

JXPert: A JAVA/XML-BASED INTELLIGENT INFORMATION SYSTEM

by

XIN ZHANG

(Under the Direction of Walter D. Potter)

ABSTRACT

An Intelligent Information system (IIS) is for storing, retrieving, manipulating, and distributing information or knowledge. In this paper, we present a Java and XML based rule authoring tool and a Web application to intelligently integrate various Forestry Decision Support Systems (FDSS). It utilizes a new XML rule description language, Simple Knowledge Markup Language (SKML) to model knowledge base and deduction rules, which can be conveniently exported across applications. It also successfully solved the problem of legacy FDSS application integration and provided an integrated solution for forestry management personnel.

INDEX WORDS: IIS, SKML, XML, FDSS

JXPert: A JAVA/XML-BASED INTELLIGENT INFORMATION SYSTEM

by

XIN ZHANG

B.S., Tsinghua University, P.R.China, 1996

M.S., Clemson University, 1998

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial
Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2002

© 2002

Xin Zhang

All Rights Reserved

JXPert: A JAVA/XML-BASED INTELLIGENT INFORMATION SYSTEM

by

XIN ZHANG

Approved:

Major Professor: Walter D. Potter

Committee: Eileen M. Kraemer
Daniel T. Everett

Electronic Version Approved:

Gordhan L. Patel
Dean of the Graduate School
The University of Georgia
August 2002

ACKNOWLEDGEMENTS

I want to thank Dr. Don Potter, my major professor, for his constant support, advice and encouragement. And also I want to thank Dr. Dan Everett and Dr. Eileen Kraemer, their valuable expertise and suggestions are so important to this project.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	vi
CHAPTER	
1 INTRODUCTION	1
2 LITERATURE REVIEW	3
2.1 System Integration Issues in the Forestry Domain.....	3
2.2 Expert System, Inference Method and Descriptive Logic	4
2.3 Current State	6
2.4 Motivation and Goals	8
3 DESIGN SPECIFICATION	10
3.1 Technological Infrastructure	10
3.2 Architecture Overview	12
3.3 Simple Knowledge Makeup Language	13
3.4 SKML Rule Authoring Tool	17
3.5 Intelligent Information System.....	20
4 DEMONSTRATION	27
4.1 SKML Authoring Tool.....	27
4.2 Intelligent Information System.....	31
5 CONCLUSION AND FUTURE WORK	34
REFERENCES	36

APPENDICES

A	THE SKML ANNOTATED DTD	38
B	THE JXPert SKML RULE SET	41
C	THE JXPert PROLOG RULE SET	50

CHAPTER 1

INTRODUCTION

A Forest Ecosystem Management Decision Support System (FEM-DSS) is computer software designed to help forest management personnel make decisions based on scientifically sound information. Usually, every software is developed independently to address specific issues with little coordination to others. It is hard for inexperienced users to select the proper software, to acquire, to install and to use it correctly on their local systems. In this research we incorporate progress in the system integration and business rule technologies to develop a Web-based Intelligent Information System (IIS) to integrate various forestry decision support systems (FDSS's).

In any application, business rules are the most dynamic component. Focusing the development process on identifying and formalizing business rules facilitates a quicker response to market and to industry changes. Identifying and defining business logic in the form of rules ensures that it is easily communicated, well understood, and can be managed separately from application code. In this research, we propose the Simple Knowledge Markup Language (SKML), an XML application suitable to represent knowledge base. As a reference implementation, the SKML rule set is mapped to a Prolog program. This module also contains a visual tool to guide the user creating a SKML document and a translator module to map a rule set to a Prolog program.

In recent years, with the excellent progress of distributed computing, Internet and Web technologies, the possibility of accessing a large number of independently designed information sources has grown. With the growing popularity of the Internet, a dedicated Web processing space is used to handle most of the presentation logic while the client merely becomes a dumb Web browser. In this research, we develop a Web application to intelligently integrate several forest decision support systems. After the user answers a series of questions, the system will automatically figure out the applicable FDSS and enable the user to run the application through the Web interface. This four-tier application integrates the AMZI! Logic Server as the Prolog rule inference engine and uses the XML scripts to define the FDSS services.

Our research improves forestry ecosystem management capability by intelligently integrating various FDSS's, which not only relieves users of the burden to acquire and to install those software, but also smoothes their learning curves to pick and use those systems. This application helps the users identify the solution efficiently and to run the legacy applications through the uniform Web interface. We also improve rule languages by providing this new rule markup language and its visual authoring tool.

CHAPTER 2

LITERATURE REVIEW

2.1 System Integration Issues In the Forestry Domain

A Forest Ecosystem Management Decision Support System (FEM-DSS) allows forest management teams to make better decisions and to improve the efficiency of their work. A typical FEM-DSS consists of a command line or graphical user interface, database, geographical information system (GIS), simulation and optimization tools, knowledge base and decision methods [1]. But usually, one system can only address topics in some specific fields under certain environmental and geographical conditions. It is difficult for inexperienced forest management personnel to pick the correct tool from the widely available collections to fulfill their customized requirements. And in today's market, there is no such application to offer a comprehensive solution to address all users' interests. On the other hand, instead of developing such a master application to discuss all important issues, it is more feasible to reuse existing software by integrating and enhancing it to provide additional services. However, integration issues in the forestry domain are complex. Usually, FDSS's are standalone applications running on the client's local machine with little or no coordination with others. They are legacy systems developed in various programming languages and platforms with primitive or no programming interface. They have proprietary input and output formats. Some have a graphical user interface, but others are merely console applications running from a batch file or in interactive

mode. Any attempt at providing a scheme for integrating a current and a future FEM-DSS should address all these problems and offer a general-purpose interoperability framework.

2.2 Expert System, Inference Method and Description Logic

Conventional programming languages such as FORTRAN and C are designed and optimized for the procedural manipulation of data. Humans, however, often solve complex problems using very abstract, symbolic approaches, which are not well suited for implementation in conventional languages. Although abstract information can be modeled in these languages, considerable programming effort is required to transform the information to a format usable with procedural programming paradigms.

One of the results of research in the area of artificial intelligence has been the development of techniques, which allow the modeling of information at higher levels of abstraction. These techniques are embodied in languages or tools, which allow programs to be built that closely resemble human logic in their implementation and are therefore easier to develop and to maintain. These programs, which emulate human expertise in well-defined problem domains, are called expert systems [2]. The availability of expert system tools has greatly reduced the effort and the cost involved in developing an expert system.

Rule-based programming is one of the most commonly used techniques for developing expert systems. In this programming paradigm, rules are used to represent heuristics, which specify a set of actions to be performed for a given situation. A rule is composed of an if portion and a then portion. The if portion of a rule is a series of patterns that specify the facts that cause the rule to be applicable. The process of matching facts to patterns is called pattern matching. The expert system tool provides a mechanism, called the inference engine, which automatically

matches facts against patterns and determines which rules are applicable. The if portion of a rule can actually be thought of as the whenever portion of a rule since pattern matching always occurs whenever changes are made to facts. The then portion of a rule is the set of actions to be executed when the rule is applicable. The actions of applicable rules are executed when the inference engine is instructed to begin execution. The inference engine selects a rule and then the actions of the selected rule are executed. The inference engine then selects another rule and executes its actions. This process continues until no applicable rules remain.

There are generally two types of inferencing: Forward chaining or data driven and Backward chaining or goal driven [3]. Forward chaining applies to reactions to a change. The new information causes us to examine a specific rule that we locate directly without having to review any of the other rules that are irrelevant at the moment. Backward chaining applies to situations where we are trying to explain the cause of what we observe. Goal-driven inference applies to situations where you are making diagnostics. It presumes that we are aware of a given situation but we do not know what caused it. Therefore, we have to ask questions and to perform some tests to determine the cause.

Description logics are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies [4]. They are usually given a declarative semantics, which allows them to be seen as sub-languages of predicate logic. The basic building blocks are concepts, roles and individuals. Concepts describe the common properties of a collection of individuals and can be considered as unary predicates which are interpreted as sets of objects. Roles are interpreted as binary relations between objects. Each description logic

also defines a number of language constructs (such as intersection, union, role quantification) that can be used to define new concepts and roles.

Description logic systems have been used for building a variety of applications including conceptual modeling, information integration, query mechanisms, view maintenance, software management systems, planning systems, configuration systems, and natural language understanding [5].

2.3 Current State

In the current rule engine market there are several commercial products such as the META Group's Brokat Advisor [6] and ILog JRules [7], SilverStream's ePortal Rules Engine [8], Haley's CIA Server [9] and IBM's AlphaWorks CommonRules [10] with different feature sets. They each have their own set of APIs, and they each have their own proprietary rule languages. As a result, rules cannot be shared across applications that use different rule engines, and applications developed using these rule engines remain tied to a specific vendor implementation.

As a metalanguage, XML can be used to define a variety of different markup languages. Because the rule engine has traditionally required specialized languages in order to solve problems, the availability of a metalanguage suitable for defining multiple specialized languages for solving problems should allow for considerable application in the rule engine field. Indeed, there have been some applications of XML to rule languages.

The RuleML Initiative represents a collaborative research effort by an international team of participants seeking to develop the shared Rule Markup Language (RuleML) [11]. This RuleML kernel language can serve as a specification for immediate rule interchange and can be

gradually extended. Rules can be stated in natural language, in some formal notation, or in a combination of both. The RuleML Initiative is working towards an XML-based markup language that permits Web-based rule storage, interchange, retrieval, and firing application.

Business Rule Markup Language (BRML) is an XML Rule Interlingua for Agent Communication, based on Courteous/Ordinary Logic Programs [12]. It is used in connection with 'CommonRules' from IBM, and was developed in connection with IBM's Business Rules for E-Commerce Project.

The Relational-Functional Markup Language (RFML) is an XML application for declarative programming and knowledge representation [13]. RFML has been implemented as output syntax for relational-functional knowledge bases and computations.

Description Logic Markup Language (DLML) is a system that allows encoding description logics expressions into XML [14]. One important motivation for description languages is to be able to embed formal knowledge in documents. Other motivations include the experiments of simple representation languages, for which description logic is well suited.

Although some of the XML rule languages are good candidates for our application, they all have some drawbacks preventing them from being applicable. We need an XML application suitable to represent a knowledge base (particularly in the forestry domain), but also simple enough to be implemented or to be converted to other proprietary rule language formats. However, most of the current XML based rule languages are designed for some specific applications or they are too complex to be implemented in a limited time frame.

For legacy forestry decision support system integration issues, a Distributed Component Object Model (DCOM) based distributed client-server intelligent information system (IIS) for

integration of FEM-DSS's has been developed [15][16]. This system consists of three modules. At the client end, it has the interface module, which provides the visual interface to the user, and the Intelligent Information Module (IIM). The IIM contains the intelligence necessary to provide additional functionality to the system. The IIM interprets the user's options to decide the appropriate actions and to activate the right applications. At the server end, it has network components and wrappers to facilitate communication with legacy applications. In this IIS system, the communication protocol between client and server is DCOM. This design provides unified, integrated, and intelligent access to different legacy FEM-DSS's. However, this design still has some limitations:

1. This system can only run on a local area network. The client must have knowledge of the server to activate the legacy FEM programs on the server. No process management exists on the server. The performance is doubtful if multiple connections are established at the same time.
2. Software must be installed on the client side. The client software contains the user interface and the Intelligent Information Module (IIM).
3. The IIM part in this IIS is still primitive. The rules are hard coded in the controller. No independent knowledge base and inference engine are used.

2.4 Motivation and Goals

In this paper, JXpert, a Web-based Intelligent Information System for Integration of Forest Decision Support System, is discussed. The JXPert uses a Simple Knowledge Markup Language (SKML), a new XML rule descriptive language, to define business logic and a Web application to integrate multiple FDSS's.

Although the motivation for this system is to intelligently integrate forest decision support systems, we do not want to build a special purpose tool with limited functions. During the design and implementation of the system, we set up the following goals:

1. Generic. The tool should be application domain independent. In other words, the rule engine is a general-purpose inference engine and the XML rule language can describe rules not limited to the forestry domain.
2. Integrated. The rule engine and FDSS's should be seamlessly integrated into the Web application.
3. Extensible. The system should be configurable to support new functions with minimum code modification or recompilation.
4. Easy to use. An intuitive graphical user interface should be provided for both administrators and end users.
5. Portable. The tool should be deployable across platforms.

The outline of the rest of this paper is as follows. Chapter 3 discusses the design, technological infrastructure, and implementation. Chapter 4 walks through the use of the JXpert system and Chapter 5 concludes with a discussion on future work.

CHAPTER 3

DESIGN SPECIFICATION

In this chapter, we introduce JXpert in depth, but before that, we briefly review the technologies utilized in the system.

3.1 Technological Infrastructure

Modern software technology provides much computing power so that difficult problems can be solved more easily today. Java is such a software platform. Its central promise, “write once, run anywhere”, has earned a reputation in today’s information technology [17]. JXpert is implemented in Java and built on the J2EE platform. The primary factors for choosing Java are that it is platform-neutral, robust, open-architecture and supports applications deployed over heterogeneous network environments. We briefly review the technologies used in this system.

The Java 2 Enterprise Edition (J2EE) is essentially a collection of APIs that can be used to build large scale, distributed, component-based, multi-tier applications [18]. The J2EE is also a standard for building and deploying enterprise applications.

Servlets are the Java platform technology of choice for extending and enhancing Web servers [19]. Servlets provide a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs. Servlets have access to the entire family of Java APIs and can also access a library of HTTP-specific calls

and receive all the benefits of the mature Java language, including portability, performance, reusability, and crash protection.

JavaServer Pages (JSP) technology allows Web developers and designers to rapidly develop and easily maintain, information-rich, dynamic Web pages that leverage existing business systems [19]. As part of the Java family, JSP technology enables rapid development of Web-based applications that are platform independent.

An XML document has both a logical and physical structure. The logical structure allows a document to be divided into named units and sub-units, called elements. The physical structure allows components of the document, called entities, to be named and stored separately. XML is actually a meta-language, meaning that it is a language that describes other languages. There is no predefined list of elements. XML provides complete freedom to employ elements with names that are meaningful to the application [20].

Prolog, which stands for PROgramming in LOGic, is the most widely available language in the logic programming paradigm [21]. Logic and therefore Prolog is based on the mathematical notions of relations and logical inference. Prolog is a declarative language meaning that rather than describing how to compute a solution, a program consists of a database of facts and logical relationships that describe the relationships, which hold for the given application. Rather than running a program to obtain a solution, the user asks a question. When asked a question, the run time system searches through the database of facts and rules to determine the answer.

3.2 Architecture Overview

The JXpert, a Web-based Intelligent Information System, is exclusively built on XML, Java, Web, and AI technology. This project consists of two major software components: a general purpose XML rule authoring tool and a Web application in the forestry domain.

First, we propose the Simple Knowledge Markup Language (SKML), an XML application suitable to represent knowledge bases. As a reference implementation, the SKML rule set is mapped to a Prolog program and the Prolog interpreter is used as the inference engine. Since the SKML is designed as a standard way to describe rules, we can also write our own inference engine to interpret the rules directly or to translate them into other rule language formats. This module contains a collection of classes to represent the underlying SKML elements, a visual tool to guide the user creating a SKML document and a translator module to map a rule set to a Prolog program.

Second, we develop a Web application to intelligently integrate several forest decision support systems. After the user answers a series of questions, the system will automatically figure out the applicable FDSS and enable the user to run the application through the Web interface. This is a standard four-tier architecture: the client tier only requires a dumb Web browser; the presentation tier contains Java Servlet, JSP and HTML components to handle user requests and responses; the business logic tier integrates the AMZI! Personal Logic Server as the Prolog inference engine and utilizes XML/XSL scripts to convert the user input information into a legacy systems' proprietary input format; finally, the Web application communicates with legacy FDSS's through the wrapper classes. In the following sections, we will discuss the design details of the JXpert system.

3.3 Simple Knowledge Markup Language

In this research, we propose a simplified, open, XML-based rule language: the Simple Knowledge Markup Language (SKML). We invent SKML based on other XML rule languages and logic programming syntax. SKML describes a generic rule language consisting of the subset of language constructs suitable for knowledge representation. Because it does not use constructs specific to a proprietary vendor language, rules specified using this Document Type Definition (DTD) can easily be exported and executed on any conforming rule engine.

The language details are listed in List 3.1, the SKML DTD. The root element of SKML is ‘rules’ that consists of multiple ‘rulePrototype’ and ‘rule’ elements. A ‘rulePrototype’ defines the name and signature of a rule, just like the abstract function of programming languages and ‘rule’ is the implementation of ‘rulePrototype’. One ‘rulePrototype’ can have multiple ‘rule’ counterparts all in a disjunctive relationship implicitly. A ‘rule’ element has three parts: one ‘formalArgs’ element which defines the parameters, one ‘initBinding’ element which binds the parameters to initial values and repetitive ‘condition’ elements. A rule is not satisfied unless all the enclosing ‘condition’ elements are succeeded. A ‘condition’ can be a binary, unary, nested binary or unary expression or a function call to other rules. Furthermore, in a ‘rule’ section, if all the parameters are bonded to constants and there are no enclosing ‘condition’ elements, this rule is degenerated to a pure fact. To demonstrate the power and flexibility of SKML, two examples: a business rule to define a discount program and a rule to define the sibling relationship are listed in List 3.2 and List 3.3.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- SKML (Simple Knowledge Markup Language) DTD -->
<!ENTITY % parameter "variable | constant">
<!ENTITY % expression "%parameter; | unaryExpr | binaryExpr">
<!ENTITY % relationalOperator "eq | neq | lt | lte | gt | gte">
<!ENTITY % arithmeticOperator "add | subtract | multiply | divide |
assign">
<!ELEMENT rules (rulePrototype*,rule*)>
<!ATTLIST rules
    name NMTOKEN #IMPLIED>
<!ELEMENT rulePrototype EMPTY>
<!ATTLIST rulePrototype
    id ID #REQUIRED
    name NMTOKEN #REQUIRED
    arity CDATA #REQUIRED>
<!ELEMENT rule (formalArgs, initBinding, condition*)>
<!ATTLIST rule
    id IDREF #REQUIRED
    name NMTOKEN #IMPLIED>
<!ELEMENT formalArgs (variable+)>
<!ELEMENT initBinding (bind*)>
<!ELEMENT bind (variable, constant)>
<!ELEMENT condition (ruleRef | %expression;)>
<!ELEMENT unaryExpr (%expression;)>
<!ATTLIST unaryExpr
    operator (plus | minus | not) #REQUIRED>
<!ELEMENT binaryExpr ((%expression;), (%expression;))>
<!ATTLIST binaryExpr
    operator (%relationalOperator; | %arithmeticOperator;) #REQUIRED >
<!ELEMENT ruleRef (%parameter;)+>
<!ATTLIST ruleRef
    id IDREF #REQUIRED
    name NMTOKEN #IMPLIED>
<!ELEMENT variable EMPTY>
<!ATTLIST variable
    name NMTOKEN #REQUIRED>
<!ELEMENT constant EMPTY>
<!ATTLIST constant
    type (string | boolean | integer | float | null) #REQUIRED
    value CDATA #REQUIRED>

```

List 3.1. Simple Knowledge Markup Language DTD

Rule: Discount

Description: If
the total purchase is > \$100.00
Then
Set the adjusted total to total * (1.00 - discount rate)

```
<!DOCTYPE rules SYSTEM "skml.dtd">
<rules>
  <!-- rule prototypes -->
  <rulePrototype id="d001" name="discount" arity="3"/>
  <!-- rules -->
  <rule id="d001" name="discount">
    <formalArgs>
      <variable name="Total"/>
      <variable name="AdjustedTotal"/>
      <variable name="DiscountRate"/>
    </formalArgs>
    <initBinding>
      <bind>
        <variable name="DiscountRate"/>
        <constant type="float" value="0.10"/>
      </bind>
    </initBinding>
    <condition>
      <binaryExpr operator="gt">
        <variable name="Total"/>
        <constant type="float" value="100.00"/>
      </binaryExpr>
    </condition>
    <condition>
      <binaryExpr operator="assign">
        <variable name="AdjustedTotal"/>
        <binaryExpr operator="multiply">
          <variable name="Total"/>
          <binaryExpr operator="subtract">
            <constant type="float" value="1.00"/>
            <variable name="DiscountRate"/>
          </binaryExpr>
        </binaryExpr>
      </binaryExpr>
    </condition>
  </rule>
</rules>
```

List 3.2 Discount SKML

Rule: parent
Description: fact
Mike is Tom's parent
Mike is Jack's parent

Rule: Sibling
Description: If
Z is X's parent
And
Z is Y's parent
And
X != Y
Then
X and Y are siblings

```
<!DOCTYPE rules SYSTEM "skml.dtd">
<rules>
  <!-- rule prototypes -->
  <rulePrototype id="r001" name="parent" arity="2"/>
  <rulePrototype id="r002" name="sibling" arity="2"/>
  <!-- facts -->
  <rule id="r001" name="parent">
    <formalArgs>
      <variable name="X"/>
      <variable name="Y"/>
    </formalArgs>
    <initBinding>
      <bind>
        <variable name="X"/>
        <constant type="string" value="Mike"/>
      </bind>
      <bind>
        <variable name="Y"/>
        <constant type="string" value="Tom"/>
      </bind>
    </initBinding>
  </rule>
  <rule id="r001" name="parent">
    <formalArgs>
      <variable name="X"/>
      <variable name="Y"/>
    </formalArgs>
    <initBinding>
      <bind>
        <variable name="X"/>
        <constant type="string" value="Mike"/>
      </bind>
      <bind>
        <variable name="Y"/>
        <constant type="string" value="Jack"/>
      </bind>
    </initBinding>
  </rule>

```

```

        </bind>
    </initBinding>
</rule>
<!-- rules -->
<rule id="r002" name="sibling">
    <formalArgs>
        <variable name="X"/>
        <variable name="Y"/>
    </formalArgs>
    <initBinding/>
    <condition>
        <ruleRef id="r001" name="parent">
            <variable name="Z"/>
            <variable name="X"/>
        </ruleRef>
    </condition>
    <condition>
        <ruleRef id="r001" name="parent">
            <variable name="Z"/>
            <variable name="Y"/>
        </ruleRef>
    </condition>
    <condition>
        <binaryExpr operator="neq">
            <variable name="X"/>
            <variable name="Y"/>
        </binaryExpr>
    </condition>
</rule>
</rules>

```

List 3.3 Sibling SKML

3.4 SKML Rule Authoring Tool

Although XML is a character-based means of information representation and can be created by any text editor, there is a sharp learning curve for a new user to write error free XML files from bottom up because of its strict syntax specified in the DTD. On the other hand, considering too much about XML specific issues will also distract the user's concentration from the underlying business logic.

To ease the authoring process, we developed a visual SKML authoring tool. This template-based tool can walk the user through each step of the SKML creation, navigate rule

collections, edit/add/remove rules, check validity and translate the SKML rule set to a Prolog program. This tool has three software components: the SKML element mapping package (jxpert.rule), the Prolog element mapping package (jxpert.rule.prolog) and the graphical user interface package (jxpert.rule.gui).

In essence, a mapping procedure is used to keep interesting information and filter out trivial or unwanted information, and transform the interesting information into the desired form. The SKML element mapping package binds selected SKML elements with corresponding Java objects to take advantage of the Java programming language's full capability to manipulate the XML structure. The Java objects are arranged in a tree-like structure and all implement a common interface. In the future, to accommodate SKML specification changes, new elements could be plugged into or removed from the overall architecture with minimum effort. The common interface script is listed in List 3.4.

```
public interface SKMLObject{
    public void transform2xml(Element element);
    public void loadFromXml(Element element);
    public boolean validate();
    public void transform2prolog(PTerm term);
    public String getError();
    public void setError(String title, String msg);
    public void setParent(SKMLObject parent);
    public SKMLObject getParent();
}
```

List 3.4 SKML object interface

The first two functions handle Java object and XML element mapping. In a mapping process, the two functions call its linked elements' transform2xml() and loadFromXml() methods recursively to complete the whole conversion. The validate() function recursively validates the current and linked elements' validity. The transform2prolog() method is a

convenient procedure to convert the Java object to the corresponding Prolog language component. As an extension, we may define other translators to translate the SKML structure to other rule languages. Finally, the setParent() method links the current object to the upper level element.

The XML rule language is a flexible and universal way to describe business rule logic, but without the help from the conforming inference engine, it cannot be used to solve any real world problems. Prolog is a mature and widely used logic programming language in expert systems and artificial intelligence computing fields. Since there exists great similarity between SKML and logic programming methods, in this research, instead of developing our own SKML rule engine, the Simple Knowledge Markup Language is mapped to a subset of the Prolog language. The Prolog mapping package contains all the necessary Java classes representing the counterparts in a Prolog's 'predict' structure. The user can intuitively create a SKML rule set through this authoring tool without awareness of the complicated SKML DTD and easily translate it to a Prolog program. List 3.5 and 3.6 show the Prolog programs generated from the SKML rule sets discussed in List 3.2 and 3.3.

Rule: Discount
Description: If
 the total purchase is > \$100.00
 Then
 Set the adjusted total to total * (1.00 - discount rate)

```
% discount/3.  
discount(TOTAL, ADJUSTEDTOTAL, 0.10) :- (TOTAL > 100.00), (ADJUSTEDTOTAL  
is (TOTAL * (1.00 - 0.10))).
```

List 3.5 Discount Prolog Program

Rule: parent
Description: fact
Mike is Tom's parent
Mike is Jack's parent

Rule: Sibling
Description: If
Z is X's parent
And
Z is Y's parent
And
X != Y
Then
X and Y are siblings

```
% sibling/2.  
% parent/2.  
parent('mike', 'tom').  
parent('mike', 'jack').  
sibling(X, Y) :- parent(Z, X), parent(Z, Y), (X =\= Y).
```

List 3.6 Sibling Prolog Program

3.5 Intelligent Information System

System interoperability has gained increasing attention because of excellent progress in Internet, Web and distributed computing infrastructure technology, leading to easy access to a large number of independently created and managed information sources. This section discusses the various issues that we should take into account when designing the architecture for integration. A good design should offer high-level abstraction to hide the underlying heterogeneity in the numerous applications. This design should be robust enough to handle protocols, data formats, network and platform differences, and the overall architecture should be flexible enough to accommodate future upgrading.

The Model-View-Controller (MVC) [22] design pattern is used through out this application. Core data access functionality is separated from presentation and control logic.

Such separation allows multiple views to share the same enterprise data model, which makes supporting multiple clients easier to implement, test, and maintain. Figure 3.1 shows the architecture of this IIS application.

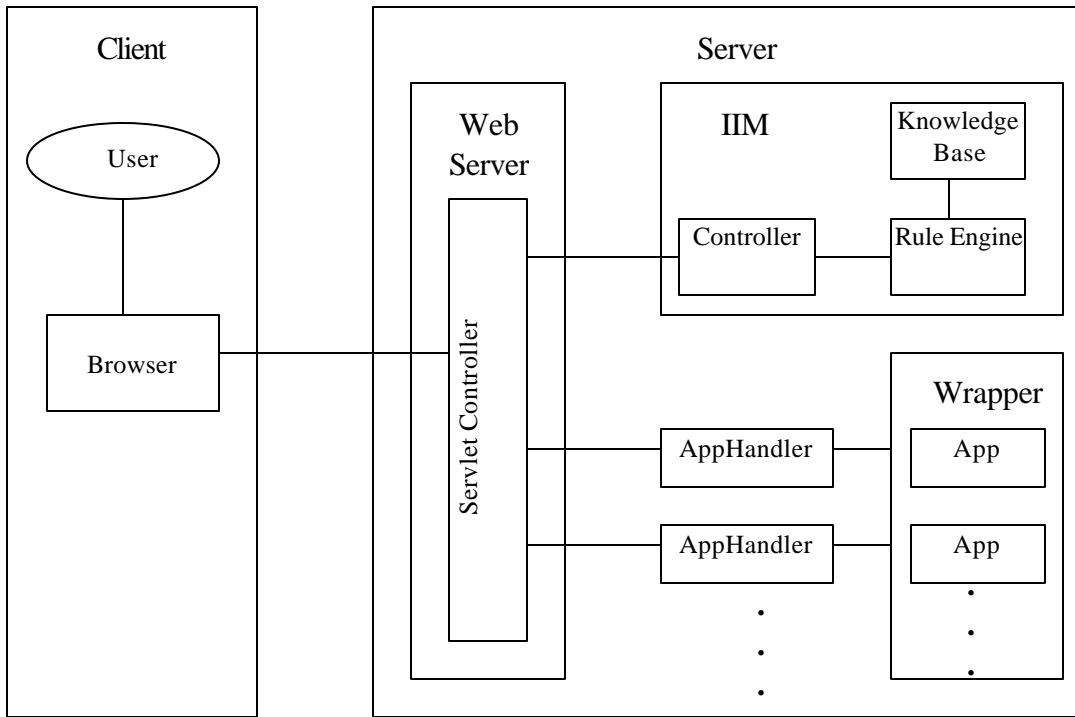


Figure 3.1 The IIS architecture

3.5.1 Legacy Systems

Currently we integrate two representative FDDS's in our Intelligent Information System (IIS) framework. The two sample systems are Forest Vegetation Simulator (FVS) [23] and FIBER [24]. The FVS simulation tool covers various geographical regions in the US and FVS is one of the most popular FDDS's used by forest management personnel. FIBER can only simulate limited vegetation types available in the northeastern region of the US, but it is capable of modeling interactions among different species.

Both systems are standalone software written in FORTRAN and C. FVS has a visual interface called SUPPOSE and the actual simulation is in a different executable. FVS needs two kinds of input information. One is the tree stand data, which gives the current tree information and the other one is keyword collection, which specifies the actions needed to be performed against the tree stand data. Since FVS can run from a batch file, we can construct the input file from the Web forms then execute it from the script. But FIBER can only run in interactive mode and all input information is obtained through menu options so we have to simulate the keystrokes in the input file then execute it in a batch. Due to these complexities, in this demo we only integrate partial functions offered by FVS and FIBER, but other functions can be easily added in the future version.

In order to access the services of these legacy programs from the Web application, we need to provide a mechanism to bridge the gap. Object wrapping is a practice that is suitable to achieve this. A wrapper class will hide the inner details of legacy systems but communicate through an abstract programming interface. In this research we create the shell scripts to run these services through a Java Native Interface (JNI) wrapper class. List 3.7 shows the DOS script used to invoke FVS. This batch file accepts two parameters: keyword file name (%1), which is a unique name generated by the system and saved in a temporary directory, and the absolute path of the FVS executable (%2), which is acquired from a Servlet configuration object. The JXPert can pass the parameters and run this DOS script through the wrapper class.

```
rem StdFVS run on DOS.
echo %1.key > %1.rsp
echo %1.tre >> %1.rsp
echo %1.out >> %1.rsp
```

```
echo %1.tr1 >> %1.rsp
echo %1.sum >> %1.rsp
echo %1.chp >> %1.rsp
%2ni.exe < %1.rsp
del %1.rsp
del %1.tre
del %1.tr1
del %1.sum
del %1.chp
```

List 3.7 FVS DOS Script

3.5.2 Intelligent Information Module (IIM)

The IIM is the core part of an intelligent information system. It provides the intelligence necessary to interpret user interactions. The IIM contains three components: the controller, the knowledge base and the inference engine.

The controller is responsible for locating and activating the rule engine, supplying input information and interpreting the rule engine output results. In this Web application, a central control servlet and its auxiliary handler classes achieve the controller's functionality.

The knowledge base is the repository of crucial information about each legacy application and provides information necessary to process the user's queries. In this research, a Prolog program supplies this purpose. We can create these rules through our SKML rule authoring tool and then translate them to a valid Prolog program.

The inference engine is used to interpret the rules and draw the conclusion based on information supplied by the controller. In this research, we integrate the AMZI! Personal Logic Server [25] as the Prolog inference engine. Amzi! offers plug-in, rule-based services for C, C++, Java, Web Servers, Delphi, Visual Basic, PowerBuilder, Access, Excel and many other tools and the Personal License is free. The integration is achieved through the Logic Server Java API. The result is a manageable and well-behaved interface that makes it possible to utilize

rule-based programming everywhere. Every time when the controller intercepts a user request, it spawns a new thread, creates a new rule engine instance, initializes the knowledge base and loads the user input information.

These three components work together to interpret the user request, process it and direct the user to the following page.

3.5.3 Data Model

In integration applications, data model is an important concept. By defining a data model, the system becomes general purpose by supporting wide varieties of data formats. The implementation solely depends on the data model instead of any particular application domain. Again, in this research, XML is used as the underlying data model because of the proven flexibility and efficiency.

The data model has two purposes: one is working as an intermediate data repository for later translation to the application specific input file; another is working as the user interface screen definition. Currently we have three XML templates for FVS, FIBER and user queries respectively and everyone is associated with two XSL templates. One is for translating the data model to an HTML page for presentation and the other one is for translating the model to the legacy application's input file. When the user's request is received, the central Servlet controller and helper handler classes will construct the XML data model in memory and load the XSL script for conversion.

By using XML as the data model, we can plug in more FDSS support by introducing new XML and XSL templates and registering them in a registry file. To expand services for the current applications, we just need to add more entries to the XML templates.

List 3.8 shows the XML template quoted from the FVS application. The ‘key’ element represents a keyword required in the input file. One keyword can have a number of parameters and the default values are given through their ‘value’ attributes. Since FVS supports dozens of keywords, we can easily expand this application by adding more keywords in the future.

```
<fvs action="fvs.out" target='jxp'>
  <key name="InvYear" required="yes">
    <description>Common starting inventory year.</description>
    <param name="Starting year: " value="2002"/>
  </key>
  <key name="NumCycle" required="yes">
    <description>Common length cycle.</description>
    <param name="Length of cycle: " value="10"/>
  </key>
  <key name="NoTrees" required="no">
    <description>Specifies that no projectable tree records will be
      used as input.</description>
  </key>
  <key name="NoTriple" required="no">
    <description>Change the number of times a tree record will be
      tripled during the first two cycles.</description>
  </key>
  .
  .
  .
</fvs>
```

List 3.8 FVS XML Keyword File

There are two XSL files associated with this XML template called fvs.in.xsl and fvs.out.xsl. fvs.in.xsl translates this XML file into proper HTML format for presentation. After the user submits the form, the XML object is populated, and then fvs.out.xsl can be used to convert it into the FVS proprietary input format.

3.5.4 Interactions Among the Components

After introducing the individual components in this IIS, we will discuss the overall architecture and interaction among those components.

This is a standard four-tier architecture. The client tier only needs a standard Web browser. The presentation tier contains a central controller and helper classes. They receive all user communications, intercept action codes from the request, look up the registry to create the proper handler objects and redirect the request to them. The business logic layer invokes the Prolog inference engine from the IIM, constructs and converts the XML data models, serves the legacy applications through the Web interfaces, activates and executes the FDDS's and presents the simulation results. The back end legacy application tier works with this Web application through the JNI wrapper classes.

CHAPTER 4

DEMONSTRATION

In this chapter, we demonstrate how JXpert can be used.

4.1 SKML Authoring Tool

The JXpert authoring tool is customized by a configuration file, `jxpert.cfg`, which defines the general user preferences such as XML parser, system ID, debug level, default working directories and window appearance.

Figure 4.1 shows the first screen of the JXpert authoring tool.

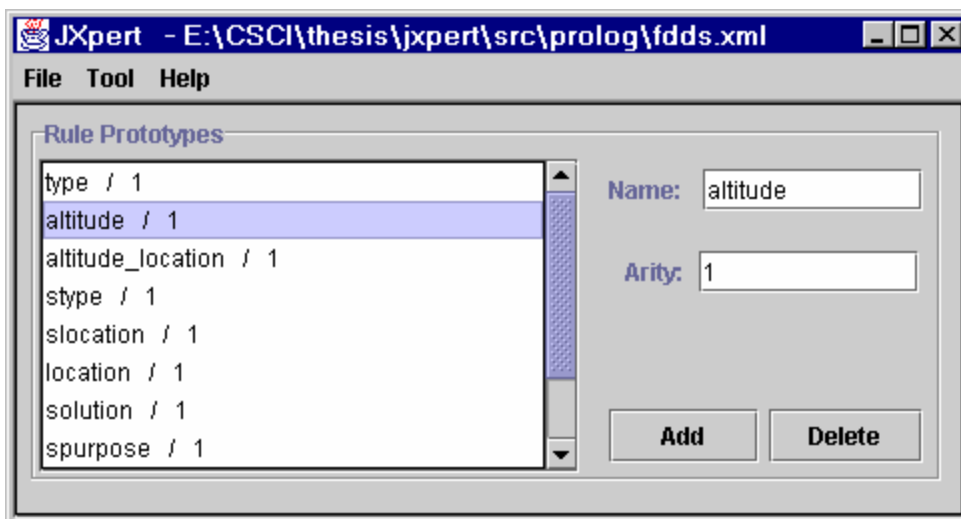


Figure 4.1 Rule prototype screen shot

The 'File' menu contains the common file operations such as 'create new file', 'open a file', 'save a file' and 'exit the application'. The 'Tool' menu consists of three operation categories: rule management, rule navigation and auxiliary functions. The rule management

includes rule creation and deletion functions; the rule navigation menu items allow the user to browse rules sequentially or jump to a particular one and the auxiliary operations include the functions to validate the current rule set and translate it to a Prolog program.

The first step of creating a rule set is to define the rule prototypes as is clear from Figure 4.1. The user can create a new rule type by entering the name and number of parameters; then a unique ID will be automatically assigned to this rule prototype.

A rule can be implemented by choosing the ‘Append New Rule’ menu item from the ‘Tool’ menu as displayed in Figure 4.2. The user should first choose a rule type from the available rule prototypes in the ‘Rule’ drop-down box, then the argument names and types need to be declared and the initial values can be assigned. The ‘Arguments’ section has three columns: type, name and value. JXpert supports four data types, which are ‘integer’, ‘float’, ‘string’ and ‘boolean’. A variable can be bonded to any data type.

A rule can contain any number of condition sections as needed and all are in an ‘and’ relationship. To add a new condition, the user clicks the ‘New Condition’ button, and then selects the condition type either as an expression or a ‘RuleRef ‘ (a call to other rules). An expression uses a free form to express any binary and unary operations. Figure 4.2 shows the discount example discussed in List 3.2 and Figure 4.3 shows the sibling example discussed in List 3.3. In Figure 4.3, the conditions are mixed with expressions and function calls.

After the user is satisfied with the rule set, the user should do a validation by choosing the ‘Validate’ menu item from the ‘Tool’ menu. If there is any error, the message will be displayed in a pop up window, otherwise the user can either save the work into a file or convert it to a Prolog program by choosing the ‘Convert to Prolog’ menu item from ‘Tool’ menu.

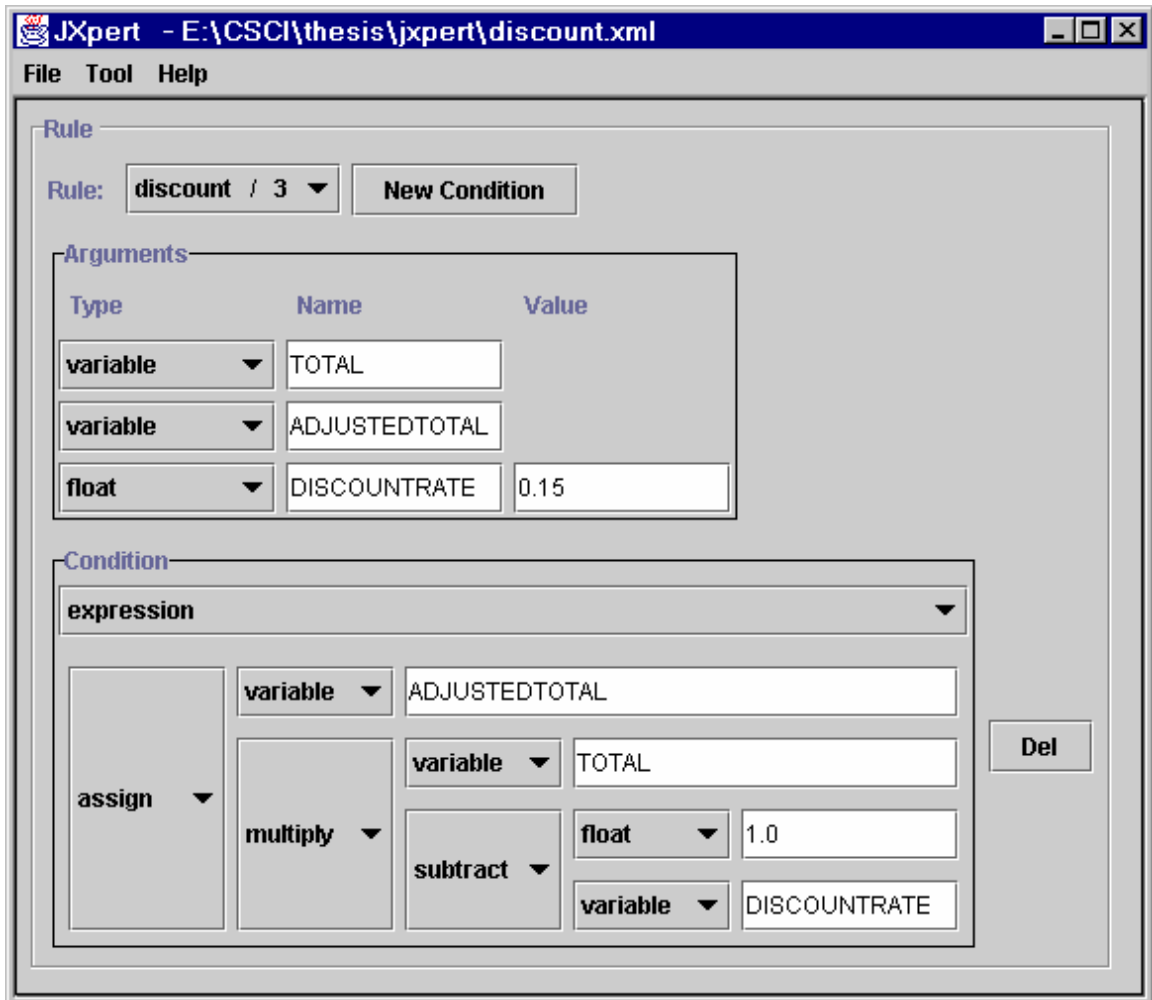


Figure 4.2 Rule screen shot one

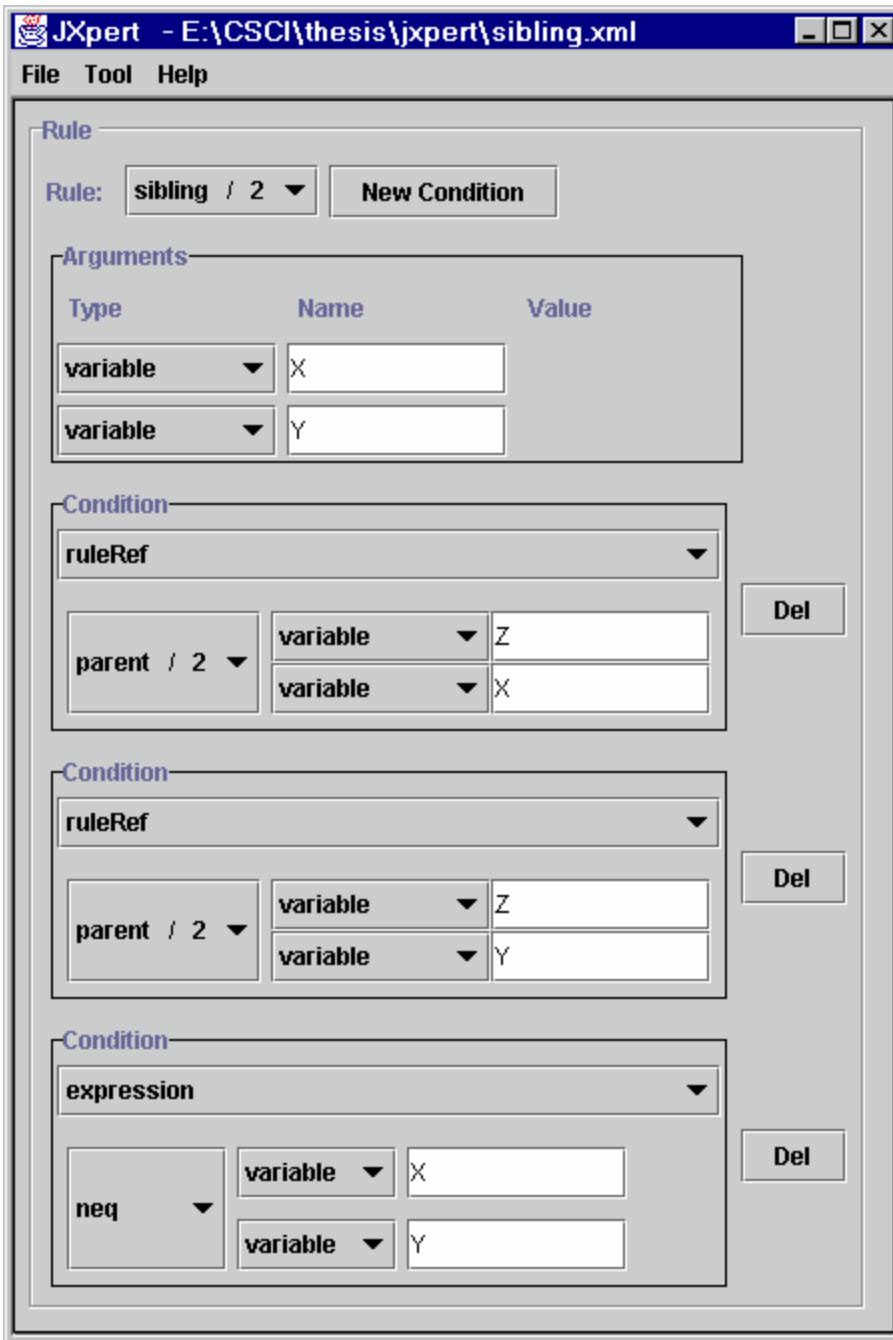


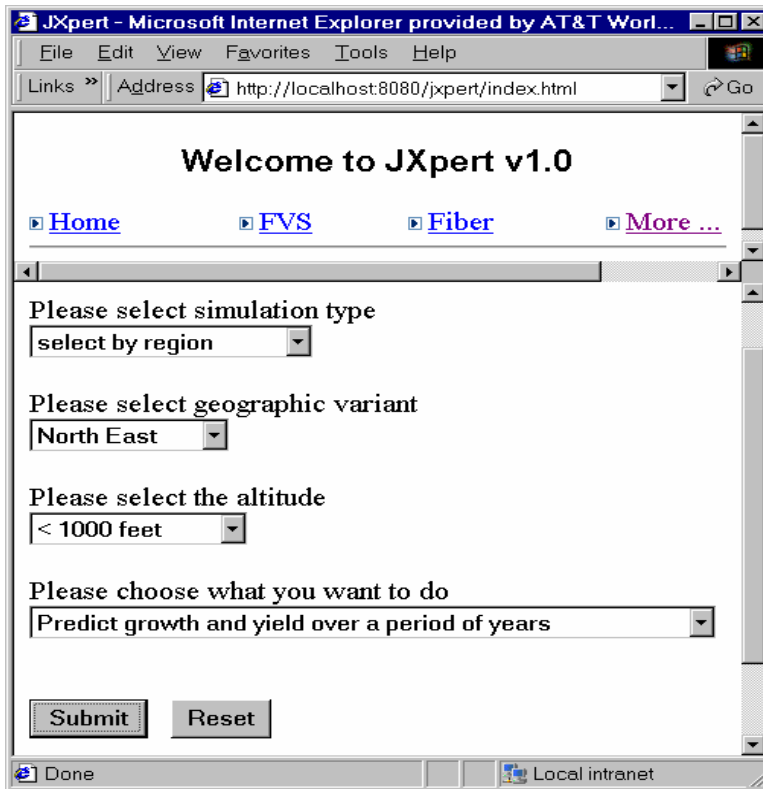
Figure 4.3 Rule screen shot two

4.2 Intelligent Information System

The Web application to integrate the FDSS's is shown from Figure 4.4 to Figure 4.7.

This is an easy four step process. Figure 4.4 shows the first screen of this application.

Currently, JXpert uses four criteria to identify the correct FDSS. They are simulation type, geographical region, altitude and purpose. Now there are only two forestry decision support systems integrated, namely FVS and FIBER, so these four criteria are enough to tell them apart. In the future, when more criteria and details are needed, we only need to update the rules from the JXpert rule authoring tool and load the compiled Prolog program to this Web application.



The screenshot shows a web browser window titled "JXpert - Microsoft Internet Explorer provided by AT&T Worl...". The address bar shows "http://localhost:8080/jxpert/index.html". The page content includes a navigation menu with links for Home, FVS, Fiber, and More ... Below the menu, there are four dropdown menus for user input: "Please select simulation type" (set to "select by region"), "Please select geographic variant" (set to "North East"), "Please select the altitude" (set to "< 1000 feet"), and "Please choose what you want to do" (set to "Predict growth and yield over a period of years"). At the bottom of the form are "Submit" and "Reset" buttons. The status bar at the bottom indicates "Done" and "Local intranet".

Figure 4.4 The user query Web page

For example, if the user wants to predict the growth and yield of a stand over a period of years in the northeast area and the altitude is less than 1000 feet, he/she fills the form and

submits. After consulting the knowledge base, the possible solutions are identified and displayed in the page illustrated in Figure 4.5.

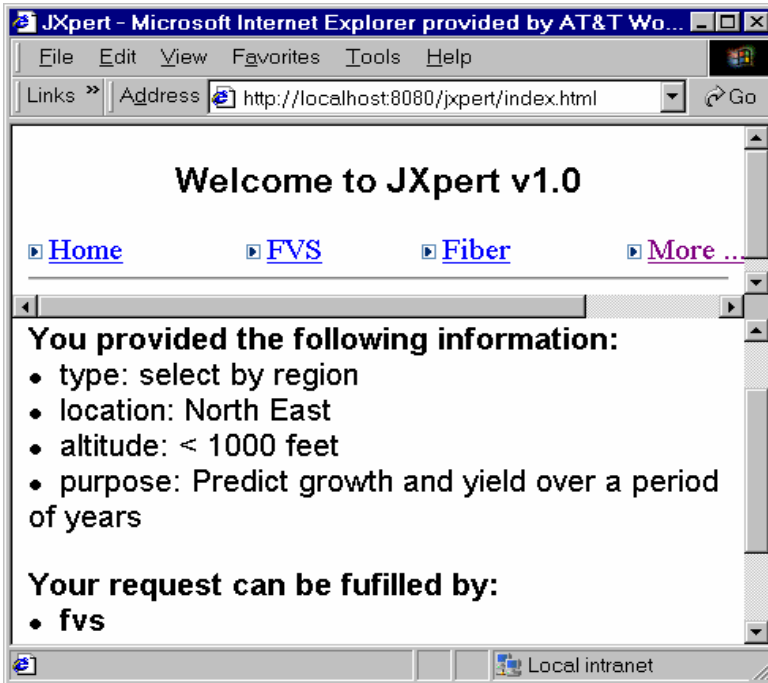


Figure 4.5 The user query answer Web page

In this case FVS can fulfill the user's requirement; then the user clicks the 'FVS' link to enter the FVS Web page (Figure 4.6). FVS needs the user to supply a bunch of required and optional keywords and parameters. Figure 4.7 shows the simulation result and the result is also available for downloading.

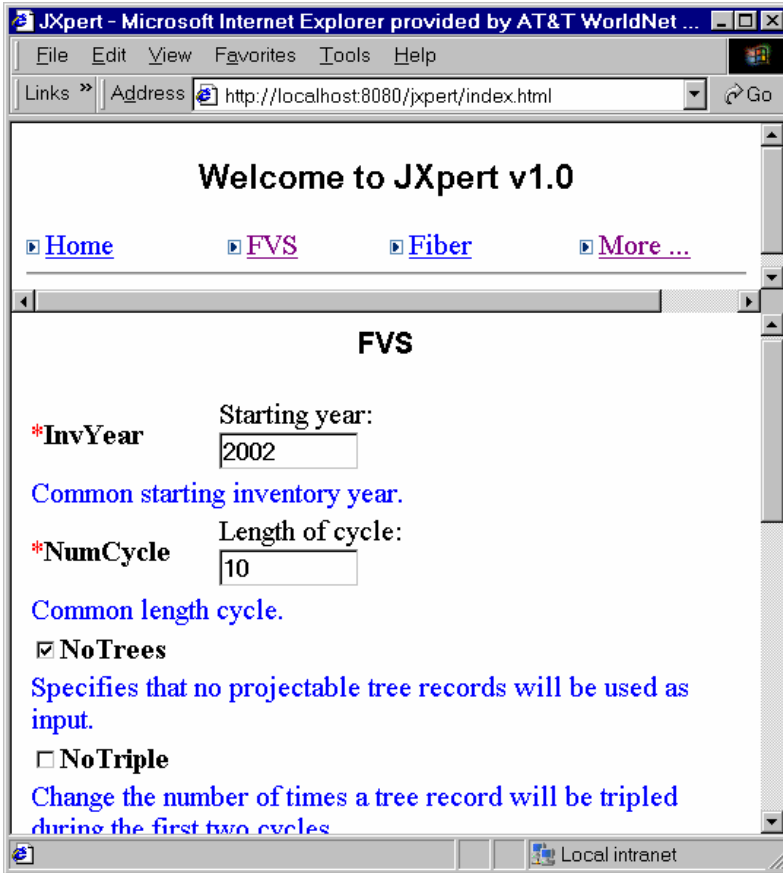


Figure 4.6 The FVS Web page

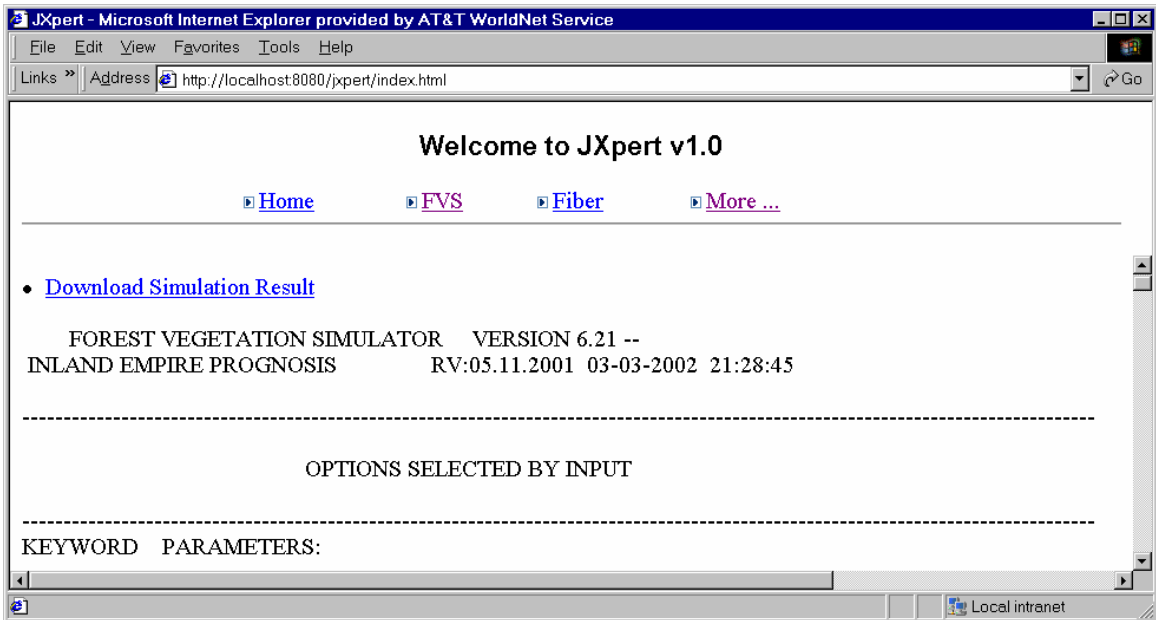


Figure 4.7 The FVS simulation result Web page

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this research, we introduce JXpert, an intelligent information system based on Java and XML technology. First, we propose a new rule markup language, the Simple Knowledge Markup Language (SKML), then we develop a visual tool to help the user create the SKML rule set and translate it to a Prolog program. The SKML and inference engine have been successfully incorporated into a Web application. This system together with other Forestry Decision Support Systems, such as FVS and FIBER, helps the user identify the solution efficiently and enables the legacy applications to be executed through the Web interface.

The key feature of this system is that it introduces SKML as the rule language. SKML borrowed some concepts from the logic programming paradigm to make it suitable for knowledge representation. And as an XML application, it inherits the merits of platform independence, open architecture and flexibility.

Extensibility is another feature that has been considered while designing the system. The extensibility is provided in two ways. First, we can easily plug in new legacy applications by registering them as new FDSS services. Second, we can migrate more functions from legacy systems to the Web by extending the corresponding XML and XSL templates.

However, there are still some issues that need to be addressed in the future. First, the intelligence has been provided by adding reaction rules to the knowledge base manually by the

user through the SKML authoring tool. Future work could eliminate this step by generating description files of underlying legacy applications and constructing the knowledge base from these files automatically. Second, we need to integrate more forestry decision support systems, such as SILVAH and NED to make this a complete system. Third, to improve usability, we need to include more functions offered by FDSS's. Fourth, we may introduce an Enterprise Java Bean (EJB) layer in this Web application to improve performance. Fifth, we need to further refine the SKML grammar to better recognize its knowledge representation purpose.

REFERENCES

1. Rauscher, H., Ecosystem Management Decision Support for Public Forests: A Review. *Forest Ecology and Management* 114, pp. 173-197, 1999
2. von Luck, K., Nebel, B., Schneider, H.-J., Aspects of Knowledge Base Management Systems, *Wissensrepräsentation in Expertensystemen*, ed, Rahmstorf, G., Springer-Verlag, Berlin, pp 146-157, 1988.
3. Qadeer, S., Algorithms and Methodology for Scalable Model Checking, Ph. D. Thesis, University of California at Berkeley, CA, 1999,
<http://www-cad.eecs.berkeley.edu/~tah/Students/qadeer.pdf>
4. Araces, C., Bouma, W., de Rijke, M., Description Logics and Feature Interaction, *Proceedings of the International Workshop on Description Logics - DL-99*, pp 28-32, 1999
5. Baader, F., Sattler, U, Knowledge Representation in Process Engineering, *Proceedings of the International Workshop on Description Logics - DL-96*, Cambridge, MA, pp 74-78, 1996
6. Brokat Advisor, Brokat Company, 2001, <http://www.brokat.com/advisor/>
7. ILog JRules, ILog Inc., 2001, <http://www.ilog.com/>
8. SilverStream ePortal Rules Engine, SilverStream Software Inc., 2001,
<http://www.silverstream.com/>
9. CIA Server, Haley Enterprise, Inc, 2001, <http://www.haley.com/>
10. AlphaWorks CommonRules, IBM, 2001, <http://www.alpha works.ibm.com/>
11. Boley, H., Said, T., Gerd, W., Design Rational of RuleML: A Markup Language for Semantic Web Rules, *International Semantic Web Working Symposium*, CA, 2001,
<http://www.semanticweb.org/SWWS/program/full/paper20.pdf>
12. Groszof, B., Labrou, Y., An Approach to using XML and a Rule-based Content Language with an Agent Communication Language, *Proceedings of the International Joint*

- Conference of Artificial Intelligence (IJCAL)-99 Workshop on Agent Communication Languages*, Stockholm, Sweden, 1999
13. Boley, H., Markup Languages for Functional-Logic Programming, *Proc. 9th Workshop on Functional and Logic Programming*, Benicassim, Spain, 2000, <http://www.dfki-uni-kl.de/~boley/flml.pdf>
 14. Calvanese, D., Giacomo, G., and Lenzerini, M., Representing and Reasoning on XML Documents: A Description Logic Approach, *Journal of Logic and Computation*, Vol. 9, No. 3, pp. 295-318, 1999
 15. Somasekar, S., An Intelligent Information System for Integration of Forest Decision Support Systems. Master's Thesis, The University of Georgia, Athens, GA, 1999
 16. Potter, W.D., Somasekar, S., Kommineni, R., and Rauscher, H.,M., NED-IIS: An Intelligent Information System for Forest Ecosystem Management, *Proceedings of American Association of Artificial Intelligence (AAAI) workshop on Intelligent Information Systems*, Orlando, FL, July, 1999, http://www.srs.fs.fed.us/pubs/ja/ja_potter001.pdf
 17. Arnold, K., Gosling, J., *The Java Programming Language Second Edition.*, Addison Wesley Longman, Inc., 1997
 18. Adatia, R., *Professional EJB*, Wrox Press Ltd., Birmingham, UK, 2001
 19. Brown, S., *Profession JSP 2nd Edition*, Wrox Press Ltd., Birmingham, UK, 2001
 20. XML, World Wide Web Consortium, 2002, <http://www.w3c.org/XML/>
 21. Konigsberger, H., *Prolog From Beginning*, McGraw-Hill Publisher, Columbus, OH, 1990
 22. Alur, D., Malks, D., and Crupi, J., *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall PTR, June, 2001
 23. Teck, R., Moeur, M., and Eav, B., Forecasting Ecosystems with the Forest Vegetation Simulator, *Journal of Forestry*, 94(12) pp. 7-10, 1996
 24. Solomon, D.S., Heraman, D.A., and Leak, W.B., *FIBER 3.0: An Ecological Growth Model for Northeastern Forest Types*, USDA, General Technical Report NE-204, 1995
 25. AMZI! Logic Server, Amzi, Inc., <http://www.amzi.com/>, 2002

APPENDIX A

THE SKML ANNOTATED DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- SKML (Simple Knowledge Markup Language) annotated DTD
-->
<!--
  A "variable" has simply a name. A "variable" can be bonded to a value
  in an "initBinding" and can be referenced in a "condition" part.
-->
<!ELEMENT variable EMPTY>
<!ATTLIST variable
  name NMTOKEN #REQUIRED>
<!--
  A "constant" has a required type (enumerated) and a value (cdata).
  The DTD provides the common types that exist in programming languages.
-->
<!ELEMENT constant EMPTY>
<!ATTLIST constant
  type (string | boolean | integer | float | null) #REQUIRED
  value CDATA #REQUIRED>
<!--
  Defines the allowed argument types for a "ruleRef"
-->
<!ENTITY % parameter "variable | constant">
<!--
  An "expression" can be an arithmetic or logical expression.
-->
<!ENTITY % expression "%parameter; | unaryExpr | binaryExpr">
<!--
  Defines the logical operators (enumerated)
-->
<!ENTITY % relationalOperator "eq | neq | lt | lte | gt | gte">
<!--
  Defines the arithmetic operators (enumerated)
-->
<!ENTITY % arithmeticOperator "add | subtract | multiply | divide |
assign">
<!--
  Defines the unary expressions. A unary expression acts on a single
  argument. The operator is specified using an enumerated type.
-->
<!ELEMENT unaryExpr (%expression;)>
<!ATTLIST unaryExpr
  operator (plus | minus | not) #REQUIRED>
<!--
```

Defines the binary expressions. A binary expression acts on two arguments. As for unary operators, binary operators are specified using enumerated symbolic names, which can be relational or arithmetic operators.

```
-->
<!ELEMENT binaryExpr ((%expression;), (%expression;))>
<!ATTLIST binaryExpr
    operator (%relationalOperator; | %arithmeticOperator;) #REQUIRED >
<!--
```

A "rules" is composed of a list of "rulePrototypes" and "rules". A "rules" has a name(optional). The "rules" is the root element of an XML document.

```
-->
<!ELEMENT rules (rulePrototype*,rule*)>
<!ATTLIST rules
    name NMTOKEN #IMPLIED>
```

<!--
A "rulePrototype" has an id (required), name (required) and arity (required). This is the interface of "rule" element.

```
-->
<!ELEMENT rulePrototype EMPTY>
<!ATTLIST rulePrototype
    id ID #REQUIRED
    name NMTOKEN #REQUIRED
    arity CDATA #REQUIRED>
```

<!--
"rule" has an id (reference to a "rulePrototype", required) and a name (optional). A "rule" has a "formalArgs", an "initBinding" and some "condition" elements. All the enclosing conditions are in a conjunctive relationship.

```
-->
<!ELEMENT rule (formalArgs, initBinding, condition*)>
<!ATTLIST rule
    id IDREF #REQUIRED
    name NMTOKEN #IMPLIED>
```

<!--
"formalArgs" defines the rule arguments. It at least has one argument.

```
-->
<!ELEMENT formalArgs (variable+)>
```

<!--
Bind the arguments to initial values.

```
-->
<!ELEMENT initBinding (bind*)>
```

<!--
Bind a value to a variable.

```
-->
<!ELEMENT bind (variable, constant)>
```

<!--
"ruleRef" calls a pre-defined rule by supplying the parameters. A "ruleRef" has an id (reference to a "rulePrototype", required) and a name (optional)

```
-->
<!ELEMENT ruleRef (%parameter;)+>
<!ATTLIST ruleRef
    id IDREF #REQUIRED
    name NMTOKEN #IMPLIED>
<!--
    The body of a "condition" is composed of "ruleRef" or "expression"
    elements.
-->
<!ELEMENT condition (ruleRef | %expression;)>
```

APPENDIX B

THE JXPert SKML RULE SET

The following SKML document is used in this JXpert web application to intelligently integrate various forest decision support systems.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rules SYSTEM "skml.dtd">
<rules>
  <rulePrototype id="I302437" name="type" arity="1" />
  <rulePrototype id="I797497" name="altitude" arity="1" />
  <rulePrototype id="I177766" name="altitude_location" arity="1" />
  <rulePrototype id="I606062" name="stype" arity="1" />
  <rulePrototype id="I989729" name="slocation" arity="1" />
  <rulePrototype id="I434654" name="location" arity="1" />
  <rulePrototype id="I849170" name="solution" arity="1" />
  <rulePrototype id="I335078" name="spurpose" arity="1" />
  <rulePrototype id="I275715" name="purpose" arity="1" />
  <rule id="I606062" name="stype">
    <formalArgs>
      <variable name="DDS" />
    </formalArgs>
    <initBinding>
      <bind>
        <variable name="DDS" />
        <constant type="string" value="fvs" />
      </bind>
    </initBinding>
    <condition>
      <ruleRef id="I302437" name="type">
        <constant type="string" value="region" />
      </ruleRef>
    </condition>
  </rule>
  <rule id="I606062" name="stype">
    <formalArgs>
      <variable name="DDS" />
    </formalArgs>
    <initBinding>
      <bind>
        <variable name="DDS" />
        <constant type="string" value="fvs" />
      </bind>
    </initBinding>
    <condition>
```

```

        <ruleRef id="I302437" name="type">
            <constant type="string" value="species" />
        </ruleRef>
    </condition>
</rule>
<rule id="I606062" name="stype">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fiber" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I302437" name="type">
            <constant type="string" value="species" />
        </ruleRef>
    </condition>
</rule>
<rule id="I335078" name="spurpose">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fvs" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I275715" name="purpose">
            <constant type="string" value="purp1" />
        </ruleRef>
    </condition>
</rule>
<rule id="I335078" name="spurpose">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fvs" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I275715" name="purpose">
            <constant type="string" value="purp2" />
        </ruleRef>
    </condition>
</rule>
<rule id="I335078" name="spurpose">

```

```

<formalArgs>
  <variable name="DDS" />
</formalArgs>
<initBinding>
  <bind>
    <variable name="DDS" />
    <constant type="string" value="fvs" />
  </bind>
</initBinding>
<condition>
  <ruleRef id="I275715" name="purpose">
    <constant type="string" value="purp3" />
  </ruleRef>
</condition>
</rule>
<rule id="I335078" name="spurpose">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fvs" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I275715" name="purpose">
      <constant type="string" value="purp4" />
    </ruleRef>
  </condition>
</rule>
<rule id="I335078" name="spurpose">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fvs" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I275715" name="purpose">
      <constant type="string" value="purp5" />
    </ruleRef>
  </condition>
</rule>
<rule id="I335078" name="spurpose">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />

```

```

        <constant type="string" value="fvs" />
    </bind>
</initBinding>
<condition>
    <ruleRef id="I275715" name="purpose">
        <constant type="string" value="purp7" />
    </ruleRef>
</condition>
</rule>
<rule id="I335078" name="spurpose">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fvs" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I275715" name="purpose">
            <constant type="string" value="purp8" />
        </ruleRef>
    </condition>
</rule>
<rule id="I335078" name="spurpose">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fvs" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I275715" name="purpose">
            <constant type="string" value="purp9" />
        </ruleRef>
    </condition>
</rule>
<rule id="I335078" name="spurpose">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fvs" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I275715" name="purpose">
            <constant type="string" value="purp10" />
        </ruleRef>
    </condition>
</rule>

```

```

        </ruleRef>
    </condition>
</rule>
<rule id="I335078" name="spurpose">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fiber" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I275715" name="purpose">
            <constant type="string" value="purp5" />
        </ruleRef>
    </condition>
</rule>
<rule id="I335078" name="spurpose">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fiber" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I275715" name="purpose">
            <constant type="string" value="purp6" />
        </ruleRef>
    </condition>
</rule>
<rule id="I989729" name="slocation">
    <formalArgs>
        <variable name="DDS" />
    </formalArgs>
    <initBinding>
        <bind>
            <variable name="DDS" />
            <constant type="string" value="fvs" />
        </bind>
    </initBinding>
    <condition>
        <ruleRef id="I434654" name="location">
            <constant type="string" value="ne" />
        </ruleRef>
    </condition>
</rule>
<rule id="I989729" name="slocation">
    <formalArgs>
        <variable name="DDS" />

```

```

</formalArgs>
<initBinding>
  <bind>
    <variable name="DDS" />
    <constant type="string" value="fvs" />
  </bind>
</initBinding>
<condition>
  <ruleRef id="I434654" name="location">
    <constant type="string" value="sn" />
  </ruleRef>
</condition>
</rule>
<rule id="I989729" name="slocation">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fvs" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I434654" name="location">
      <constant type="string" value="se" />
    </ruleRef>
  </condition>
</rule>
<rule id="I989729" name="slocation">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fvs" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I434654" name="location">
      <constant type="string" value="cs" />
    </ruleRef>
  </condition>
</rule>
<rule id="I989729" name="slocation">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fvs" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I434654" name="location">
      <constant type="string" value="cs" />
    </ruleRef>
  </condition>
</rule>

```

```

</initBinding>
<condition>
  <ruleRef id="I434654" name="location">
    <constant type="string" value="ls" />
  </ruleRef>
</condition>
</rule>
<rule id="I989729" name="slocation">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fiber" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I434654" name="location">
      <constant type="string" value="ne" />
    </ruleRef>
  </condition>
</rule>
<rule id="I989729" name="slocation">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fiber" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I434654" name="location">
      <constant type="string" value="cs" />
    </ruleRef>
  </condition>
</rule>
<rule id="I177766" name="altitude_location">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fvs" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I989729" name="slocation">
      <constant type="string" value="fvs" />
    </ruleRef>
  </condition>

```

```

<condition>
  <ruleRef id="I797497" name="altitude">
    <variable name="ATL" />
  </ruleRef>
</condition>
<condition>
  <binaryExpr operator="lt">
    <variable name="ATL" />
    <constant type="integer" value="4000" />
  </binaryExpr>
</condition>
</rule>
<rule id="I177766" name="altitude_location">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fiber" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I434654" name="location">
      <constant type="string" value="ne" />
    </ruleRef>
  </condition>
  <condition>
    <ruleRef id="I797497" name="altitude">
      <variable name="ATL" />
    </ruleRef>
  </condition>
  <condition>
    <binaryExpr operator="lt">
      <variable name="ATL" />
      <constant type="integer" value="2000" />
    </binaryExpr>
  </condition>
</rule>
<rule id="I177766" name="altitude_location">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding>
    <bind>
      <variable name="DDS" />
      <constant type="string" value="fiber" />
    </bind>
  </initBinding>
  <condition>
    <ruleRef id="I434654" name="location">
      <constant type="string" value="cs" />
    </ruleRef>
  </condition>

```

```

<condition>
  <ruleRef id="I797497" name="altitude">
    <variable name="ATL" />
  </ruleRef>
</condition>
<condition>
  <binaryExpr operator="gt">
    <variable name="ATL" />
    <constant type="integer" value="2000" />
  </binaryExpr>
</condition>
<condition>
  <binaryExpr operator="lt">
    <variable name="ATL" />
    <constant type="integer" value="4000" />
  </binaryExpr>
</condition>
</rule>
<rule id="I849170" name="solution">
  <formalArgs>
    <variable name="DDS" />
  </formalArgs>
  <initBinding />
  <condition>
    <ruleRef id="I606062" name="stype">
      <variable name="DDS" />
    </ruleRef>
  </condition>
  <condition>
    <ruleRef id="I335078" name="spurpose">
      <variable name="DDS" />
    </ruleRef>
  </condition>
  <condition>
    <ruleRef id="I177766" name="altitude_location">
      <variable name="DDS" />
    </ruleRef>
  </condition>
</rule>
</rules>
<!--/* *****
* This program is generated by JXpert *
* author Xin Zhang *
* copyright 2002 *
*****/
-->

```

APPENDIX C

THE JXPert PROLOG RULE SET

This is the Prolog program converted from the SKML document listed in Appendix B.

```
/* *****  
 * This program is generated by JXPert *  
 * author Xin Zhang *  
 * copyright 2002 *  
 *****/  
% type/1.  
% altitude/1.  
% altitude_location/1.  
% stype/1.  
% slocation/1.  
% location/1.  
% solution/1.  
% spurpose/1.  
% purpose/1.  
stype('fvs') :- type('region').  
stype('fvs') :- type('species').  
stype('fiber') :- type('species').  
spurpose('fvs') :- purpose('purp1').  
spurpose('fvs') :- purpose('purp2').  
spurpose('fvs') :- purpose('purp3').  
spurpose('fvs') :- purpose('purp4').  
spurpose('fvs') :- purpose('purp5').  
spurpose('fvs') :- purpose('purp7').  
spurpose('fvs') :- purpose('purp8').  
spurpose('fvs') :- purpose('purp9').  
spurpose('fvs') :- purpose('purp10').  
spurpose('fiber') :- purpose('purp5').  
spurpose('fiber') :- purpose('purp6').  
slocation('fvs') :- location('ne').  
slocation('fvs') :- location('sn').  
slocation('fvs') :- location('se').  
slocation('fvs') :- location('cs').  
slocation('fvs') :- location('ls').  
slocation('fiber') :- location('ne').  
slocation('fiber') :- location('cs').  
altitude_location('fvs') :- slocation('fvs'), altitude(ATL), (ATL <  
4000).  
altitude_location('fiber') :- location('ne'), altitude(ATL), (ATL <  
2000).  
altitude_location('fiber') :- location('cs'), altitude(ATL), (ATL >  
2000), (ATL < 4000).  
solution(DDS) :- stype(DDS), spurpose(DDS), altitude_location(DDS).
```