-3 The interactions between the retailer and its suppliers to completing an online order.................................................................................. 119
META-RULE ENHANCED INTEROPERATION OF RULES AND PROCESSES FOR ACHIEVING DYNAMIC INTER-ORGANIZATIONAL COLLABORATION

By
Xuelian Xiao

May 2012

Chair: Stanley Y. W. Su
Major: Computer Engineering

In today's complex and changing world, government and business organizations worldwide often find the need to share their resources and coordinate their activities in order to solve many complex problems and stay competitive. Resource sharing should not be limited to data and computing resources. The sharing and interoperation of organizational policies, constraints, regulations and processes are also very important for achieving inter-organizational collaboration. Organizational resources as well as organizations' partnership can change at any time in a dynamic environment. Thus, inter-organizational processes that carry out inter-organizational collaboration and coordination must be dynamically constructed and processed.

This work aims to research on and develop an integrated rule and process specification language, an Event-Triggered Knowledge network (ETKnet 2.0) system, and a meta-rule enhanced, distributed rule processing and process enactment technique to achieve the sharing and interoperation of organizational resources, and the dynamic construction and processing of inter-organizational processes for achieving inter-organizational collaboration and coordination. The integrated specification
language is for use by collaborating organizations to independently specify 1) events of common interest, 2) policies, constraints and regulations in terms of different types of business rules (integrity constraints, derivation rules and action-oriented rules), 3) manual/automated operations, and 4) workflow processes. A process is defined in the form of a structure of activities, which invoke different kinds of rules and/or automated/manual operations. An action-oriented rule, when triggered by an event instance, can enact a process, which may contain rules to conditionally enact other processes, making rules and processes interoperable. The specified rules and processes are automatically translated into Java code and then packaged as Web services for their uniform discovery, processing, sharing and interoperation in a Web service infrastructure. The meta-rule enhanced, distributed rule processing and process enactment technique implemented in ETKnet 2.0 enables the dynamic construction of inter-organizational processes in a declarative way, and the processing of inter-organizational processes in a decentralized way. Meta-rules are used to enforce some constraints in the processing of distributed rules and processes defined by collaborating organizations. The introduction of meta-rules also allows collaborating organizations to conduct “what-if” analyses to test or exercise their collaboration in a changing environment. The proposed specification language, the network system, and the rule processing and process enactment technique are applied in two application areas, namely e-government and e-business, to demonstrate their utilities for inter-organizational collaboration and coordination.
CHAPTER 1
INTRODUCTION

Government organizations worldwide are facing many complex problems such as disease detection and control, border control, immigration, disaster assistance and recovery, and terrorism. The solutions to these and other problems require these organizations to collaborate, share their resources, and coordinate their activities rapidly and effectively. In the business area, the rapid globalization also requires business organizations to share their resources and collaborate with one another in a timely manner in order for them to stay competitive and to be successful. In a collaborative environment, resource sharing should not be limited to data and computing resources. The sharing and interoperation of organizational policies, constraints, regulations and processes are also very important for achieving inter-organizational collaboration. Hence, there is a need for a high-level specification facility for collaborating organizations to specify their sharable data, application operations, business rules and processes, and a way to make these resources sharable and interoperable to support seamless collaboration and coordination among organizations.

Organizational policies, constraints and regulations have been specified in terms of different types of rules (Business Rules Group, 2000; Rouvellou et al., 2000; Sowa, 2000): integrity constraints (Ullman, 1982), derivation rules (Ullman, 1988), and action-oriented rules (Widom and Ceri, 1996). Business rules model organizational knowledge in a declarative way. Their specifications are easier for non-technical users to understand than the programming code that implement them. Organizational behaviors that can be expressed in terms of manual operations, automated operations and processes are defined using a workflow specification language such as the Workflow...
Process Definition Language (WPDL) (WfMC, 1998), or a business process modeling language such as the Business Process Modeling Notation (BPMN) (OMG, 2009-b) or the Web Service Business Process Execution Language (WSBPEL) (OASIS, 2007). Data, rules and business processes are usually managed by different types of software systems such as a database management system (e.g. MySQL), a rule management system (e.g. (RedHat)'s JBoss Enterprise BRMS), and a workflow management system (e.g. IBM’s MQ Workflow (IBM)) or a business process execution system (e.g. Microsoft’s BizTalk (Microsoft, 2009)). One problem with this traditional method of data, rules and processes management is that it is costly to develop/purchase and maintain different kinds of software systems. What is more, it is difficult to make these systems, as well as the data, operations, rules and processes that they manage, interoperable. By interoperability, we mean that the availability of some data may require some knowledge rules to be activated automatically. The processing of these rules may infer more data or enact some business processes, which perform some manual or automated operations. The rule activation and process enactment would produce new data, which may require additional rules and processes to be activated. This type of interoperation can continue until all relevant operations, rules and processes have been performed.

Business rules can be used in business process modeling as shown in (Bae et al., 2004; Charfi and Mezini, 2004; Eijndhoven van, Lacob, and Ponisio, 2008; Kappel, Rausch-Schott, and Retschitzegger, 2000; Knolmayer, Endl, and Pfahrer, 2000; Meng et al., 2006). It has been recognized that combining business process modeling and business rule specification is beneficial (Herbst et al., 1994; Prezel, Gašević1 and
Milanović, 2010; Recker et al., 2005). However, these works either model business processes entirely by using business rules (Kappel, Rausch-Schott and Retschitzegger, 2000; Knolmayer, Endl and Pfahrer, 2000), or use business rules (especially ECA rule) to handle exceptions (Bae et al., 2004), or achieve process adaptability at various points of process execution (Charfi and Mezini, 2004; Eijndhoven van, Lacob, and Ponisio, 2008; Meng et al., 2006; Rosenberg and Dustdar, 2005). In these works, business rules and business processes are defined by different definition languages or handled by different execution engines. They are not fully integrated and specified in a single specification language. Recent efforts reported in (Graml, Bracht and Spies, 2008; Milanovic and Gašević, 2009) have started to investigate the integration of process-oriented business process modeling with rule-oriented business process modeling. Milanovic and Gasević (2009) point out that “the relationship between process-oriented business process modeling and business rule specification is one of the most interesting research directions”. They propose a hybrid language named rBPMN by integrating the existing process modeling language BPMN with the existing rule specification language REWERSE I1 Rule Markup Language (R2ML) (Rewerse Working Group I1, 2006). In rBPMN, a business rule is used to represent a small chunk of business logic and a business process is modeled by the process modeling language BPMN. Business rules are incorporated in the BPMN’s element named Gateway, and are executed presumably by invoking an existing rule execution engine. However, no implementation of such an engine was mentioned in their publications. In our work, we aim to use business rules to model entire business logics, regulations and processes. A process can be specified by a structure of different types of rules and
manual/automated operations by using structural constructs, and is enacted by an
action-oriented rule. Rules and processes are all automatically translated into Java code
and wrapped and executed as Web Services. Our work provides a new way to fully
integrate the rule specification and the process modeling. We argue that it is beneficial
to come up with more ways to fully integrate the two.

Another important issue that needs to be addressed is the dynamic nature of
government/business organizations and their collaborative relationship. Organizations’
data, policies, constraints, regulations, services and processes can change at any time.
Process changes become necessary when, for example, new policies come into effect,
business goals need to be modified and temporary partnerships need to be created.
This means that the rules and processes, which capture organizations’ resources,
knowledge and collaborative activities, will need to allow for modifications and/or
additions to occur at any time. Traditionally, an inter-organizational process is pre-
defined at build-time, and processed by a workflow management system or a business
process management system at runtime. Once defined, it is difficult to make changes to
accommodate the changing demands. The need for flexible process modeling and
execution has long been recognized by the workflow and business process
communities. Meeting this need will enable organizations to adapt to changing
environments and partnerships as pointed out in (Heinl et al., 1999).

Systems developed in the business process management domain have widely
adopted the workflow technology. A business process model specifies precisely how a
given set of activities should be performed (i.e. the order of processing these activities is
explicitly defined). This modeling approach makes a business process definition very
rigid, and hence difficult to capture and adapt to the changing demands of business and business organizations (Nurcan, 2008). Several past efforts have attempted to overcome this limitation (Eijndhoven van, Lacob and Ponisio, 2008; Meng et al., 2006; Müller, Greiner and Rahm, 2004). There are also efforts to find alternative ways from different perspectives to model and enact inter-organizational processes that achieve inter-organizational collaboration (Alexopoulou et al., 2008; Asunction, Lacob and van Sinderen, 2010; Blake and Gomaa, 2005; Buhler and Vidal, 2004; Grefen, 2009a; Orriens and Yang, 2006). To some degree, these existing approaches still require that a global specification of inter-organizational collaborative activities be defined in advance. The global specification can be a complete inter-organizational process, but augmented with flexible elements, like business rules, to make the process adaptive. The global specification can also be an abstract one, for example defining by an e-contract (Kutvonen, Metso and Ruokolainen, 2005), which defines the agreement and/or behavior of collaborating organizations and is later decomposed into the local processes defined by the involved collaborating organizations. However, in a dynamic environment, organizations can join or leave a collaboration federation freely. The participating organizations may be anonymous to one another and may not have the global knowledge about their collaboration. It is difficult to design a global specification in advance in such an environment.

To address the above challenges, we have developed in this work an integrated rule and process specification language for collaborating organizations to specify their organizational policies, constraints, regulations and processes. We also developed a meta-rule enhanced, distributed rule processing and process enactment technique to
enable the interoperation of organizational data, rules and processes, and the dynamic construction and processing of inter-organizational processes without having to define a global specification in advance.

We first introduce an XML-based Integrated Rule and Process Specification Language (IRPSL) (Chapter 3) for specifying business rules and processes. The language provides a declarative way to specify human and organizational knowledge and collaborative processes. Distinct from the traditional process specification language, a process in our language can be defined as a structure of activities that can activate three types of business rules (integrity constraint rules, derivation rules and action-oriented rules), manual operations and automated operations. An action-oriented rule can enact a process specified in the rule’s action clause. The process may contain rules to further enact other processes, which make rules and processes interoperable. The rules and processes specified in our language are first automatically translated into Java code and then packaged and exposed as Web services for their uniform discovery, processing, sharing and interoperation among collaborating organizations in a Web service infrastructure. Our work provides a new way to combine business rule specification and business process modeling by extending process modeling to include rule specifications, and by extending rule specifications to enact processes. The advantages of our approach are as follows. First, rules and processes are defined in an integrated specification language. It is not necessary for organizations to treat rules and processes as separated entities and define them by using two different specification languages. Second, the defined rules and processes can be uniformly processed as Web services to enable their interoperability. They do not have to be processed
separately by a rule processing engine and a workflow/business process processing system.

In this work, we also have implemented a distributed Event Triggered Knowledge network 2.0 (ETKnet2.0) system (Chapter 4) based on the meta-rule enhanced, distributed rule processing and process enactment technique mentioned before. In addition to the facilities for collaborating organizations to define rules, processes and manual or automated operations, the system also provides the facility for these organizations to define different types of events. Here an event is anything of common interest to these organizations, such as the outbreak of a disease, the placement of an order, the announcement of a new product or service, a change in the internal state of a database, or a real world incident, or a slip in a production schedule. Sharable events, rules and processes are registered at the Host site of ETKnet2.0. Hence they are searchable and accessible by all end users of collaborating organizations. We further allow organizations to subscribe to events, and define triggers to link events to rules. When an event occurs (i.e. an event instance), data relevant to that event instance (i.e. the event data) is transferred to all the subscribers through an event notification mechanism. Additionally, the event data is automatically sent to the network sites of organizations that contain rules, which can make use of the event data (or part of it) as their input data (Chapter 4). We call these rules "applicable rules" (i.e., applicable to the event instance). The processing of these rules may enact organizational processes defined by collaborating organizations. These processes may contain different types of rules and operations. The data produced by these rules and operations may in turn activate other distributed rules and processes. The above interoperations will continue
until all rules, operations and processes that are relevant to an event instance have been performed. We also introduce meta-rules to enforce some constraints in processing distributed rules and processes upon an event occurrence. They can be used to specify the constraints on the selection and execution of applicable rules (Chapter 5). For example, a meta-rule may state that “If rule A is selected for processing and rule C is not, then rule B should be suppressed”, or “If rule A and rule B are both selected, rule A should be processed before rule B”. Collaborating organizations can use meta-rules to achieve the desired collaboration without precisely predefining global inter-organizational processes. We call the above distributed processing technique the "meta-rule enhanced, event-triggered, distributed rule processing and process enactment technique". The adaptation to changing execution contexts is achieved by 1) using flexible constructs in the specification language such as conditional branches in the activity structure of a process; 2) allowing organizations to define meta-rules to specify the desired collaboration without precisely predefining inter-organizational processes; 3) allowing organizations to add/change their business rules, processes and meta-rules at build-time as well as runtime; 4) dynamically constructing and processing inter-organizational processes to achieve resource sharing and collaboration among organizations. Collaborating organizations can also use meta-rules to conduct “what-if” analyses to test or verify their assumptions of their collaborative relationship.

We have evaluated our research results in two application domains. The first application of our developed technology is in the e-government domain. We use IRPSL and the ETKnet2.0 system to implement the USDA’s National Plant Diagnostic Network Chain of Custody and Chain of Communication Standard Operating Procedure (NPDN
SOP) (National Plant Diagnostic Network, 2008) defined by USDA’s National Plant Diagnostic Network. We test and evaluate our system by creating application scenarios to exercise the Standard Operating Procedure and by conducting a field test with the participation of the NPDN staff. The second application is in the e-business domain. We use a scenario abstracted from the real e-business world to demonstrate the general application of the developed technology and thus its broader impact.

The significance and contributions of this research can be summarized as:

It provides a new way to integrate business rule specification and business process modeling. Our integrated rule and process specification language allows the business rules, processes and operations of collaborating organizations to be defined in an integrated manner and thus facilitates their interoperability. Their high-level specifications are easier for the users of these organizations to understand, modify and use than the program code that implements them.

The automatic translation of three different types of rules and processes into Web services allows them to be discovered, shared and processed uniformly in a Web service infrastructure just like other Web services. Organizations do not have to write programming code to implement them.

The event-triggered, distributed rule processing and process enactment technique allows for the inter-operation of sharable data, different types of rules, processes and operations, without having to use multiple rule engines and business process execution systems.

The use of meta-rules to enforce constraints in the processing of distributed rules and processes provides an alternative way of dynamically constructing an inter-organizational process at runtime. Meta-rules are also useful for organizations to perform "what-if" analyses to exercise or test their collaboration in a changing environment.

Since new rules, processes and meta-rules can be introduced and existing rules, processes and meta-rules can be modified at any time, the implemented system is very flexible and can easily adapt to changes in business and collaborative agreement between organizations. It is also a very general system and can be applied in different application domains such as the e-government and e-business domains as shown in this dissertation as well as the e-learning domain presented in (DePree, Su and Xiao, 2009; DePree, Xiao and Su, 2010). Also, the system is highly scalable since the software components of the system can be easily replicated at different network sites with the growth of the number of member organizations.
The rest of this PhD dissertation is organized as follows. Chapter 2 surveys some related works and points out the main differences between them and our work. Chapter 3 details the specification language IRPSL with examples, and shows how rules and processes can be automatically translated into Java code, and packaged and exposed as Web services. Chapter 4 presents the architecture of our distributed network system, and how inter-operation of sharable knowledge can be achieved by its event-triggered, distributed rule processing and process enactment technique. Chapter 5 explains the use of meta-rules to enhance the rule processing and process enactment technique to achieve the dynamic construction and processing of inter-organizational processes. Chapter 6 covers the implementation of NPDN SOP using our integrated specification language and our meta-rule enhanced distributed rule processing and process enactment technique, and the results of a field test of the developed system carried out by the NPDN staff. Chapter 7 covers another application of the developed technology in the e-business domain that uses our language and system to achieve inter-organizational collaboration. The Chapter 8 summarizes our research problems, our approaches to solve these problems, and our main research contributions. Some issues for future research are also presented in this chapter.
CHAPTER 2
RELATED WORKS

We survey some research works in three areas that are related to our work: business rule language design, the integration of business rules in business process application, and the dynamic construction and execution of inter-organizational business processes.

2.1 Business Rule Language Design

A business rule is “a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business” (Business Rules Group, 2000). Business rules describe business logic, policies, regulations and processes that apply to business organizations. A business rule language is a formal specification that is designed to formalize the organization’s business rules in a declarative way that managers and technologists can understand.

Several industry standards have been established for business rule language specifications. Some research works have also proposed their own rule languages. We briefly describe them below.

2.1.1 General-Purpose Rule Specification Languages

Simple Rule Markup Language(SRML). SRML (Cover, 2001), proposed by ILOG, S.A. in 2001, aims to address the problem of sharing rules among applications that use different java rule engines. It was realized that Java rule engines on the market at that time had their own set of Java APIs, and their own proprietary rule languages. SRML is provided as an interlingua for rule exchange between these rule engines. It describes a generic rule language consisting of a subset of language constructs
common to the popular forward-chaining rule engines. Rules in this language have a condition part and an action part. The action in the action part can include variable declarations and assignments, as well as the traditional assertion, retraction and modification statements of rule languages. However, the action part of our language can express much more complex structures. Other types of rules supported in our work are not specifiable in this language.

**Rule Markup Language (RuleML).** RuleML (Rule Markup Language Initiative, 2010), was launched in 2000 by the Rule Markup Language Initiative, formed by an open network of individuals and groups from both industry and academia, and is still evolving. Its main goal is “to provide a basis for an integrated rule-markup approach”. This is achieved by “having all participants collaborate in establishing translations between existing tag sets and in converging on a shared rule-markup vocabulary”. It aims to cover the entire rule spectrum, from derivation rules to transformation rules to reaction rules. Transformation rules could be conversely reduced to derivation rules over special relations that have an extra argument for transformation values. Based on its latest version (v1.0), its current design has well-defined derivation rules, but reaction rules are only briefly mentioned and their definitions and syntaxes are not provided. Our specification language adopts some syntactic constructs from its derivation rule’s specification.

**Extensible Rule Markup Language (XRML).** XRML (Lee and Sohn, 2003) is designed for knowledge sharing between human and software agents. It is an extension of XML with additional capabilities for representation of rules. Therefore, it can represent and process rules that are implicitly embedded in a web page. It has three
components: the Rule Identification Markup Language (RIML) which can be used to specify rules in a web page, the Rule Structure Markup Language (RSML) which converts rules embedded in the web into a formal structure usable by an expert system, and the Rule Triggering Markup Language (RTML), which defines the conditions under which those rules will be triggered. This language only supports deductive rules.

**Semantic Web Rule Language (SWRL).** SWRL (Horrocks et al., 2003) aims to extend the set of the Web Ontology Language (OWL) (McGuinness and Van Harmelan, 2004) axioms to include Horn-like (deductive) rules. Rules in SWRL are expressed in terms of OWL constructs like individuals, properties, literals and classes. Rules are in the form of an implication between an antecedent and a consequent. The antecedent is the rule body, and the consequent is the rule head. It means that when the conditions specified in the rule body are true, the conditions specified in the rule head must also be true. While SWRL is not a standard, it is a widely-used language supported by several commonly used reasoners, like Pellet (Sirin et al., 2007).

**Rule Language in OWL (ROWL).** ROWL (Gandon, Sheshagiri and Sadeh, 2004) is developed by the mobile commerce lab at Carnegie Mellon University for representing user preferences (including complex privacy/confidentiality preferences) in the context of their pervasive computing work. It allows for the specification of deductive rules (in the form of Horn clauses) using an ontology in OWL and provides the facility for translating these rules into Java Expert System Shell (JESS) rules.

**Semantics of Business Vocabulary and Rules (SBVR).** SBVR (OMG, 2008) is an adopted standard of the Object Management Group (OMG) intended to be the basis for a formal and detailed natural language declarative description of a complex entity,
such as a business. SBVR contains a vocabulary for conceptual modeling, and captures expressions specified in this vocabulary as formal logic structures. The SBVR vocabulary allows one to formally specify representations of concepts, definitions, instances, and rules of any knowledge domain in a natural language. These features make SBVR well suited for describing business domains and requirements for information systems to implement business models. SBVR's schema includes the syntaxes of constraints and derivation rules.

**Production Rule Representation (PRR).** PRR (OMG, 2009a) was defined by vendors of business rules engines such as ILOG, Fair Isaac, LibRT, IBM, Pega, Corticon, TIBCO, academic community (RuleML.org), and UML tool vendors. The current version is now adopted as an OMG standard, and a formal model for production rules. A production rule is a statement of programming logic that specifies the execution of one or more actions in the case that some data conditions are satisfied. It is typically represented as “if [condition] then [action-list]”. Some implementations extend this definition to include an “else” construct as follows: “if [condition] then [action-list] else [alternative-action-list]”. The latter production rule can be represented by two condition-action rules in our language: one with the condition being true and the other with the condition being false. What is more, the action clause in our action-oriented rule can activate a process with a much more complex structure than is possible with PRR.

**REWERSE I1 Rule Markup Language (R2ML).** R2ML (Rewerse Working Group I1, 2006) was originally developed by the REWERSE Working Group I1 to support rule interchanges between different systems and tools. It is also a comprehensive rule modeling language, which integrates the Object Constraint Language (OCL), SWRL,

The above general-purpose rule languages support only one or at most two types of rules that are supported by our rule language, or have limited support for structuring a set of rules (Bry et al., 2006) such as R2ML. In practice, structuring a set of rules and treating it as a reusable module is very important for reducing the complexity of a business process specification and the tediousness of rewriting the same rules in different business process specifications. Compared with the above mentioned rule languages, our language provides good support for not only different types of rules, but also the specification of processes, which are structures of different types of rules and/or operations and can be enacted using action-oriented rules. Thus, individual rules as well as structures of rules and operations modeled as processes can be made reusable.

2.1.2 Our Own Earlier Effort

The rule specification language reported in (Degwekar, 2007) is our earlier effort on rule language design. It can be used to specify three types of rules: constraint rules, logic derivation rules and action-oriented rules. The language also supports the specification of a "rule structure" by using four structural constructs: link, split, and-join, and or-join. The action clause of an action-oriented rule can only be a primitive operation. In our language IRPSL, we adopt and support the specifications of the same types of rules. However, separate from our earlier work, an action-oriented rule can
enact either a primitive operation or a workflow process. A workflow process can be specified as a complex structure of activities that invoke different types of rules, and/or manual and/or automated operations. Thus, the rule structure of our earlier work is a special case of our workflow process. The structural constructs allowed in our process definition are: Sequential, Switched, Unordered, Selective, Repeated, AND/OR/XOR-Split, AND/OR-Join. Furthermore, different from our earlier work, which aims to enhance knowledge sharing among organizations, our current work aims to achieve, not only knowledge sharing, but also dynamic and flexible inter-organizational process modeling and execution.

2.2 Integration of Business Rules in Business Process Application

The works on using business rules in business process application can be categorized based on the two approaches used. One approach is to model business processes using only business rules. This approach is taken in (Bry et al., 2006; Kappel, Rausch-Schott and Retschitzegger, 2000; Knolmayer, Endl and Pfahrer, 2000; Zeng et al. 2002). These works consider only one type of rule (i.e., the action-oriented rule). For example, (Knolmayer, Endl and Pfahrer, 2000) demonstrates how workflow patterns, including sequence, parallel, alternative and iteration, can be modeled by Event-Condition-Action-Alternative-action (ECAA) rules. Taking the sequential order as an example, it can be specified by an ECAA rule in the following way: “On (the termination of the predecessor) If (condition) Then (execute the successor) Else (do nothing)”. In these works, business process enactment is achieved by using reasoning algorithms. However, the execution of a business process by using reasoning algorithms might “lead to some unexpected behavior hard to determine upfront” (Milanovic and Gaševic, 2009), and the entire process specification in terms of rules will be rather long and
difficult to read because each rule only captures a small chunk of business logic and there are many rules needed to model a process. Distinct from this approach, our work extends a rule language to incorporate process enactment in an action-oriented rule, and a process is defined by a structure of rules of different types and/or manual/automated operations/services. Business process enactment is achieved by a distributed rule processing and process enactment technique (Chapter 4) instead of using reasoning algorithms.

The other approach to using business rules in business process applications is to incorporate business rules in business process modeling and enactment. Bae et al. (2004) incorporate ECA rules in a process model to deal with the exceptions thrown during a workflow enactment. Muller, Greiner and Rahm (2004) use ECA rules to specify exceptions and support workflow adaptation. Meng et al. (2006) incorporates ECA rules in process modeling by introducing the Before-Activity-Event, After-Activity-Event, and External Event in the workflow modeling language WPDL. These events can trigger the processing of business rules during the enactment of a workflow process to enforce business constraints and policies, and hence dynamically change the execution behavior of an inter-organizational business process. Works in (Charfi and Mezini, 2004) and (Rosenberg and Dustdar, 2005) incorporate different types of rules in a BPEL process. However, in (Charfi and Mezini, 2004), rules are represented as an aspect and are hard-coded in the process using aspect-oriented programming (AOP). Rosenberg and Dustdar (2005) proposes to incorporate rules by introducing before/after-interceptor in a BPEL process specification. Business rules associated with the before-interceptor are executed before the actual BPEL activity, and those associated with the after-
interceptor are executed after the BPEL activity. These two works only provide an execution infrastructure rather than a formal specification language. Eijndhoven van, Lacob. and Ponisio (2008) propose a method to implement the variable parts (called variation points) of a process flow by using so-called workflow patterns discussed in (Russell et al., 2006), and then by representing these patterns using business rules in order to make the business process flexible. The business rules used in the examples given in the paper are ECA rules; no other kinds of rules are discussed. Furthermore, the work presents a method on how to implement business processes, rather than a specification language that incorporates three types of rules in process modeling as proposed in our work. Graml, Bracht and Spies (2008) point out that it is beneficial to incorporate business rules in business process modeling. It uses examples to demonstrate how to express three group patterns (control flow decisions, data constraints and process composition) by using three kinds of business rules (derivation, constraint and ECA rules). However, the work does not provide a formal specification language to couple business rules and business processes. Such a language is said to be its future work.

The work presented in (Milanovic and Gaševic, 2009) is very relevant to ours but using totally different methods. It proposes a Rule-Based BPMN (rBPMN) language. To the best of our knowledge, rBPMN is the only language beside ours that is designed to support the integration of rule specification and process modeling. rBPMN combines the specification facilities of two existing languages: BPMN (OMG, 2009b) and R2ML (Rewerse Working Group I1, 2006). It extends the BPMN’s element Gateway, which is used for control flow, by defining a new element named RuleGateway to refer to R2ML.
rules. In this way, an R2ML rule can be placed into a process model to model a small chunk of business logic. rBPMN is designed to support a rule-enhanced process orchestration and choreography modeling as presented in (Milanović et al., 2009; Milanović and Gasević, 2010). rBPMN and our language were developed concurrently and independently. rBPMN integrates rule specification and process modeling by extending BPMN’s element Gateway to incorporate rules, while our language achieves the integration by extending an action-oriented rule’s action clause to enact a process, and by including rules in a process model (i.e., rules are integral parts of a process structure rather than things appended to a process structure). Although our language does not support ECA rule directly as in R2ML, its function can be achieved by using our Trigger element, which can link an event to an action-oriented rule (Chapter 4). The inter-organizational process is explicitly modeled in the language rBPMN, whereas it can be achieved through our event-triggered rule processing and process enactment technique (Chapter 4). Also, the execution of rules in rBPMN is presumably achieved by making Web service calls to some existing rule engine. There is no evidence that an engine has been implemented to process the language rBPMN. In our case, we have implemented a rule server, which can be replicated and installed at the network sites of collaborating organizations for processing different types of rules and business processes in a distributed manner.

Our work distinguishes itself from the approaches discussed above by using three kinds of rules to specify organizational policies: regulations, constraints, and workflow processes; not just to handle exceptions, but to specify control flow such as synchronizations or asynchronizations, and model small chunks of business logic. What
is more, our work formally defines an integrated specification language, which enables a complex business process to be defined as a structure that contains different types of business rules and manual/automated operations interconnected by a variety of structural constructs. The defined processes as well as the rules specified in them are automatically translated into Java code and exposed as Web services for their reuse and interoperation.

### 2.3 Dynamic Construction and Execution of Inter-organizational Processes

Systems in the business process management domain have widely adopted the workflow technology. Workflow Management Systems (WFMS) have been positioned as appropriate technological solutions for integrating process islands at a high level [Reijers, 2006]. There are also efforts to develop alternative techniques to coordinate activities across organizational boundaries using techniques that are event-driven (Alexopoulou et al., 2008), rule-based (Meng et al., 2006; Orriens and Yang, 2006), agent-based (Blake and Gomaa, 2005; Buhler and Vidal, 2004; Jiang, Dignum and Tan, 2011; Taveter and Wagner, 2001), contract-driven (Grefen et al., 2000; Kutvonen, Metso and Ruokolainen, 2005), and service-oriented (Asuncanion, Lacob and van Sinderen, 2010; Grefen et al., 2000; Grefen et al., 2009a, 2009b, Kapuruge, Han. and Colman, 2010). Some works apply multiple techniques. In this section, we give an overview of the existing works on dynamic/flexible inter-organizational process modeling and execution that are related to our work, and point out their differences and similarities with ours.

An early effort on cross-organizational workflows was undertaken in the Workflow-based Internet SErvice (WISE) project (Alonso et al., 1999; Lazcano et al., 2001). WISE provides an Internet-based software infrastructure for supporting business-to-business
electronic commerce within a so-called trading community. Companies in a trading community can post their services in a catalogue so that other companies can search and find them. However, in WISE, cross-organizational business processes that describe the cooperation between organizations are specified at process definition time. They are manually defined by process designers who explicitly incorporate other companies' services in their own business processes.

CrossFlow (Grefen et al., 2000) goes a step further toward cross-organizational collaboration. It aims to develop and implement a mechanism for connecting Workflow Management Systems (WfMS) used in different organizations. The organizational collaboration is based on a dynamic service consumer/provider paradigm. An organization (the service consumer) outsources a pre-defined part of its business process to an organization that can perform this service (the service provider). The interaction between a service consumer and a service provider is based on an electronic contract, in which relevant details of service offerings are specified. Service offerings and service requests are later matched according to their well-defined interfaces through a matchmaking facility presented in (Klingemann, Wäsch and Abere, 1999). In CrossFlow, cross-organizational process enactment is performed by linking the workflow management infrastructures of the involved organizations. In our work, the collaborating organizations’ activities and services are specified by different kinds of rules and processes. These rules and processes are converted into and exposed as Web services, which can be searched and processed uniformly in the Web service infrastructure. The cross-organizational process can either be explicitly defined in our language at build-time, or dynamically constructed at runtime.
DynaFlow presented in (Meng et al., 2006) uses the concepts of event, rule and triggers to achieve flexible and adaptive workflow processing. This work extends the definition of activities in WPDL to include Before-Activity-Event, After-Activity-Event and External Event, and to also define triggers that link events with rules. During the execution of a workflow process, these events are posted to trigger some action-oriented rules, which perform operations to adapt the workflow instance to suite the changing business environment, report the processing milestones of an enacted business process, and/or handle an exception condition. In DynaFlow, inter-organizational processes are pre-defined. The system uses only action-oriented rules in the form of “condition-action-alternative-action” instead of all three types of rules used in our work. The action clause of a rule can only perform a single operation, whereas it can enact a process in our work. Besides user-defined triggers that explicitly link events with rules, our system can automatically create triggers if the data associated with events can provide the input data for rules. Furthermore, the dynamic and adaptive inter-organizational processes can be constructed and enacted at runtime in our work.

Alexopoulou et al. (2008) propose an event-based approach to compose a business process at runtime. The main idea of this approach is to use event chains to compose business processes. An event may either initiate an action, or cause another event, or both. Every Event-Action (EA) pair constitutes an autonomous relationship, and is pre-defined at build-time. The event in an EA pair can be an actual single event or a conceptual single event. The latter is a complex event (David, 2002) which is defined by an expression containing a number of events and logical operators such as AND, OR and XOR. A business process is then the event-action sequences that are
involved at runtime. A conceptual architecture for an event-based IT infrastructure is proposed in the paper, but no implementation is discussed. The cross-organizational issue is not specifically addressed in this paper. In our work, an event instance can trigger different kinds of rules, and processes enacted by action-oriented rules. An inter-organizational business process is either explicitly defined in our language, or dynamically constructed through multiple-rounds of rule processing and process enactment that are triggered by an event instance.

Based on the agent-oriented and service-oriented technologies developed on top of an Internet infrastructure, CrossWork (Grefen et al., 2009a, 2009b) presents a system for collaborating organizations to create and operate an Instance Virtual Enterprise (IVE). Like CrossFlow, it aims to connect the workflow management systems and/or the business management systems of different organizations. An inter-organizational process is defined at build-time in the following way. A pre-defined high-level goal is first decomposed into a set of operational business goals with the help of a domain expert, and the partners that can fulfill these goals are selected in the market by the system with the assistance of a market expert. Later, the system retrieves the external specifications of those selected partners’ local business processes and then semi-automatically composes a structured global business process based on these local business processes by using the algorithm presented in (Eshuis and Grefen, 2009). The algorithm first automatically constructs an abstract dependency graph by using the input/output dependencies among these local services/processes, and then a concrete dependency graph is generated with the help of the domain expert by specifying the branch type such as split or join. A structured global business process is
then generated based on the concrete dependency graph. Therefore, the global business process is explicitly specified at build-time. All the selected partners execute their respective local business processes according to this global business process. Although our work also makes use of input/output dependencies to determine the rule/process dependencies, an inter-organizational process is constructed dynamically at runtime. What is more, the branch types, which are specified manually in this related work, are automatically generated at runtime by using meta-rules introduced in our work. Our system provides a mechanism to dynamically construct a global/inter-organizational business process at runtime without having to define a dependency graph at build-time.

Aiming to increase the flexibility of enterprise application integration, Asunction, Lacob and van Sinderen (2010) integrate enterprise applications based on a goal-based, model-driven and service-oriented framework presented in (Iacob and Jonkers, 2008). In particular, the organization requirements are first specified as goal models (Lamsweerde, 2008) at a high level. Those parts that are more likely to change over time are separated from a process, and are specified in terms of business rules. A rule engine exposes its functions as Web services. These business rules are then executed by making Web service calls to the rule engine, and hence are incorporated into the definition of the business process as services. In our work, processes are enacted through action-oriented rules, and rules are automatically converted into Java code and packaged as Web services. At run-time, these Web services are executed without the use of a rule engine.
To summarize the main differences between our work and the works surveyed in this chapter (i.e., our main contributions), our work introduces a new integrated specification language, which can be used to define an organizational business process or an inter-organizational process by a structure having a mixture of different kinds of business rules and manual/automated operations connected by a variety of structural constructs. Action-oriented rules specified in a process can in turn activate other processes. Business rules and processes defined by different organizations are automatically converted into Java code and exposed and executed uniformly as Web services in a Web service infrastructure without having to use different types of rule engines and a workflow management or business process execution system. Organizations can add and change their business rules and processes at any time. Inter-organizational processes can either be predefined or constructed at run-time using an even-triggered, distributed rule processing and process enactment technique introduced in this work. This processing technique is further enhanced by the use of meta-rules to enforce constraints on the selection and execution of distributed, applicable rules that are triggered by an event instance. The distributed network system that implements this enhanced processing technique enables government/business organizations to share not only data and application operations but also business rules and business processes. The enhanced processing technique also enables the flexible and dynamic construction and execution of inter-organizational processes needed to achieve inter-organizational collaboration and coordination.
CHAPTER 3
INTEGRATED RULE AND PROCESS SPECIFICATION LANGUAGE (IRPSL)

In this chapter, we present the language IRPSL with examples, and show how operations, rules and processes can be automatically translated into code, and packaged and exposed as Web services. This piece of work has been published in (Xiao et al., 2008; Xiao, DePree and Su, 2011).

A business rule is a declarative statement that specifies business logic, constraints, policies or regulations that an organization must apply or enforce in order to achieve its business objectives. There are three popular types of business rules that are commonly used in current rule-based systems (Business Rules Group, 2000; Rouvellou et al., 2000; Taveter and Wagner, 2001): integrity constraints, derivation rules, and action-oriented rules. Integrity constraint rules (Ullman, 1982), originally used in database systems, specify the condition(s) to which some data must adhere as required by an application. Derivation rules, commonly used in expert systems for decision-support (Ullman, 1988), are also known as inference rules or deductive rules. They define how new knowledge can be inferred from the current knowledge (Business Rules Group, 2000). Action-oriented rules define the actions that an organization must perform under certain conditions. They are commonly used in an event-condition-action (ECA) system. Business rules model organizational knowledge and behaviors in a declarative way.

A business process consists of a structure of activities that should be carried out in a specific order that conforms to an organization’s business requirement. In this work, we extend the traditional workflow process specification by including not only manual operations and automated operations, but also three types of business rules. We also
take advantage of the existing Web Service Business Process Execution Language (WS-BPEL) (OASIS, 2007) and the Workflow Process Definition Language (WPDL) (WfMC, 1998) by combining the control patterns(or structural constructs) offered by these two languages and adopting them in our language for specifying the business processes of collaborating organizations.

We extend the action clause of an action-oriented rule to allow the invocation of either a primitive operation or a process. The latter can contain different types of rules and/or operations. In this way, we achieve the integration of business rule modeling and business process modeling. Furthermore, operations, processes and different kinds of rules can interoperate. For example, an action-oriented rule may enact a process, which may invoke other rules and operations to produce new data that can serve as the input to some other rules or operations.

3.1 Operations

Operations can be manual operations or automated operations. A manual operation is a task that has to be done off-system. Each manual operation should be assigned to a certain role(s) and/or a specific person. The system that we developed will send a notification with instructions via email to all the people who play the role(s) or to a specific person. In the former case, one of them can choose to perform the operation. After performing the operation, the performer needs to log into the system to report its completion and provide the data resulting from the operation. The specification of a manual operation includes input data items, output data items (if any), operation instructions that will be sent to the performer(s). An automated operation is executed automatically by the system. It can be either a Web service call, a call to a local
program, a rule activation, or a posting of an event to indicate its occurrence. Its metadata includes input data items and output data items (if any).

Examples of a manual operation

"TriageLab_Inform_Campus_Safety_Office_Of_Housing_Suspected_Sample()" and an automated operation "Retrieve_Lab_Information()" are shown in Figure 3-1. The manual operation has an instance of entity "npdn.Sample" as its input. The operation instruction is included in the “Message” field of the xml specification. The system will send the operation instructions and the input data values to all the users who play the role “Diagnostician”. The automated operation has “npdn.Sample.labId” and “npdn.General.has_distance_diagnosis_capability” as its outputs (we shall skip the prefix (entity name) and use the attribute name directly in the following section). It will invoke the piece of local program code that accesses the database to retrieve the values of current lab’s id and lab’s property “has_distance_diagnosis_capability”, and return these two values.

3.2 Rules

3.2.1 Integrity Constraints

The enforcement of an integrity constraint ensures that any change to data does not break the data consistency requirement imposed by an application. Since we use an object-oriented model to model the data transferred between organizations, constraints on data can be specified by the constraints on the values of an object’s attributes (i.e. attribute constraint) or on the relationship between attributes (i.e. inter-attribute constraint). The former limits the acceptable values that an attribute may have. Examples for this type of rule are ‘Sample.localLabSampleID>0’ and ‘Sample.classification in ("unclassified","suspected","presumptive"
positive”, “positive”, “negative”). The latter states the relationship between attributes modeled either in a mathematical expression (e.g., ‘Order.itemAmount * Item.unitPrice < Customer.creditLine’) or a conditional expression (e.g., if Sample.classification = ‘positive’ then Shipping.method is ‘UPS’). We use the same syntax as that used in our previous work (Degwekar, 2007) to specify these two types of constraints.

3.2.2 Derivation Rules

A derivation rule produces new data if some preconditions on existing data are satisfied, which is also known as an inference rule, or a deductive rule. It captures implicit knowledge that can be derived from existing knowledge. This kind of rule has the following form: $P \implies Q$, which means that, given that the body of the implication $P$ is true (i.e., data expressed by $P$ exist in the current data set), the head or conclusion $Q$ is also true (i.e. data expressed by $Q$ should be added to the current data set.). One example of this rule type is $Customer.numOfPurchase > 3 \implies Customer.discountLevel = 3$. We use the same syntax for the specification of this type of rule as that used in (Degwekar, 2007).

3.2.3 Action-Oriented Rules

Action-oriented rules are used for invoking actions in response to the occurrence of events. They are generally expressed in the form of event-condition-action (ECA) rules or event-action (EA) rules (Krishnamurthy and Rosenblum, 1995). An ECA rule states that when an event $E$ occurs, the action $A$ should be executed if the condition $C$ is evaluated to true. Like the work in (Lee, Su and Lam, 2004), we separate the event specification $E$ from the CA rule specification to allow events and rules to be independently defined by different organizations. An event defined by one organization can be conditionally linked to the rules defined by the same or other organizations using
triggers (Chapter 4) in order to coordinate the actions taken by collaborating organizations. The condition part is optional.

In IRPSL, the action clause of an action-oriented rule can enact either a single operation or a process. The latter is a structure of activities that can activate three types of business rules, manual operations and/or automated operations. Examples of action-oriented rules are shown in Figure 3-2 and 3-3.

*SPRO_Prepare_For_Response_Plan* is the name of a manual operation. “*ProcessExampleOnTriageLabReceivingSample*” is the name of a process. Its definition is shown in Figure 3-4. It is required that the operation and process have been defined and exposed as Web services (refer to the following subsections for details) before they can be referenced by the action clauses of these rules.

### 3.3 Processes

A workflow process is modeled by a structure of activities, each of which can invoke a rule, manual operation or automated operation. It is enacted by an action-oriented rule.

#### 3.3.1 Structural Constructs Defined in a Process

By integrating the process patterns defined in WPDL (WfMC, 1998) and WS-BPEL (OASIS, 2007), we use the following structural constructs for defining processes: Sequential, Switched, Unordered, Selective, Repeated, AND/OR/XOR-split, and AND/OR-join.

**Sequential Construct.** It specifies a sequential execution order between two activities. The second activity is activated after the completion of the first one.

**Switched Construct.** It specifies a conditional processing of activities. It is similar to the switch statement in a programming language. The conditions specified on the
branches of this structural construct are evaluated in the order in which they appear, and the first activity whose condition is evaluated to true will be activated. If no conditional branch is taken, then a default branch is taken.

**Unordered Construct.** The activities listed in this construct may be executed in any order, but not concurrently.

**Selective Construct.** This construct allows a number of activities to be selected randomly from a set of activities for execution in an unordered fashion. The exact number is specified in the NUM field of this construct.

**Repeated Construct.** This construct specifies that either a single activity or an ordered set of structural constructs, which make references to either activities or other structural constructs, should be processed repeatedly as long as a specified data condition evaluates to true at the beginning of each iteration.

**Split Construct.** This construct is used to specify that several activities can be executed in parallel. There are three types of split: AND-Split, OR-Split, and XOR-Split. The AND-Split specifies that all successor activities can be executed in parallel after the completion of the predecessor. The OR-Split specifies that a specified number of activities can start their execution in parallel after the completion of the predecessor. The specific number is given in the NUM field of the construct. Furthermore, these activities have to satisfy the data conditions specified for their processing. The XOR-Split is very similar to the switched construct. The differences are that, in the XOR-Split, there is no default branch and only one (not in any specific order) can be activated. Thus, zero or one of the successor activities may be executed.
**Join Construct.** This construct specifies a synchronization point. There are two types of Join constructs: AND-Join and OR-Join. In an AND-Join, the successor activity can only be executed after the completion of all of its predecessors. In an OR-Join, the successor activity can only be executed if the specified number of predecessors has been completed and the data conditions associated with their outgoing transitions are satisfied. The number is given in the NUM field of the construct.

A process can thus be viewed as a directed graph, with some combination of three kinds of rules, automated operations and/or manual operations as its nodes, which are connected by the constructs mentioned above. In order to describe a process in a graph more easily, we also introduce auxiliary activity nodes: Dummy-activity and Route-activity. The Dummy-activity invokes nothing in its body. The Route-activity is used to indicate what the construct that follows belongs to. It can be a selective router, unordered router or repeated router. The rules and operations referenced in a process can be defined by the same organization as the one that defines the process. They can also be defined by other organizations. Not only operations/rules’ names, but also their metadata (input data items, output data items (if any) and location where the operations/rules are defined) are included in the specification of a process. The inputs of a process are the union of the inputs of its rules and operations (i.e., the data needed to process the rules and operations), and its outputs are the union of the outputs of these rules and operations (i.e., the data produced by the rules and operations). However, if an input data item of a rule/operation is in the output data set of the rules/operations that precede it (i.e., the data item is produced by preceding rules/operations), then this data item will not be included in the process’s input.
3.3.2 An Example of a Process

Figure 3-4 shows an example process. It is an example taken from the implementation of NPDN SOP (National Plant Diagnostic Network, 2008) (Chapter 6). The process describes the procedure after a Triage Lab receives a suspected sample from a sample submitter. In this process, firstly, the automated operation “Retrieve_LabInformation()” accesses a database to retrieve the lab’s id and the value of the lab’s capacity “has_distance_diagnosis_capability”. An assignment operation follows to set the sample’s status as “triagelab”. “Examine_Sample()” is a manual operation which instructs the users to examine the received sample. After the operation is completed, the value of the attribute “Sample.classification” can be assigned. Then “Derivation Rule1” will change the sample’s status accordingly. An OR-Split construct is next. In this structure, either the manual operation “Store_Sample()” or the manual operation “Store_Sample_In_Secure_Place()” will be executed according to the sample’s classification. Another action-oriented rule “TriageLabCARule2” is invoked after the completion of “Store_Sample_In_Secure_Place()”. The rule is followed by a split construct, whose branches include three action-oriented rules, “TriageLabCARule3”, “TriageLabCARule4” and “TriageLabCARule5”. These branches are synchronized by an action-oriented rule, “TriageLabCARule6”, which specifies the procedure to prepare a follow-up diagnosis request, including the packing and shipping of the sample. The last manual operation “Inform_Campus_Safety_Office()” follows “TriageLabCARule6” sequentially.

3.4 Events

We separate the Event part from the Condition-Action part of an Event-Condition-Action rule to allow these two parts to be defined by different organizations.
independently. An event can be anything of interest to collaborating organizations that occurs at a particular point in time. Examples of events include the announcement of a special sale, the breakdown of a delivery truck that can affect a delivery schedule, the breakage of a machine that can affect a production schedule, the receipt of a new sample that calls for its diagnosis. Each event is defined by a name, a human-readable description of its meaning, and a schema that specifies the data entities and their attributes. For example, Figure 3-5 shows the schema of the event “TriageLabE1” (Triage Lab receiving a new suspected sample) (Chapter 6). When the event occurs, the data values generated for these data items form the “event data”. An event can be local or global. A local event is only of interest to the organization that defines it. Its occurrence may trigger some rules defined by the site that defined the event. A global event is of interest to some or all of the collaborating organizations. Its occurrence may trigger the processing of rules and processes defined by these interested organizations. All global events are published at the host site of a collaborative federation, and are thus searchable and sharable by collaborating organizations. An organization can subscribe to an event and receive its event data when that event occurs by defining a trigger (Chapter 4), which conditionally links an event to a knowledge rule. Hence, a global event which occurs at one site can automatically trigger the processing of applicable rules defined by all sites, including the site of the event occurrence.

3.5 Sharable Operations, Rules and Processes as Web Services

In a distributed, collaborative environment, operations, rules and processes are defined independently by collaborating organizations. We need a way to uniformly process them and make them interoperable. The approach we take is first to translate the defined operations, rules and processes into Java code, and then package them as
Web services, which shall be called “Operation Web Services (OWSs)”, “Rule Web Services (RWSs)”, and “Process Web Services (PWSs)”, respectively. These Web services are then deployed on the server at the site that defined them, and their metadata, including their names, input data items and output data items are also stored in a local database.

OWSs, RWSs and PWSs are created in the way shown in Figure 3-6. When operations, rules and processes are defined through the user interface, their specifications are stored in XML specification files. These specifications are used to generate Java source code files (lines 1-5). Each source code file contains an interface (header) file and an implementation file. The implementation file is the source code generated from the specification of an operation/process/rule. Each operation/rule/process is translated and packaged as a web service with a Web service operation implemented by a method in the Java language. The specification of input and output parameters for the operation is represented by the method’s signature. If more than one data item is included in the output parameters, then the method returns a java.util.HashMap data structure in the Java language. The source code is then compiled to generate the corresponding Java class files (line 6).

After this translation process, configuration files required for their successful compilation and deployment as Web services are then created (line 7). These configuration files would depend on the specific application framework used. In our work, we deploy these Web services on the Sun Application Server 9.0. The configuration files include web.xml, sun-web.xml, config.xml and jaxrpci-ri.xml, which are needed for a web application deployed on the Sun Application Server. Line 8 calls
the “wscompile” command of the Sun Application Server to generate a WSDL file, a model file and a mapping file. After all these files are ready, they are then packed as a deployable package and deployed to Sun Application Server (line 9).

3.5.1 Operations as Web Services

Manual operations and automated operations defined in IRPSL are all exposed as Web services. The Web service operation of an IRPSL manual operation will send an operation instruction to all possible performers, and wait for one of them, who takes on the responsibility of performing the operation, to indicate its completion and enter the data resulting from the operation into the system. The Web service operation of an IRPSL automated operation is to execute the local programming code or invoke a Web service. The OWS has an operation/method with the same name as that of the manual or automated operation. For example, if an IRPSL operation named myOp is defined, then the generated Web service will have a method named myOp, and the name of the OWS is “myOpClass”. The Java code method myOp(…) that implements the OWS operation sends an email to notify the operation’s potential performer(s) and wait for the indication of completion, if the operation is a manual operation, or to invoke a Web service, or a local program code, if it is an automated operation. The method also has the input data items of the IRPSL operation as its parameters, and returns the values of the data items that are specified in the “Returns” field of the IRPSL operation specification.

3.5.2 Rules as Web Services

We use the same algorithm used in (Degwekar, 2007) to translate an integrity constraint rule or a derivation rule into Java code. We define the input of a constraint rule as all of the attributes that are referenced in the rule. The output is the truth value
indicating whether the constraint is satisfied or not. The input and output information of this and other rules, as well as processes, are used by the system to determine the applicability of rules and processes to an event instance (Chapter 4). Each integrity constraint rule is represented as a Web Service with an operation named check. The Java code method check(…), which implements the RWS’s operation, examines the supplied value of input and returns a Boolean value which is true if the constraint is satisfied or false otherwise. All data items referenced in the rule constitute the input to Java code method check(…). This input is also the input of the RWS operation. Each derivation rule is represented as a Web Service with an operation named implies. The Java code method implies(…), which implements the RWS operation, examines the input data to determine if the body of the implication is true. If so, it returns the values of data items specified in the head of the implication. All data items referenced in the body of the implication and in the “value” part (where the “value” part is another data item) of the head constitute the input to the method implies(…). This input is also the input of the RWS operation, as well as the input of the rule.

Each action-oriented rule is represented as a Web Service with an operation named perform. Figure 3-7 shows the general structure of the generated Java code for an action-oriented rule. The Java code, which implements this operation (i.e. the method perform(…) in the figure), examines the input data to see if the condition is satisfied. If so, the IRPSL operation/process specified in the action clause is executed by invoking the corresponding OWS/PWS, and the output of the invocation is returned. The input of the action-oriented rule is the union of the data items referenced in the condition clause, and the input data items of the IRPSL operation/process specified in
the action clause. The output of the action-oriented rule is the output of the IRPSL operation/process referenced in the action clause.

We require the following precondition to be met in order for the generated RWS of an action-oriented rule to work correctly: The IRPSL operation or process referenced in an action-oriented rule must have already been defined and deployed as a Web service before the rule is defined. The OWS/PWS must use the name of the IRPSL operation/process as the name of its Web service operation. Hence, the execution of the operation/process given in the action-oriented rule is achieved by invoking the corresponding OWS/PWS with the same name (the name of the Web service of an operation named myOp is myOpClass).

3.5.3 Processes as Web Services

A process is defined as a structure of activities. Each activity can be a rule or an operation. In the following, when we say an “activity”, it either refers to an operation or a rule. We use AWS to refer to the generated Web service of an activity. We require the following precondition to be met to generate a PWS. The activity referenced in a process specification must have already been defined and deployed as an AWS before the process is defined. The AWS must have the same name as the name of the activity. The PWS has the same name as the process, and has an operation named perform. The Java method perform(...) that implements the PWS operation takes in all the input data items required by all the activities in the process (if an input data item required by an activity can be produced by its ancestor activity, then it will not be included in the method’s input), and returns the output data items produced by all these activities by adding them to a hash table data type in the Java language named java.util.HashMap.
Figure 3-8, 3-9 show the general structure for a process code generated by our algorithm.

Before implementing the method \textit{perform}(\ldots), our algorithm first defines an invoker thread class named \textit{callActivityName} for each distinct activity referenced in the process specification (here ActivityName shall be replaced by a real activity name). Since each activity is executed by invoking the corresponding AWS, the invoker is actually a Web service call, but is wrapped in a Java thread class as shown in Figure 3-8. An invoker thread object will be created for each activity instance (an activity may be referenced more than once) in the process specification. The structure of activities that defines a process is decomposed into a set of structural constructs that form the structure. Each structural construct specifies a sequential, switched, unordered, selective, repeated, AND/OR/XOR-Split, or an AND/OR-Join relationship between two or more activities or constructs. Programming code that will cause the activity to be executed in the manner specified by the construct is then generated for each construct. The Java code method \textit{perform}(\ldots) that implements the PWS operation is constructed by putting together all the code generated for these constructs.

The method \textit{perform}(\ldots) is, essentially, a do-while loop. The loop keeps on checking a vector named \textit{toBeExecuted}, which includes all the activity instances that need to be executed, until the vector is empty. When an activity instance is ready for execution, its activity instance is inserted into the vector \textit{toBeExecuted}, and its corresponding invoker thread object is started. Once the thread is finished, it will be removed from this vector, and its output values will also be copied to the process’s member variables and used by the activity instances that follow the current activity
instance. The predecessor(s) specified in a structural construct are invoked first, followed by the successor(s) specified in the same construct. Besides the completion of the predecessor(s), the processing of a successor would depend on whether the data condition specified for the transition between the predecessor and the successor is satisfied. It also depends on whether its processing will follow the restriction that is inherent in the structural construct. For example, the successor in the AND-Join construct can only be processed after the completion of all the predecessors. Only one successor can be executed after the completion of the predecessor in the XOR-Split construct. Our algorithm checks whether an activity invoker thread has been started or terminated successfully according to the thread’s two Boolean type member variables \textit{is\_started} and \textit{is\_finished}. These two variables have the value ‘false’ before the thread is started, and the value of \textit{is\_started} becomes ‘true’ after the thread is started and the value of \textit{is\_finished} becomes ‘true’ when the thread terminates successfully. In order to avoid the situation that a condition is not satisfied first but becomes satisfied later when the data gets updated by the subsequent activities, we introduce a vector named “\textit{terminatedEvent}”. After an activity instance is checked for execution, it is added to this vector no matter whether it can be executed or not. By doing so, we can make sure that each activity instance is only checked once for its execution.

For illustration purposes, we show the algorithm for generating code for the switched construct in Figure 3-10.
Figure 3-1. Examples of manual operation and automated operation.

...<Rule>
  <RuleName>CA1</RuleName>
  <RuleDescription>an example of action-oriented rule</RuleDescription>
  <RuleState>SUSPENDED</RuleState>
  <Sharing>local</Sharing>
  <CARule>
    <Condition>
      <CondExpr>
        <BooleanExpr>
          <PredicateExpr>
            <Expr>
              <Term>
                <DataType>
                  <Name>npdn.General.SPRO_informed_presumptive_positive_sample</Name>
                  <Type>boolean</Type>
                </DataType>
              </Term>
            </Expr>
            <RelOp>eq</RelOp>
            <Expr><Term><Value value="true">true</Value></Term></Expr>
          </PredicateExpr>
        </BooleanExpr>
      </CondExpr>
      <Action>
        <Operation>
          <Name>SPRO_Prepare_For_Response_Plan</Name>
          <Location>localhost</Location>
          <Inputs>
            <setDataItem><Name>npdn.Sample.*</Name><Type>npdn.Sample</Type></setDataItem>
          </Inputs>
          <Returns>
            <setDataItem><Name>npdn.ResponsePlan.*</Name><Type>npdn.ResponsePlan</Type></setDataItem>
          </Returns>
          <Type>manual</Type>
        </Operation>
      </Action>
    </Condition>
  </CARule>
  ...

Figure 3-2. An example of action-oriented rule whose action part is to invoke an operation.
Figure 3-3. An example of action-oriented rule whose action part is to invoke a process.
Figure 3-4. An example process named “ProcessExampleOnTriageLabReceivingSample”.

Figure 3-5. An example of an event named “TriageLabE1”.

Sample.labld, has_idistance_diagnosis_capability = Retrieve_Lab_Information()

Sample.status = “triagelab”

Sample.classification = Examine_Sample(Sample.labld, Sample.localLabld, Sample.sampleDate)

Derivation Rule 1: if Sample.classification = “negative” then
Sample.status = “confirmed diagnosis”

Sample.classification = “negative”

Or-split:

Sample.classification = “presumptive positive”

OrJoin:

Store_Sample(Sample.sampleDate, Sample.localLabld, Sample.labld)

Inform_Termination(Sample.*)

TriageLabCARule2

TriageLabCARule3

TriageLabCARule4

TriageLabCARule5

TriageLabCARule6 (Process further diagnosis request)

Inform_Campus_Safety_Office(Sample.*)
Function **CreateWebServices**(XML s)

1. if (s is a operation) opHandler(s); end if
2. else if (s is an integrity constraint) icHandler(s); end if
3. else if (s is a derivation rule) drHandler(s); end if
4. else if (s is an action-oriented rule) caHandler(s); end if
5. else if (s is a process) processHandler(s); end if
6. compileSourceCode();
7. createConfigFiles();
8. callWSCompile();
9. deploy();

End Function

**Figure 3-6.** The routine for creating OWSs, RWSs and PWSs.

```
package myrules.rulename;
import statements;
public class RuleName implements RuleNameIF {
    public java.util.HashMap<Object, Object> perform(input data item values) {
        java.util.HashMap outputValues = new java.util.HashMap<Object, Object>();
        if(code for Condition) {
            Prepare the code to invoke the corresponding Web service of the operation/process specified in the action clause of the rule;
            Invoke the Web service;
            Get the return of the Web service, and put all the output items in the hashmap "outputValue" to be returned;
        }
        return outputValues;
    }
}
```

**Figure 3-7.** General structure of code for an action-oriented rule.

```
class callActivityName extends Thread{
    declaration of variables for activity input and output parameters;
    public boolean is_started = false;
    public boolean is_started = false;
    public java.util.HashMap<String, Object> mapResults = new java.util.HashMap<String, Object>();
    //constructor to assign values to the member variables
    callActivityName(values){
        assignment statement for Initialization of member variables
    }
    public void run(){
        is_started = true;
        try{
            //prepare for invoking Web service
            Statements for preparing for invoking Web service;
            Invoke the Web service;
            Get the return of the Web service, and put them in the member variables representing activity's output parameters
            is_finished = true;
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

**Figure 3-8.** General structure for generated invoker thread class code (it is a part of generated process code showed in Figure 3-9).
package myprocesses.processname;

import statements;

//before you begin the process class, write invoker thread for each activity
Code for defining invoker thread class for each distinct activity in the process

public class ProcessName implements ProcessNameIF {
    //here, declare ALL the inputs and outputs, just so they are available to all activities
declaration of variables for process input and output parameters;

    public java.util.HashMap<String, Object> perform(process's input data items) {
        //prepare to hold returned items
        java.util.HashMap<String, Object> outputValues = new java.util.HashMap<String, Object>();

        //here, declare ALL invoker object for the activities
        for each activity instance in the process
callActivityName activityNameID = new callActivityName(activity's input data items);

        try {
            java.util.Vector<String> terminatedEvent = new java.util.Vector<String>();
            java.util.Vector<String> toBeExecuted = new java.util.Vector<String>();

            do{
                the code for each construct con in the process. Here only take a sequential construct as an example
                //pred <- con.predecessor; succ <- con.successor
                if (!pred.is_started & & toBeExecuted.contains(pred.id)) {
                    terminatedEvent.add(pred.id);
                    toBeExecuted.add(pred.id);
                    for each input data item of predecessor, update its value
                    pred.start(); // start to execute the predecessor
                }

                if (pred.is_finished & & toBeExecuted.contains(pred.id)) {
                    for each output data item of predecessor, update process’s corresponding output variable member.
                    toBeExecuted.remove(pred.id);
                }

                if (pred.is_finished & & !toBeExecuted.contains(pred.id)) {
                    if (!succ.is_started & & toBeExecuted.contains(succ_id) & & !terminatedEvent.contains(succ.id)) {
                        terminatedEvent.add(succ.id);
                        if (condition) { //condition is the data condition along the transition between pred and succ.
                            toBeExecuted.add(succ.id);
                            for each input data item of successor, update its value
                            succ.start();
                            }
                    }
                }

                if (succ.is_finished & & !toBeExecuted.contains(succ.name)) {
                    for each output data item of succ, update process’s corresponding output variable member.
                    toBeExecuted.remove(succ.name);
                }

            } while (toBeExecuted.size() != 0);

            put all the produced output items in the HashMap outputValues

        } catch (Exception e) {
            e.printStackTrace();
        }
        return outputValues;
    }
}

Figure 3-9. General structure for a process code.
Function writeSwitchedActHandler(Element switchedActivities) {
    code = "";
    predecessor = switchedActivities->predecessor;

    // If the code for executing this activity has not been created. It is possible that the predecessor is a successor in the other constructs that have been processed.
    if (!handledActivities.contains(predecessor.id)) {
        // This thread has still not started processing
        code += "if (!predecessor.is_started && terminatedEvent.contains(predecessor.id)) { ";
        code += " terminatedEvent.add(predecessor.id);"
        code += " toBeExecuted.add(predecessor.id);"
        for each input inputItem of predecessor
            code += "predecessor." + inputItemName + "+" = inputItem + "",
        end for

        // Start to execute the predecessor activity
        code += "predecessor.start();"

        // Predecessor is terminated, now get the output of the activity
        code += "if (predecessor.is_finished && toBeExecuted.contains(predecessor.id)) { 
            for each output outputItem
                code += this." + outputItemName + "+" = predecessor.outputItemName;
            end for
            code += toBeExecute.remove(predecessor);
            code += "}" // End for predecessor is terminated

            handledActivities.add(predecessor.id);
            code += "}"
        end if

        // Get successors
        succList = switchActivities->successors;
        code += "if (predecessor.is_finished && toBeExecuted.contains(predecessor.id)) { 
            for each Activity act in succList:
                code += "if (act.is_started && terminatedEvent.contains(act.id)) { 
                    code += " terminatedEvent.add(act.id);""
                    if it is in a case branch, then code += "if (" + code for the condition expression + ") { ";
                    if it is a "otherwise" branch, then code += "else ( 
                    code += toBeExecuted.add(" + act.id + ");"
                    for each input dataltem
                        code += act.dataltemName + "= this." + dataltemName;
                    end for
                    code += act.start();

                    code += "if (act.is_finished && toBeExecuted.contains(act.id)) { 
                        code += toBeExecute.remove(act.id);
                        for each output outputItem of act
                            code += this." + outputItemName + "= act.outputItemName +";
                        end for;
                        code += "}" // End for if (act.is_finished.....
                        code += "}"
                    code += "}" // End for if (predecessor.is_finished.....
                    handledActivities.add(act);
                    return code;
            }
        }
    }

Figure 3-10. The algorithm for generating code for the switched construct.
CHAPTER 4
ETKNET2.0 SYSTEM ARCHITECTURE AND ITS EVENT-TRIGGERED PROCESSING TECHNIQUE

In this chapter, we present an implemented system, named Event-Triggered Knowledge Network 2.0 (ETKnet2.0), and the event-triggered rule processing and process enactment technique for achieving the interoperation of event data, knowledge rules, operations and processes, and the dynamic construction and enactment of inter-organizational processes. A part of this work has been published in (Su et al., 2011; Xiao, DePree and Su, 2011).

4.1 ETKnet2.0 System Architecture

ETKnet2.0 has the similar architecture as that of our earlier system ETKnet (Degwekar, 2007; Degwekar and Su, 2008; Su et al., 2008). It is a peer-to-peer server-based system as shown in Figure 4-1. Each organization, called a collaborating site, has identical software installed on it. The software components include a User Interface, a Rule Server, an Event Server, and a database system. The User Interface provides facilities for the users of each organization to define events, event data, triggers, rules, operations, processes and meta-rules. It generates language specifications for internal processing based on what has been defined. This tool is important and useful because, in a distributed, collaborative environment, it gives the flexibility and power to specify business logic, policies, regulations, constraints and processes of individual organizations. In addition to the above facilities for knowledge management, the User Interface also provides facilities for configuration management, administrative management, task management and ontology management. It has been developed by another student, Jeff DePree, in our research group, so it will not be covered in this dissertation.
The Rule Server component is responsible for managing the operations, rules, processes and meta-rules defined at each site, for translating the defined operations, rules and processes into Java code and packaging them as Web services, and for invoking the applicable rules at that site after receiving an event notification and the event data. The Event Server component is responsible for managing the events and triggers defined at that site. When an event occurs at that site, the Event Server will act as the coordinator of a dynamic inter/intra-organizational process construction and processing session initiated by the event instance.

One of the collaborating sites plays the role of Host for the collaborative federation. The Host has an additional software component: a Web Service Registry. The specifications of sharable operations, rules and processes of all the collaborating organizations are registered with the Web Service Registry. Therefore, they are searchable and sharable by members of the federation. The Rule Server component of the Host, named the Host Rule Server, is responsible for carrying out the registration of sharable operations, rules and processes in the Web Service Registry. Meta-rules defined by collaborating organizations are made available to the Host but are not registered with the Web Service Registry because they are not converted into Web Services at design time. Rather, they are interpreted at run-time. Since the Host has all the information about sharable rules and meta-rules, it also determines those distributed rules that are applicable to an event instance and their processing structure based on the meta-rules and the event data sent from the coordinator. The Event Server component, named the Host Event Server, is responsible for registering global events.
and triggers, and forwarding the event data received from the coordinator to the Host Rule Server.

Another additional software component at the Host is the Ontology System. The Ontology System is responsible for semantic matching among entities and attributes. The collaborating organizations might use their own terms to define events, operations, rules and processes. It is often the case that the terms they use have semantic discrepancies. Thus, an ontology management system is needed to reason over the underlying meanings of the terms used, and resolve the semantic discrepancies among them. This part of the work is done by another member in our group as shown in (Zhou et al., 2010). This dissertation will not address ontology issues. We shall assume that the semantic ambiguities and inconsistencies of all the data items used by the collaborating organizations can be resolved by querying the ontology management system. The underlying implementation method that we use is as follows: when we save the metadata of events, operations, rules and processes, we also save the corresponding unique object ids used and stored in the ontology management system. When we search for specific data items, we use their object ids to do the matching.

We have implemented the ETKnet2.0 system in the Java language and have hosted it on the Sun Application Server Platform Edition 9. The Event Server and the Rule Server at each site are implemented using the Enterprise JavaBean 2.1 framework. We published the Web services that implement application operations, rules and processes in the private Web Service Registry of the Host. The registry makes use of a MySQL database to store the Web services’ meta information, including inputs, outputs, and their WSDL file’s addresses. The MySQL database is also used by the
Event Server and the Rule Server to store the meta information of events, triggers, rules, processes and operations locally.

**4.2 Event-triggered Rule Processing and Process Enactment Technique**

Traditionally, data, operations, knowledge rules and processes have been managed by different types of software systems such as database management systems, rule processing systems, application systems and business process execution systems (or workflow management systems). These systems, and the data, rules and processes they manage, do not interoperate. By interoperation, we mean that, for example, the availability of some data may cause business rules to be activated automatically. The processing of business rules may in turn enact some processes, which perform some manual or automated operations. The operations may then produce data, which may subsequently cause additional rules to be evaluated. Also, an inter-organizational process is usually pre-defined and processed by a business process execution system or a workflow management system. This “template approach” will not work well in a loosely coupled collaborative environment because the business rules and processes and other resources of collaborating organizations can be removed and/or updated at any time. It is costly to redefine and process an inter-organizational process each time an organizational resource has been changed. Also, automatic runtime modification of the process to accommodate changes is not simple and may not be workable if the changes to the pre-defined process are drastic. Ideally, an inter-organizational business process should be constructed and processed dynamically.

In this section, we explain our event-triggered distributed rule processing and process enactment technique, and show how it can achieve the interoperation of
sharable data, rules and processes. This technique can also be used to achieve dynamic construction and processing of inter-organizational processes.

4.2.1 Event Instance and Event Data

When an event occurs, it forms an event instance and the data associated with the event instance constitute the event data. The event data include all the data items that are assigned values upon an event occurrence, and those data that are added and/or modified later during the processing of rules, which are either explicitly or implicitly triggered by the event instance (the details will be given below). Event data are saved in an XML document, which is transmitted between collaborating organizations during the collaboration session triggered by the event instance. The XML schema of the event data is provided in Appendix A.

In ETKnet2.0, an event instance can be posted either synchronously or asynchronously. If an event instance is posted asynchronously, the procedure is returned immediately without waiting for the result of processing the event instance. If it is posted synchronously, the procedure waits for the result. When an event instance is posted synchronously, a time interval can be specified. If the time interval is expired but the event processing is still not finished, the current event processing result (if any) is returned.

4.2.2 Triggers

As we have explained before, we separate the Event part from the Condition-Action part of an event-condition-action rule. These two parts can be defined independently by collaborating organizations. A trigger is a specification in the format of “Condition; Event; Rule”, which conditionally links an event to a rule. It states that if the event data associated with an event instance satisfies a certain data condition, the
specified rule should be processed. The *Condition* part could be null. In that case, the rule is processed unconditionally upon the occurrence of the event. The condition part is useful if an organization wants to be notified only when the event data satisfies a certain condition. For example, a supplier might want to be notified with a quote request event for a quantity above a certain level.

An organization can explicitly define a trigger to link a local event with a rule defined by the organization itself, or by some other organization. It can also define a trigger to link a global event defined by another organization with a rule defined locally. Multiple triggers defined by many organizations can link the same event to different rules. The occurrence of an event will trigger the processing of all these local and global rules as well as the processes enacted by action-oriented rules. The above approach maximizes sharing of events, rules and processes.

Defined triggers are called *explicit triggers* and the organizations that define the rules are *explicit subscribers*. Triggers can also be automatically and dynamically generated by the system. If the event data items associated with an event instance can provide the input data required by a rule, we regard this rule as potentially *applicable* to the event. The applicability of rules will be explained in detail in the next section. The processing of applicable rules and the processes enacted by them may change the event data or produce new data, thus adding new information to the event data. System-defined triggers are called *implicit triggers* and the organizations that have applicable rules are *implicit subscribers*. When an event occurs, the event data will be sent to all the subscribers’ sites to trigger the processing of their applicable rules. The inclusion of implicit subscribers in event notification and event data transfer is important.
because these organizations may not know of the existence of the event nor its occurrence, or may not know that the event data can provide input to their rules to produce useful data and/or to enact relevant processes.

4.2.3 Applicability of Rules

We have explained how rules can be translated into Java code and exposed as RWSs and PWSs in Chapter 3. Each rule has a set of input data items and a set of output data items. Its applicability to an event instance is determined by the Host using a syntactic matching process followed by a semantic matching process. The syntactic matching is based on the names of data entities and their attributes referenced in the two sets: the event data item set and the rule’s input data item set. It checks whether the entity and attribute names of the event data item set is a superset of those entity and attribute names referenced in the rule’s input data item set. If not, the rule is not applicable. If so, the event data can potentially provide the input data needed to process the rule. However, this matching process can only determine if the rule is "potentially applicable" because the data values of the event data and the rule’s input data have not been considered. The semantic matching process is thus carried out to determine if the data condition(s) specified by the rule can be satisfied by the event data values. If so, the rule is said to be "applicable" because the rule can be processed to produce new data or modify the event data.

The syntactic matching process enables our system to pre-filter out those distributed rules, the inputs of which cannot be satisfied by the event data. The semantic matching process enables the system to identify those distributed rules that are truly applicable. Thus, the event data can be transferred to only those sites that have applicable rules. These two matching processes allow the system to avoid
unnecessary processing of business rules that are not truly applicable, and to save data transmission time.

4.2.4 Event-Triggered, Distributed Rule Processing and Process Enactment Technique

Similar to the ETKnet system, the ETKnet2.0 system processes distributed rules using an event-triggered mechanism. When a global event occurs at a site, the site becomes the coordinator of the collaboration and knowledge sharing session. The event data is sent by the Event Server of the coordinator’s site to the Event Servers of all the sites that have applicable rules. These sites are the explicit and implicit subscribers of the event. When the subscriber sites receive the event data, the applicable and most up-to-date rules are activated automatically by their corresponding Rule Servers. The processing of these rules may in turn enact business processes specified in action-oriented rules to perform the manual and/or automated operations and/or rules that define the processes. The execution of rules and operations may produce new data and/or update the existing event data. After processing all of the applicable rules, the explicit and implicit subscriber sites would return the new and/or updated data to the coordinator. When the coordinator receives these data from the subscriber sites, it merges the returned data with the original event data to form a new version of the event data. The coordinator site’s rule that is linked to the same event can then be applied to process this new version. This processing produces yet another new version of the event data, which may cause some other rules at the same and/or different sites to become applicable. In that case, the coordinator will send the new version to explicit and implicit subscribers in another round of event notification and rule processing. Multiple rounds can take place until no new data and/or updated data have been
generated and no applicable rules can be found. At that time, the collaboration and
knowledge sharing session terminates. Through this event-triggered processing of
distributed organizational rules and embedded processes, ETKnet2.0 achieves the
interoperation of sharable data, rules, operations and processes and dynamically
constructs and enacts an inter-organizational process that carries out inter-
organizational collaboration or coordination.

The above processing technique allows all explicit/implicit subscriber sites to be
notified of the event occurrence. The ETKnet 2.0 system also supports a restricted
event notification. That is, only a subset of subscriber sites are notified. This is achieved
by adding a field named “RestrictedSites” in the event data that is sent to the Host. The
Host then only checks the global rules defined by these restricted sites. This processing
technique is useful when the coordinator wants only a subset of subscribers to be
notified of the event occurrence and to receive the event data.

Our event-triggered rule processing and process enactment technique enable the
processing of all applicable rules and organizational and predefined inter-organizational
processes and operations. After rules processing session is finished, all relevant
organizations and their staffs will received all the data associated with the original event
as well as the data generated by these rules, processes and operations.

We should also point out here that the above distributed rule processing technique
produces a similar effect to that of a logic-based deductive system in that rules are
repeatedly processed as new data are produced in the previous round of rule
processing. However, the main differences are that our system processes three
different types of business rules and business processes in an integrated and
interoperable fashion, and rules and processes are distributed and processed uniformly as Web services in a Web service infrastructure.

4.3 The Need to Enhance the Rule Processing and Process Enactment Technique

In our system, an organization can define an inter-organizational process that contains a structure of rules and operations shared by multiple organizations. This kind of predefined inter-organization process is a “stable” process that collaborating organizations know ahead of time to be useful and should be carried out upon the occurrence of an event. At runtime, our current event-triggered rule processing and process enactment technique may trigger other distributed applicable rules that are not involved in predefined inter-organizational processes. The multiple rounds of rule processing and pre-defined processes enacted by these rules constitute an inter-organizational process. Such a process is “dynamic” in that it is constructed on-the-fly.

However, the above approach of constructing an inter-organizational process may not be adequate to set up a desirable interaction among organizations. As the application environment changes, collaborating organizations may want to enforce some constraints in processing those applicable rules that are not embedded in pre-defined inter-organizational processes. For example, in a certain circumstance, collaborating organizations may want to ensure that Rule A defined at one site be processed before Rule B of another site; or they may want rule B not to be processed at all even though it is applicable to an event instance; or they may want to give a higher priority to an organization over other(s); i.e., the rules (and the processes activated by them) provided by one organization is more important or reliable than some other organization(s). In order to constrain the processing of distributed rules that have been determined to be applicable to an event instance, we propose using “meta-rules”. A
meta-rule is a specification of a constraint on the execution of applicable rules or a constraint on the priorities of applicable sites. We shall explain the meta-rules, as well as the meta-rule enhanced, dynamic construction and enactment of inter-organizational process in detail in Chapter 5.

Figure 4-1. The architecture of ETKnet2.0 system, and its local and remote calls among distributed components.
CHAPTER 5
META-RULE ENHANCED DYNAMIC CONSTRUCTION AND PROCESSING OF INTER-ORGANIZATIONAL PROCESSES

In Chapter 4, we explained our event-triggered rule processing and process enactment technique. We also pointed out that the processing technique can, not only achieve the interoperation of sharable data, rules, operations and processes, but also trigger the processing of applicable distributed rules, which may or may not be referenced in pre-defined inter-organizational processes. Those distributed rules that are not referenced in pre-defined processes are processed at different sites in parallel (i.e., not in a specific order or structure). As we pointed out in Chapter 4, such an order or structure may be needed in some situations.

One way to enforce their order/structure of processing is to put them in a pre-defined, inter-organizational process as explained in Chapters 3 and 4. However, such a process is static and not easy to change. Some rules even cannot be decided at build-time. Another method that we propose here is to introduce "meta-rules" to constrain the processing of these rules. A meta-rule is a specification of a constraint on the execution of applicable rules or on the priorities of collaborating organizations. It can be introduced or removed at any time (i.e., dynamically) with the agreement of collaborating organizations.

Meta-rules are optional. They can be defined for an event by collaborating organizations and are interpreted at runtime to enforce the order/structure of rule processing when the event occurs. Their use is a flexible way to capture and react to the changing requirements of collaboration in a dynamic environment because they can be introduced, modified and removed at runtime.
Organizations can also use meta-rules to perform a “what-if” analysis based on the distributed rules and processes that have been defined. These organizations may suspect or assume, but not necessarily be sure that, by applying some constraints in the processing of distributed, applicable rules, they may produce some desirable data and/or achieve some desirable inter-organizational interactions. Meta-rules can be used to verify their suspicions or assumptions.

5.1 Meta-rules

A meta-rule is a specification of a constraint on the execution of applicable rules or on the priorities of collaborating organizations. There are three types of constraints that can be specified by meta-rules: selection constraint, structure constraint and priority constraint. Selection constraints specify whether applicable rules should be conditionally or unconditionally selected for execution. Structure constraints specify the execution order/structure of applicable rules. Priority constraints specify the priorities of organizations; thus the precedence of the meta-rules defined by them (Section 5.2.3).

5.1.1 Meta-rules on Selection Constraints

Selection constraints can be specified by three meta-rules: the “Suppress()” meta-rule, the “Onetime()” meta-rule and the conditional suppression meta-rule. Their syntaxes are defined as follows.

Suppress ($A_1$, ..., $A_n$). It specifies that any rule in the set ($A_1$, ..., $A_n$) ($n\geq 1$) should not be selected.

Onetime ($A_1$, ..., $A_n$). It specifies that any rule in the set ($A_1$, ..., $A_n$) ($n\geq 1$) can only be executed once in multiple rounds of rule execution triggered by an event instance.
If (ConditionExpr) then Suppress(B₁, …, Bₘ). It is called the conditional suppression meta-rule. It specifies that if the “ConditionExpr” is evaluated as true, then all the rules in the set (B₁, …, Bₘ) (m>=1) should not be selected. The definition of “ConditionExpr” in the Backus-Naur Form (BNF) syntax is “ConditionExpr := RuleBooleanExpr {and/or RuleBooleanExpr} |ConditionExpr {and/or ConditionExpr}”. The “RuleBooleanExpr” is a Boolean expression of rules, and defined as “RuleBooleanExpr := [Not]Rule”. The curly brackets denote that the item enclosed by them can appear zero or more times. The square brackets denote that an item enclosed by them is optional; i.e. the item can appear either zero or one time. The “Suppress()” meta-rule and the “Onetime()” meta-rule are special cases of the conditional suppression meta-rule.

In the above definitions of meta-rules as well as the ones given later, by saying that a rule is selected, we mean that the rule is selected for processing from the set of rules that has been determined to be applicable to an event instance using the syntactic and semantic matching processes introduced in our work. All applicable rules that are not suppressed by meta-rule(s) are processed by the system.

5.1.2 Meta-rules on Structure Constraints

Structure constraints can be specified by three meta-rules: the “Prerequisite(and(),)” meta-rule, the “Prerequisite(or(),)” meta-rule, and the “Order()” meta-rule. Their definitions are shown below:

Prerequisite (and(A₁, ...,Aₙ), B). It is also called the and-Prerequisite meta-rule. It specifies that rule B can only be processed if all of rules A₁,...,Aₙ have been processed (n>=1). If n is equal to 1, then the constraint can be rewritten as Prerequisite(A₁, B).
Prerequisite(or(A₁, …, Aₙ), B). It is also called the or-Prerequisite meta-rule. It specifies that rule B can only be processed if one of Aᵢ (1≤ᵢ≤ₙ) has been processed (n>1).

Order (A₁, …, Aₙ). For any i and j, where 1 ≤ i < j ≤ n, Aᵢ should be processed before Aⱼ.

We note here that we do not have to use a structure constraint to specify the parallel execution of rules because our event-triggered, distributed rule processing and process enactment technique explained in Chapter 4 executes all applicable rules in parallel by default.

5.1.3 Meta-rules on Priority Constraints

A priority constraint specifies the relative priorities assigned to collaborating organizations as shown below:

\[
\text{Priority}((O₁, ..., Oₙ) > (O_{n+1}, ..., O_{n+m}) > ... > (O_{p+1}, ..., O_{p+k}))
\]

Organizations shown in the inner pair of parentheses have the same priority, and the set of organizations shown on the left of ">" has a higher priority than the one shown on the right. If no priority is assigned to an organization, then it is given the lowest priority. The priorities of organizations determine the precedence of meta-rules defined by them (more details will be given below).

5.2 Application of Meta-rules

After determining the applicability of rules by the syntactic and semantic matching processes, meta-rules are applied to filter out some of these applicable rules and/or enforce their processing order/structure.
5.2.1 Application of Meta-rules on Selection Constraints

Meta-rules on selection constraints are first applied to filter out some rules from the set of applicable rules resulting from the syntactic and semantic matching processes. They are applied in the following ways:

**Suppress** \((A_1, \ldots, A_n)\). Rules \(A_1, \ldots, A_n\) will be suppressed, that is, removed from the set of applicable rules in each round of rule processing.

**Onetime**\((A_1, \ldots, A_n)\). For any rule \(A_i\) \((1 \leq i \leq n)\), if it has been executed in a previous round, then it will be removed from the set of applicable rules in the current round as well as in the subsequent rounds.

**Conditional suppression meta-rule**, i.e. “If (ConditionExpr) then Suppress \((B_1, \ldots, B_m)\)”: If the “ConditionExpr” is evaluated as true, then rules \(B_1, \ldots, B_m\) are removed from the set of applicable rules in the current and/or later rounds. Otherwise, those rules in \((B_1, \ldots, B_m)\) that are applicable in the current round can be selected. A rule specified in a RuleBooleanExpr, say “RuleA” is evaluated as true if it is either applicable in the current round or has been executed in a previous round. "Not RuleA" in a RuleBooleanExpr is evaluated as true if RuleA is neither applicable in the current round nor has been executed in an earlier round.

5.2.2 Application of Meta-rules on Structure Constraints

Meta-rules on structure constraints are used to enforce the processing order/structure of the applicable rules determined by the syntactic and semantic matching processes and the selection constraints.

**Prerequisite_AND** \((A_1, \ldots, A_n, B)\). This meta-rule is applied only if rule B is applicable in the current round. If all rules \(A_1, \ldots, A_n\) either have been processed in the previous round(s) or are applicable in the current round, then B is processed right after
those A’s that are applicable in the current round. Otherwise, B’s processing is delayed until all A_i (1<=i<=n) have been processed. If there is any A_i that is never triggered, then rule B will not be processed.

**Prerequisite(or(A_1, ..., A_n), B).** This meta-rule is applied only if rule B is applicable in the current round. If there is a rule A_i (1<=i<=n) that has been processed in an earlier round, then rule B is processed just like any other applicable rule. If A_i is applicable in the current round, B is processed right after A_i. If none of A_1, ..., A_n were processed in an earlier round or are applicable in the current round, B’s processing is delayed. The delayed rule B will not be processed if none of A_1, ...A_n are executed at the end of rule processing.

**Order (A_1, ..., A_n).** For any pair (A_i, A_j) where (1<=i<j<=n), if both A_i and A_j are applicable in the current round, then A_i will be processed before A_j. If only A_i or A_j is applicable, then A_i or A_j will be processed.

### 5.2.3 Application of Priority Meta-rules

The meta-rules defined by the collaborating organizations may have conflicts. A conflict is resolved according to meta-rules’ precedence. When a conflict is detected by the system, the meta-rule with a lower precedence will be disabled.

The priorities of organizations are used to determine the precedence of meta-rules defined by these organizations. The meta-rules defined by the organization with a higher priority will have a higher precedence. For those meta-rules defined by the same organization or by the organizations that have the same priority, we give the following precedence to different types of meta-rules. Meta-rules on selection constraints are given a higher precedence than meta-rules on structure constraints because the former are used to filter out some applicable rules and the latter are used to construct the
processing order/structure of the remaining applicable rules. Among the meta-rules on selection constraints, we give a higher precedence to the “Suppress()” meta-rule and the conditional suppression meta-rule than to the “Onetime()” meta-rule because if a rule specified in the “Onetime()” meta-rule is suppressed, there is no need to enforce the one-time semantics on that rule. Furthermore, the “Suppress()” meta-rule has a higher precedence than the conditional suppression meta-rule because the former can be viewed as a special case of the latter that has "True" as its condition specification. Among the meta-rules on structure constraints, the “Prerequisite(and(),)” meta-rule and the “Prerequisite(or(),)” meta-rule are given a higher precedence than the “Order()” meta-rule because the former will be enforced across multiple rounds of rule processing and are special cases of the latter. Furthermore, the “Prerequisite(and(),)” meta-rule has a higher precedence than the “Prerequisite(or(),)” meta-rule because the former specifies a stronger precondition than the latter.

In a specific round of rule processing, meta-rules are applied according to their precedence. The meta-rule with a higher precedence is applied first.

5.3 Meta-rule Enhanced Rule Processing and Process Enactment Technique

In this sub-section, we explain how meta-rules and meta-rule enhanced rule processing and process enactment techniques are implemented in our system. We also explain how they can achieve the dynamic construction and processing of inter-organizational processes.

5.3.1 Determining the Applicable Rules and their Processing Order/Structure

Like the definitions of rules and processes, the specifications of meta-rules are xml-based. Their schemas are given in Appendix A. A priority meta-rule is associated with a global event, which can only have one such meta-rule. We assume that there is a
high-level decision body that makes such priority decisions for collaborating organizations. Priority meta-rules for global events, if defined, are registered with the Host.

Other types of meta-rules can also be defined for a global event. They can be defined by the organization that defines the event, or by some other organizations. After a meta-rule is defined, it is stored at the local site and also registered with the Host site.

At design time, when an organization defines a meta-rule other than a priority meta-rule, the Host will check if the rule conflicts with any other meta-rule that has been defined for a global event. If so, the conflict will be brought to the attention of and resolved by the person or persons who defined them. For example, the meta-rule “Order(A, B)” conflicts with the meta-rule “Order(B, A)”; and the meta-rule “If (A) then Suppress(C)” conflicts with the meta-rule “Prerequisite (A, C)”. However, some conflicts in meta-rules cannot be determined at design time because the applicability of rules can only be determined at runtime. For example, “Order(C, A)” conflicts with “Order(A, B)” and “Order(B, C)” only if rule A, B and C are all applicable in the current round of rule processing. “Prerequisite(A,C)” conflicts with “Prerequisite(B,A)” only if rules A and C but not B are applicable. The former requires that A be processed right before C but the latter requires that A should be deferred until B is processed. For those conflicts that cannot be determined at design time, they will be detected at runtime by the Host. These conflicts are resolved by disabling the meta-rule with a lower precedence in the current round of rule processing. If conflicting meta-rules have the same precedence, then the one applied later will be disabled. The Host will send a notification to all the
organizations that defined conflicting meta-rules. They can then resolve their differences offline and modify their rules.

At runtime, when a global event occurs at a site, the site becomes the coordinator for processing the event instance and the distributed rules triggered by the event. As explained, the coordinator gets applicable site and applicable rule information from the Host by sending it the event data in an XML file. This file contains a data field named “ExecutedRules”, which records all the rules that have been executed so far. The file also contains a data field named “DeferredRules”, which lists those rules whose processing have been delayed.

When the Host receives the event data from the coordinator, it first performs the syntactic and semantic matching processes, as we have explained Chapter 4, to determine which distributed rules are applicable to the event instance. We shall call this set of applicable rules the "applicable-rules" set. Then the Host would follow the algorithm shown in Figure 5-1 and apply the meta-rules that have been defined for the event on this set of applicable rules to remove those rules that need to be suppressed in the current round, and construct the processing order/structure for the remaining applicable rules (lines 9 - 28 of the algorithm). The processing order/structure is represented by an acyclic directed graph G. Each applicable rule has a vertex in the graph. If there is an edge going from vertex v_i to vertex v_j, then rule v_i is processed before rule v_j. Applicable sites at this time are those sites that define the applicable rules whose corresponding vertices have no incoming edges in the processing graph. The processing graph is finally converted into an XML document. Each vertex and its incoming and outgoing edges are converted to a “Node” element in the xml file. The
“Node” element has sub-elements “Id” and “Rule” which specify the rule that the vertex represents, including the rule name and the site that defined the rule. The sub-element “ChildRules” lists all the rules that immediately follow this rule in the graph. The sub-element “ParentRules” lists all the rules that this rule immediately follows in the graph. The algorithm also maintains a set called the “associated-meta-rules” for each applicable rule, which stores all the meta-rules that have been applied and are related to this rule.

Meta-rules are applied in descending order of their precedence. As we have explained before, their precedence are determined first by the priorities of organizations that define them, and then by their rule types. If a meta-rule tries to suppress or defer an applicable rule whose “associated-meta-rules” set is not empty, then it conflicts with the meta-rule(s) that has been applied so far; i.e., it conflicts with the meta-rule(s) that have higher precedence than it. If a meta-rule adds an edge that introduces a cycle in G, then it conflicts with the meta-rule(s) that have been applied; i.e., the latter has a higher precedence than the former. We can use the depth-first search method to check whether a directed graph has a cycle. When a conflict is detected, the meta-rule with a lower precedence is disabled in the current round and also saved to the log. The organization that defines the disabled meta-rule will be informed.

For each "Suppress(A_1,\ldots,A_n)" meta-rule, A_i (1<=i<=n) is removed from the “applicable-rules” set and the processing graph G, if its “associated-meta-rules” set is empty (Figure 5-2). A conditional suppression meta-rule “If (ConditionExpr) then Suppress(B_1,\ldots,B_m)” can be applied if its condition expression is evaluated as true. The “ConditionExpr” of a conditional suppression meta-rule is a Boolean expression of rules,
like “A₁ and (A₂ or (Not A₃))”. “A₁” is evaluated as true if rule “A₁” has been executed in a previous round or is in the applicable-rules set. “Not A₃” is evaluated as true if rule “A₃” is neither in the applicable-rules set nor has been executed in an earlier round. If the condition is satisfied and rule Bᵢ’s (1≤i≤m) “associated-meta-rules” set is empty, rule Bᵢ is removed from the “applicable-rules” set and the graph G. If the condition is satisfied but rule Bᵢ’s “associated-meta-rules” set is not empty, rule Bᵢ will be selected. If this meta-rule has ever been applied successfully for at least one Bᵢ, then it is added to the “associated-meta-rules” sets that correspond to the rules that are evaluated as true in the conditional expression. Figure 5-3 shows the algorithm to process a conditional suppression meta-rule. For each rule Aᵢ (1≤i≤n) referred in a “Onetime(A₁,…,Aₙ)” meta-rule, if it is in the “executed-rules” set and its “associated-meta-rules” set is empty, it is removed from the “applicable-rules” set as well as from the graph G. The algorithm to process a “Onetime()” meta-rule is shown in the Figure 5-4.

A Prerequisite(and(A₁,..,Aₙ), B) meta-rule will be applied only if rule B is applicable in the current round as shown in the algorithm given in Figure 5-5. Otherwise, this meta-rule is ignored and A₁,..,Aₙ are processed normally. If the precondition of rule B’s execution is satisfied (i.e., all the rules Aᵢ (1≤i≤n) are either in the “executed-rules” set, or in the “applicable-rules” set, or in both sets), then the algorithm adds a directed edge <Aᵢ,B> from vertex Aᵢ to vertex B for each Aᵢ that is in the “applicable-rules” set. If adding the new edge introduces a cycle into the graph, then this meta-rule is ignored in this round. All edges that have been added for this meta-rule will be removed from G (lines 2-18 of algorithm). The algorithm also adds this meta-rule to the “associated-meta-rules” sets of all applicable rules Aᵢ (1≤i≤n) and rule B if this meta-rule is applied.
successfully. If there is a rule in \((A_1, \ldots, A_n)\) that is neither in the “executed-rules” set nor in the “applicable-rules” set (i.e., the precondition of B’s execution is not satisfied), and B’s “associated-meta-rules” set is empty (i.e., rule B is not referenced in any meta-rules that have been applied so far), then rule B is removed from the “applicable-rules” set and added to the “deferred-rules” set (i.e., B’s execution is deferred until its precondition is satisfied) (lines 23-25 of algorithm). If the precondition of B’s execution is not satisfied (i.e., there is a rule in \((A_1, \ldots, A_n)\) that is neither in the “executed-rules” set nor in the “applicable-rules” set), but B’s “associated-meta-rules” set is not empty, then this meta-rule is ignored in the current round because removing rule B may cause a conflict in the meta-rules that have been applied before.

A Prerequisite(or\((A_1, \ldots, A_n), B)\) meta-rule will be applied only if rule B is applicable in the current round. Otherwise, the meta-rule is ignored and rules \(A_1, \ldots, A_n\) are processed normally. This meta-rule is implemented using the algorithm shown in Figure 5-6. If none of rules \(A_i (1 \leq i \leq n)\) are in the “executed-rules” set or in the “applicable-rules” set, and rule B’s “associated-meta-rules” set is empty, then B is removed from the “applicable-rules” set and added to the “deferred-rules” set. If none of \(A_i (1 \leq i \leq n)\) are in the “executed-rules” set or in the “applicable-rules” set, but rule B’s “associated-meta-rules” set is empty, then this meta-rule is ignored in the current round. Otherwise, it is implemented as a Prerequisite\((A_i, B)\), where \(A_i\) is the rule that has been executed in a previous round (if there is such a rule) or is applicable in the current round.

An Order\((A_1, \ldots, A_n)\) meta-rule will enforce the processing order of each pair of rules \((A_i, A_j) (1 \leq i < j \leq n)\) where \(A_i\) and \(A_j\) are both in the “applicable-rules” set of the current round (Figure 5-7). For each such pair, if there is no path from vertex \(A_i\) to vertex
A_j in the graph, and adding an edge <A_i, A_j> will not introduce a cycle to the graph, then add the directed edge <A_i, A_j> to the graph. If adding the edge <A_i, A_j> would introduce a cycle in the processing graph G, then this meta-rule is ignored in the current round.

5.3.2 Meta-rule Enhanced Rule Processing Technique

At runtime, when a global event occurs at a site, the site becomes the coordinator for processing the event instance and the distributed rules triggered by the event. The coordinator sends the event data to the Host site to get applicable rules and their processing structure.

After receiving event data from the coordinator, the Host carries out the syntactic and semantic matching processes and uses meta-rules to determine the applicable rules and their processing order/structure in a processing graph. The Host returns the processing graph in an XML document, the applicable rule and applicable site information (at this time, applicable sites are only those sites that have applicable rules without incoming edges in the processing graph) to the coordinator. The coordinator then sends the event data and the processing graph to the applicable sites. After a site receives the event data from another site (it could be the coordinator or any other applicable site), it will execute its global and local applicable rules. A new thread is created to determine the applicability of local rules and process these rules. Since the data produced by processing the local rules will not be included in the event data sent to the other applicable sites or back to the coordinator, we will skip the local rules’ execution in the following explanation. Again, since the Host has checked the global rules’ applicability and constructed the processing graph for them, it is not necessary to check the applicability of global rules and enforce the meta-rules again after a site
receives the event data. The receiver site can process its global rules according to the received processing graph directly.

After a site receives the event data and the processing graph from a site (it could be the coordinator or any other site, or even the same site), it checks whether its applicable rules can be executed right away. An applicable rule can be executed only if the site has received all the event data produced by the rule’s predecessor rules shown in the processing graph, or if the rule has no predecessor rules. Otherwise, it stores the received event data in a static hash table, which will later be merged with other received event data that are needed for processing this applicable rule at this site. For each applicable rule, when it is processed, the site checks whether the rule has any successor rule(s) in the processing graph that needs to be executed. If so, it sends the updated event data and the processing graph to the site(s) that define the successor rule(s). If there is no successor rule, then the site will send the updated event data, along with the executed rules, back to the coordinator.

The coordinator then merges all the returned event data to create a new version of event data (the coordinator knows it has received all the event data by checking whether the executed rules cover all the applicable rules in this round). If the new version contains any update or addition to the original event data, or the “DeferredRules” data field in the original event data is not empty, it sends the new version of event data to the Host to start a new round of rule processing and process enactment. If the version of the event data at the beginning of the current round has not been changed by the current round of rule processing and process enactment and there are no deferred rules, the process triggered by the event instance terminates. This is
because all the rules that are applicable to the current version of event data have been processed and the processed rules and the processes enacted by these rules have not added any new information that warrants another round of rule processing.

The syntactic and semantic matching processes introduced in this work identify those distributed rules that are applicable to the event data upon an event occurrence. The processing of these rules and static processes enacted by them forms an inter-organizational process, which can be further refined by meta-rules. Since meta-rules can be added, modified and removed at runtime, the refined inter-organizational process can be more accurate in capturing and carrying out the desired collaborative operations of organizations, and more flexible and suitable for managing the business of these organizations in a dynamic environment. In Chapter 6 and Chapter 7, we will use two example scenarios taken from e-government domain and e-business domain to demonstrate the applications of our specification language, processing technique and system.
// determine the applicable rules and construct a processing graph for these applicable rules

Algorithm determineApplicableRulesAndProcessingGraph (event-data)

(1) deferred-rules = getDeferredRules(event-data);
(2) executed-rules = getExecutedRules(event-data);

//determine the potential applicable rules by syntactic matching process
(3) applicable-rules = syntactic_matching(event-data);

(4) applicable-rules = applicable-rules + deferred-rules;

//filter out some rules from the applicable-rules set by semantic matching process
(5) applicable-rules = semantic_matching(event-data, applicable-rules);

(6) Initialize each applicable rule’s associated-meta-rules set as empty;
(7) Initialize G.v = {rules in the applicable-rules set}, G.e = {};

(8) sort all meta-rules according to their precedence in descending order;
(9) for each meta-rule R in the sorted order do
(10)     if R is a “Suppress(A1,…,An)” do
(11)        processingSuppressMetarule(R);
(12)     end if
(13)     else if R is a conditional suppression “If (Condition) then Suppress(A1,…,An)” do
(14)        processingConditionalSuppressionMetarule(R);
(15)     end else if
(16)     else if R is a “Onetime(A1,…,An)” do
(17)        processingOnetimeMetarule(R);
(18)     end else if
(19)     else if R is “Prerequisite(and(A1,…,An), B)” do
(20)        processingAndPrerequisiteMetarule(R);
(21)     end else if
(22)     else if R is “Prerequisite(or(A1,…,An), B)” do
(23)        processingOrPrerequisiteMetarule(R);
(24)     end else if
(25)     else if R is “Order(A1,…,An)” do
(26)        processingOrderMetarule(R);
(27)     end else if
(28) end for

(29) applicable-sites <- those sites that define applicable rules with no incoming edge in the graph;
(30) convert the graph G to an xml document;
(31) return applicable-sites, applicable-rules and the xml document;

Figure 5-1. Algorithm to determine the applicable rules and construct a processing graph.

Function ProcessingSuppressMetarule(SuppressMetarule suppress(A1,…,An))
(1) for each rule Ai (1<=i<=n) do
(2)     if Ai’s associated-meta-rules set is empty do
(3)        remove Ai from the applicable-rules set
(4)     end if
(5) end for
End Function

Figure 5-2. Algorithm for processing “Suppress()” meta-rule.
Function `processingConditionalSuppressionMetarule`("If (Condition) then Suppress(B1,...,Bm)"

1. \( (A_1,...,A_n) = \text{evaluateCondition}(\text{"Condition"}) \); //Ai is the rule used in the Condition that is selected to be true;
2. if \((A_1,...,A_n)\) is not empty do // the "Condition" is evaluated true
3. \( \text{is\_meta\_rule\_applied} <- \text{false}; \)
4. for each rule Bi \((1<=i<=m)\) do
5. \( \text{if Bi is in the applicable-rules set do} \)
6. \( \text{if Bi's associated-meta-rules set is empty do} \)
7. \( \text{remove Bi from the applicable-rules set;} \)
8. \( \text{is\_meta\_rule\_applied} <- \text{true}; \)
9. end if
10. end if
11. end for
12. if \((\text{is\_meta\_rule\_applied} == \text{true})\) do
13. for each Ai in \((A_1,...,A_n)\) do
14. \( \text{if Ai is in the applicable-rules set do} \)
15. \( \text{add this meta-rule to Ai's associated-meta-rules set;} \)
16. end if
17. end for
18. end if
19. end if
End Function

Figure 5-3. Algorithm for processing conditional suppression meta-rule.

Function `ProcessingOnetimeMetarule`(Metarule Onetime(A1,...,An) )

1. for each rule Ai \((1<=i<=n)\) do
2. \( \text{if Ai \((1<=i<=n)\) is included in the executed-rules set do} \)
3. \( \text{if Ai's associated-meta-rules set is empty do} \)
4. \( \text{remove Ai from the applicable-rules set;} \)
5. end if
6. end if
7. end for
End Function

Figure 5-4. Algorithm for processing "Onetime()" meta-rule.
Function processingAndPrerequisiteMetarule("Prerequisite(and(A1,...,An), B)")
(1)   if rule B is in the applicable-rules set do
(2)      if all Ai (1<i<n) are either in the executed-rules set or in applicable-rules set do
(3)         meta-rule-applicable <- true;
(4)      for each Ai that is in the applicable-rules set do
(5)         add a directed edge <Ai, B> in G;
(6)      if it introduces a cycle in G, do
(7)         meta-rule-applicable<- false;
(8)      log();
(9)     break;
(10)   end if
(11)  end for
(12)  if meta-rule-applicable == false do
(13)    remove all new added edges;
(14)    ignore this meta-rule in the current round;
(15)   end if
(16)   else
(17)    add this meta-rule to the associated-meta-rules sets that correspond to all applicable rules Ai and B;
(18)   end if
(19)  else if if B's associated-meta-rules set is not empty do
(20)    ignore this meta-rule in the current round;
(21)   end if
(22)  else do
(23)    remove rule B from the applicable-rules set and G, and add B to the deferred-rules set;
(24)  end else
(25)  end else
(26) end if
End Function

Figure 5-5. Algorithm for processing “Prerequisite(and(), )” meta-rule.

Function processingOrPrerequisiteMetarule("Prerequisite(Or(A1,...,An), B)")
(1)   if rule B is in the applicable-rules set do
(2)      if there is a rule Ai (1<i<n) is in the executed-rules set do
(3)         add this meta-rule to rule B's associated-meta-rules set; // B is processed normally
(4)      end if
(5)    else if there is rule Ai (1<i<n) in the applicable-rules set do
(6)       add a directed edge <Ai, B> in G;
(7)      if this edge introduces a cycle in G do
(8)         remove the edge<Ai, B>, and ignore this meta-rule in the current round;
(9)      log();
(10)     else
(11)       add this meta-rule to the associated-meta-rules sets that correspond to rule Ai and B;
(12)    end else if
(13)  else if if B's associated-meta-rules set is not empty do
(14)    ignore this meta-rule in the current round;
(15)   end if
(16)  else do
(17)    remove rule B from the applicable-rules set and graph, and add B to the deferred-rules set;
(18)  end else
(19) end else
(20) end if
End Function

Figure 5-6. Algorithm for processing “Prerequisite(or(), )” meta-rule.
Function processingOrderMetarule("Order(A1, ..., An)")

1. let A'1, ..., A'm be a list of applicable rules that are in (A1, ..., An). Each pair of rules has the same order as they are in the list A1, ..., An.

2. is_meta-rule_applied <- true

3. for each pair of A'i, A'i+1 (1 <= i < m), do

4. if there is no path from Ai to Aj do

5. add an directed edge <Ai, Aj> in G;

6. if introducing a cycle after adding the new edge do

7. is_meta-rule_applied <- false;

8. log();

9. break;

10. end if

11. end if

12. end for

13. if is_meta-rule_applied = false do

14. remove all edges that have been added for this meta-rules;

15. skip this meta-rule in this round;

16. end if

17. end for

End Function

Figure 5-7. Algorithm for processing “Order()” meta-rule.
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). NPDN includes five regions: the Great Plains Diagnostic Network (GPDN), the Northeastern Plant Diagnostic Network (NEPDN, the North Central Plant Diagnostic Network (NCPDN), the Southern Plant Diagnostic Network (SPDN), and the Western Plant Diagnostic Network (WPDN). The University of Florida was selected as the regional center of SPDN. The specific purpose of NPDN is to provide a nationwide network of public agricultural institutions with a cohesive, distributed system to quickly detect high consequence pests and pathogens that have been introduced into agricultural and natural ecosystems, identify them, and immediately report them to appropriate responders and decision makers (The United States Department of Agriculture, Cooperative State Research and Education and Extension Service, 2002). NPDN defines a document of Custody and Chain of Communication Standard Operating Procedure (SOP) (National Plant Diagnostic Network, 2008) to specify the constraints, policies, regulations and procedures on 1) how a disease sample should be transferred, and diagnosed in different levels of diagnostic labs after its collection; 2) how related labs/offices can communicate with one another about the sample status and diagnosis results effectively. We have identified the need for event notification, automatic event data delivery, and distributed processing of knowledge rules and processes in order to achieve close communication and coordination among these organizations. We have applied the results of our R&D efforts in the definition and enactment of the SOP. Our
technology will enable the organizations (labs) in NPDN to effectively share their event data, knowledge rules, operations and processes, provide a timely exchange of information about any disease or pest outbreak, and receive assistance on appropriate emergency response, thus establishing better communication, coordination and collaboration among them. To demonstrate and evaluate the integrated rule and process specification language, the distributed processing technique, and the system architecture, we have implemented the SOP using our language and have run through several test scenarios with NPDN members from University of California at Davis and Kansas State University. The participants agree that, overall, the ETKnet2.0 system provides an efficient way to keep people engaged in the SOP by facilitating and guiding the communication channels. It helps each participant know who should do what and when. It provides a good framework for communication and helps participants understand how the SOP works. It can be a good training tool as well as a useful tool for exercises and/or the management of actual events (Su et al., 2011).

6.1 Events, Rules, Operations, Processes, and Meta-rules Used to Implement NPDN's SOP

There are several categories of organizations/labs/offices involved in the operation of the SOP, including the Triage Lab, NPDN Regional Hub Lab, Expert Lab, State Plant Regulatory Official (SPRO), State Plant Health Director (SPHD), National Identification Service (NIS), Confirming Diagnosis Designate (CDD), Emergency and Domestic Programs (EDP) and Regional Program Manager (RPM). Their names and functions are listed in Appendix B.

Each state has its growers, pest advisors or other sample submitting entities to collect and submit samples to a triage lab designated to receive and examine suspect
samples. After the triage lab receives the sample, the following procedure is followed according to the NPDN SOP (National Plant Diagnostic Network, 2008). The triage lab's diagnostician would examine the sample. If the sample is determined to be of suspect regulatory significance, then the triage lab would send this presumptive positive sample to the NPDN Regional Hub Lab or CDD according to the CDD's requirement, and notify the related staffs/labs of the presumptive positive sample being in the system. Once the NPDN Regional Hub Lab receives a presumptive positive sample, the diagnostician of the lab would examine the sample with or without the collaboration of a Local Expert, and then send the sample to the CDD according to the CDD’s requirement. The CDD conducts a confirming diagnosis of the received presumptive positive sample. After a confirmed result is released, all other interested organizations are informed of it by designated organizations in the following order: CDD first informs the NIS of a confirmed diagnostic result. Then NIS emails the result to the EDP staff. The EDP staff forwards the result to the appropriate APHIS-PPQ headquarters and RPMs, the state of origin SPHD and the SPRO. After the RPM gets the confirmed result, he/she contacts the SPHD and SPRO to determine who will contact the triage lab with the result. The SPHD or SPRO contacts the triage lab with the result accordingly. The triage lab should also acknowledge to related labs the receipt of the result and contact the Campus Safety Officer and the regional Hub Lab if it is engaged. At the same time, after the regional director gets the result, he/she contacts the regional directors in other regions with the confirmed result if engaged.

Each organization/lab/office is a network site in ETKnet2.0. It defines its own events, operations, rules, processes and triggers to be processed at its network site.
Figures 6.1 – 6.9 show the rules, operations and processes defined by various organizations/labs/offices needed to implement the SOP. The defined operations, rules and processes are translated into program code and packaged as Web services using the methods explained in Chapter 3. The program code of these Web services resides at the network sites where operations, rules and processes are defined. The meta-data of shared operations, rules, and processes are registered with the private registry installed at the Host site.

In order to implement the NPDN SOP using our language, IRPSL, we also define a few data entities. They include “npdn.Sample”, “npdn.Result”, “npdn.Packing”, “npdn.ResponsePlan” and “npdn.General”. The entity npdn.Sample (simply called Sample in the remainder of this chapter) is used to describe a sample in the system. The attribute combination (localLabSampleID, labID, sampleDate) is the unique identifier of a sample instance. The entity npdn.Result (Result) is used to describe the diagnosis result of a unique sample. The entity NPDN.PackingRequirement (PackingRequirement) is the packing requirement for a sample that will be shipped to another lab. The SOP has different packing requirements for plant samples and insect samples. The entity npdn.Shipment (Shipment) denotes the shipping result of a sample that is sent to another lab to do further diagnostics. The entity npdn.ResponsePlan (ResponsePlan) is used to describe the response information that SPRO and SPHD should have if the presumptive positive sample in the system is confirmed to be positive. The entity npdn.General includes all the program variables that are used to save the values generated by operations or rules during their execution. The detailed specifications of these entities are given in Appendix B.
In the following sections, we explain the operations, rules and processes defined for implementing the SOP. We will not explain the details of operations here due to space limitations. Basically, an automated operation is the Java code that performs a specific task such as querying the database, assigning values to data items, or invoking a Web Service. A manual operation is implemented by sending an email notification to persons who play a specific role(s) that is suited to perform the task. After the completion of a manual operation, the person who committed to perform the task should log in to ETKnet2.0 to indicate the task's completion and provide the data produced in the task.

6.1.1 Operations, Rules and Processes

**NPDN Triage Lab.** Once the Triage Lab receives a new suspected sample, an automated operation named “Retrieve_Lab_Information()” will retrieve the lab’s configuration information, including the lab id, and whether the lab has the distant diagnosis capability. The sample is then examined by the manual operation “Examine_Sample()”, and the sample’s classification can be preliminarily determined at this time. If the sample is determined to be “negative”, then no further processing needs to be done. The procedure terminates after storing the sample by the manual operation “Store_Sample()” and informing the user by automated operation Inform_Termination(). Otherwise, the “Store_Sample_In_Secure_Place()” operation asks the staff to store the sample in a secure location. After the sample has been examined, its status will be changed accordingly as specified in derivation rule1. The above procedure is defined by the rule “TriageLabCARule1”. The CA rule “TriageLabCARule2” guides the staff to inform other related organizations of the fact that a presumptive positive sample is in the system. The above two rules are shown in
Figure 6-1. If it is appropriate to conduct the distant diagnosis to the sample, then the CA rule “TriageLabCARule3” and “TriageLabCARule4” are activated to guide the staff to carry out different operations according to whether the lab has the distant diagnostic capability. The packing and shipping requirement and procedure are specified in the rule “TriageLabCARule6”. Finally, the manual operation “Inform_Campus_Safety_Office” activated by “TriageLabCARule7” notifies the triage lab staff to contact the Campus Safety Officer to inform him/her of the presumptive positive sample in the system. Rule “TriageLabDR6” and “TriageLabDR7” change the sample’s status accordingly after the Triage Lab ships the sample to the next lab. The details of these rules are shown in Figure 6-2.

Once the triage lab receives the message on the confirmed diagnostic result of the suspected sample, it first acknowledges to the related organizations that it has received the result, and also notifies the Regional Hub Lab, if it is involved and has not been informed, of this confirmed result by the rule “TriageLabCARule8”. If a portion of sample was retained in the lab, then rule “TriageLabCARule9” guides the triage lab staff to deal with the retained sample. The rule “TriageLabCARule10” asks the Triage Lab to submit a record to NPDN databases at state, regional and national level. The above rules are shown in Figure 6-3.

**SPRO.** Once the SPRO is informed that a suspected sample is in the system, he/she contacts the SPHD to discuss plans and prepare for a response. This is expressed by the rule “SPROCARule1” shown in Figure 6-4.

Once the SPRO gets the confirmed diagnostic result, he/she conducts the manual operations “SPRO_Inform_SDA_Of_Result()”,

93
“SPRO_Discuss_GrowerSubmitter_Notification_Process_With_SPHD()”, and “SPRO_Inform_TriageLab_Of_Result()” to inform other organizations of the confirmed result, and discuss the sample submitter’s notification process with the SPHD. A new data item “triagelab_gets_result” is created with true as its value. The whole procedure is described in the rule “SPROCARule2” and “SPROCARule3” shown in Figure 6-4.

SPHD. Once the SPHD is informed of a suspected sample in the system, he/she performs the manual operation “SPHD_Contact_ARD_Of_Presumptive_Sample()” to inform related organizations of the presumptive positive sample in the system. This action is described in the rule “SPHDCARule1” shown in Figure 6-5. Once SPHD gets the result, if he/she is the person who has been determined to be the one to contact the triage lab about the result, then he/she performs the operation “SPHD_Inform_TriageLab_Of_Result()”, and a new data item “triagelab_gets_result” is created with true as its value. The above procedure is described in the rule “SPHDCARule2” shown in Figure 6-5.

NPDN Regional Hub Lab. Once the Regional Hub Lab receives a suspected sample from Triage Lab, the procedure that needs to be carried out is specified by the CA rules “HubLabCARule1”, “HubLabCARule3” and “HubLabCARule4” shown in Figure 6-6. It first performs a diagnosis of the sample (“HubLab_Examine_Sample()”) with the assistance of a local expert (“Get_LocalExpert_Opinion()”). It then informs the related organizations of the preliminary result to determine if it is necessary to conduct a further diagnosis. This is done by a manual operation “Contact_Director_CDD_with_preConclusion()”. “HubLabCARule3” describes the procedure that needs to be carried out, if it is necessary to conduct a further diagnosis.
The first four derivation rules are used to determine the packing and shipping requirements, which are needed in the later operations “Packing_Sample()” and “Shipping_Sample()”. “HubLabCARule4” describes the procedure that needs to be carried out, if it is not necessary to conduct a further diagnosis. It sets the status of the sample to "confirmed" and the triage lab is informed by a manual operation.

Once the Regional Hub Lab receives the confirmed result, the process that needs to be carried out is specified in the CA rule “HubLabCARule2” (Figure 6-6): The Hub Lab first informs its own state’s SPRO and SPHD of the result, processes the retained sample, submits Form4, which is a report on the identification of a select agent or toxin in a clinical or diagnostic laboratory, if necessary, and then informs its own Campus Safety Officer of the result.

**NPDPN Regional Director.** Once the NPDPN Regional Director is informed of a presumptive positive sample in the system, as described in the rules “RegionalDirectorCARule1”, “RegionalDirectorCARule2”, “RegionalDirectorCARule5” and “RegionalDirectorCARule3” shown in Figure 6-7, he/she forwards the message to the NPDPN Program Leader, and gets instructions on whether to inform other regional directors and other labs. If the confirmation will take more than 7 days or it is deemed appropriate to contact other regional directors, the NPDPN Regional Director would inform other regional directors of the presumptive positive sample in the system.

Once the NPDPN Regional Director knows the confirmed result, he/she first informs the Program Leader of the result, and, if deemed appropriate, also contacts other regional directors with the confirmed result. This procedure is specified in the rule “RegionalDirectorCARule4” shown in Figure 6-7.
CDD. “CDDCARule1” shown in Figure 6-8 describes the process where, once the CDD receives the suspected sample, it acknowledges the receipt of the sample to the sender, and informs the sender of the approximate expected date and time to receive the confirmed result. “CDDCARule2” describes the procedure where the CDD conducts the diagnostic procedure on the sample and makes the confirming diagnosis. At that time, the sample status is changed to “confirmed”, and an instance of the entity Result.* is created by an assignment operation.

Once CDD gets the confirmed result, it informs the NIS of the result, and a new data item NIS_gets_result with true as its value is created by the rule “CDDCARule3”.

NIS. Once the NIS gets the confirmed result, it emails the EDP with the result, and a new data item EDP_gets_result, with true as its value, is created by the rule “EDPCARule1” shown in Figure 6-9.

EDP. Once the EDP gets the confirmed result, it forwards the confirmation to the appropriate labs according to whether the diagnosis result is new to the US, and whether it is significant or not. New data items RPM_gets_result, Regional_Director_in_StateOfOrigin_gets_result, SPHD_gets_result, and SPRO_gets_result are created. All these new data items will have the value true. The above process is described in the rule “EDPCARule1” shown in Figure 6-9.

RPM. Once the RPM gets the confirmed result, it first contacts the regional director to discuss diagnostic needs and capabilities, and then contacts SPHD and SPRO to determine who will contact the triage lab about the result. A new data item triagelab_notifier will be produced by the manual operation.
“RPM_Co ntact_SPRO_SPHD()”. The above process is specified in the rule “RPM_CARule1” shown in Figure 6-9.

6.1.2 Event and Meta-rules

A global event named “TriageLabE1” (Triage Lab receiving a new suspected sample from a sample submitter) is defined at the Triage lab site. The event data included in this event is an instance of the entity Sample. The event is posted when the Triage Lab receives a new suspected sample from a sample submitter. Meta-rules defined for this event are shown in Table 6-1.

The enforcement of NPDN’s SOP requires the close communication, coordination and collaboration of many organizations/labs/offices at the local, state, regional, and national levels. In this section, we have shown that the SOP can be implemented in terms of sharable distributed data, events, rules, processes, manual and automated operations, triggers and meta-rules specified by these collaborating organizations/labs/offices. In the next section, we shall explain how these distributed data, rules, processes and operations can interoperate upon the occurrences of events and how inter-organizational processes are dynamically formed and processed.

6.2 A Scenario on Dynamic Construction and Processing of Inter-organizational Collaboration

We use a scenario that exercises a part of the SOP as an example to explain our meta-rule enhanced, event-triggered, distributed rule processing and process enactment technique. The scenario and its steps are shown in Figure 6-10. The rules and meta-rules that correspond to these steps are shown in Figures 6-1 - 6-9 and Table 6-1. The scenario begins with the Triage Lab receiving a new suspected plant sample from a sample submitter. Once the Triage Lab receives a new suspected sample, the
staff posts an event “TriageLabE1” (Triage Lab receiving a new suspected sample from a sample submitter). The occurrence of the event causes the event data (denoted by ED1 in Figure 6-10) that contains an instance of entity Sample, which is sent to the Host in an XML document. The Host determines that only rule “TriageLabCARule1” is applicable currently, and rule “TriageLabCARule7” is deferred by the meta-rule “Prerequisite(TriageLabCARule6, TriageLabCARule7)”. It sends this information back to the NPDN Triage Lab, which serves as the coordinator of this event instance. The coordinator receives applicable sites and applicable rules information, and sends the event data to the applicable site (in this case the applicable site is the coordinator’s site). After the Triage Lab processes the rule “TriageLabCARule1”, new event data items will be added to the event data and some old event data items’ values will also be updated. Let us assume that the sample is diagnosed as “presumptive positive”, which means that the sample needs further diagnosis by other labs such as NPDN Regional Hub Lab and/or CDD, and the value of the attribute “sample.classification” is changed to “presumptive positive”.

Since there have been new data added/updated, the coordinator sends the new version of the event data ED2 to the Host to check whether there are other rules that need to be processed. After the syntactic and semantic matching process and checking of the meta-rules, the Host determines that only rule “TriageLabCARule2” is applicable, and rule “TriageLabCARule6” and “TriageLabCARule7” are deferred by the meta-rule “Prerequisite(Or(TriageLabCARule3, TriageLabCARule4, TriageLabCARule5), TriageLabCARule6)” and “Prerequisite(TriageLabCARule6, TriageLabCARule7)”. The Triage Lab processes “TriageLabCARule2” by informing the related labs with the
presumptive positive sample information and reporting whether it is appropriate to conduct a distant diagnosis now. In this scenario, let us assume that it is appropriate to conduct a distant diagnosis and that this lab also has distant diagnosis capability. That is, the data item “is_distant_diagnosis_appropriate” and “has_distant_diagnosis_capability” have the value “true”.

Similarly, the coordinator sends the new version of the event data, i.e. ED3 in Figure 6-10, to the Host site to start the third round of event notification and rule processing. In the third round, “TriageLabCARule3”, “TriageLabCARule6”, “TriageLabCARule7”, “SPROCARule1”, “SPHDCARule1” and “RegionalDirectorCARule1” are determined to be applicable after the syntactic and semantic matching processes at the Host site. Furthermore, “TriageLabCARule3”, “TriageLabCARule6” and “TriageLabCARule7” should be processed sequentially as required by the meta-rules “Prerequisite(or(TriageLabCARule3, TriageLabCARule4, TriageLabCARule5), TriageLabCARule6)”, and “Prerequisite(TriageLabCARule6, TriageLabCARule7)”. In this round, the applicable sites include “Triage Lab”, “SPRO” and “NPDN Regional Director”. After “SPRO”, “SPHD” and “NPDN Regional Director” receive the event data from the coordinator, they process their rules “SPROCARule1”, “SPHDCARule1” and “RegionalDirectorCARule1” respectively, and send the new event data back to the coordinator. In the mean time, the Triage Lab processes its applicable global rules “TriageLabCARule3”, “TriageLabCARule6” and “TriageLabCARule7” sequentially, and sends the updated event data back to the coordinator. Let us assume that Hub Lab is the next lab that will conduct a diagnosis on the sample. That is, the
data item “next_sampler_receiver” produced by Triage Lab site has the value “HubExpertLab”.

In the fourth round, the CA rule “RegionalDirectorCARule2” and the derivation rule “TriageLabDR6” are applicable. The rule “TriageLabDR6” is executed at the Triage Lab site, and it updates the value of the attribute sample.status as “hublab”. “RegionalDirectorCARule2” is executed at the Regional Director site without adding/updating any event data items.

Since there are new added/updated event data, the coordinator sends the new version of the event data, i.e. ED5, to the Host to start the fifth round of event notification and rule processing. In this round, the Host determines that “HubLabCARule1” is applicable after performing syntactic and semantic matching processes and applying meta-rules. The coordinator sends ED5 to the “Hub Lab” site to process the rules “HublabCARule1”. The Hub Lab processes the “HubLabCARule1” by further examining the sample with a local expert, and determining whether the sample needs to be diagnosis by the CDD Lab. The rule produces several event data items, including an instance of the entity “Shipment” and the data item “is_further_diagnosis_necessary”. Let us assume that the Hub Lab has come to the confirmed diagnostic result to the sample, which means that the data item “is_further_diagnosis_necessary” has the value “false”. The Hub Lab site sends the new added/updated event data back to the coordinator.

The coordinator sends the latest version of the event data, i.e. ED6, to the Host again since there are new added event data items. In this round, the Host determines that only “HubLabCARule4” is applicable. The coordinator sends the event data ED6 to
the applicable site “Hub Lab” to process the rule “HubLabCARule4”, which changes the value of the sample’s attribute “status” to “confirmed”, creates an instance of the entity “Result”, and assigns the value “true” to the data item “hublab_gets_result” and “trialeglab_gets_result”.

The seventh round of processing will take place because there are new data created in the previous round. In this round, the Host constructs a processing graph as shown in Figure 6-11 for applicable rules “HubLabCARule2”, “TriageLabCARule8”, “TriageLabCARule9” and “TriageLabCARule10” based on the defined meta-rules. The applicable site is the “Triage Lab” site. The “Triage Lab” site processes the rule “TriageLabCARule8” and “TriageLabCARule9” in parallel. After “TriageLabCARule8” is finished, the “Triage Lab” sends the event data to the “Hub Lab” to process the rule “HubLabCARule2”. The rule “TriageLabCARule10” is processed after the rule “TriageLabCARule8” and “TriageLabCARule9” are both completed at the “Triage Lab” site.

After the coordinator receives the returned event data from the Hub Lab site and the Triage Lab site, it sends the new version of event data to the Host again. The Host determines that there are no more applicable rules. The rule processing triggered by the occurrence of the event “TriageLabE1” terminates at this point. Figure 6-10 shows the event data, the executed rules, and the interaction between involving organizations/labs in this scenario.

The above scenario and its processing show that, in ETKnet2.0, an event occurs at one site would trigger the processing of distributed rules defined by collaborating organizations/labs/offices. A CA rule can conditionally or unconditionally enact an
organizational process, which can activate the rules and manual and/or automated operations that define it. The processing of rules and operations may produce data or alter the event data, which in turn trigger more applicable rules. Meta-rules are interpreted at runtime to process the applicable rules in a desirable order/structure.

Through the multiple rounds of event data transmission, rule processing, and process enactment, the sharable event data, operations, rules and processes defined by different organizations/labs/offices can interoperate to produce the data needed for inter-organizational collaboration and coordination. Thus, an inter-organizational process is dynamically constructed and enacted at runtime rather than pre-defined at build-time by a process designer. This rule processing and process enactment technique would allow collaborating organizations to change their rules, processes, operations and meta-rules at any time as their operating conditions or policies require, and the changes would be reflected instantaneously in the subsequently formed inter-organizational processes. In Chapter 7, we will provide another example taken from the business domain to explain and demonstrate the utility of our integrated rule and process specification language and distributed rule processing and process enactment technique.

<table>
<thead>
<tr>
<th>Organization</th>
<th>Meta-rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPRO</td>
<td>Prerequisite(TriageLabCARule2, SPROCARule1); Prerequisite(EDPCARule1, SPROCARule2); Prerequisite(EDPCARule1, SPROCARule3); Onetime(SPROCARule1, SPROCARule2, SPROCARule3);</td>
</tr>
<tr>
<td>SPHD</td>
<td>Prerequisite(TriageLabCARule2, SPHDCARule1); Prerequisite(EDPCARule1, SPHDCARule2); Onetime(SPHDCARule1, SPHDCARule2);</td>
</tr>
<tr>
<td>Organization</td>
<td>Meta-rules</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NPDN Triage Lab</td>
<td>Prerequisite(TriageLabCARule1, TriageLabCARule2);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(TriageLabCARule2, TriageLabCARule3);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(TriageLabCARule2, TriageLabCARule4);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(TriageLabCARule2, TriageLabCARule5);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(TriageLabCARule6, TriageLabCARule7);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(Or(SPROCARule3, SPHDCARule2, HubLabCARule4), TriageLabCARule8);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(And(TriageLabCARule8, TriageLabCARule9), TriageLabCARule10);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(Or(TriageLabCARule3, TriageLabCARule4, TriageLabCARule5), TriageLabCARule6);</td>
</tr>
<tr>
<td>NPDN Regional Hub Lab</td>
<td>Prerequisite(TriageLabCARule6, HubLabCARule1);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(HubLabCARule1, HubLabCARule3);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(HubLabCARule1, HubLabCARule4);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(TriageLabCARule8, HubLabCARule2);</td>
</tr>
<tr>
<td></td>
<td>Onetime(HubLabCARule1, HubLabCARule3, HubLabCARule4, HubLabCARule2);</td>
</tr>
<tr>
<td>NPDN Regional Director</td>
<td>Prerequisite(TriageLabCARule2, RegionalDirectorCARule1);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(Or(RegionalDirectorCARule2, RegionalDirectorCARule5), RegionalDirectorCARule3);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(EDPCARule1, RegionalDirectorCARule4);</td>
</tr>
<tr>
<td></td>
<td>If (RegionalDirectorCARule2) then Suppress (RegionalDirectorCARule5);</td>
</tr>
<tr>
<td></td>
<td>If (RegionalDirectorCARule5) then Suppress (RegionalDirectorCARule2);</td>
</tr>
<tr>
<td></td>
<td>Onetime(RegionalDirectorCARule1, RegionalDirectorCARule2, RegionalDirectorCARule3, RegionalDirectorCARule4);</td>
</tr>
<tr>
<td>CDD</td>
<td>Prerequisite(CDDCARule1, CDDCARule2);</td>
</tr>
<tr>
<td></td>
<td>Prerequisite(CDDCARule2, CDDCARule3)</td>
</tr>
<tr>
<td></td>
<td>Onetime(CDDCARule1, CDDCARule2, CDDCARule3);</td>
</tr>
<tr>
<td>bNIS</td>
<td>Prerequisite(CDDCARule3, NISCARule1);</td>
</tr>
<tr>
<td></td>
<td>Onetime(NISCARule1);</td>
</tr>
<tr>
<td>EDP</td>
<td>Prerequisite(NISCARule1, EDPCARule1);</td>
</tr>
<tr>
<td></td>
<td>Onetime(EDPCARule1);</td>
</tr>
<tr>
<td>RPM</td>
<td>Prerequisite(EDPCARule1, RPMCARule1);</td>
</tr>
<tr>
<td></td>
<td>Onetime(RPMCARule1);</td>
</tr>
</tbody>
</table>
**Rule: “TriageLabCARule1”** (TriageLab receives and examines the new sample, and communicate with the related offices with information)
Condition: Sample.localLabSampleId != null && Sample.classification == "unclassified"
Action: (Process "TriageLabProcess1")

```
Sample.labid, has_distance_diagnosis_capability = Retrieve_Lab_Information()
Sample.status = "triagelab"
Sample.classification = Examine_Sample(Sample.labid,
Sample.localLabId, Sample.sampleDate)
```

Derivation Rule1: if Sample.classification == "negative" then Sample.status = "confirmed"
If Sample.classification == "negative"

```
Inform_Termination(Sample.)
```

**Rule: “TriageLabCARule2”** (Inform related offices of having presumptive positive sample in the system)
Condition: Sample.classification != negative && Sample.status == "triagelab"
Action: (Process "TriageLabProcess2")

```
TriageLab_Inform_Related_Offices_Of_Receiving_Presumptive_Positive_Sample(Sample.sampleDate,
Sample.localLabSampleId, Sample.labId)
```

**Rule: “TriageLabCARule3”** (Conduct distant diagnosis (1)) has distance diagnosis capability
Condition: is_distant_diagnosis_appropriate == true && has_distance_diagnosis_capability == true
Action: TriageLab_Conduct_Web_Based_Diagnosis(Sample.sampleDate, Sample.localLabId, Sample.labId)

**Rule: “TriageLabCARule4”** (Conduct distant diagnosis (2)) has no distance diagnosis capability
Condition: is_distant_diagnosis_appropriate == true && has_distance_diagnosis_capability == false
Action: TriageLab_Email_Picture(Sample.sampleDate, Sample.localLabId, Sample.labId)

**Rule: “TriageLabCARule5”** (It is not appropriate to conduct distance diagnosis).
Condition: is_distant_diagnosis_appropriate == false
Action: Dummy_Activity()

**Rule: “TriageLabCARule6”** (Process further diagnosis request)
Condition: Sample.classification == "presumptive positive" && Sample.status == "triagelab"
Action: (Process "TriageLabProcess6")

```
next_sample_reciever = get_instruction_from_CDD_for_further_diagnosis(Sample.)
```

Derivation Rule2: if sample.type == insect, then
PackingRequirement.vial = true;
PackingRequirement.in_appropriate_crush_proof = true
PackingRequirement.mark = "urgent ID requested"

```
dummy_activity = Dummy_Activity()
```

Derivation Rule3: if Sample.type == "plant" then
PackingSample.doublebag = true;
PackingSample.Distant_diagnosis_level = "negative"
PackingSample.mark = "plant sample for diagnosis"

```
Inform_Termination(Sample.)
```

**Rule: “TriageLabDR5”**
Condition: Sample.classification != negative && Sample.status == "triagelab"
Action: (Process "TriageLabProcess5")

```
next_sample_reciever = "HubExpertLab" && Sample.status = "triagelab"
```

**Rule: “TriageLabDR7”**
Condition: next_sample_reciever == "CDD" && Sample.status == "triagelab"

Figure 6-1. NPDN Triage Lab rules on receiving a new sample (1)

Figure 6-2. NPDN Triage Lab rules on receiving a new sample (2)
Figure 6-3. NPDN Triage Lab rules on receiving the confirmed result.

Figure 6-4. SPRO rules on being informed of a presumptive positive sample in the system and the confirmed result.

Figure 6-5. SPHD rules on being informed of the confirmed result.
Figure 6-6. NPDN Regional Hub Lab rules on receiving a presumptive positive sample and being informed of the confirmed result.
Figure 6-7. NPDN Regional Director rules on being informed of 1) a presumptive positive sample in the system, 2) the estimation that the confirmation may take more than 7 days and 3) the confirmed result.

Figure 6-8. APHIS-PPA CDD rules on the receipt of a presumptive positive sample, and the release of the confirmed result.
Figure 6-9. NIS, EDP and RPM rules on receiving the confirmed result.

Figure 6-10. Part of event data transmission, rule processing and process enactment in the NPDN Domain (the Host site is not shown here).
Figure 6-11. The processing graph generated by the Host in the seventh round
CHAPTER 7
CASE STUDY 2: AN EXAMPLE SCENARIO IN THE E-BUSINESS DOMAIN

In this chapter, we use an e-business scenario to demonstrate the utility of the developed technology in achieving interoperation of sharable business data, operations, rules and processes, and the dynamic construction and processing of inter-organizational processes. In the scenario, we assume that retailers and suppliers (i.e., distributors and/or manufacturers) of a certain category of products form a collaborative federation in order to achieve their business goals. They all register with our system, ETKnet2.0, as members of the federation. As members, they can publish their products and services and subscribe to events to receive event data when events occur. Any member can join or leave a federation, and can define, delete, and modify events, business rules and processes at any time, thus forming a very dynamic, collaborative business environment, and the partnership may change over time. It is a good example for demonstrating that our system can function well in such a dynamic environment. This part of the work is included in the published journal paper (Xiao, DePree and Su, 2011).

In this scenario, the retailer receives an order online from a customer. The order specifies a certain quantity of a product that the customer wants to purchase. After receiving the order, the retailer first check her inventory to see whether she can fulfill the order. If yes, she then processes the order to carry out the usual interaction with the customer to make the sale; otherwise, she will get the product from her partners, i.e. suppliers who have joined the collaborative federation, to fulfill this order. Since retailers and suppliers can join or leave the federation at any time, the available suppliers may change over time. The retailer who receives the order can just post a global event to
query all available suppliers for a certain product. The suppliers can respond to the query with their supply. Then the retailer can make a decision as to which supplier will fulfill the order.

The order received from a customer specifies a certain quantity of a product that the customer wants to purchase and some information about the customer. Receiving an order online is treated as an event named “ReceivingOnlineOrderEvent” defined at the retailer site. The event data contain an entity CustomerOrderRequest(ID, productId, quantity) and an entity CustomerInfo(Name, ContactInfo,…). The retailer defines a trigger to link the event with one of her action-oriented rules named “OnlineOrderProcessingRule”. Therefore, once a customer places an online order, the system will automatically activate the rule “OnlineOrderProcessingRule” at the retailer site to process this online order.

The action part of this rule specifies a process named “OnlineOrderProcessingProcess”. The invocation of this rule would enact the process. The first rule “InventoryCheckingRule” specified in this process checks the retailer’s inventory to get the available quantity of a given product according to her inventory policy. In the following Or-Split structure, if the quantity ordered by the customer can be satisfied by the retailer, the rule “FulfillOnlineOrderByOwnInventoryRule”, which will not be explained in this scenario, is processed to carry out the usual interaction with the customer to make the sale; otherwise the rule “FulfillOnlineOrderWithPartnersRule” shown in Figure 7-1 is processed to fulfill the order with the help of collaborating suppliers.
The first operation “getRetailerInformation()” in the process “FulfillOnlineOrderWithPartnersProcess” prepares the retailer’s information to be sent to suppliers. The second operation “calculateRequestAmount()” figures out how many units of the product should be ordered from suppliers based on the quantity the retailer already has, and creates an instance of the entity “QuoteRequest” which forms a part of the event data to be sent out later. The third operation “Post_Syn_Event()” posts a global event named “QuoteRequestEvent”. The event is posted synchronously. The retailer site waits for responses from collaborating business partners within a specified time interval enforced by a timeout mechanism. The event data contain the following entities: QuoteRequest (ID, ProductId, UnitsRequested), RetailerInfo (Name, ContactInfo,…), SupplierInfo (SupplierName, CreditRating, ContactInfo, …), QuoteResponse(ID, ProductId, UnitsOffered, UnitPrice, PaymentOptions), and Shipping (ShippingOption, Price). Here, the first two entities specify what the retailer wants the suppliers to know, and the last three entities specify what responses the retailer expects from the suppliers. Some suppliers may have subscribed to this event by explicitly defining triggers that link the event to their business rules. They also can specify the condition when their business rules can be processed through the conditional part of the trigger. For example, they can specify that they only want to process the quotation request whose required units are more than or less than a certain amount. These suppliers are most likely the suppliers that do business with the retailer regularly. Other suppliers may not have subscribed to the event but may have rules that can be processed to respond to the event “QuoteRequestEvent”. Both types of suppliers should receive the event notification and their rules and processes should be executed. Since
members of a federation can join and leave the federation freely, some retailers and suppliers may not know one another and may not have explicitly defined triggers to link the events defined by other members to their rules. The ability to send event notifications and event data to *implicit subscribers* (i.e., those that have applicable rules as determined by the system) in addition to *explicit subscribers* is a key feature of our system. Such a feature is important and useful for achieving dynamic construction and enactment of inter-organizational business processes.

Continuing our scenario, all the suppliers' sites that receive the event notification and event data would activate their applicable rules. For example, a rule at one site may activate a structure of operations. The first operation checks her inventory to determine if the units of the product requested by the retailer are available. If not, the retailer is informed of this fact. Otherwise, an inventory policy rule is activated to determine how many units of the product should be offered to the retailer so that her own inventory will not drop below a certain level.

All suppliers that can fully or partially satisfy the retailer's request for quote would then return the following data to the retailer's site: *SupplierInfo* (*SupplierName, CreditRating, ContactInfo, …*), *QuoteResponse* (*ProductId, UnitsOffered, UnitPrice, PaymentOptions*), and *Shipping* (*ShippingOption, Price*).

Upon the receipt of the data from the suppliers, the retailer site would merge these data with the original event data to produce a new version of the event data. This merge operation is automatically done by the ETKnet 2.0 system as a part of the system’s rule processing and process enactment. The retailer has a rule named "*QuoteRequestEventPostprocessingRule*" to process the merged event data, and a
trigger to link the event “QuoteRequestEvent” to the rule. After merging the event data received from the supplier sites, the Rule Server at the retailer site would automatically invoke the rule “QuoteRequestEventPostprocessingRule” to process the merged event data. The invoked rule is an action-oriented rule whose action clause invokes a process. The first rule in this process would examine the data returned from the suppliers and select a subset of suppliers who have responded to the “QuoteRequestEvent” positively. This selection may be based on a number of selection criteria such as the credit rating of a supplier, the past experience of the retailer with regard to the supplier’s timeliness of product delivery, etc. The next rule unconditionally performs a number of operations. The first operation would check if the units of the product offered by the selected suppliers meet the needs of the retailer. This is done by reducing the value of the attribute QuoteRequest.UnitsRequested given in the original QuoteRequest instance based on the units provided by the selected suppliers. The next operation checks if the value of said attribute has been reduced to zero. If it has not been, the supplier-provided data are removed from the current version of the event data to produce another version of the event data, and the new version is sent to new suppliers who may have joined the federation since the last round of event notification and rule processing, and who have either subscribed to the event or defined some applicable rules. It is possible that some suppliers who have responded to this event instance before would receive the new version of event data because their rules are applicable to this new version. If these suppliers do not want to respond to the same quote request more than once, they can either define “Onetime” meta-rules to prevent their rules from being executed more than once for the same event instance, or they can add a
condition to their rules to prevent the processing of those quote requests whose IDs are already in their databases. As in the first round, any suppliers who can provide some units of the product can respond positively to the retailer. This procedure may repeat until the retailer gets all the units she needs, or when no new suppliers can be found, or the time set by the retailer for this request has expired.

The automatic operation getQuoteResult() following the event posting operation can further process the received updated event data to create an instance of the entity “QuoteResponse” and calculate the total units responded by suppliers. If the required units of the product have been found (checked by a switched construct followed by the operation getQuoteResult as shown in Figure 7-1), the following sequence of operations is performed: calculate the total price of the customer’s order, create an instance of the entity “CustomerOrderResponse,” inform the customer of the total price and the shipping charges of the selected suppliers, and charge the customer for the product he/she ordered. The next operation, named CompletePurchaseOrderWithPartners in Figure 7-1, forms a purchase order, which contains a specific supplier’s name, the number of units ordered from the supplier and other pertinent information, and posts a global event named “PurchasingOrderRequestEvent” for each of the selected suppliers. Even though all the suppliers might have subscribed to the event, only those selected suppliers will be notified by using the restricted event notification mechanism in the ETKnet2.0 system. The event data transferred would contain the following added information: PurchasingOrder (ProductId, SupplierName, UnitsOrdered, SelectedShippingMethod, SelectedPaymentMethod, PaymentInfo, OrderDate, …) and RetailerInfo (Name, ContactInfo, …).
The receipt of the event data at the suppliers’ sites would trigger their rules that process the received purchase order. Let us consider a possible rule named “PurchasingOrderProcessingRule” that is executed at Supplier C’s site (Figure 7-2), which enacts a process named “PurchasingOrderProcessingProcess”. The first CA rule updates the inventory of the product being ordered based on the units requested by the retailer. The second rule (a derivation rule) checks if the remaining inventory amount drops below 10% of the quota set by Supplier C. If so, a CA rule following it would alert the supplier and ask her to restock the product. Another rule, “FulfillPurchasingOrderRule”, which can be processed in parallel with the derivation rule described above, is carried out to fulfill the purchase order. This rule has a structure of operations. Two parallel operations are executed first. One would send a message to the person who is in charge of doing the packaging and shipping to carry out the manual operation. The other one is to charge the retailer. The next operation, the successor of an AND-Join construct, would send the information such as the shipping date, shipping company name, tracking number, etc. to the retailer.

After receiving the shipping information from the suppliers, the retailer site would forward all this information to the customer. This ends the example scenario. Figure 7-3 shows the interactions between the retailer and the suppliers to complete an online order.

In this scenario, retailers and suppliers define their business rules and processes independently. By using the event-triggered, distributed rule processing and process enactment technique implemented in ETKnet2.0, the retailer is able to complete a purchase order with the help of partners without having to pre-specify the partnership
with these suppliers. The interactions between the retailer and suppliers form an inter-organizational process that is dynamically constructed. The scenario demonstrates the key features of our system and its potential application in the e-business domain.

Figure 7-1. The rule “FulfillOnlineOrderWithPartnersRule” defined by the retailer for fulfilling an online order with the help of partners.
CARule: “PurchasingOrderProcessingRule”
Condition: true
Action: (process: “PurchasingOrderProcessingProcess”)

CA Rule: UpdateInventoryAfterOrderRule
Condition: None
Action: 
Derivation Rule: If the product’s inventory amount drops below 10% of its quota, then it is in short status.
Rule Name: CheckProductInShortStatusRule
If getRemainingAmount(Product.ID) < 0.1*getProductQuota(Product.ID)
Then Product.in_short_status = true

CA Rule: FulfillPurchasingOrderRule
Condition: None
Action:
chargeRetailer (PurchasingOrder.*)
Shipping.* = packAndShipProduct (PurchasingOrder.*)
informRetailerOrderStatus (Shipping.*, RetailerInfo.*)

Figure 7-2. The rule “PurchasingOrderProcessingRule” defined by a supplier to fulfill the retailer’s purchase order.
Figure 7-3. The interactions between the retailer and its suppliers to completing an online order.
CHAPTER 8
CONCLUSION AND FUTURE WORK

We have pointed out the need and importance for collaborating organizations to share not only data and computing resources, but also their policies, constraints, regulations and processes. We also stressed the need for a network infrastructure and a distributed processing technique to enable the interoperation of these resources, and the dynamic construction and enactment of inter-organizational processes in order to achieve inter-organizational collaboration and coordination. To meet these needs, we have developed in this work an integrated rule and process specification language to enable collaborating organizations to define events of common interest, organizational policies, constraints, regulations, and all sorts of human and organizational knowledge in terms of three different kinds of knowledge rules, and organizational and inter-organizational processes in terms of structures of operations and knowledge rules. Algorithms have been developed to translate each rule or process into Java code and wrap it as a Web Service. All the sharable Web Services are registered with the Web Service Registry for their discovery and sharing. A distributed network system, ETKnet2.0, has been implemented. It uses a meta-rule enhanced, event-triggered, distributed rule processing and process enactment technique to achieve 1) the interoperation of sharable data, rules, manual and automated operations and workflow processes, and 2) the dynamic construction and processing of inter-organizational processes. To demonstrate the utility of the integrated language, the distributed rule processing and process enactment technique and the implemented network system, we have used them to successfully implement and demonstrate the Standard Operating Procedure developed by National Plant Diagnostics Network of USDA. We have also
applied the R&D results in the area of e-business to demonstrate their general application and thus their broad impact.

We believe that our R&D results are important for the following reasons:

First, unlike the existing works that use different specification languages to define different types of rules and workflow processes, this work uses an integrated specification language to define them. The language presented in Chapter 3 provides the facilities for specifying logic derivation rules, action-oriented rules and integrity constraint rules, as well as the facility for defining workflow processes in terms of structures of, not only manual and automated operations like in a traditional workflow management system, but also different types of knowledge rules. The defined rules and processes are automatically translated and deployed as Web services. Since rules and processes can be specified by using the same language and processed uniformly as Web services in our system, distributed rules and processes can easily interoperate. Furthermore, there is no need to use multiple rule engines and a workflow management system or a business process execution system to manage and process them. This work provides a new way to combine business process modeling and business rule specification and a new way to achieve the interoperation and uniform processing of distributed rules and processes.

Second, high-level specifications of events, knowledge rules, processes and meta-rules are easier for collaborating organizations to understand, modify and use. Organizations do not have to write application code to implement them. The automatic translation of three different types of rules and processes into Web services allows them to be discovered and shared in a Web service infrastructure just like other Web services.
This approach makes the business rules and processes independently defined by organizations sharable and reusable.

Third, the architecture of ETKnet2.0 presented in Chapter 4 is highly scalable. As more and more organizations join a collaborative federation, the software components of ETKnet2.0 can be replicated and installed at the new members' network sites. The execution of distributed rules and organizational processes are carried out in parallel. Thus, as the number of organizations, rules and processes grow, the computational power grows. Since different events can occur at different sites, these sites will serve as the coordinators of the corresponding events to manage different sessions of rule processing and process enactment. Thus, no single site will be overburdened. In the present version of ETKnet2.0, there is a single host to, among other tasks, manage the registered events, rules and processes defined by all organizations. If it becomes a bottleneck, it is possible to establish multiple hosts to share the management and processing load (e.g., each manages a subset of events and their corresponding rules, processes and meta-rules). The multiple hosts would communicate with one another to determine, for example, a set of global rules that are applicable to an event instance.

Fourth, unlike most of the traditional workflow or process management systems, in which inter-organizational processes are pre-defined, this work provides a new way to achieve the dynamic construction and processing of inter-organizational processes. The event-triggered, distributed rule processing and process enactment technique presented in Chapter 4 ensures that all applicable business rules and pre-defined organizational and inter-organizational processes are executed to produce useful data needed for inter-organizational collaboration and coordination. Multiple rounds of distributed
processing of rules and processes constitute an inter-organizational process that is dynamically constructed and processed.

Fifth, the use of meta-rules at runtime (presented in Chapter 6) to suppress the processing of some applicable rules, structure the applicable rules, and/or resolve conflicting rules based on organizations’ priorities provides a dynamic way to tailor an inter-organizational process to meet the changing needs and partnership of collaborating organizations.

Sixth, the implemented and demonstrated ETKnet2.0 is a very general system and has many applications. As shown in Chapters 6 and 7, it can be applied in both e-government and e-business domains. We believe that it can be used to facilitate, not only the collaboration among business and government organizations in different environments, but also among people who undertake tasks of common interest such as scientists who collaborate in a research project or experiment, or students who collaborate in a joint project (e.g., DePree, Su and Xiao, 2009). Our R&D results can have a broader impact than what has been presented in this dissertation.

There are several R&D issues that are out of the scope of this research and are worthwhile for future investigation.

One is security in our network system. During the event-triggered rule processing and process enactment, we send all the event data to all the subscriber sites and the sites that have applicable rules. However, some of the event data should only be seen by some and not all of the sites due to security or privacy reasons. Hence, it is necessary for collaborating organizations to negotiate and establish some kind of trust and security policies to be enforced during a collaboration and knowledge sharing
session. Also, in our current system, we assume that all global rules and processes defined by collaborating organizations can be used by all organizations. It is possible in some cases that they can be used only by some but not others. Although we believe that security and trust policies as well as access control to event data, business rules and processes can all be specified in terms of three types of rules that our system can process and enforce, this issue needs further investigation.

The second issue that we have not dealt with is transaction management in our current system. In our system, each distributed rule processing and process enactment session triggered by an event instance forms a transaction. We have not dealt with the problem of transaction recovery when software or hardware problems occur. The traditional ACID properties of a transaction may not be applicable. For example, a business process that includes manual operations can last a long time. If we treat the multiple rounds of rule processing and process enactment that are triggered by an event instance as a transaction, and apply a locking mechanism, it will require locking resources for a long period of time. What is more, the conventional transaction rollback mechanism may not be desirable for our system. Consider the following case: if a collaborating site aborts for some reason, it may not be desirable to rollback the data generated in the session because the data generated by a subset of applicable rules may still contain information that is valuable to collaborating organizations. We need to examine and possibly extend the conventional ACID enforcement mechanism.

The third issue is about the efficiency of rule management at the Host site. In the current version, the specifications of rules defined by different organizations are registered with the Host. These rules are organized by the organizations that define
them. As more and more organizations participate in a collaborative federation, the number of sharable rules will continue to grow. For efficiency reasons, it can be beneficial to organize these rules based on some added semantic descriptions such as their functionalities, behaviors and/or the categories of organizations in an application domain. For example, rules defined by different organizations but serving a similar function can be grouped together as a rule group. The determination of the applicability of rules can then take the functionalities of rule groups into consideration, and select the group of rules whose functionality matches with the needs of an event instance, rather than simply include syntactic and semantic matches between their input data items and event data. This will avoid the need to perform the matching on all the registered rules. Furthermore, we can also extend our meta-rule specification to include meta-rules that specify constraints, not only on the processing of specific rules, but also on the processing of rule groups. For example, we can define a meta-rule which constrains the processing of a rule group that verifies the customer’s credit history to precede the processing of a rule group that is useful for completing an order instead of using many meta-rules to specify the precedence of individual rules.
APPENDIX A
THE XML SCHEMA OF IRPS, META-RULES AND EVENT DATA

The XML schemas for different types of rules, processes, meta-rules and event data are listed here.

1. The xml schema of integrity rules, derivation rules and action-oriented rule.

```xml
<?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.util.Date"/>
      <xs:enumeration value="Object"/>
      <xs:enumeration value="npdn.General"/>
      <xs:enumeration value="npdn.Sample"/>
      <xs:enumeration value="npdn.PackingRequirement"/>
      <xs:enumeration value="npdn.ResponsePlan"/>
      <xs:enumeration value="npdn.Result"/>
      <xs:enumeration value="npdn.Shipment"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="Expr">
    <xs:complexType>
      <xs:choice>
        <xs:sequence>
          <xs:element ref="rulebase:Term"/>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="rulebase:MathOp"/>
            <xs:element ref="rulebase:Term"/>
          </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:sequence>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<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: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:sequence>
  </xs:complexType>
</xs:element>

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

<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:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="IntCons"/>
<xs:element name="Implies"/>
<xs:element name="CAARule"/>
<xs:element name="Rule"/>
<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:element name="Head">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Atom"/>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="rulebase:AndOp"/>
        <xs:element ref="rulebase:Atom"/>
      </xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>

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

<xs:element name="FormulaCons">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Expr"/>
      <xs:element ref="rulebase:RelOp"/>
      <xs:element ref="rulebase:Expr"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="PredicateExpr">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="rulebase:Expr"/>
      <xs:element ref="rulebase:RelOp"/>
      <xs:element ref="rulebase:Expr"/>
    </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: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:element name="Action">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ActionName" type="xs:string"/>
      <xs:element name="ActionDescription" type="xs:string" minOccurs="0"/>
      <xs:choice>
        <xs:element ref="rulebase:Operation"/>
        <xs:element ref="rulebase:Process"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Operation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:NCName"/>
      <xs:element name="Location" type="xs:anyURI"/>
      <xs:element name="Inputs" minOccurs="0">
        <xs:complexType>
          <xs:sequence 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 name="Type" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="manual"/>
            <xs:enumeration value="automated"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
2. The xml schema of Process

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.dbcenter.cise.ufl.edu/process"
xmlns:process="http://www.dbcenter.cise.ufl.edu/process"
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:annotation>
    <xs:documentation>
      Schema for Process specification. 05/29/2010
    </xs:documentation>
  </xs:annotation>
  <xs:element name="Process">
    <xs:annotation>
      <xs:documentation>
        This is the root element for a process
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ProcessName" type="xs:NCName"/>
        <xs:element name="ProcessDescription" type="xs:string"/>
        <xs:element name="ProcessState" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="ACTIVE"/>
              <xs:enumeration value="SUSPENDED"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
<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 name="StructuredActivitiesSet"/>

<xs:element name="StructuredActivity">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="process:Sequential"/>
      <xs:element ref="process:Switched"/>
      <xs:element ref="process:Selective"/>
      <xs:element ref="process:Repeated"/>
      <xs:element ref="process:AndSplit"/>
      <xs:element ref="process:OrSplit"/>
      <xs:element ref="process:XorSplit"/>
      <xs:element ref="process:AndJoin"/>
      <xs:element ref="process:OrJoin"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="Sequential">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="process:Predecessor"/>
      <xs:element ref="process:Successor"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Switched">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="process:Predecessor"/>
      <xs:element name="Successors">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="case" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element ref="rulebase:IfExpr"/>
                  <xs:element ref="process:Activity"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="otherwise">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="process:Activity"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Selective">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Predecessor">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Activity">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Router">
                    <xs:simpleType>
                      <xs:restriction base="xs:NMTOKEN">
                        <xs:enumeration value="selective"/>
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="Id" type="xs:NMTOKEN" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Successors">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="subSuccessor" minOccurs="2" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element ref="process:Activity"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Repeated">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Predecessor">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Activity">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Router">
                    <xs:simpleType>
                      <xs:restriction base="xs:NMTOKEN">
                        <xs:enumeration value="repeated"/>
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="Id" type="xs:NMTOKEN" use="required"/>
              </xs:complexType>
            </xs:element>
            <xs:element ref="rulebase:IfExpr"/>
            <xs:element name="LoopBody">
              <xs:complexType>
                <xs:choice>
                  <xs:element ref="process:Activity"/>
                  <xs:element ref="process:StructuredActivitiesSet"/>
                </xs:choice>
              </xs:complexType>
            </xs:element>
            <xs:element>
              <xs:complexType>
                <xs:sequence>
                  <xs:element ref="process:Predecessor"/>
                  <xs:element ref="process:Successors" minOccurs="0"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="AndSplit">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Predecessor">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Activity">
              <xs:complexType>
                <xs:sequence>
                  <xs:element ref="process:Predecessor"/>
                  <xs:element ref="process:Successors"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="XorSplit">
              <xs:complexType>
                <xs:sequence>
                  <xs:element ref="process:Predecessor"/>
                  <xs:element ref="process:Successors"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="OrSplit">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="process:Predecessor"/>
      <xs:element name="Num" type="xs:nonNegativeInteger"/>
      <xs:element ref="process:Successors"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="AndJoin">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="process:Predecessors"/>
      <xs:element ref="process:Successor"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="OrJoin">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Num" type="xs:nonNegativeInteger"/>
      <xs:element ref="process:Predecessors"/>
      <xs:element ref="process:Successor"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Predecessor">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="process:Activity"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Successor">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="process:Activity"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Predecessors">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="subPredecessor" minOccurs="2" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="rulebase:IfExpr" minOccurs="0"/>
            <xs:element ref="process:Activity"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Successors">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="subSuccessor" minOccurs="2" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="rulebase:IfExpr" minOccurs="0"/>
            <xs:element ref="process:Activity"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Activity">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element name="Dummy"/>
        <xs:element ref="process:Router"/>
        <xs:element ref="process:Operation"/>
        <xs:element ref="process:Rule"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="Id" type="xs:NMTOKEN" use="required">
      <xs:annotation>
        <xs:documentation>unique identifier of the object</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="Execution">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="ASYNCHR"/>
          <xs:enumeration value="SYNCHR"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="Router">
  <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="switched"/>
      <xs:enumeration value="selective"/>
      <xs:enumeration value="unordered"/>
      <xs:enumeration value="repeated"/>
      <xs:enumeration value="dummy"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="Operation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:NCName"/>
      <xs:element name="Location" type="xs:anyURI"/>
      <xs:element name="Inputs" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
3. The xml schema of meta-rule

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:metarules="http://www.dbcenter.cise.ufl.edu/metarules"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.dbcenter.cise.ufl.edu/metarules"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="Metarules">
        <xs:annotation>
            <xs:documentation>Specification of one or more meta-rules</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="metarules:Metarule" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Metarule">
        <xs:annotation>
            <xs:documentation>Specification of one meta-rule</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="MetaruleName" type="xs:string"/>
                <xs:element name="Description" type="xs:string" minOccurs="0"/>
                <xs:element name="MetaruleState" 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 ref="metarules:Organization"/>
                <xs:choice>
                    <xs:element ref="metarules:Suppress"/>
                    <xs:element ref="metarules:Onetime"/>
                    <xs:element ref="metarules:ConditionalSuppression"/>
                    <xs:element ref="metarules:andPrerequisite"/>
                    <xs:element ref="metarules:orPrerequisite"/>
                    <xs:element ref="metarules:Order"/>
                    <xs:element ref="metarules:Priority"/>
                </xs:choice>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Suppress">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="metarules:Rule" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Onetime">
        <xs:complexType>
            <xs:sequence>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```
<xs:element ref="metarules:Rule" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:element name="ConditionalSuppression">
<xs:complexType>
<xs:sequence>
<xs:element ref="metarules:RuleCondition"/>
<xs:element name="thenSuppress">
<xs:complexType>
<xs:element ref="metarules:Rule" maxOccurs="unbounded"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="RuleCondition">
<xs:complexType>
<xs:sequence>
<xs:choice>
<xs:sequence>
<xs:element ref="metarules:RuleBooleanExpr" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="metarules:LogicalOp"/>
<xs:element ref="metarules:RuleBooleanExpr"/>
</xs:sequence>
<xs:sequence>
<xs:element ref="metarules:RuleCondition"/>
<xs:element ref="metarules:LogicalOp"/>
<xs:element ref="metarules:RuleCondition"/>
<xs:sequence minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="metarules:LogicalOp"/>
<xs:element ref="metarules:RuleCondition"/>
</xs:sequence>
<xs:sequence>
<xs:element ref="metarules:RuleCondition"/>
<xs:element ref="metarules:LogicalOp"/>
<xs:element ref="metarules:RuleCondition"/>
<xs:sequence minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="metarules:LogicalOp"/>
<xs:element ref="metarules:RuleCondition"/>
</xs:sequence>
<xs:sequence>
<xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="Not" type="xs:boolean"/>
<xs:element name="LogicalOp">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="AND"/>
<xs:enumeration value="OR"/>
</xs:restriction>
</xs:complexType>
<xs:element name="RuleBooleanExpr">
<xs:complexType>
<xs:sequence>
<xs:element ref="metarules:Not" minOccurs="0"/>
<xs:element ref="metarules:Rule"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="andPrerequisite">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="andPredecessors">
        <xs:complexType>
          <xs:element ref="metarules:Rule" maxOccurs="unbounded"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Successor">
        <xs:complexType>
          <xs:element ref="metarules:Rule"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="orPrerequisite">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orPredecessors">
        <xs:complexType>
          <xs:element ref="metarules:Rule" maxOccurs="unbounded"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Successor">
        <xs:complexType>
          <xs:element ref="metarules:Rule"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Order">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="metarules:Rule" minOccurs="2" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Priority">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="metarules:equalPriority" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="equalPriority">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="metarules:Organization" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Organization">
  <xs:complexType>
4. The schema of EventData

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:eventdata="http://www.dbcenter.cise.ufl.edu/eventdata"
targetNamespace="http://www.dbcenter.cise.ufl.edu/eventdata" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xsd:element name="EventData">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="EventName" type="xsd:string"/>
                <xsd:element name="EventID" type="xsd:string"/>
                <xsd:element name="RoundID" type="xsd:string"/>
                <xsd:element name="Coordinator" type="xsd:string"/>
                <xsd:element name="RulesExecuted">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element ref="eventdata:Rule" minOccurs="0" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="RulesDeactivated" minOccurs="0">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element ref="eventdata:Rule" minOccurs="0" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="RulesDeferred" minOccurs="0">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element ref="eventdata:Rule" minOccurs="0" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="RestrictedSites" type="xsd:string" minOccurs="0"/>
                <xsd:element name="DataItem" minOccurs="0" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Name" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```
<xsd:element name="OID" type="xsd:string"/>
<xsd:element name="DataType" type="xsd:string"/>
<xsd:element name="Value" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="source" type="xsd:string" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="OldDataItem" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="OID" type="xsd:string"/>
      <xsd:element name="DataType" type="xsd:string"/>
      <xsd:element name="Value" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:attribute name="source" type="xsd:string" use="optional"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Organization">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="OrgName" type="xsd:string"/>
      <xsd:element name="SiteIP" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Rule">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="RuleName" type="xsd:string"/>
      <xsd:element name="Location" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
APPENDIX B
THE DEFINITION OF ORGANIZATIONS AND ENTITIES USED IN NPDN SOP

There are several categories of organizations/labs/offices involved in the operation of SOP. Their names and functions are listed below:

**Triage Lab.** The state facility designated to receive and examine suspect samples. This lab is a Land Grant University Lab, State department of Agriculture lab or University Experiment Station Lab. In some states, more than one type of triage lab may exist and be a member of the NPDN.

**NPDN Regional Hub Lab.** The key coordinating lab for an NPDN region. These labs provide coordination, training, funding, and surge capacity support to the NPDN triage labs within their regions and occasionally to other regions.

**Expert Lab.** A lab which has been approved or provisionally approved by the Animal and Plant Health Inspection Service, Plant Protection and Quarantine (APHIS-PPQ) to conduct a specific diagnostic test. The diagnosticians of this type of lab receive additional training from APHIS-PPQ diagnostician and the labs are equipped to handle surge overflow from triage labs.

**SPRO.** The State Plant Regulatory Official. The highest ranking state plant regulatory official. SPRO is employed by the State Department of Agriculture.

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

**NIS.** The APHIS-PPQ National Identification Service. The USDA-authorized lab for diagnosing plant diseases (fungal and viral). This lab also coordinates insect diagnosis with the Systematic Entomology Laboratory, Communication and Taxonomic Services Unit (USDA-ARS SEL CTSU) for triage and hub labs.

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

**EDP.** The APHIS-PPQ Emergency and Domestic Programs.

**RPM:** The Regional Program Manager

There are a few data entities used in the SOP implementation. The entity

*npdn.Sample (Sample)* is used to describe a sample in the system. It contains the
following attributes. As the diagnosis proceeds, these attributes get assigned appropriate values. The tuple \((localLabSampleID, labID, sampleDate)\) is the unique identifier of a sample instance.

localLabSampleID of type string. It is the sample index in a local lab, which is the triage lab that receives the sample.

labId of type string. It is the lab Id of the triage lab.

sampleDate of type date. The date when the sample is collected.

sampleType of type string. The valid values of sampleType are “plant” and “insect”.

classification of type string. The valid values of classification are “unclassified”, “suspected”, “presumptive positive”, “positive” and “negative”.

stateOfOrigin of type string. It is the state where the sample is collected.

status of type string. Its valid values are “triagelab”, “hubexpertlab”, “CDD”, and “confirmed”.

The entity \(npdn.Result (Result)\) is used to describe the result of a diagnosis to a unique sample. It contains the following attributes. The \((localLabSampleID, labID, sampleDate)\) is used to identify a result record.

localLabSampleID of type string. It has the same value as that of the corresponding sample.

labId of type string. It has the same value as that of the corresponding sample.

sampleDate of type date. It has the same value as that of the corresponding sample.

sampleType of type string. The valid values of sampleType are “plant” and “insect”.

classification of type string. The valid values of classification are “positive” and “negative”.

stateOfOrigin of type string. It is the state where the sample is collected.

status of type string. Its valid values are “confirmed”.

The entity \(npdn.Shipment (Shipment)\) contains the following attributes.

- sender of type string. It is the sender of this shipment.
• receiver of type string. It is the receiver of this shipment.
• trackingNumber of type string. It is the tracking number of this shipment if has.
• timeOfDelivery of type date. The date when the shipment happens.
• methodOfDelivery of type string. It is the method used for this shipment.
• properlyPackaged of type boolean. It is used to indicate whether the sample included is packaged properly according to the sample type.
• submissionFormIncluded of type boolean. It is used to indicate whether the required submission form is included in this package.
• businessCardIncluded of type boolean. It is used to indicate whether the required business card is included in this package.
• arrivalCondition of type string. Its valid values are “poor” and “good”.
• notification of type string. It is used to indicate who should be notified for this shipment.

The entity NPDN.PackingRequirement (PackingRequirement) is the packing requirement for a sample that will be shipped to another lab. SOP has different packing requirement to plant samples and insect samples.

sealed_in_a_sturdybox of type boolean. It is used to indicate whether the sample should be sealed in a study box.

vial of type boolean. It is used to indicate whether the sample should be packed in a vial.

in_appropriate_crush_proof of type boolean. It is used to indicate whether the sample should be packed in crush proof.

double_bag of type boolean. It is used to indicate whether the sample should be packed in a double bag.

mark of type string. It is additional note for the packing requirement.
The entity NPDN.ResponsePlan (ResponsePlan) is used to describe the response information that SPRO and SPHD should have if the presumptive positive sample in the system is confirmed to be positive. It contains the following attributes.

- **officialsToBeNotified** of type string.
- **preEmptiveAction** of type string.
- **pressReleaseAction** of type string.
- **postConfirmationResponseAction** of type string.

The entity *npdn.General* includes all variables that are used to save the temporary values generated by operations or rules during their execution.

- **ok_to_contact_other_directors** of type boolean. Whether the NPDN regional director in the state of origin should contact directors in other regions.
- **regionaldirector_knows_presumptive_positive_sample** of type boolean. Whether the NPDN regional director in the state of origin has been informed of a presumptive positive sample in the system.
- **SPHD_knows_presumptive_positive_sample** of type boolean. Whether SPHD in the state of origin has been informed of a presumptive positive sample in the system.
- **SPRO_knows_presumptive_positive_sample** of type boolean. Whether SPRO in the state of origin has been informed of a presumptive positive sample in the system.
- **has_distant_diagnosis_capability** of type boolean. Whether the current triage lab has distant diagnosis capability.
- **is_distant_diagnosis_appropriate** of type boolean. Whether it is appropriate to conduct the distant diagnosis to the sample.
- **NIS_gets_result** of type boolean. It is used to indicate whether NIS gets the confirmed diagnostic result.
- **RPM_gets_result** of type boolean. It is used to indicate whether RPM gets the confirmed diagnostic result.
- **SPRO_gets_result** of type boolean. It is used to indicate whether SPRO gets the confirmed diagnostic result.
• **SPHD** _gets_result_ of type boolean. It is used to indicate whether SPHD gets the confirmed diagnostic result.

• **regionaldirector_in_stateoforigin_gets_result** of type boolean. It indicates whether the NPDN regional director in the state of origin gets the confirmed diagnostic result.

• **trialeglab_notifier** of type string. It is used to indicate who will contact triage lab with the confirmed diagnostic result. Its valid values are “SPRO” and “SPHD”.

• **morethan7days** of type boolean. Whether the confirmation will take more than seven days.

• **is_sample_new_to_USA** of type boolean. Whether the sample is new to USA.

• **is_sample_recognized_as_significant** of type boolean. Whether the sample is recognized as significant.
LIST OF REFERENCES


BIOGRAPHICAL SKETCH

Xuelian Xiao was born in Guangdong, China. She received her B.S. degree and M.S. degree from Sun Yat-sen University, Guangzhou, China, in 2000 and in 2003 correspondingly, both in computer science. She joined the Department of Computer and Information Science and Engineering (CISE) graduate program at the University of Florida in August 2006. Her research areas include knowledge management and sharing, Web services, e-business and inter-organizational process and collaboration.