

# **Rule-based Process Support for Enterprise Information Portal**

MASTER THESIS

Submitted by:

Henry Kamau Gichahi  
Matriculation Number: 17029

A thesis submitted in partial fulfillment of the requirements for the degree of  
Master of Science in Information and Media Technologies

Technische Universität Hamburg-Harburg  
GERMANY

Supervised by:

Prof. Dr. J.W Schmidt  
Prof. Dr. F. Matthes

February 2003



# Table of Contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>1</b>
1.1	Objective and Goals .....	1
1.2	Related Work.....	1
1.3	Structure of the Work.....	2
<b>2</b>	<b>RULE MANAGEMENT .....</b>	<b>3</b>
2.1	Business Rules.....	3
2.2	Business Rule Management.....	4
2.3	Business Rule Management Versus Workflow Management and Business Process Management.....	4
2.4	The Need for Rule Management.....	5
2.5	Rule Management Environment.....	6
2.5.1	Rule Repository.....	6
2.5.2	The Business Terminology Glossary .....	6
2.5.3	Rule Traceability .....	6
2.6	Key Concepts of Rule Management .....	7
2.6.1	Specifying Business Rules .....	7
2.6.2	Executing Business Rules .....	9
2.6.3	Managing Business Rules .....	14
<b>3</b>	<b>RULE LANGUAGE FOUNDATIONS .....</b>	<b>15</b>
3.1	Languages and Language Classes.....	15
3.1.1	Formal Languages, Grammars and Automata.....	15
3.1.2	Language Classes .....	17
3.1.3	Language Complexity Analysis .....	19
3.2	Rule Languages .....	20
3.2.1	Overview .....	21
3.2.2	Production Rules .....	21
3.2.3	Prolog System Rules: Definite Clauses.....	21
3.2.4	Event-Condition-Action (ECA) Rules .....	22
3.2.5	Independent Event-Condition-Action (IECA) Rules .....	22
3.2.6	Case Study: IECA .....	22
<b>4</b>	<b>ANALYSIS AND DESIGN OF RULE MANAGEMENT SYSTEMS .....</b>	<b>25</b>
4.1	Requirement Analysis of Rule Management Systems .....	25
4.1.1	Performance Considerations and Platform Support.....	26

4.1.2	Rule Engine-Specific Requirements .....	27
4.1.3	Rules Management GUI-Specific Requirements .....	27
4.1.4	Rules Management Integration Specifications.....	27
<b>4.2</b>	<b>Rule Management System Architecture .....</b>	<b>28</b>
4.2.1	Rule Engine and API for Integration Layer .....	28
4.2.2	Rules Editor Layer .....	29
4.2.3	Rules Management GUI Layer .....	29
<b>4.3</b>	<b>Rule Management System Products and Prototypes.....</b>	<b>29</b>
4.3.1	Overview .....	29
4.3.2	The ILOG Rule Management System.....	30
4.3.3	The Rule Engine Prototype .....	35
<b>5</b>	<b>CASE STUDY: AN IECA RULE MANAGEMENT SYSTEM .....</b>	<b>37</b>
<b>5.1</b>	<b>Rule Engine.....</b>	<b>37</b>
5.1.1	Evaluation Context and Variables.....	37
5.1.2	Rules.....	38
5.1.3	Conditions .....	38
5.1.4	Actions .....	39
<b>5.2</b>	<b>Implementation Platform and Implications .....</b>	<b>41</b>
5.2.1	InfoAsset Broker Software as Development Platform.....	41
5.2.2	Implications of using InfoAsset Broker Software as Development Platform.....	41
5.2.3	Rule Management Integration into the WIPS IT1 Portal.....	42
<b>5.3</b>	<b>Applications of Rule Management System in the WIPS Scenario .....</b>	<b>42</b>
5.3.1	The Ship Survey Process.....	42
5.3.2	Support for Process Management .....	43
<b>5.4</b>	<b>Rules Management GUI for WIPS IT 1 Portal .....</b>	<b>44</b>
5.4.1	User Roles and Use Cases.....	45
5.4.2	Life Cycle of Rules and Rule States.....	50
<b>6</b>	<b>SUMMARY AND OUTLOOK .....</b>	<b>53</b>
<b>6.1</b>	<b>Summary .....</b>	<b>53</b>
<b>6.2</b>	<b>Outlook.....</b>	<b>53</b>
	<b>APPENDIX A: RULE ENGINE API .....</b>	<b>55</b>
	<b>APPENDIX B: SURVEY ITEMS GENERATION LANGUAGE GRAMMAR.....</b>	<b>59</b>
	<b>BIBLIOGRAPHY .....</b>	<b>61</b>

## Table of Figures

FIGURE 2-1 A TABULAR SHIPPING RATES EXAMPLE .....	9
FIGURE 2-2 ORIGINAL JESS NODES NETWORK FOR RULES 2-4 AND 2-5 .....	12
FIGURE 2-3 REFINED JESS NODES NETWORK WITH REDUNDANCY.....	13
FIGURE 2-4 OPTIMIZED JESS NODES NETWORK.....	13
FIGURE 3-1 THE CHOMSKY LANGUAGE HIERARCHY .....	18
FIGURE 4-1 THE ARCHITECTURE DIAGRAM FOR RULES MANAGEMENT SYSTEM .....	28
FIGURE 4-2 THE GRAPHICAL USER INTERFACE OF THE RULE BUILDER'S PROFILER.....	31
FIGURE 4-3 THE GRAPHICAL USER INTERFACE OF THE RULE BUILDER.....	32
FIGURE 4-4 THE GRAPHICAL USER INTERFACE OF THE RULE BUILDER'S QUERY.....	33
FIGURE 4-5 THE GRAPHICAL USER INTERFACE OF THE RULE EDITOR.....	34
FIGURE 5-1 CLASS DIAGRAM FOR EVALUATION CONTEXT AND VARIABLES .....	38
FIGURE 5-2 CLASS DIAGRAM FOR RULESET .....	38
FIGURE 5-3 CLASS DIAGRAM FOR CONDITION AND RELATED CONCEPTS .....	39
FIGURE 5-4 CLASS DIAGRAM FOR ACTIONS AND RELATED CONCEPTS .....	40
FIGURE 5-5 THE UML CLASS DIAGRAM FOR DATATYPE AND RELATED CONCEPTS.....	42
FIGURE 5-6 EXAMPLE OF USE OF CONDITIONS IN PROCESS MANAGEMENT.....	43
FIGURE 5-7 EDITOR'S USE CASE DIAGRAM .....	45
FIGURE 5-8 THE RULE MANAGEMENT GUI FOR THE RULE EDITORS .....	46
FIGURE 5-9 VERIFIER'S USE CASE DIAGRAM.....	47
FIGURE 5-10 INSPECTOR'S USE CASE DIAGRAM.....	48
FIGURE 5-11 NOTIFICATION SERVICE USE CASE DIAGRAM .....	49
FIGURE 5-12 RULE STATE DIAGRAM .....	50



# 1 Introduction

In recent years it has been realized that business rules constitute an entire body of knowledge that has not been adequately addressed in either the analysis or design phases of system development. Typically, business rules have been buried in program code or in database structures. It turns out, however, that the identification of business rules is important in its own right. Moreover, it is different from the definition of data structure in a data model, and from the definition of processes.

In this document business rules are described from the enterprise perspective. The document does not attempt to develop a methodology, rather to create a model that supports a range of methodological approaches. One way to think of the business rules is as a core component of a business. If it were implemented, the result would be a repository in which business rules could be stored and related to other information about the enterprise, whatever methodology were used for discovering and defining them.

## 1.1 Objective and Goals

The main goal of the WIPS IT 1- *Internet Services for Ship Technology* project is to develop an Online Service Portal for various tasks related to ship classification. This portal will provide process support, knowledge management and integrated services. It is intended to be used by customers, surveyors, external staff and head office employees of a ship classification society. [WIPS01].

The main objective of this research work is to provide a rule-based process support in the WIPS online portal for the ship inspection process.

The goal of this research is to build a prototype rule management system for the WIPS IT 1 Online Portal that will support business process (control flow), support the management of ship inspection rules and generation of survey item lists.

## 1.2 Related Work

Rules management technology has been around in various forms for over 20 years. It was first used widely in the mid-80s when expert systems or “Artificial Intelligence Technology” became briefly popular. Companies were faced with the need to combine domain expertise with the flexibility to write lots of “if x, then y” statements over a wide range of variables without resorting to spaghetti code. Computer scientists and programmers began developing rule languages and the

corresponding engines that could handle the conditions and actions needed to satisfy the wide range of rules that needed to be written.

There have been many efforts to commercialize expert systems. One of the most successful branches was initially referred to as “production rules.” In this branch of rules management technology, real world rules are expressed in “if-then” form with the ability to use functions (or calculations) and data objects as necessary. Since an application could have hundreds of rules, an efficient representation and execution scheme was required to quickly identify the subset of rules that were relevant in a given situation. As those rules fired, the execution scheme would quickly identify the next set of relevant rules. The most successful approach for doing this has proven to be the Rete algorithm [Fo82] developed at Carnegie Mellon in Pittsburgh in the 80s. Named after the Latin word for comb, the Rete algorithm quickly identifies all rules relevant to a given set of conditions and scales very well into rule bases with thousands of rules.

Over the last ten years, rule engines have evolved beyond their expert systems heritage by adopting distributed object technology such as EJB and COM+ and thereby integrating diverse information sources. They have also developed simpler and easier to understand syntaxes for writing rules, gained substantial performance improvements, and added extensions for rule sequencing, branching, maintenance, and function optimization.

### **1.3 Structure of the Work**

This master thesis consists of three main parts: an introduction to rule management systems, investigation of rule management systems, and development of a prototype rule management system.

The first part of this document (Chapters 2-3) describes business rules technology in general. In chapter 2, Business rules are defined. The key concepts and importance of rule management is also defined and discussed. This is followed by a study of rule languages that are used to define business rules.

The second part (Chapters 4) investigates the existing rule management systems and environment and to obtain knowledge about rule management systems including the requirements and architecture of rule management systems.

The last part (Chapters 5) consists of analysis, design and implementation of a prototype rule management system for WIPS IT 1 online portal.

Finally, a summary and future work of this project is presented in chapter 6.

## 2 Rule Management

In this chapter, business rules are defined as well as need to manage them. The rule management environment will be described, followed by the key concepts and importance of rule management.

### 2.1 Business Rules

The term, “**business rules**” has with different meanings for both business and information system (I/S) professionals. Within I/S, “business rules” has a different meaning for business and I/S professionals. Within I/S, “business rules” can have different connotations depending on whether the perspective is “data-oriented”, “object-oriented” or “expert system-oriented”.

#### Definition

From the information system perspective,

*“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 [BRG02].

From the business perspective,

*“A business rule is a directive, which is intended to influence or guide business behavior, in support of business policy that is formulated in response to an opportunity or threat.”* Business Rules Group [BRG02].

To summarize, a business rule is independent of the modeling paradigm or technical platform and is defined and owned by business people. Business rules may be found in policy manuals, regulatory mandates, programming code running corporate or departmental systems or may simply be in the minds of employees who have learned a “best practices” approach to solving problems through practical experience. The biggest challenge is simply tracking them down, and organizing a more effective management approach. In every case there is need for Business Rules management.

Business Rules represent core business policies. They literally control or influence business behavior. They indicate what is possible or desirable in running business and what is not. Business Rules play a major role especially when establishing a relationship between business people and IT. Automating business processes in this way requires support for operational transactions,

workflow management, application integration and sophisticated business decision management. In many business processes the highest cost and least automated steps are those that rely on business decisions to determine an appropriate action. In a process to manage warranty claims, for example, the decision as to what is and what is not covered by a warranty depends on what product is being discussed, when and where it was bought and what kind of problem required it to be fixed. Getting these types of decisions wrong has a tremendous impact on customer satisfaction and bottom line profitability. Yet these types of decisions may be overlooked when automating large-scale enterprise business processes. For instance, organizations may place guidelines in printed policy manuals, relying on employees to read, remember, and apply the proper policies to different business situations.

## **2.2 Business Rule Management**

The biggest challenge of business rules is tracking them down, and organizing a more effective management approach. In every case there is need for business rules management. Business rules management comprises the definition, storage, and application of the many rules used in business operations to provide organizations with greater automation, more responsiveness to change and less expensive distribution and maintenance of their business guidelines.

Business rules governing operational processes may be complex, with many interrelating considerations. They may be simple but extremely numerous, covering a wide variety of different situations. And most difficult of all for organizations to manage, they may change frequently based on internal policy decisions and external competitive or regulatory factors. Rules management offers a solution to meet the requirements of changing business rules.

## **2.3 Business Rule Management Versus Workflow Management and Business Process Management**

### **Business Process Management**

Business Process Management (BPM) is concerned with the overall tasks used to drive a complete process from start to finish. This may include assignment and movement of physical resources, information gathering and processing, online and offline communications, approval and referral sequences, determination of prerequisites for future steps, and the integration of automated systems with human employees [Blaze02].

### **Workflow Management**

Workflow tends to concentrate more on the steps used to drive information or products through a subset of an overall business process, perhaps concentrating on physical production of an item or on an approval process necessary to implement a design change [Blaze02].

Real-time business decisions differ from workflow and BPM in that they focus on taking a specific action or making a particular decision based on the conditions present at a given time. Such decisions may be used as decision-support tools to drive the succeeding step in a larger business process. It may be helpful to think of business decisions as a subset or component of a workflow or business process.

There are software packages on the market that are built specifically to help organizations model and manage workflow and BPM. These contain special mechanisms to assist in defining sequences of tasks e.g. involving physical shipments, information retrieval and transfer, human intervention, and wait/continue conditions that might be encountered during the process. Workflow technologies usually include special queue management capabilities to route information or physical items to specific workers at specific times, and then reintroduce their modifications back into the main flow. Some of these “workers” may be IT systems doing some form of processing, but many of the branching tasks that require decisions assume that the decisions are made by people or are simple to describe. By and large these software packages provide support for managing manual approval processes and queuing, with support for branching decisions based on relatively simple mechanisms. More complex decisions typically require external programming to accomplish. Business rules management addresses the unique needs of condition-based decision making. It separates the logic behind business policies and practices from the mechanics involved in carrying out the actions recommended in the decisions. As such, business decisions may be shared between different business processes and workflows without requiring them to be specified and maintained in duplicate systems. Decision logic can also be more easily defined, located, and maintained when the context of the surrounding business process does not affect their construction. Business rules are a part of every business process in every organizational function. When these rules are complex, numerous, interdependent or subject to change, it makes sense to manage them with systems intended specifically for that purpose.

## **2.4 The Need for Rule Management**

As pointed out above, decision-making is core functionality in many different business applications. Traditionally, these decisions have been programmed as part of the general application code that runs the rest of the system’s processing.

When decisions are translated into unstructured computer programming languages and are embedded in applications, it becomes difficult, costly, and time-consuming to make changes to them. Eventually, the application code becomes so entrenched and interdependent that changing it is no longer practical. The code and all the business logic contained in it becomes ‘legacy code’ within the organization.

Some enterprise applications are purely composed of legacy code. Legacy code may sometime be stable, proven in practice, and preferable to writing and installing entirely new applications and systems. But the business logic incorporated in these systems can be expected to change — sometimes over long periods of time, sometimes on a daily basis. Business rules, properly implemented, allow legacy systems to continue to support business processes while accounting for changes to business policies, practices, and procedures.

Rule Management helps organize large number of rules as well as making it easier to identify and modify rules. Rule Management makes it easier to determine where the rule is implemented and reasons why the rule was implemented. Rule Management involves managing rules in a consistent and coherent manner.

Since business rules represent business logic, - not programming logic, the one challenge of Rule management is to transfer control of Rule management from IT personnel to Business analyst. The intention of Rule Management is to give business analysts the ability to manage and access that business logic directly.

## **2.5 Rule Management Environment**

Generally decisions in a business may depend on several Business Rules. Business Rules are interrelated and one Business Rule affects another Business Rule. Therefore there is need for comprehensive traceability in the Rule Management environment. One way to persist rules is by 'Databasing' them. Equally important is providing business analysts with tools to manage rules. In part, this can be addressed by offering predefined reports and queries providing canned support for typical needs. Beyond that, visualization techniques are useful, especially for making the front-ends friendlier and for communicating the knowledge more effectively.

### **2.5.1 Rule Repository**

A rule repository is a central place where managers, analysts, and software developers can define, share, and maintain the business rules of a company.

### **2.5.2 The Business Terminology Glossary**

The most fundamental in this regard is vocabulary management. When dealing with large number of rules exercising control over the consistent and correct use of business terminology becomes paramount. Once an enterprise definition for terms is completed, everyone should have a common understanding of the terms, thus facilitating better communication between business and systems groups.

### **2.5.3 Rule Traceability**

Another important feature of Rule management is the ability to manage “experience” in business that would otherwise not be documented, but would remain in the heads of a few experts in the business. Take for example, documenting all conditions that were in play when a particular decision was taken.

Rule Management should address the following questions:

- Where can more information about the rule be found?
- To what areas of the business does the rule apply?
- What work tasks does the rule guide?
- Where is the rule implemented?
- What new design deliverables need to address the rule?
- In what jurisdictions is the rule enforced?
- Who can answer particular questions about the rule?
- When was the rule created?
- Over what period of time is the rule enforced?

Another question crucial to managing rules is being able to address relationships between rules - that is, rule-to-rule associations. There are many ways in which rules can be interconnected, as the following list suggests. Being able to trace these relationships is also crucial to Rule Management.

- A rule is an exception to another rule.
- A rule enables another rule.
- A rule subsumes another rule.
- A rule is semantically equivalent to another rule.
- A rule is similar to another rule.
- A rule is in conflict with another rule.
- A rule supports another rule.

## 2.6 Key Concepts of Rule Management

The key concepts supporting separation of the business rules from the remainder of application functionality are threefold:

- Specify the rules in a way that does not depend on the mechanisms used to obtain and update data or on the mechanisms used to carry out recommended actions.
- Create a way to choose and execute the right rules at the right time in the right order without involving control by the application code.
- Allow organizational control and management of the rules in a way that does not depend on or affect the rest of the application code.

Most business rules management systems incorporate all the three elements through language and interface constructs, a processing “engine,” and independent rule management facilities. The following sections discuss each of these aspects of business rules management

### 2.6.1 Specifying Business Rules

Specification of business rules for effective use requires a number of technical prerequisites. First and foremost is rigid adherence to the concept that business logic is independent from the mechanisms used to manipulate data or implement decisions. A business process should be able to make use of rules to reach operational decisions that can be carried out by a variety of systems. For instance, when responding to a customer asking for product support, the company needs to make a number of decisions regarding the customer’s status, support level, location, problem area, and so on. These considerations follow a single defined decision process, even though they may be used by completely separate applications managing website interaction, phone center response by live agents, or automated phone response systems. The physical interaction with the customer is irrelevant as far as the underlying logic is concerned. This is one of the great limitations of business rules included with software applications designed for a single interaction method such as web-based commerce. The rules may work within the application, but they typically are intricately

bound to specific interaction methods. Once the company wishes to expand the decision logic to encompass other types of systems, it has to rewrite the rules to meet the implementation requirements of the new systems. Only an independent business rules system offers the flexibility and neutrality to work identically within multiple implementations.

To achieve these, rules are implemented independent of any external system that may use it. For example in a process that requires a mechanical task Rules management system does not include facilities to physically control input and output. Instead, it contains clearly defined integration points for systems that are built to accomplish these mechanical tasks.

## **Rule Languages**

Whatever technique is used to visualize business rule, a language specifying the syntax, and semantics of the business rules is required. Chapter three will presents a detailed review of languages, classification of languages and their complexity.

Writing business rules in a formalized, executable fashion requires an language for expressing the business logic. The semantic of rules should be intuitive to a business user, even if s/he is not trained as a programmer. For example:

```
If customer.debt > customer.assets then  
  
customer.application.status = Declined.
```

### **EXAMPLE 2-1: SAMPLE RULE IN RULE LANGUAGE**

```
If the debt of the customer is greater than the assets of  
the customer then the status of customer's application is  
Declined.
```

### **EXAMPLE 2-2: SAMPLE RULE IN BUSINESS-LIKE RULE LANGUAGE**

The above formats represent the same rule and execute with equivalent efficiency and styles may be mixed and matched to provide the most convenience and familiarity for the users of the system.

As pointed out earlier vocabulary management ensures a consistent use of business terminologies across the system. In the forgoing example, from a chief executive officer's perspective, he thinks of a customer as a physical person. Whereas a designer thinks of a customer as a logical representation of the customer. There should be a clear system wide understanding of what a customer and an asset is.

When business rules use a familiar format and language, business policymakers can have direct participation in the development, testing, and modification process. System implementation times are reduced and the potential for interpretation errors between business intent and programming implementation are lessened.

Consider, for example, rules that apply when sending a parcel. Shipping cost depends on source and destination location, package weight, packaging, and priority. Writing rules in a table format is sometimes easier for business people to write, modify and understand without involving developers. Figure 2-1 [Fedex02] show a typical shipping rates rules for a shipping company written in a table format.

Rate Scales								
	A	B	C	D	E	F	G	...
Up to 8 oz.	\$ 20.75	\$ 23.50	\$ 23.50	\$ 28.00	\$ 28.00	\$ 28.00	\$ 28.00	...
1 lb.	37.45	39.03	37.97	44.41	44.41	44.41	44.41	...
2 lbs.	42.67	44.04	42.98	51.65	51.65	51.65	51.65	...
3 lbs.	46.75	49.05	51.00	60.76	60.76	60.76	60.76	...
4 lbs.	50.26	54.06	56.28	68.94	68.94	68.94	68.94	...
5 lbs.	53.27	59.07	61.56	77.12	77.12	77.12	77.12	...
...	...	...	...	...	...	...	...	...

FIGURE 2-1 A TABULAR SHIPPING RATES EXAMPLE

The Table format (see figure 2-1), for example, provides a more natural way to think about many types of rules. It provides increased clarity and decreased development time and effort in specifying business logic. The above rules would otherwise translate into the following set of rules.

If the package weight is less than 8oz and in rate scale A then shipment cost = \$ 20.75.

If the package weight is less than 8oz and in rate scale B then shipment cost = \$ 23.50.

If the package weight is less than 8oz and in rate scale C then shipment cost = \$ 23.50.

If the package weight is less than 8oz and in rate scale D then shipment cost = \$ 28.00.

If the package weight is less than 8oz and in rate scale E then shipment cost = \$ 28.00.

EXAMPLE 2-3: SAMPLE SHIPPING RATES CODE.

The code for the remaining rows would follow making the code difficult to read and maintain.

**2.6.2 Executing Business Rules**

One of the key benefits of having a complete business rules system is the reduction in complex programming application logic to support execution of the business logic. In order to achieve these benefit, the business rules system must incorporate sophisticated software components designed to accomplish tasks that would otherwise need to be written in the application code. Most rule

processing system contains components to manage simultaneous requests for decisions from different applications or for different users; condition-based selection of the rules to be executed, in the right order; auditing software to record what rules were used in the course of making a decision; and support for complex decision chains without needing explicit control from the calling application.

Rule processing engine is usually a callable component of an Enterprise application, providing decisions based on data about the case in question. That data may be read from database tables, XML documents, or passed parameters from the calling application. By the same token, the Rule processing engine can make calls to external applications, functions, and business systems from within rules to initiate external processes, make complex calculations, or update values in external data stores.

Rule processing engine performance can vary widely depending on the precise nature of the implementation. In most cases, performance degrades as rules are added to the rule base. However, certain algorithms used in commercial rule engines in particular those written in the manner of the Rete Algorithm see below have been shown to have performance that is “asymptotically independent of the number of rules.” A detailed review of the Rete Algorithm [Fo82] is outside the scope of this work, but the pertinent essence is that only the Rete Algorithm is known to provide performance that is both fast and scalable [Halley01a]. It may be easiest to show the concepts and performance of Rete Algorithm first by looking at an example of its implementation.

### 2.6.2.1 The Rete Algorithm

The information in this section is provided for an understanding of the Rete algorithm.

A typical rule engine has a fixed set of rules while the knowledge base changes continuously. However, it is an empirical fact that, in most Rules management systems, much of the knowledge base is also fairly fixed from one rule operation to the next. Although new facts arrive and old ones are removed at all times, the percentage of facts that change per unit time is generally fairly small. The obvious implementation would be to keep a list of the rules and continuously cycle through the list, checking each one's left-hand-side (LHS) against the knowledge base and executing the right-hand-side (RHS) of any rules that apply. This is inefficient because most of the tests made on each cycle will have the same results as on the previous iteration. However, since the knowledge base is stable, most of the tests will be repeated. You might call this the rules finding facts approach and its computational complexity is of the order of  $O(RF^P)$ , where R is the number of rules, P is the average number of patterns per rule LHS, and F is the number of facts on the knowledge base. This escalates dramatically as the number of patterns per rule increases.

Most Rule engines uses a very efficient method known as the Rete (Latin for *net*) algorithm. The classic paper on the Rete algorithm [Fo82] became the basis for a whole generation of fast Rule Engines: OPS5, its descendant ART, CLIPS, Blaze Advisor, and ILOG. In the Rete algorithm, the inefficiency described above is alleviated (conceptually) by remembering past test results across iterations of the rule loop. Only new facts are tested against any rule LHSs. Additionally, as will be described below, new facts are tested against only the rule LHSs to which they are most likely to be relevant. As a result, the computational complexity per iteration drops to something more like  $O(RFP)$ , which is linear in relation to the size of the fact base.

The discussion of the Rete algorithm is necessarily brief. Its is based on implementation of the Rete algorithm by the Java Expert System Shell (JESS). For more information on Rete algorithm, refer to the Forgy paper [Fo82] or to Giarratano and Riley paper [GiRi93].

### 2.6.2.2 Implementation of the Rete algorithm in JESS

The Rete algorithm is implemented by building a network of nodes, each of which represents one or more tests found on a rule LHS. Facts that are being added to or removed from the knowledge base are processed by this network of nodes. At the bottom of the network are nodes representing individual rules. When a set of facts filters all the way down to the bottom of the network, it has passed all the tests on the LHS of a particular rule and this set becomes an activation. The associated rule may have its RHS executed (fired) if the activation is not invalidated first by the removal of one or more facts from its activation set.

Within the network itself there are broadly two kinds of nodes: one-input and two-input nodes: One-input nodes perform tests on individual facts, while two-input nodes perform tests across facts and perform the grouping function. Subtypes of these two classes of node are also used and there are also auxiliary types such as the terminal nodes mentioned above.

An example is often useful at this point.

```
(defrule library-rule-1
  (book (name ?x)      (status late) (borrower (name ?y))
  (borrower (name ?y) (address ?z))
  =>
  (send-late-notice ?x ?y ?z))
```

#### EXAMPLE 2-4: A JESS RULE

This rule might be translated into a pseudo-English rule language as follow:

Library rule #1:

```
If
  a late book exist, with name X, borrowed by someone named Y
And
  that borrower's address is known to be Z
Then
  send a late notice to Y at Z about the book X.
```

#### EXAMPLE 2-4 B: LIBRARY RULE #1 IN PSEUDO-ENGLISH RULE LANGUAGE

```
(defrule library-rule-2
  (book (name ?x) (status late) (borrower (name ?y))
  =>
  (charge-late-fee ?x ?y))
```

#### EXAMPLE 2-5: A JESS RULE

This rule might be translated into a pseudo-English rule language as follow:

Library rule #2:

If

a late book exist, with name X, borrowed by someone named Y

Then

charge a late fee to Y on the book X.

EXAMPLE 2-5 B: LIBRARY RULE #2 IN PSEUDO-ENGLISH RULE LANGUAGE

The above rules can be transformed into the following condition/action evaluation trees:

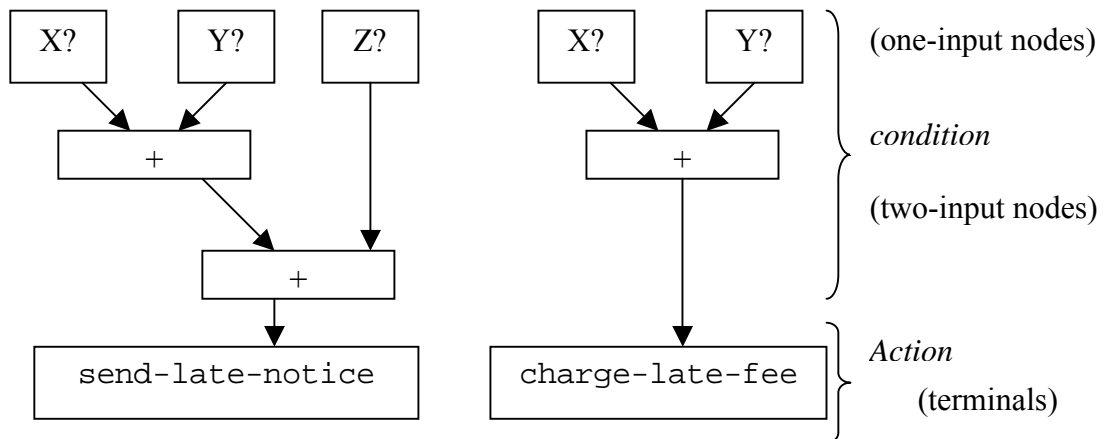


FIGURE 2-2 ORIGINAL JESS NODES NETWORK FOR RULES 2-4 AND 2-5

The nodes marked x?, etc., test if a fact contains the given data, while the nodes marked + remember all facts and fire whenever they've received data from both their left and right inputs. To run the network, Jess presents new facts to each node at the top of the network as they are added to the knowledge base. Each node takes input from the top and sends its output downwards. A single input node generally receives a fact from above, applies a test to it, and, if the test passes, sends the fact downward to the next node. If the test fails, the one-input nodes simply do nothing. The two-input nodes have to integrate facts from their left and right inputs, and in support of this, their behavior must be more complex. First, note that any facts that reach the top of a two-input node could potentially contribute to an activation: they pass all tests that can be applied to single facts. The two input nodes therefore must remember all facts that are presented to them, and attempt to group facts arriving on their left inputs with facts arriving on their right inputs to make up complete activation sets. A two-input node therefore has a left memory and a right memory. It is here in these memories that the inefficiency described above is avoided. A convenient distinction is to divide the network into two logical components: the single-input nodes comprise the pattern network, while the two-input nodes make up the join network.

There are two simple optimizations that can make Rete even better: The first is to share nodes in the pattern network. In the network above, there are five nodes across the top, although only three are distinct. We can modify the network to share these nodes across the two rules (the arrows coming out of the top of the x? and y? nodes are outputs) see figure 2-3.

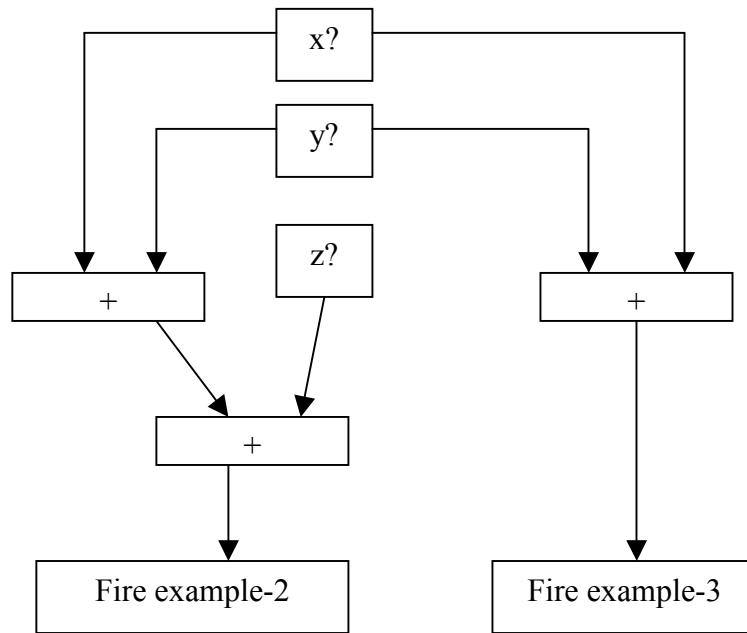


FIGURE 2-3 REFINED JESS NODES NETWORK WITH REDUNDANCY

But that's not removing all the redundancy in the original network. Now we see that there is one join node that is performing exactly the same function (integrating x,y pairs) in both rules, and we can share that also:

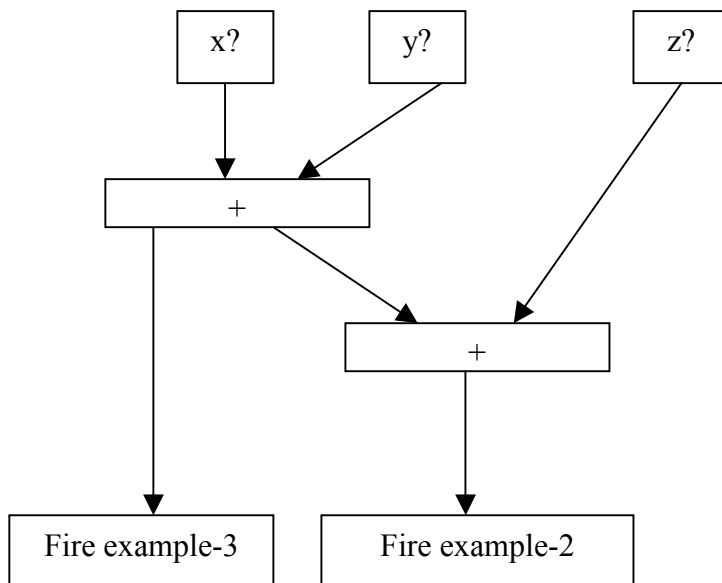


FIGURE 2-4 OPTIMIZED JESS NODES NETWORK

The pattern and join networks are collectively only half the size they were originally. This kind of sharing comes up very frequently in real systems and is a significant performance booster.

### **2.6.3 Managing Business Rules**

Business rules constitute the part of an application most prone to change over time. New or altered regulations must be adhered to, new business methods get implemented, competitive pressures force changes in policy, new products and services are introduced, and so on. All of these require changes to the decisions that control behaviors and actions in business applications. The people best able to gauge the need for new operational behaviors, envision the new decision criteria and responses, and authorize implementation of new business policies are seldom technically trained in programming techniques. Policymakers want business applications designed to let them accomplish business tasks such as introducing new promotions, changing discount levels, altering rating criteria, or incorporating new regulations. But creating such modification and management applications is a task often comparable to building the underlying business systems themselves! Traditional organizational behavior is for business policymakers to gather a set of business changes that should be made, submit them as a formal request to a programming department, sign off the programming interpretation of the changes, and wait for a scheduling opportunity to have the changes implemented in a new software release. The delays in this type of cycle are apparent, but there is no alternative in traditionally implemented software systems.

Because business rules are separated from and independent of the underlying system code that keeps a business application operating, they should be changed without impacting the application logic.

#### **Auditability**

An aspect of business rules management that should not be overlooked is the importance of keeping track of tasks carried out and decisions made. Since business rules control key decision processes, it is crucial to have clear access to what rules were in play when decisions were made, who made changes to rules, and why changes were made. Part of this process is dependent on a commitment to good change management procedures within the organization. But the software being used to manage the rules should also contribute functionality to support enterprise control and auditing. For example, to record the state of an entire rule service at any point in time, a 'snapshot' may be taken of the files containing the rules. This can be recovered, reused, or examined later for auditing and results testing / comparison against later versions.

Rules management and maintenance applications should also add documentary information to rule creation, and changes showing the author, time and date of the change.

## 3 Rule Language Foundations

The purpose of this chapter is to introduce rule languages that form the foundation of building business rules. Languages, language classes, and languages complexity analysis is discussed followed by description of rule languages and their standards.

### 3.1 Languages and Language Classes

#### 3.1.1 Formal Languages, Grammars and Automata

There are three fundamental concepts in languages:

- **Languages**
  - A language is a subset of the set of all possible strings formed from a given set of symbols.
  - There must be a membership criterion for determining whether a particular string in the set.
- **Grammars**
  - A grammar is a formal system for accepting or rejecting strings.
  - A grammar may be used as the membership criterion for a language.
- **Automata**
  - An automaton is a simplified, formalized model of a computer.
  - An automaton may be used to compute the membership function for a language.
  - Automata are outside the scope of this work and will not be discussed.

##### 3.1.1.1 Definition of a Language

An alphabet is a finite, nonempty set of symbols. The symbol  $\Sigma$  is used to denote this alphabet. *Note:* Symbols may be more than one English letter long, e.g. **while** is a single symbol in Java.

A string is a finite sequence of symbols from  $\Sigma$ .

The length of a string  $s$ , denoted  $|s|$ , is the number of symbols in it.

The empty string is the string of length zero. It really looks like this: ' ' but for convenience the symbol lambda ( $\lambda$ ) is used.

$\Sigma^*$  denotes the set of all sequences of strings that are composed of zero or more symbols of  $\Sigma$ .

$\Sigma^+$  denotes the set of all sequences of strings composed of one or more symbols of  $\Sigma$ . That is,  $\Sigma^+ = \Sigma^* - \{\lambda\}$ .

A *language* is a subset of  $\Sigma^*$  [PU01].

### 3.1.1.2 Definition of a Grammar

A *grammar*  $G$  is a quadruple  $G = (V, T, S, P)$  where

- $V$  is a finite set of (meta) symbols, or *variables*.
- $T$  is a finite set of *terminal symbols*.
- $S \in V$  is a distinguished element of  $V$  called the *start symbol*.
- $P$  is a finite set of *productions* (or *rules*).

A *production* has the form  $X \rightarrow Y$  where

- $X \in (V \cup T)^+$ , and
- $Y \in (V \cup T)^*$ .

Putting this in words read:

- $X \in (V \cup T)^+$  :  $X$  is a member of the set of strings composed of any mixture of variables and terminal symbols, but  $X$  is not the empty string.
- $Y \in (V \cup T)^*$  :  $Y$  is a member of the set of strings composed of any mixture of variables and terminal symbols;  $Y$  is allowed to be the empty string.

### 3.1.1.3 Productions

Productions are rules that can be used to define the strings belonging to a language. Suppose language  $L$  is defined by a grammar  $G = (V, T, S, P)$ . You can find a string belonging to this language as follows:

1. Start with a string  $w$  consisting of only the start symbol  $S$ .
2. Find a substring  $x$  of  $w$  that matches the left-hand side of some production  $p$  in  $P$ .
3. Replace the substring  $x$  of  $w$  with the right-hand side of production  $p$ .
4. Repeat steps 2 and 3 until the string consists entirely of symbols of  $T$  (that is, it doesn't contain any variables).
5. The final string belongs to the language  $L$ .

Each application of step 3 above is called a *derivation step*.

Suppose  $w$  is a string that can be written as  $uxv$ , where

- $u$  and  $v$  are elements of  $(V \cup T)^*$
- $x$  is an element of  $(V \cup T)^+$
- there is a production  $x \rightarrow y$

Then we can write

$uxv \Rightarrow uyv$

and we say that  $uxv$  directly derives  $uyv$ .

Notation:

- $S \Rightarrow T$  : S derives T (or T is derived from S) in exactly one step.
- $S \stackrel{*}{\Rightarrow} T$  : S derives T in zero or more steps.
- $S \stackrel{+}{\Rightarrow} T$  : S derives T in one or more steps.

For an example of language grammar, refer to appendix B: Survey items generation language grammar.

### 3.1.2 Language Classes

There are many different classes of languages that exist. However there are four “**classic**” categories as shown in table 2.1 that are generally accepted. It should be noted that not all language classes fit neatly into a hierarchy.

#### The Chomsky Hierarchy

The Chomsky hierarchy, as originally defined by Noam Chomsky, comprises four types of languages and their associated grammars and machines.

- Type-0 grammars (unrestricted grammars) include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. The language that is recognized by a Turing machine is defined as all the strings on which it halts. These languages are also known as the recursively enumerable languages. Note that this is different from the recursive languages which can be decided by an always halting Turing machine.
- Type-1 grammars (context-sensitive grammars) generate the context-sensitive languages. These grammars have rules of the form  $\alpha A \beta \rightarrow \alpha \gamma \beta$  with A a nonterminal and  $\alpha$ ,  $\beta$  and  $\gamma$  strings of terminals and nonterminals. The strings  $\alpha$  and  $\beta$  may be empty, but  $\gamma$  must be nonempty. It can also include the rule  $S \rightarrow \epsilon$ . If it does, then it must not have an S on the right side of any rule. These languages are exactly all languages that can be recognized by a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input.
- Type-2 grammars (context-free grammars) generate the context-free languages. These are defined by rules of the form  $A \rightarrow \gamma$  with A a nonterminal and  $\gamma$  a string of terminals and nonterminals. These languages are exactly all languages that can be recognized by a nondeterministic pushdown automaton. Context free languages are the theoretical basis for the syntax of most programming languages.
- Type-3 grammars (regular grammars) generate the regular languages. Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed by a single nonterminal. The rule  $S \rightarrow \epsilon$  is also here allowed if S does not appear on the right side of any rule. These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of formal languages can be obtained by regular expressions. Regular languages are

commonly used to define search patterns and the lexical structure of programming languages.

Every regular language is context-free, every context-free language is context-sensitive and every context-sensitive language is recursively enumerable. These are all proper inclusions, meaning that there exist recursively enumerable languages which are not context-sensitive, context-sensitive languages which are not context-free and context-free languages which are not regular.

The following table summarizes each of Chomsky's four types of grammars, the class of languages it generates, the type of automaton that recognizes it, and an example of such language.

Language	Grammar	Machine	Example
Recursively enumerable language	Unrestricted grammar	Turing machine	Any computable function
Context-sensitive language	Context-sensitive grammar	Linear-bounded automaton	$a^n b^n c^n$
Context-free language	Context-free grammar	Nondeterministic pushdown automaton	$a^n b^n$
Regular language	Regular grammar <ul style="list-style-type: none"> <li>• Right-linear grammar</li> <li>• Left-linear grammar</li> </ul>	Deterministic or nondeterministic finite-state acceptor	$a^*$

TABLE 2.1 THE CHOMSKY LANGUAGE HIERARCHY [W102B].

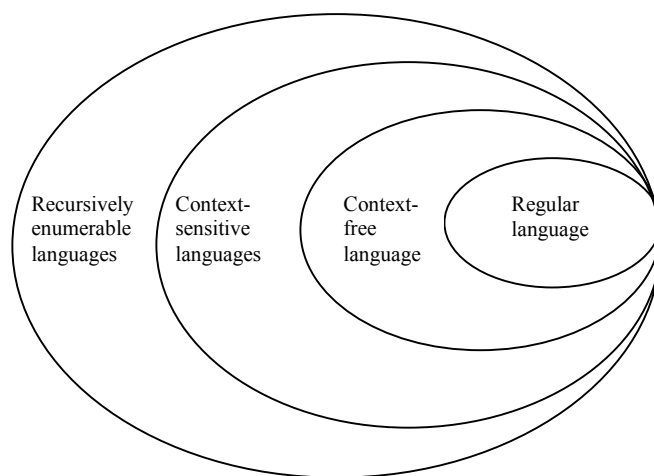


FIGURE 3-1 THE CHOMSKY LANGUAGE HIERARCHY

These languages form a strict hierarchy; that is, regular languages  $\subset$  context-free languages  $\subset$  context-sensitive languages  $\subset$  recursively enumerable languages.

A rule language is context-sensitive as, e.g., variables used in rules must be defined before they can be used in the rules. Usually the variables are defined outside the rules, in the rule system. Therefore the rules themselves can be parsed by a context-free language parser with additional context-sensitive checking (variable definition, type checking, etc.).

### 3.1.3 Language Complexity Analysis

**Complexity** concerns itself with two kinds of measures: time and space. Time complexity is a measure of how long a computation takes to execute. For a rule language, this could be measured as the number of moves required to evaluate one rule.

**Space complexity** is a measure of how much storage is required for a computation. Both of these measures are functions of a single input parameter, the number of the elementary expressions in the rule. Again, this can be defined in terms of bytes.

For any given number of the elementary expressions, different expressions typically require different amounts of space and time. Hence we can discuss for either the average case or for the worst case. Usually worst-case complexity is chosen because

- It may be difficult or impossible to define an "average" case. For many problems, the notion of "average case" doesn't even make sense.
- It is usually much easier to compute worst-case complexity.

In complexity equations are subjected to some extreme simplifications. For example, if a given algorithm takes exactly  $5(n^3 + 2n^2 - n + 1003)$  machine cycles, where  $n$  is the size of the input, it is simplified to  $O(n^3)$  (read: Order  $n$ -cubed). This is called an **order statistic**. Specifically,

- Drop all terms except the highest-ordered one, and
- Drop the coefficient of the highest-ordered term.

Justifications for this procedure are:

*For very large values of  $n$ , the effect of the highest-order term completely swamps the contribution of lower-ordered term.*

Tweaking the factor computation can improve the coefficients, but the order statistic is a function of the algorithm itself.

### Polynomial-Time Algorithms

A **polynomial-time algorithm** is an algorithm whose execution time is either given by a polynomial on the size of the input, or can be bounded by such a polynomial. Problems that can be solved by a polynomial-time algorithm are called **tractable** problems.

For example, most algorithms on arrays can use the array size,  $n$ , as the input size. To find the largest element in an array requires a single pass through the array, so the algorithm for doing this is  $O(n)$ , or **linear time**.

Sorting algorithms usually require either  $O(n \log n)$  or  $O(n^2)$  time. Bubble sort takes linear time in the best case, but  $O(n^2)$  time in the average and worst cases. Heapsort takes  $O(n \log n)$  time in all cases. Quicksort takes  $O(n \log n)$  time on average, but  $O(n^2)$  time in the worst case.

### Complexity classes P and NP

In complexity theory, the class **P** consists of all those decision problems that can be solved on a deterministic sequential machine in an amount of time that is polynomial in the size of the input; the class **NP** consists of all those decision problems whose positive solutions can be verified in polynomial time given the right information, or equivalently, whose solution can be found in polynomial time on a non-deterministic machine.

### NP-completeness

In complexity theory, the complexity class **NP-complete** is the set of problems that are the hardest problems in **NP**, in the sense that they are the ones most likely not to be in P. If you could find a way to solve an **NP-complete** problem quickly, then you could use that algorithm to solve all **NP** problems in polynomial time.

### Example of an NP-complete Problem

#### Traveling salesman problem

The travelling salesman problem (TSP) is a prominent illustration of a class of problems in computational complexity theory. The problem can be stated as: Given a number of cities and the costs of travelling from one to the other, what is the cheapest roundtrip route that visits each city exactly once and then returns to the starting city?

The most direct answer would be to try all the combinations and see which one is cheapest, but given that the number of combinations is  $N!$  (the factorial of the number of cities), this solution rapidly becomes impractical.

#### How fast are the best known deterministic algorithms?

The problem has been shown to be NP-hard, and the decision version of it ("given the costs and a number  $x$ , decide whether there is a roundtrip route cheaper than  $x$ ") is **NP-complete**.

The problem is of considerable practical importance, and various approximation algorithms, which "quickly" yield "good" solutions with "high" probability, have been devised. An approximate solution for 15,112 cities in Germany was found in 2001 by Princeton University scholars [PUTSP01].

## 3.2 Rule Languages

A rule language defines syntax and semantics to express rules in a structured way that can be interpreted by rule engines. Most rule languages follow the same structure of expressing rules, i.e., as condition-action pairs, but employ different syntax.

### 3.2.1 Overview

Nowadays, many rule languages exist. Some are defined by rule language vendors especially for their commercial products, and research has led to many research rule systems with their own languages.

To date, there is no standard rule language but there is much effort toward standardizing RuleML (Rule Markup Language) [RuleML01]. Most rule languages are Context-free language. RuleML is an early-phased standard effort on a markup language for exchange of rules in XML. The goal of this effort is eventual adoption as a web standard, e.g., via the World Wide Web Consortium (W3C) within its new Semantic web Activity [W3C01].

RuleML is an XML syntax for rule knowledge representation (KR), that is inter-operable among major commercial rule systems. It is especially oriented towards four currently commercially important families of rule systems: SQL (relational databases), Prolog, production rules (c.f. OPS5, CLIPS, JESS) and Event-Condition-Action (ECA). These kind of rules today are often found embedded in systems built using Object-Oriented (OO) programming Languages (e.g., C++ and Java), and are often used for business process connectors / workflow management.

### 3.2.2 Production Rules

Production rules are one of the most popular and widely used knowledge representation languages. Early expert systems used production rules as their main knowledge representation language. For example, MYCIN, which is also considered one of the first research works in medical informatics, has production rules as its knowledge representation language.

Production rule system consists of three components, i.e., working memory, rule base and interpreter. The working memory contains the information that the system has gained about the problem thus far. The rule base contains information that applies to all the problems that the system may be asked to solve. The interpreter solves the control problem, i.e., decide which rule to execute on each selection-execute cycle.

### 3.2.3 Prolog System Rules: Definite Clauses

Prolog (PROgramming LOGic) rose within the realm of Artificial Intelligence (AI). It originally became popular with AI researchers, who know more about "what" and "how" intelligent behaviour is achieved. The philosophy behind Prolog deals with the logical and declarative aspects. A Prolog developed system decides the way to solve the problem, including the sequences of instructions that the computer must go through to solve it. It solves problems by searching a knowledge base [Prolog03].

The Prolog language is based on *definite clauses*, which are *Horn clauses* [Wi03a], that have exactly one positive literal [SICS03]. Following are two examples of *non-unit clauses*:

$P :- Q,R,S$        $P$  is the conclusion of the clause, while  $Q,R,S$  are preconditions. The system would try to satisfy  $Q,R$  and  $S$  to satisfy the goal  $P$ .

Example:       $\text{employed}(X) :- \text{employs}(Y,X), \text{isEmployer}(Y)$

Semantics:      Any  $X$  is employed if a  $Y$  exists that is an employer and  $Y$  employs  $X$ .

?- Q , R            Query: The system would try to satisfy Q and R.

Example:            ?- isStudent(X), isEmployed(X).

Semantics:            Is there any X for which the following is true: X is a student and X is employed.

### **3.2.4 Event-Condition-Action (ECA) Rules**

Event-condition-action (ECA) rules take the following form:

*on* event

*if* condition

*do* actions

ECA rules have been used in many settings, including active databases , workflow management, network management, personalisation and publish/subscribe technology , and specifying and implementing business processes .When multiple ECA rules are defined within a system, their interactions can be difficult to analyse, since the execution of one rule may cause an event which triggers another rule or set of rules. These rules may in turn trigger further rules and there is indeed the potential for an infinite cascade of rule firings to occur. as an example of ECA rules, database triggers is described below.

#### **Relational Database Triggers**

Modern relational databases provide for the attachment of procedures that can be “triggered” whenever a row in the table is added, removed, or changed. By attaching the appropriate “how” routine for each condition of a rule to the corresponding table business rules can be brought to bear without modifying application procedures that manipulate the database.

### **3.2.5 Independent Event-Condition-Action (IECA) Rules**

Independent Event-Condition-Action rule are identical to Event-Condition-Action rule except that the execution of its action does not alter the evaluation of the condition of any rule, i.e. the terms used in conditions are strictly separated from those used in the actions. Without the interdependency ECA rules are identical to IECA hence IECA rule languages are a subset of the ECA rule languages. IECA systems are used where the execution of an action does not affect the evaluation of the condition of any other rule in the system. For example, in the ship inspection process, where the terms used in the conditions are distinct from the survey items in the action, the survey items to be generated are independent of the condition to be satisfied.

### **3.2.6 Case Study: IECA**

In our case study (see more in chapter 5), during the ship inspection process, survey items are generated by running a rule evaluator. The Survey Items Generation Rule Language is a general rule language specially designed for defining rules for generating survey items based on survey type and ship information. The Survey Items Generation Language grammar is described in appendix B.

### 3.2.6.1 Survey Items Generation Rule Evaluation Complexity

The complexity of rule languages deals with evaluation of rule conditions. The execution of action if the rule hold is usually considered as an external process with its own complexity. Hence omitted in this work.

#### Time complexity

Time Complexity expresses the order of time needed to evaluate one rule. As stated before, evaluating a rule refers to evaluate its condition. The execution of the action is considered to be outside the analysis, as this could be, e.g., the complex operation on business objects in an information system. Generally a condition is made up of one or more elementary expressions joined by **AND** or **OR** operators. An elementary expression can also be negated by a **NOT** operator.

The execution time for one elementary expression is  $O(1)$ .

The execution time of a negated elementary expression is the sum of execution time for the elementary expression  $O(1)$  and the negation itself  $O(1)$  hence the execution time of an elementary expression or a negated elementary expression can be expressed as  $O(1) + O(1) = 2 O(1) = O(1)$  in **order statistic**.

The execution time for n pure elementary expression or negated elementary expression joined by AND or OR operators is  $(2n-1)$  times  $O(1)$ . Again dropping all terms except the highest-ordered one, and dropping the coefficient of the highest-ordered term. **And assuming a maximum number of n elementary expressions eg 32,<sup>1</sup> the execution time in order statistic is reduced to  $O(1)$ .**

#### Evaluating a Rule Set

A rule set is composed of one or more rules. Each rule consist of a validity condition an actual rule condition. Expressing the execution time in *order statistic* we find:

Evaluation of validity condition  $= O(1)$ .

Evaluation of main rule condition  $= O(1)$ .

Total execution time  $= 2 \times O(1) = O(1)$ .

Assuming a rule set is composed of m number of rules and rules in a rule set are independent of each other then, the execution time for a rule set in *order statistic* can be expressed as  $m \times O(1) = O(m)$ .

#### Space complexity

It refers to how much memory is required to evaluate a rule set. Following the argument above, it can be shown that the amount of memory required to evaluate a rule set is  $S(m)$  in *order statistic*.

---

<sup>1</sup> n becomes fixed because it has an upper bound.

### **3.2.6.2 Complexity Analysis of Existing Rule Languages**

Most Rule languages are proprietary and the level of expressiveness and complexity differ from one vendor to another. ILOG Rule Language [ILOG02a] and Java Expert System Shell (JESS) [JESS02] Rule Language are examples. Unfortunately, the level of expressiveness and complexity for most of these languages is not made public. However, it is clear that rules in these languages are interdependent. For example, an action of one rule may cause another rule to fire. The interdependent increases the order of complexity as compared to our case study (Survey Items Generation Rule) language, which consist of independent rules.

## 4 Analysis and Design of Rule Management Systems

Business rule applications exist in virtually every business domain. They can be defined as those applications that contain business rules of a very dynamic nature e.g. business rules that change frequently and, as such, require:

- System support to quickly change rules without modifying application code
- A high-level business rule language that can be understood by business administrators and/or users of the application
- Tools like rule editors that support the definition and maintenance of business rules.
- A mechanism for executing rules that integrates easily into application architectures

Applications suitable for business rules can be found wherever there exists a need to quickly and easily implement changes in an organizations declarative knowledge. From Customer Relationship Management to Financial Management applications, business rules are used to implement more flexible, and easier to maintain applications.

### 4.1 Requirement Analysis of Rule Management Systems

The main challenge of Rules Management systems is the ability to allow users that are not IT experts to change business rules without any help from IT or the service provider.

A rules management system requires a number of capabilities to be successfully implemented. The following are general requirements of a rules management system.

#### **Support of Business Process Management**

Enterprises run their businesses with repeatable business processes driven by general business rules with specific rules for specific situations and customers. Thus, supporting the process requirements of business is fundamental. Rules, at their most elemental level, are just clearly expressed “If X is true then do Y” statements. During execution rules may need to interact to complete a business process, and they need to be easy to build and manipulate in modular units with adequate controls. In addition, process support requires that rules can use information from diverse sources and create temporary records to allow calculations and decisions to be collected and stored in memory [Blaze02]. Rules need to calculate values and make assessments. Exceptions e.g. missing and

unavailable data events must be handled. Finally and most importantly, rules are evaluated or executed in a sequence.

Taken together, these capabilities allow enterprises to do business using independent rule services made up of executable, sequenced, declarative rules, rather than being forced to integrate the logic as code into a system.

### **Support of Application Maintenance**

Current Enterprise applications require a new application maintenance paradigm that can deliver faster, easier application modification. Business rule changes are first identified by the users of the system. They are often domain experts and process owners who have the authority to authorize application modifications within the area of their responsibility. Rule changes are identified by users of the system themselves. The fastest and safest way to empower these users is to give them the tools they need to make the application changes themselves. This can be achieved by giving them access to easy-to-use rule maintenance applications that allow them to maintain the policies, procedures and rules for which they are responsible. And for more and more applications, control belongs to the individual in a user-centric world. By taking IT “out of the loop” for everyday application maintenance, advanced rules management technology can make applications more adaptable.

### **Support of Enterprise Policy Consistency**

This refers to using information consistently across the enterprise. As the number of applications in an enterprise grows, specific individual pieces of business logic are needed in more and more places. Increasingly, different applications with overlapping or identical purposes are being deployed over different communication mediums. Web customer self-service applications, call center applications and mobile applications are obvious cases in point. When business rules change, it is increasingly common that more than one application must be changed. And when business rules change, all application changes should generally be done at the same time.

By separating business logic from application code, rules management system allows the creation of sharable business logic services. This means that the same business logic can be applied across multiple applications including the increasingly important mobile applications.

## **4.1.1 Performance Considerations and Platform Support**

The efficiency of a rule management system can be measured in terms of scalability and performance as described below.

### **Scalability**

With many enterprises applications taking advantage of rule based programming, it is vital that a rules management system is able to efficiently deal with growing number of rules as they are added over time.

### **Performance**

Especially in batch environments, performance requirements can be very high. Rule processing engine performance can vary widely depending on the precise nature of the implementation. In most cases, performance degrades as rules are added to the rule base. However, certain algorithms

used in commercial rule engines in particular those written in the manner of the Rete Algorithm have been shown to have performance that is “asymptotically independent of the number of rules.” for more information on the Rete Algorithm, refer to chapter 3.

#### **4.1.2 Rule Engine-Specific Requirements**

The Rule Engine component requires a number of capabilities for the Rules Management System to function properly. There are at least six major ones in a complete system, including:

- Support refining existing object/ data models.
- Support interoperation with other data sources.
- Manage simultaneous requests for decisions from different applications or for different users.
- Perform condition-based selection of the right rules to fire, in the right order.
- Integrate with auditing software to record what rules were used in the course of making a decision.
- Offer support for complex decision chains without needing explicit control from the calling application.

#### **4.1.3 Rules Management GUI-Specific Requirements**

The Rules Management GUI requires a number of capabilities for the Rules Management System to function properly. There are at least three major ones in a complete system, including:

- Support management of rule set and service partitioning (role-based access), service assembly and deployment
- Support execution path testing
- Support generation rule maintenance applications that completely hide the structured rule language-the applications that allow rules to be safely constructed and modified by the right people with the right data entry controls.

#### **4.1.4 Rules Management Integration Specifications**

For effective use of a Rules Management System, business logic should be independent from the mechanisms used to manipulate data or implement decisions. Rules should be implemented independent of any external system that may use it. For example in a process that requires a mechanical task e.g. lighting a bulb, rules management system should not include facilities to physically control input and output. Instead, it should contain clearly defined integration points for systems that are built to accomplish these mechanical tasks.

## 4.2 Rule Management System Architecture

The following architectural diagram describes the concepts identified in a rules management system.

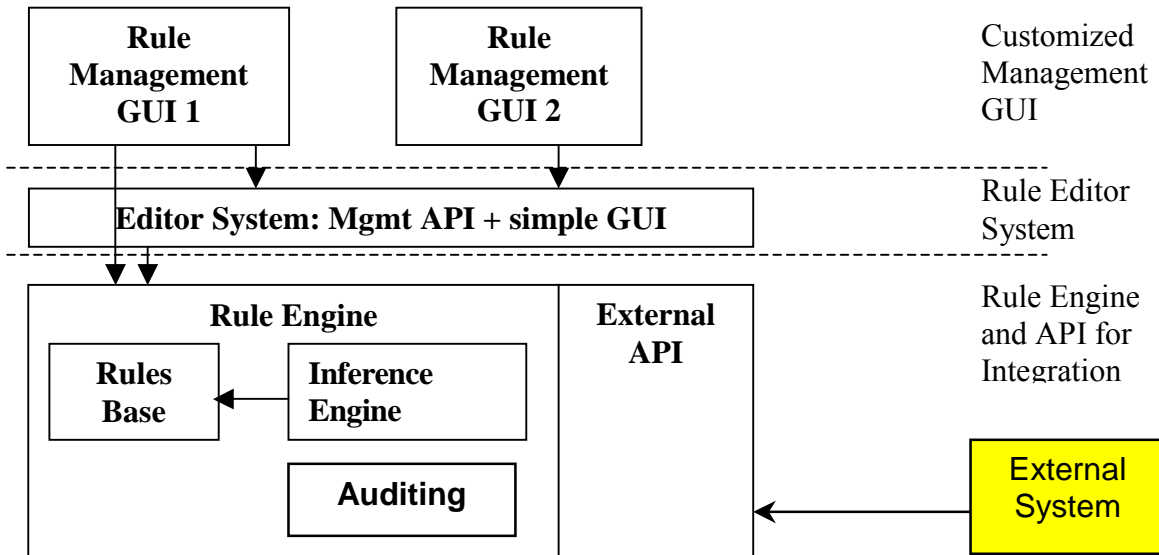


FIGURE 4-1 THE ARCHITECTURE DIAGRAM FOR RULES MANAGEMENT SYSTEM

A rules management system consists of the execution and integration layer, a generic rule editor, and a customized management GUI as shown in figure 4-1 above. All components are interconnected through clearly defined interfaces.

Most rules management systems lack support for a customized GUI. However for non-technical people to fully understand and utilize rules management systems, there is a need for rules management GUI's layer which in most cases are not provided by existing rules management systems. See section 4.3 for a detailed review of existing rules management systems. Anyhow each of the above layers is explained in the following sections.

### 4.2.1 Rule Engine and API for Integration Layer

Rule engine and the external API form the bottom layer of a rules management system.

#### 4.2.1.1 The External API

The external API provides an interface to external systems. The API provides the bridge between external system and rules management system. The interface allows management of rules. The API provides the integration point for the rules management system into the application architectures.

#### 4.2.1.2 Rule Engine

The rule Engine is a software agent that executes some actions resulting from pattern matching. Generally Rule Engine consist of the following components:

- Rule Language: The Rule Engine language defines the syntax and semantics of rules. A detailed review of rule languages is described in chapter three.
- Rules base: The inference engine manages the interactions between user applications and the rules-base. It is responsible for matching, selecting and firing of rules and for the execution of resulting actions.
- Inference engine: Rules are stored in a common repository where they can be managed and accessed during execution.
- Collection of tools: The auditing component provides access to [rule management and execution history e.g., it allows to track] what rules were in play when decisions were made, who made changes to rules, and why changes were made.

### 4.2.2 Rules Editor Layer

A good rules management system allows the business logic of a system to be specified external to the system itself. Rules can be changed directly by rule maintainers and editors. Many rules management system provide natural-language or wizard-style GUIs for designing rules, allowing rule editors e.g. product managers or business analysts to actually specify the logic. However most rule editor systems are not specific for a domain, so rule maintainers must learn the syntax and structure of the rule language.

### 4.2.3 Rules Management GUI Layer

A Rules Management GUI allows rules to be safely constructed and modified by the rule editor with the right data entry controls. The GUI hides the structured rule language and the programming involved and provide user-specific / application-specific customization as demonstrated in example 2-2 and figure 2-1.

## 4.3 Rule Management System Products and Prototypes

There are currently many commercial and open-source implementations of rules management systems. The commercial vendors have undoubtedly good algorithms and have spent considerable time and effort on the user interfaces and rule specification languages.

### 4.3.1 Overview

Some products like Blaze Advisor from HNC allows non-technical business people to control, maintain, and evolve business applications. Business managers can control business policies and procedures throughout multiple systems, in a zero-programming environment. Additional features of Blaze Advisor include automated wizards that allow users to easily create rule and action templates to author business rules in a simple to use web-based maintenance application.

Refer to [Blaze02] for more information on Blaze Advisor. Other products providing rule management technology include:-

- **Blaze Advisor:** <http://www.blazesoft.com/>,
- **ILOG JRules:** <http://www.ilog.com/>,

- **Haley Eclipse:** <http://www.haley.com/>,
- **Sandia JESS:** <http://herzberg.ca.sandia.gov/jess/>,
- **CLIPS:** <http://www.ghg.net/clips/CLIPS.html>,
- **Rule Engine Prototype** [Gi02].

The following section describes rules management tools provided by ILOG JRules. For detailed review of each product, refer to their respective websites.

## **4.3.2 The ILOG Rule Management System**

The following section describes set of tools for managing, editing, and debugging rules in enterprise development provided by ILOG JRules and ILOG Rules. ILOG rules are ECA rules.

### **4.3.2.1 Management tools**

The following are management tools provided by ILOG JRules and ILOG Rules. The business rule repository, logging, and locking are part of the rule engine component while permission management and versioning are part of rule editor management as illustrated in rule management system architecture diagram in figure 4-1.

#### **Business Rule Repository**

The Business Rule Repository provides a central location for storing business rules and business rule metadata. The Repository can be persisted to files or databases, forming the foundation for business rules management functionality.

#### **Logging**

Creation, modification or deletion of business rules and business rule properties can be logged in the Repository's history log.

#### **Locking**

The Repository's locking feature lets multiple users access and modify rules. The locking mechanism prevent multiple users from modifying the same repository element simultaneously.

#### **Permission management**

The Repository Permission Manager controls access to the Repository, allowing definition and enforcement of user or group permissions. Developers and business users can be assigned specific roles and permissions, restricting access and modification privileges. The permission manager interfaces with external authentication and authorization system, including LDAP, JAAS, Unix, Windows, Kerberos or Keystore.

## Versioning

The Repository supports creation and viewing versions of business rules, allowing users to track policy changes. Using versioning support, users can view all versions of a business rule, reverting to a previous version if necessary.

### 4.3.2.2 Profiling tools

Rule execution can be visualized using the profiler. The profiler can be used to collect summary statistics about all rules in a given session:

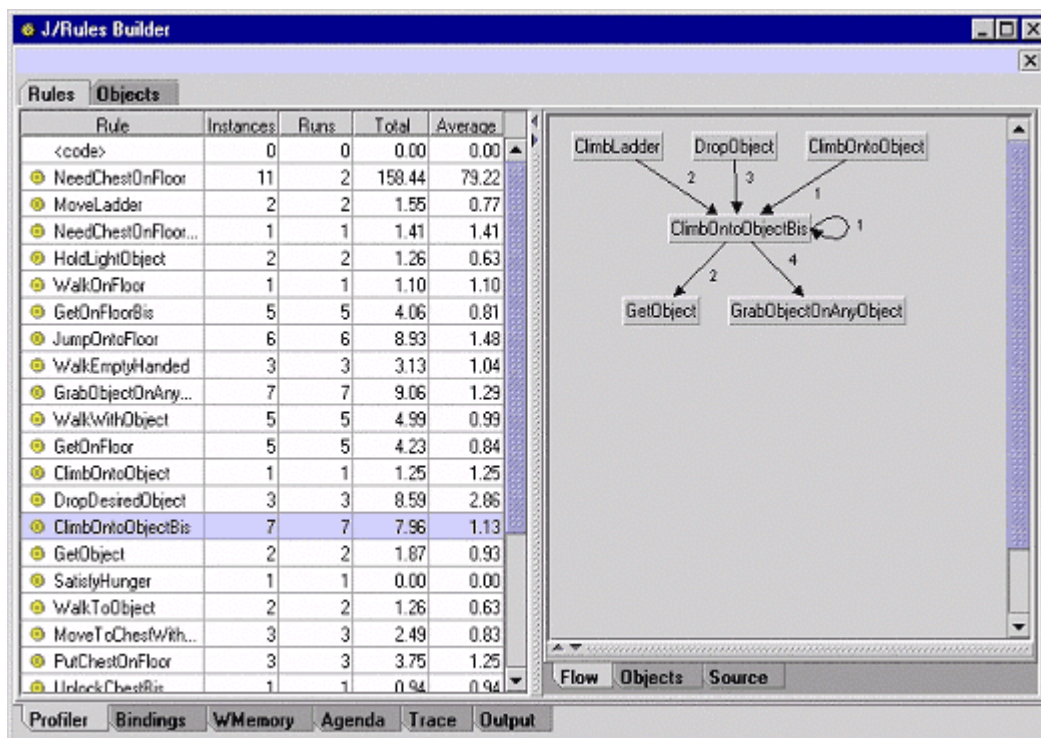


FIGURE 4-2 THE GRAPHICAL USER INTERFACE OF THE RULE BUILDER'S PROFILER

- The number of times that each rule was instantiated
- The number of times each rule was fired
- The total and average time that was spent executing each rule

Users can visually navigate rule flow, one rule at a time, displaying all rules that caused a given rule to fire. Additionally, the profiler displays objects that were asserted, retracted, or updated, and provide information at the object level, allowing a view of the rules associated with application objects.

### 4.3.2.3 Debugging and Tracing tools

By use of breakpoints, one can step through the execution of the Rule Engine rule-by-rule, or action-statement-by-action-statement. Breakpoints can be set on a rule, the action part of a rule, a class, or an object. Selected events can be traced, and the trace output collected and displayed through Rule Builder's tracer panel.

#### 4.3.2.4 Development tools: Rule Builder

The Rule Builder is a graphical environment for managing business rules and developing and debugging rule applications. Managing projects stored in the Business Rule Repository, the Rule Builder provides a set of editors to create and modify rules and rule properties. It also features integrated debugging, inspection and tracing tools used in application development. The Rule Builder can connect with multiple engines concurrently, executing remotely or in the application's process.

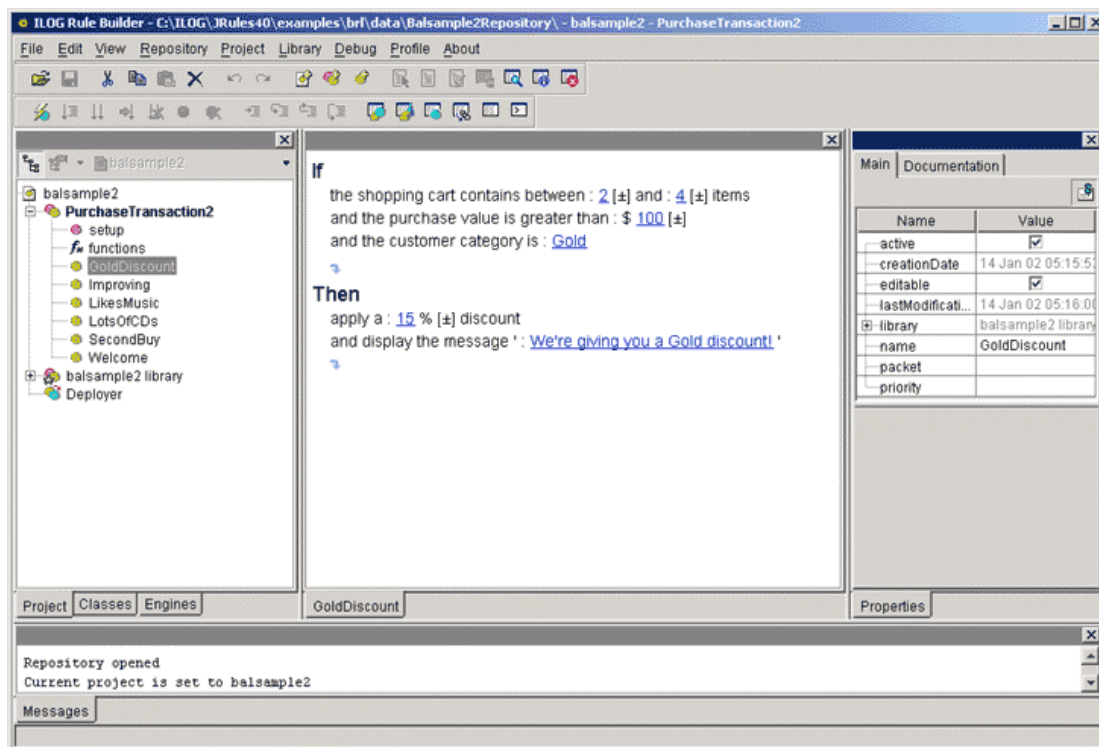


FIGURE 4-3 THE GRAPHICAL USER INTERFACE OF THE RULE BUILDER

The Rule Builder options lets users:

- Edit in different rule languages
- Employ different styles for rule editing
- Customize look and feel, including icons, menubars, toolbars and pop-up menus

#### Query support

The Rule Builder includes a query feature that allows users to locate business rules. The Rule Builder's query feature let users:

- Find and view rules that will be affected by policy changes
- Find and view rules that will be affected by changes in the Business Object Model
- Make global changes to a set of related rules.

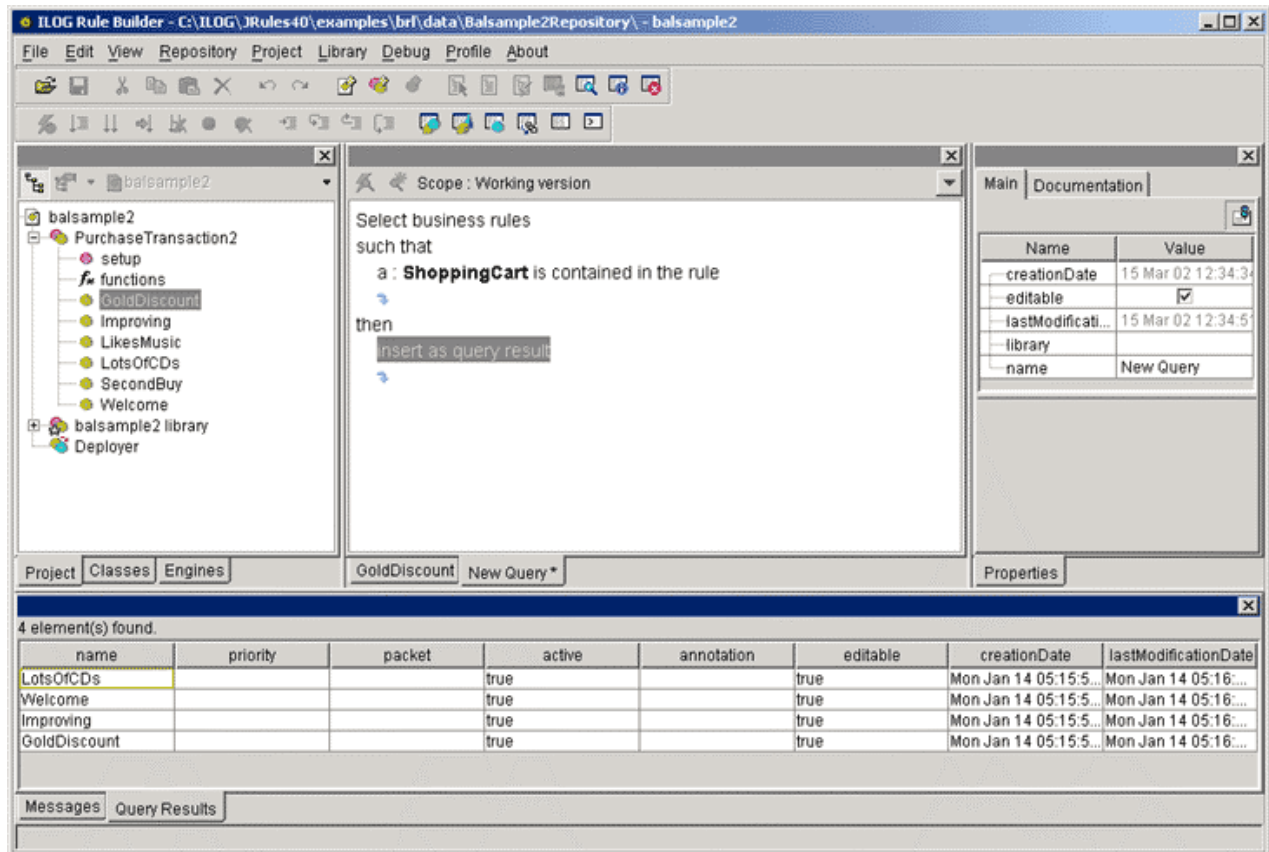


FIGURE 4-4 THE GRAPHICAL USER INTERFACE OF THE RULE BUILDER'S QUERY

## Project Management

The Rule Builder is used in project management. Users can define and access multiple projects stored in the Business Rule Repository. Projects are defined using a deployer, and consist of one or more rulesets and their associated application classes.

## Rule Development: Rule Editor

In the Rule Builder environment, rules can be developed in either a plain text rule editor, or a WYSIWYG syntactic rule editor that graphically steps users through the process of writing rules.

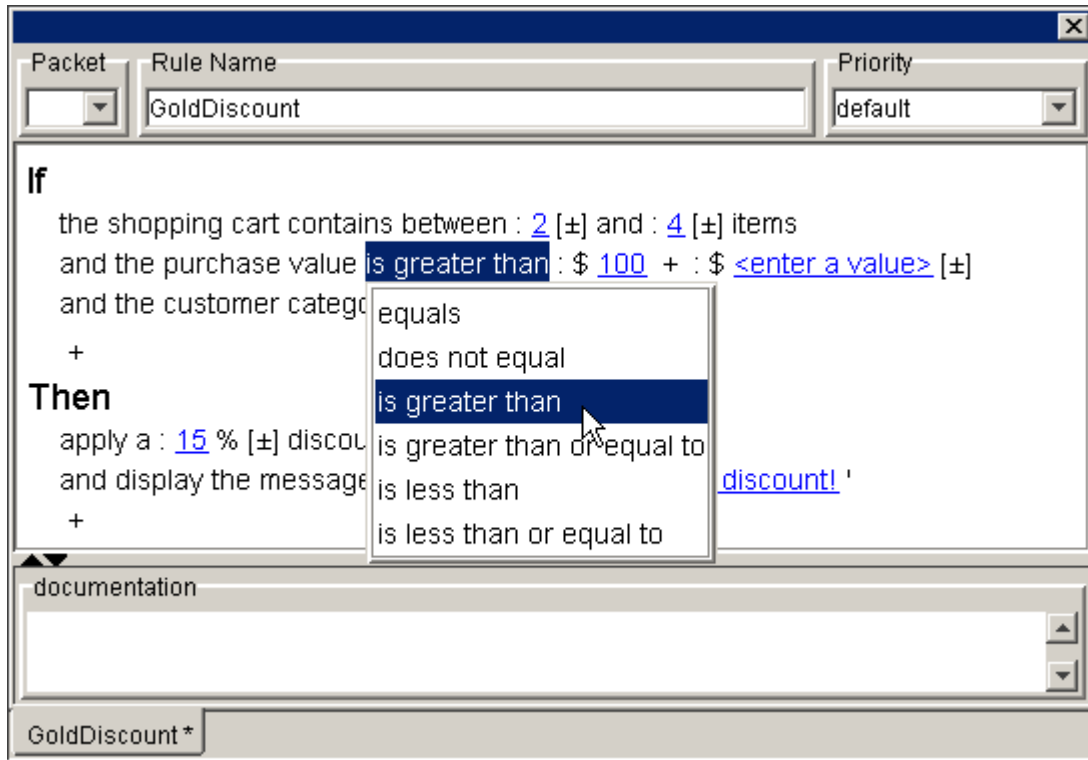


FIGURE 4-5 THE GRAPHICAL USER INTERFACE OF THE RULE EDITOR

The Rule Editor displays business rules in an English-like syntax that allows rules to be written and modified by non-technical users. The business rule language can be customized to support the specific application. The Rule Editor is provided as a JavaBean that can be embedded in a Java application or applet. It is also available as a Web component that can be integrated into a browser.

### Editor as a JavaBean

The Rule Editor as a JavaBean component enables rules to be created with the ease of drop-down pick lists.

### Editor as a Web Component

The Web-based editor component extends the functionality of the syntactic editor to Web-based applications. Behavior of this editor is identical to the JavaBean, except that it can be deployed in a servlet or a JSP page where Web users can use it to edit business rules across the Web. Rules can be viewed using the separate browser component and the syntactic editor in combination.

#### 4.3.2.5 Business Rule Language Support

Business rule language support enables users to customize rule languages to specific applications or domains.

### Business Action Language

Business action Language is a business rule language and can be extended and customized to specific business domains.

### **Business Rule Templates**

Business Rule Templates are rule forms, suitable for untrained users. In essence, a rule template is a partially expanded business rule. New rules can be created by cloning an existing template, Templates help users:

- Develop libraries that embed domain knowledge
- Avoid tedious and repeated editing when creating many similar rules
- Restrict the type of rules that can be written by final users, for security

### **Business Rule Language Definition Framework**

The Business Rule Language Definition Framework (BRLDF) provides a mechanism for defining and customizing a business rule language. With the BRLDF, languages can be expressed in a BNF-like format using XML, and transformed into executable rules using XSLT or Java.

### **4.3.3 The Rule Engine Prototype**

The rule engine prototype was part of research projects carried out at the department of STS at TUHH, Hamburg. For an in-depth and detailed understanding of the prototype rule engine refer to the Analysis and Design of a Rule-based System for Process Support in a Maritime Information Portal [Gi02]. Rule in the prototype are independent hence IECA. However the prototype does not encompass all aspects of a rule management system hence there is a need to build a new system with, in addition, a GUI Editor, an integration component, and a customized GUI for developing rules in the Ship Survey Process. A case study of a rule management system is described in details in chapter five.



## 5 Case Study: An IECA Rule Management System

The following chapter discusses the identified concepts of rule engine component, integration of the rules management system in the WIPS IT 1 portal and rules management GUI for WIPS IT1 portal.

### 5.1 Rule Engine

In the previous section, analysis of the requirements has finally led to the design and implementation of a rule engine and management GUI specifically customized for the usage within the WIPS IT 1 Portal in the ship survey process.

The discussion of rule engine is necessarily brief. It is based on a rule engine that has been designed as one of the research projects carried out at the department of STS at TUHH, Hamburg. For more information on rule engines, refer to Analysis and Design of a Rule-based System for Process Support in a Maritime Information Portal [Gi02].

#### 5.1.1 Evaluation Context and Variables

The evaluation context associates current execution of rules with working memory that contain working memory. The evaluation context comprises of variables, which have a qualified names and are bound to a value, which is used for evaluation at runtime. For example, if the ship survey process involves a ship named *Pinta*, then *Pinta* will be bound to the variable named *Ship.Name*. Variable values can be set and retrived using `setVariable()` and `getVariable()` methods. For more information on evaluation context and variables, refer to Appendix A: Rule Engine API. The evaluation context and variables is illustrated in the figure 5-1 below.

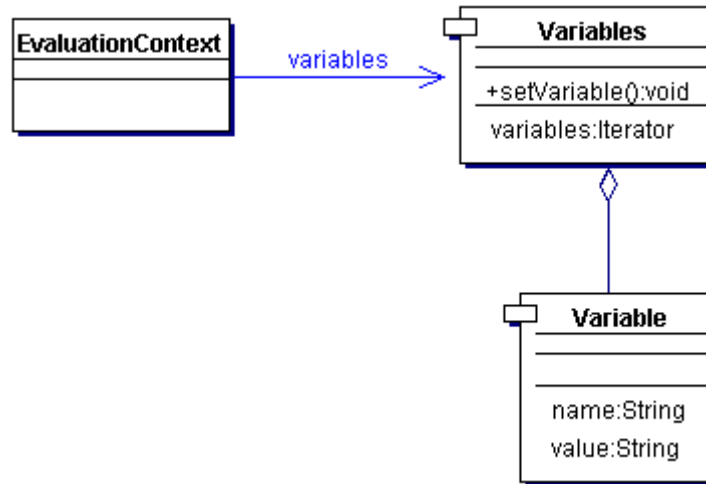


FIGURE 5-1 CLASS DIAGRAM FOR EVALUATION CONTEXT AND VARIABLES

### 5.1.2 Rules

A **Rule** is the core of a Rule Engine. It consists of a **main condition**, an **action**, and an **enabling condition** parts. Likewise the process of pattern matching sets out to find whether or not the **condition** in a rule is satisfied. **Rule** can be represented in an object model as shown in Figure 5-2 below.

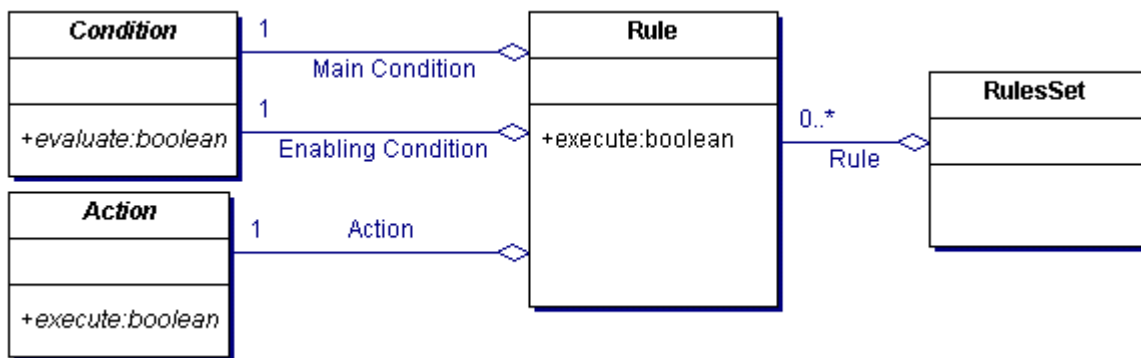


FIGURE 5-2 CLASS DIAGRAM FOR RULESET

Related rules can be grouped together to form a rule set for segmented control over definition and maintenance, conditional execution, reuse, and ease of management and documentation. Rulesets limits the scope of potential interaction between rules and reduces the number of rules that need to be tested for potential conflicts. Rulesets are also separate maintenance and management of different functional aspects of business operations.

### 5.1.3 Conditions

Patterns are commonly referred to as conditions. As a loose definition, we might say that a pattern is a coded expression that is capable of designating one or more objects of that kind. Each condition implements the method evaluate that returns a boolean value. Condition is implemented

as an abstract class, and each concrete subclass defines a specific condition types. Conditions type include expression (simple condition), a negated condition, and two conditions joined by an ‘AND’ or an ‘OR’ boolean operator. *Condition* is designed in the WIPS IT1 rule engine as shown in Figure 5-3 below.

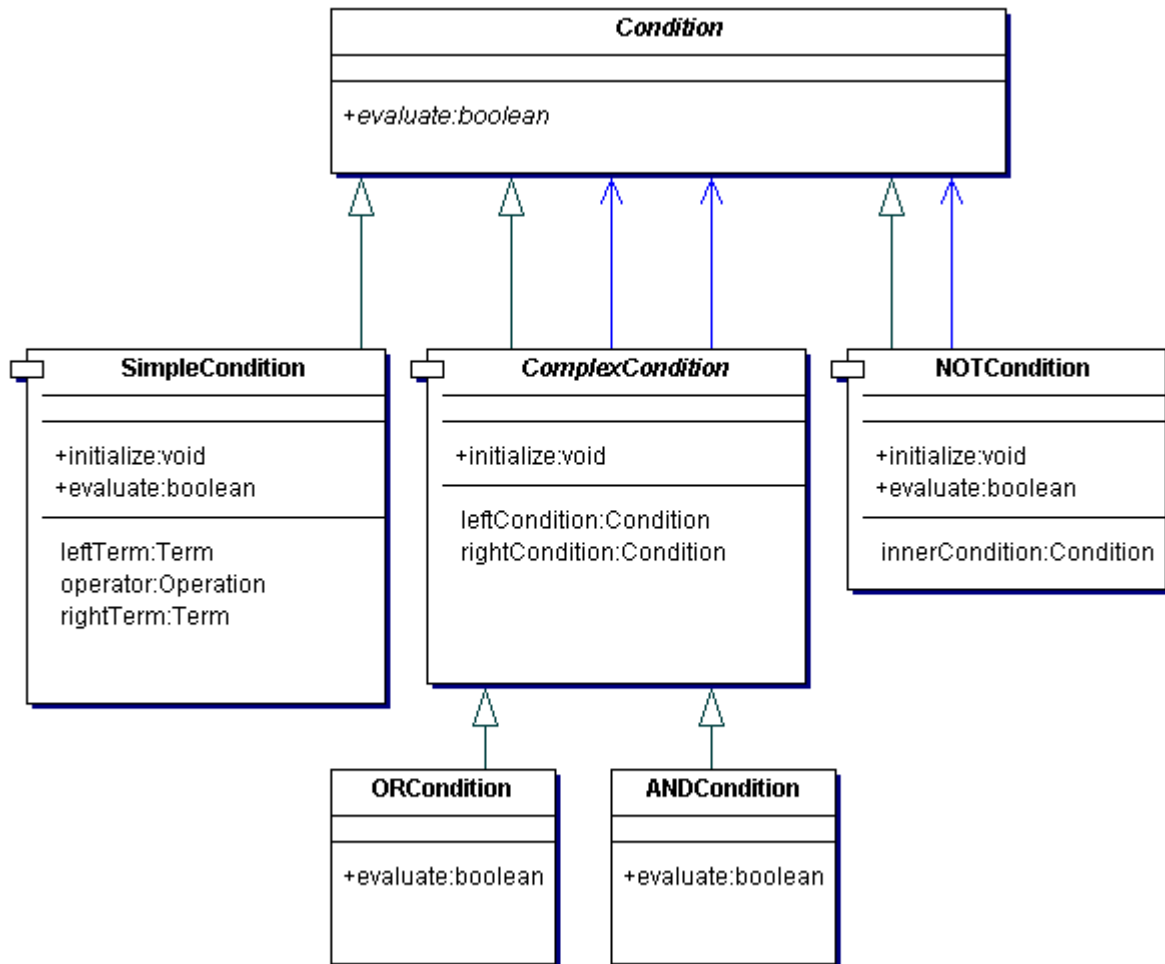


FIGURE 5-3 CLASS DIAGRAM FOR CONDITION AND RELATED CONCEPTS

### 5.1.4 Actions

Actions are description of operations to be performed whenever a rule’s condition is evaluation to true. As mentioned earlier the business logic should be independent from the mechanisms used to manipulate data or implement decisions. The action object model of Actions is illustrated in the figure 5-4 below.

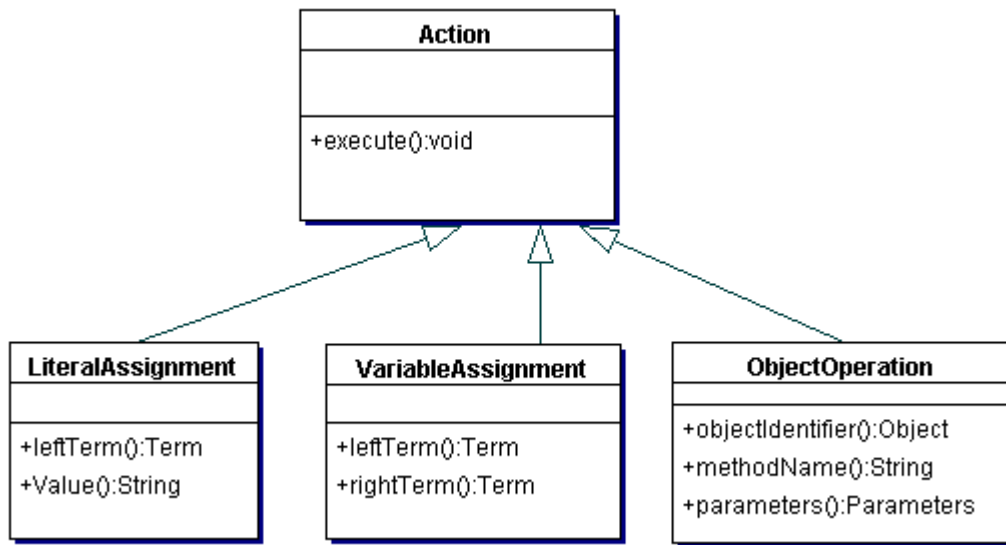


FIGURE 5-4 CLASS DIAGRAM FOR ACTIONS AND RELATED CONCEPTS

The different action types are defined below.

### Literal Assignment

Literal assignments define assigning values to a left term (leftTerm). The left term can be object attributes or context values. For example:

```
Ship.Name = "Titanic"
```

### Variable Assignment

Variable assignment is identical to literal assignment except that the left term (leftTerm) and right term (rightTerm) are variables. For example:

```
Ship1.Hull = Ship2.Hull
```

### Object Operation

An object operation refers to executing an object's (identified by objectIdentifier) method (named methodName) and passing parameters if any. It is easiest to understand this concept by looking at an example from the WIPS inspection process. Assume we have survey items asset container SurveyItems and a check list container CheckList as shown in the diagram below.

```
SurveyItems.createSurveyItem():SurveyItem
```

```
CheckList.addSurveyItem(surveyItem:SurveyItem):void
```

An object operation involving the above objects will have `CheckList` as object identifier, `addSurveyItem` as methodName and `SurveyItem` as the parameter. The rules management is independent of the method implementation. The method may have zero, one or more parameters depending on type of action to be performed. For example such an action may be starting an external mechanical task that require no parameter.

For more information on rule execution and condition evaluation, refer to Appendix A: Rule Engine API.

## 5.2 Implementation Platform and Implications

### 5.2.1 InfoAsset Broker Software as Development Platform

The *InfoAsset Broker* provides a system platform for the construction of enterprise portals and corporate knowledge portals [Info01].

*InfoAsset Broker* is used as the development platform because the WIPS IT 1 portal is realized with this system software. The *InfoAsset Broker* is described in more details in the [Info01].

### 5.2.2 Implications of using InfoAsset Broker Software as Development Platform

The *InfoAsset Broker* is web-based platform-independent server software and is the platform for the WIPS online portal where rule-based process support shall be modelled and implemented. However the version of the *InfoAsset Broker*, which was available at the time of implementation, did not support subclassing of business object types. The strategy pattern was used to overcome this problem.

#### The Strategy Pattern

The Strategy Design Pattern basically consists of decoupling an algorithm from its host, and encapsulating the algorithm into a separate class. More simply put, an object and its behaviour are separated and put into two different classes. This allows one to switch the algorithm that one is using at any time.

The use of the strategy pattern provides an alternative to defining subclasses with different behaviours. When subclassing an object is used to change its behaviour, that behaviour that it executes is static. To change what the object does, a new instance of a different subclass has to be created and replace the original object with it. With strategies, however, switching the object's strategy will immediately change how it behaves.

In WIPS IT 1 portal, single business object class is defined for several business object types and delegate the behaviour to transient, non-business objects. Thus one business object can represent several business object types Figure 5-5 shows how the strategy pattern is realized in the WIPS IT 1 online portal.

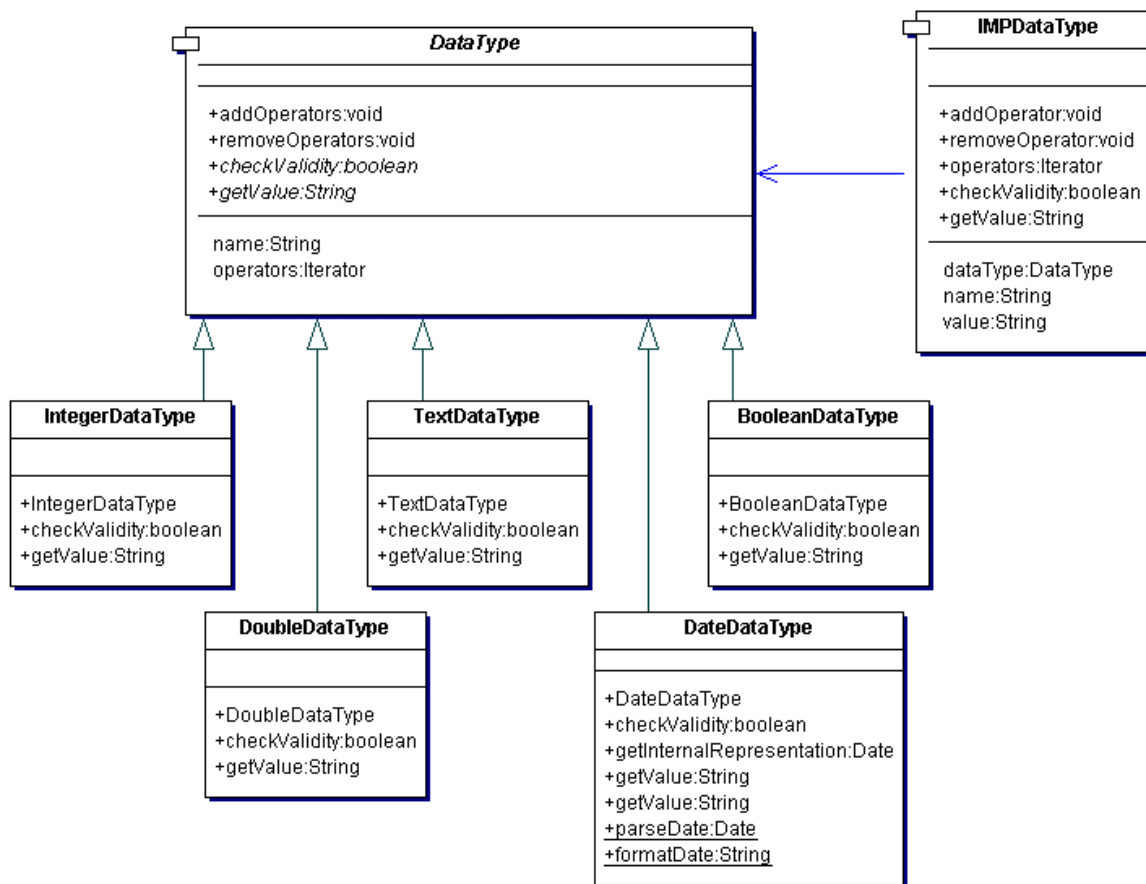


FIGURE 5-5 THE UML CLASS DIAGRAM FOR DATATYPE AND RELATED CONCEPTS

The dataType attribute in the IMPDataType class above is *transient*. The dataType field is set before IMPDataType Object can be used. The methods checkValidity() and getValue() correspond to their respective methods of a specific DataType.

### 5.2.3 Rule Management Integration into the WIPS IT1 Portal

The rule management technology aims at decoupling rules from the rest of the application code. To achieve this goal a rule engine API is provided. For more information on the rule engine API, refer to Appendix A. Apart from rules; the rule engine API describes how independent conditions are created modified and evaluated.

## 5.3 Applications of Rule Management System in the WIPS Scenario

The survey item generation in the ship survey process and support for process management were the main application areas of rules management system identified in the WIPS scenario.

### 5.3.1 The Ship Survey Process

The business process “*ship survey*” includes physical checking of different *parts* of ships at some frequent interval of time for the running vessels or when a new ship is constructed, on the basis of ship surveys, fitness certificates for ships are granted, without this *fitness*

*certificate* no ship is allowed to carry out its activities. The German Lloyd (GL), the ship classifier, administrates ships surveys.

### Process Description

All ship's parts (e.g. hull, machinery, etc.) must be surveyed, that is physically inspected, either on a regular basis (for the ship's main parts), or, if some alterations have been made to a ship, these changes have to be pre approved by the company ship classifier. Surveys are structured into survey types. Each survey type consists of a survey aspect (like the machinery, or hull) and a survey occurrence (annual or renewal). As ships usually are not built in mass production but a single ship at a time usually, only a specific list of survey types applies to each ship.

Survey types are further broken down into survey items, which are much more detailed and specific to ship types. The inclusion of survey items per ship inspection is ship dependent of the ship type. Defining which survey items to include in a survey type can best be expressed by rules that take into consideration the type of survey and the ship type. The rules management system allows rule editors to define rules that specify conditions corresponding actions. The users, e.g., the ship inspectors, can use the rule management system to create a list of survey items for the ship they shall inspect.

### 5.3.2 Support for Process Management

The rules management system supports process management in the WIPS IT1 portal by:

- Providing support for branching decisions
- Providing support for pre and post condition for execution of an activity
- Supporting transition based on condition evaluation

The process management system [Ah03], use conditions as a prerequisite in executing an activity and in determining transition from one activity to the next as shown in the figure 5-6 below.

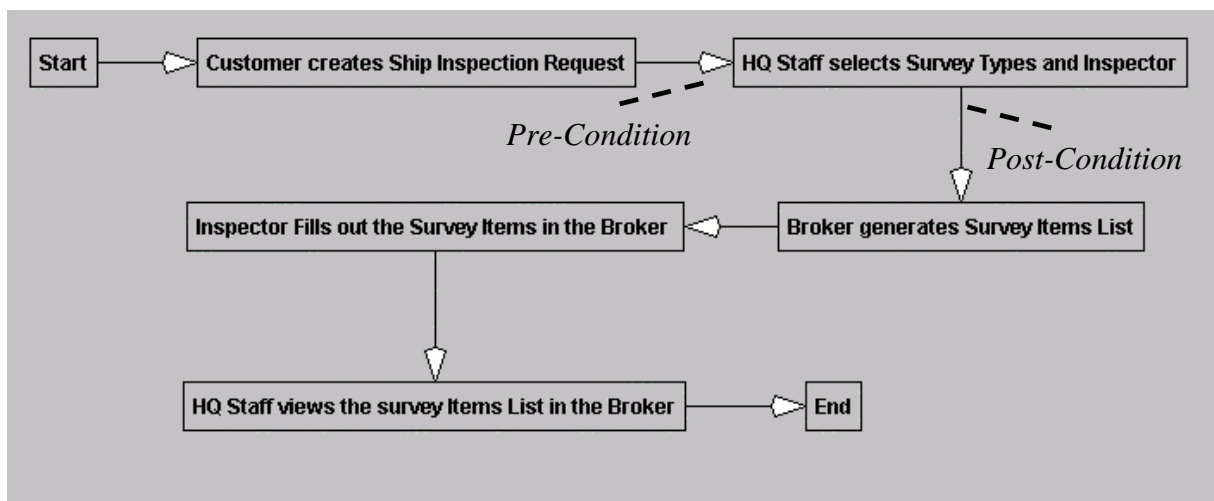


FIGURE 5-6 EXAMPLE OF USE OF CONDITIONS IN PROCESS MANAGEMENT

The process engine of the process management system uses an evaluation context to define variable bindings and then evaluates conditions against the evaluation context. For more information on process management system, refer to Prototype of Software Components for Process Support in an Enterprise Information Portal [Ah03].

### 5.4 Rules Management GUI for WIPS IT 1 Portal

Generally, the rules management GUI should provide an interface that allow rules to be safely constructed and modified by the rule editors with customized visualization, filters and input controls. It should completely hide the structured rule language and the programming involved. Editors should be able to understand the rules without interpretation from the IT specialists.

For example, in a ship inspection process, rules editors may not understand an object operation, which has the following structure: `<object>.<method>(< parameter > [ ,<parameter>])` (see section 5.1.3 for more information on actions). Presenting the rules like in table 5-1 below was most appropriate according to editors in the WIPS scenario.

Rule Identification	Enabling Condition		Main Condition	Add Survey Item
	Start Date	End date		
1001	01-Jan-2001	31-Dec-2005	Class == 100A5 AND (ShipType == LiquefieldGasTanker OR ClassNotation == LiquefieldGasTanker)	SI 11.1
1004	01-Jan-2001	31-Dec-2002	Class == 100C1 AND ShipType == LiquefieldGasTanker	SI 14.1

TABLE 5-1 THE GUI FOR THE WIPS RULE EDITOR

The mapping of the GUI objects to the rule engine objects is done by the rules management system relieving rule editors from understanding the rule language structure. The first rule example in table 5-1 would be interpreted as follow.

```

Rule Identifier      : 1001

Enabling Condition  : if Today IN
                    [01-Jan-2001 .. 31-Dec-2005]

Main Condition      : if Class == 100A5 AND
                    (ShipType == LiquefieldGasTanker OR
                    ClassNotation == LiquefieldGasTanker)

```

Action : Add Survey Item SI 11.1  
to the check list.

The following section describes the rules management GUI specifically designed to meet the requirements of rules management system of Ship inspection process for the WIPS IT 1 portal.

### 5.4.1 User Roles and Use Cases

The following roles have been identified in the WIPS scenario related to rule management.

#### 5.4.1.1 The Rule Editor

An Editor refers to the person responsible for rule creation and modifications. Business rules may be found in policy manuals, and regulatory mandates. New rules are especially added as a result of regulatory change.

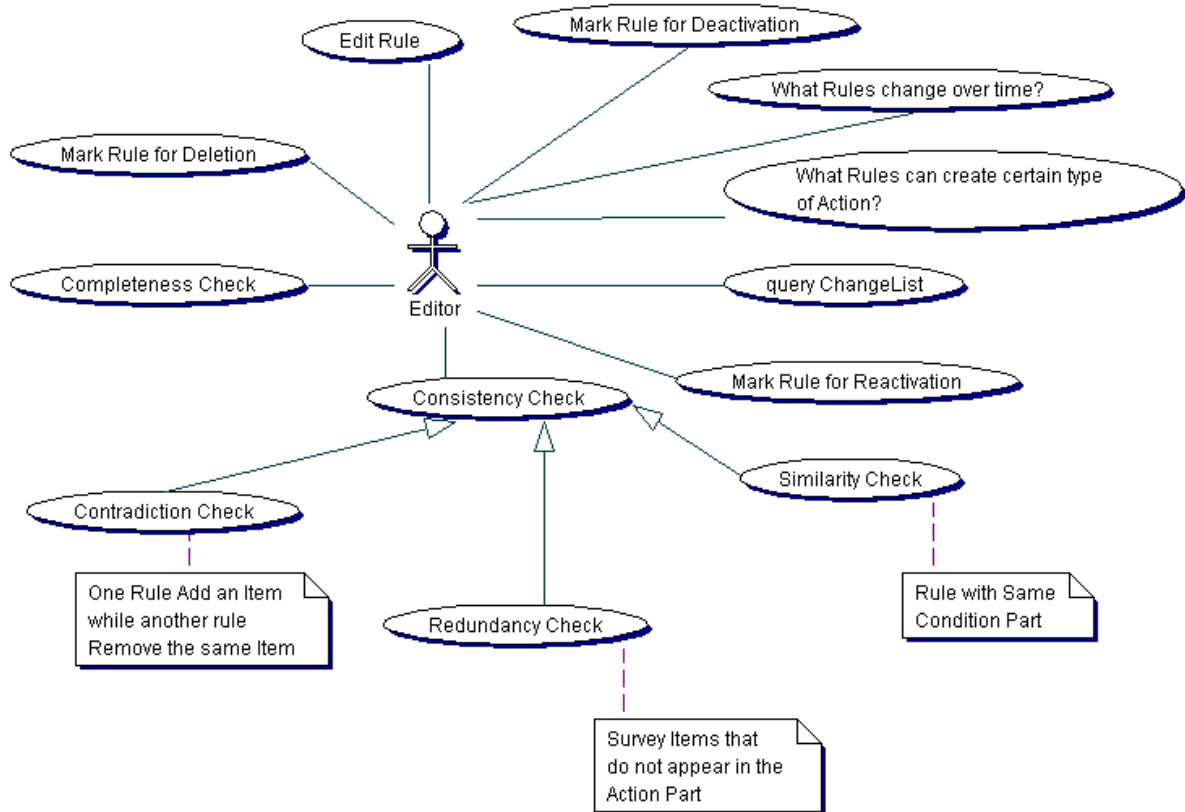


FIGURE 5-7 EDITOR’S USE CASES

#### Completeness Check

Completeness check refers to checking result set against predefined result set. It reports:

- a) Survey items that are generated by the rule engine for the tested rule set and do not appear in the predefined list and

- b) Survey items that are in the predefined list but are not automatically generated by the Rule engine when the rule set is tested.

### Redundancy Check

Redundancy check involves checking for survey items that do not appear in the Action part of any Rule thereby making them redundant, as they will never be generated.

### Similarity Check

Similarity check involves checking for rules that have same condition part.

### The Rule Editor's GUI design

A GUI implementation of the editor's use cases can be summarized in the figure 5-8 below

**Rules**

**Filter Rules:**

Show only rules for: Action part contains "SI 2122" action

Show only rules with: Class == 100A5 Condition

Show rule identified by: 1001 Search

[Show all rules](#)

**Changing Rules:**

Changes on 01-Jan-1970 (date) compared to (date);

[Show](#) Rules that become valid or invalid

3 4 5

<a href="#">Delete</a>	Sort By Start Date   Sort By End Date Sort By Start Date and End Date			<a href="#">Create new Rule</a>		
<input type="checkbox"/> All	<a href="#">Status</a>	<a href="#">Rule Identifier</a>	<a href="#">Enabling Condition</a>	Rule Condition	<a href="#">Action</a> (Add Survey Items)	<a href="#">Edit</a>
<input type="checkbox"/>	Active	1001	Valid From 01-Feb-2002 To 31-Dec-2002	Ship.Country == USA	SI 192,SI 329	<a href="#">Edit</a>
<input type="checkbox"/>	Active	1002	Valid from 01-Jan-1900	Class == 100A5 AND (ShipType == LiquefieldGasTanker OR ClassNotation == LiquefieldGasTanker)	SI 11.1	<a href="#">Edit</a>

6

FIGURE 5-8 THE RULE MANAGEMENT GUI FOR THE RULE EDITORS

The management GUI provides functionality for creating (link marked 5), editing (link marked 6), querying (link marked 1 & 2), of rules. By using the drop down list (marked 3) selected rules can be deleted, and deactivated. Displayed rules can be sorted using the column heads or by date (link marked 4).

### 5.4.1.2 The Rule Verifier

Once a rule is added or modified, it has to be approved before it become active into the system. A verifier is responsible for approving new or modified rules. Like the rule editor, the verifier can also perform completeness, contradiction, redundancy, and similarity checks.

The rule verifier's use cases can be summarized in the figure 5-9 below

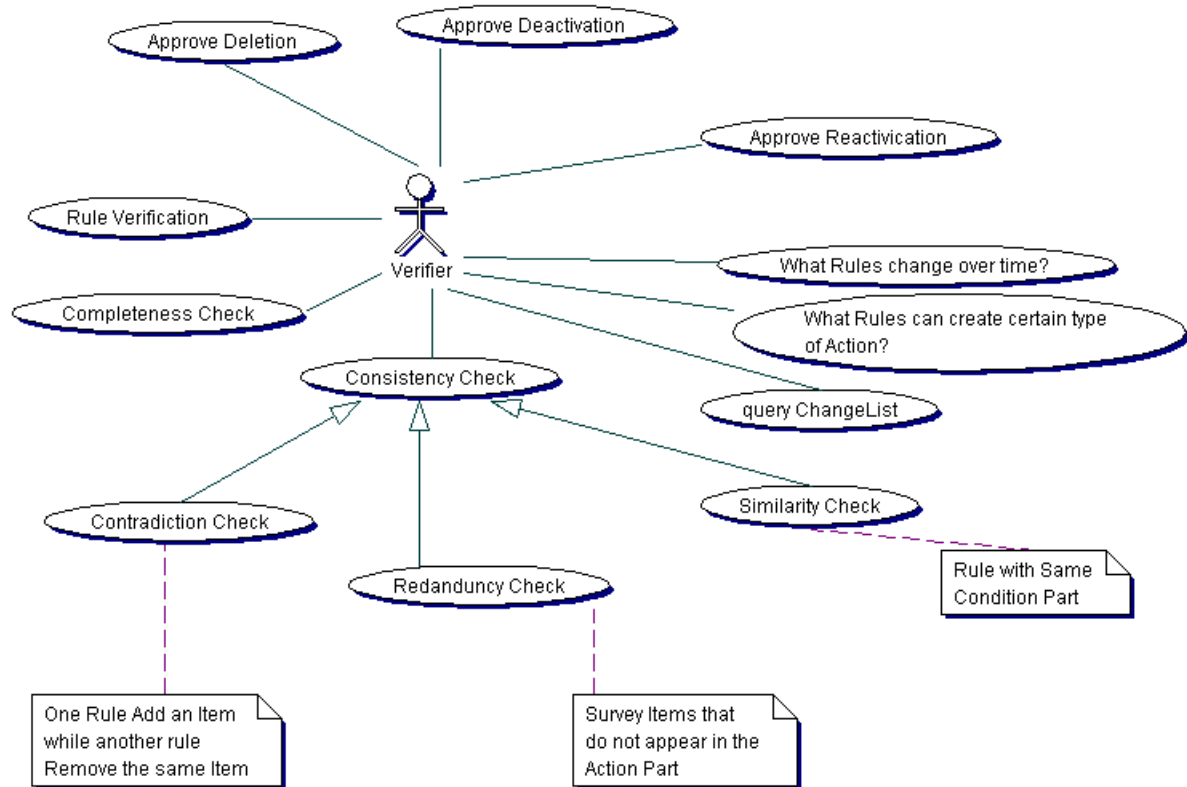


FIGURE 5-9 VERIFIER'S USE CASES

### 5.4.1.3 The Inspector

Inspectors are based in different ports all over the world and conduct physical inspection of the ships. They write reports of the survey, based on their observations after inspection of ships and send their reports to the main office of the ship classifier.

The inspector can generate checklist, add or remove survey items from the checklist, and request for a checklist. For easier viewing of rules, the inspector can order rules by classification or where they apply as well as by the survey item they generate. The inspector can also query the system for rules that change over time and can find rules that generate a certain survey item. Finally the inspector can request for a report of rules that have changed by querying the changelist. Unlike the rule editor, the inspector can only perform completeness check.

The inspector's use cases can be summarized in the figure 5-10 below

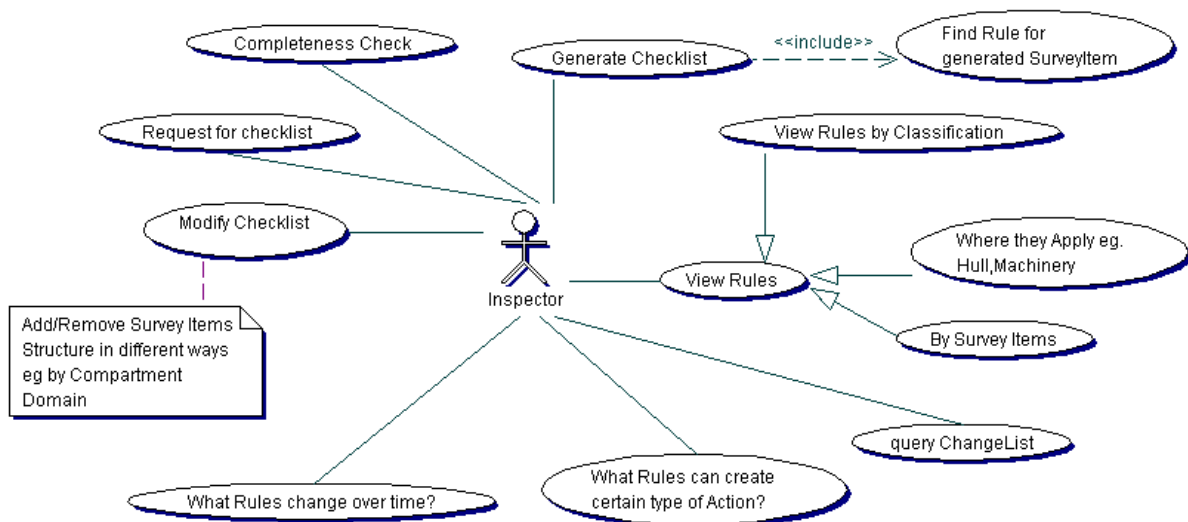


FIGURE 5-10 INSPECTOR'S USE CASES

### The Ship Inspection

The inspector will receive from ship classifier the following:

- The list of selected survey types,
- A rule set and
- The main ship data.

Inspector will generate from the survey type list, a list of survey items. Survey items are “atomic” inspection items, i.e. these cannot be inspected partially and they are much more detailed than the survey types.

The survey items are generated from the survey types by running a rule evaluator on the rules. Depending on the ship data and the selected survey types, the rules define what survey items will be included in the survey item list (e.g. If the ship class notation is 100A5 standard AND the annual machinery inspection is selected, the survey item 1703 will be added to the survey item list).

The inspector will verify the survey item list, and, if necessary, will remove survey items or add further survey items to the list. He will then print out the list and inspect the ship physically.

After the inspection of the ship, the inspector consolidates the survey item list and, depending whether the survey succeeded, creates a temporary certificate from the survey item list. He gives the temporary certificate to the ship captain. If the survey did not succeed, the ship captain must initiate the required repairs. Without certificate, the ship may not leave the harbour.

The inspector writes a report and sends his report (stating also costs), and the filled out survey item list to the main administration office of the ship classifier.

#### 5.4.1.4 Notification Service

Rule changes are propagated to rule editors, inspectors and ship owners through a notification service. The Rule editors, inspectors and ship owners subscribe to a notification service with parameters (what to subscribe for, frequency of updates). The rule changes include rules becoming active or inactive and introduction of new rules as shown in the figure 5-11 below.

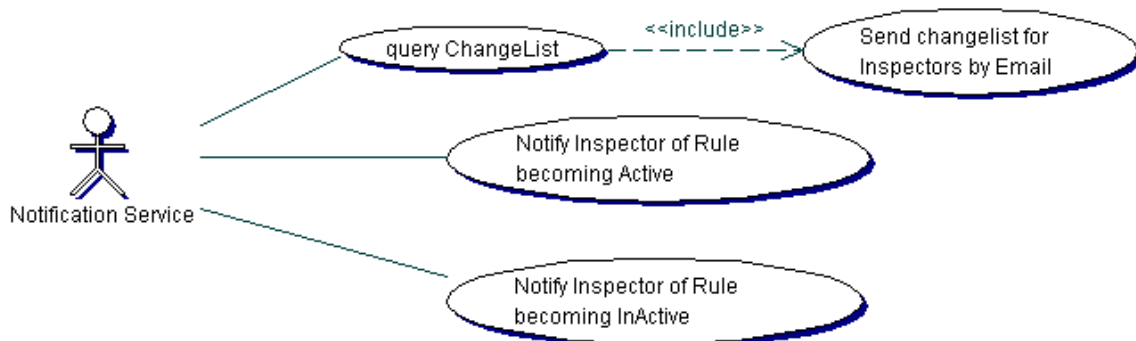


FIGURE 5-11 NOTIFICATION SERVICE USE CASES

## 5.4.2 Life Cycle of Rules and Rule States

Rules can be created, modified, deactivated in the rule management system as illustrated in the figure 5-12 below.

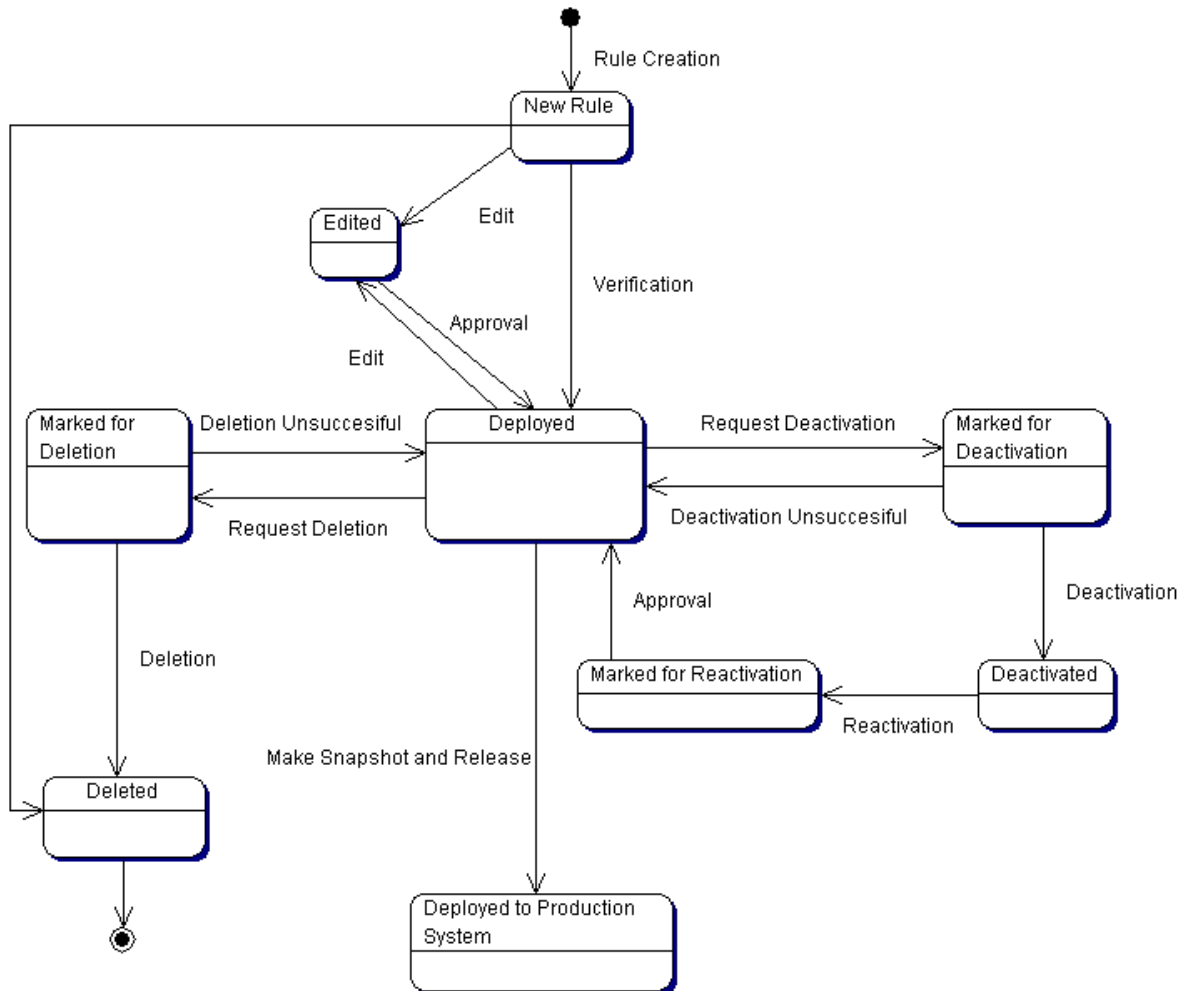


FIGURE 5-12 RULE STATE DIAGRAM

### Rule Creation

The creation of rules is done by rule editors. A new rule is available for editing but has to be approved for deployment.

### Editing Rules

Editing rules is the modification of the main condition, the enabling condition or the action part of a rule. To keep track of changes, only new or deployed rules can be edited. Deactivated rules must be reactivated before they can be modified.

### **Deactivation and reactivation of Rules**

A rule can be manually or automatically be deactivated. For example, a rule is automatically deactivated on 1 January 2005 if it is time constrained to function between 01 January 2001 and 31 December 2004. An editor may manually deactivate a rule especially when the regulatory changes making some rules invalid e.g. a rule that applied to all ships manufactured in North Korea may no longer be valid. Rule verifier may reactivate a manually deactivated rule.

### **Deletion of Rules**

Rules that are no longer in use in the system can be removed from the system by deletion.

### **Deployment of Rules**

Rules are deployed into the production system by making a snapshot of the deployed rules in the development system.



## **6 Summary and Outlook**

In this chapter, a summary of the work and further works of this research is presented.

### **6.1 Summary**

In this report, an introduction to rule management from the enterprise perspective was presented. First, the benefits of rule management technology, including: technical independence; faster application development; better quality requirements; ease of change; and a balance between flexibility and centralized control was presented. Secondly, after describing the key concepts of rule management, rule languages complexity and expressiveness was discussed. Then, a concrete rule management system architecture, which satisfies the requirement of decentralization, independence, and flexibility, was presented. In addition, need to build a customized, user-specific, domain specific rule management GUI, was demonstrated. Finally, some relevant aspects of the current prototype implementation of the rule management system customized to meet the requirements of ship survey process were given.

### **6.2 Outlook**

Since the rule management technology described in this report is not only appropriate to provide support for business people to manage application without interpretation from IT staff, but might obviously extend its application in a very dynamic manner, more practical research seems to be necessary to explore the potential as well as risk of such rule-sensitive applications. Moreover, the implemented prototype is not only restricted to ship survey process, but also applicable to software components (services) in general. Therefore, there is a need to extend and improve the current prototype, for example ability to evaluate complex rules and interoperate with existing rule management systems using XML rule language standards. Since the rule engine is specifically designed and customized to meet the requirements of ship survey process, the rule engine cannot handle the expressiveness of the language. Therefore further research is needed to identify rule sets that are outside the engine's scope and delegate the evaluation of rules to a rule engine that can handle higher expressiveness.



## Appendix A: Rule Engine API

The WIPS IT 1 Rule Engine interfaces is located in the package `de.tuhh.sts.ruleEngine.interfaces.*`.

### Adding and Removing Conditions

You can use the methods of the class `services.getConditions()` to dynamically add/remove conditions to/from your system.

### Creating an Expression

An Expression is a simple condition that consists of a Left Term, an Operator and a Right Term. An Expression can be created by calling the `createExpression()` method. The left and right term must be set by calling the methods `setLeftTerm(term)` and `setRightTerm(term)` respectively. A Term can be a date, string, integer, double or a boolean value. The operator of the expression is set by using the `setOperator(operator)` method. The operator can be “=”, “<=”, “>=”, “!=”, “<”, “>”.

See the class `condition` in the *WIPS Rule Engine Documentation* for more Information.

### Creating an AND Condition

You can add an AND condition to your system by using the method `createANDCondition(leftCondition, rightCondition)`. If the operation succeeds, a new AND condition is returned.

See the class `conditions` in the *WIPS Rule Engine Documentation* for more Information.

### Creating an OR Condition

You can add an OR condition to your system by using the method `createORCondition(leftCondition, rightCondition)`. If the operation succeeds, a new OR condition is returned.

See the class `conditions` in the *WIPS Rule Engine Documentation* for more Information.

## Creating a NOT Condition

You can add a NOT condition to your system by using the method `createNOTCondition(innerCondition)`. If the operation succeeds, a new NOT condition is returned.

See the class `conditions` in the *WIPS Rule Engine Documentation* for more Information.

## Naming a Condition

You can name a condition in your system by using the method `setConditionName(ConditionName)`.

## Removing a Condition

You can remove a condition in your system by using the method `remove(ConditionID)`. However use it with caution!

## Displaying the Condition expression(s)

You can display the condition expression(s) in your system by using the method `printExpression()`. If the operation succeeds, a string representation of the condition is returned.

See the class `condition` in the *WIPS Rule Engine Documentation* for more Information.

## Variable Binding

You can use the methods of the class `de.tuhh.sts.ruleEngine.interfaces.variables` to dynamically add/remove variables to/from your system.

### Variable Names and values.

Variable names refer to qualified names of variables whose values are determined at runtime. Variable values are always represented in their string representation.

There are two ways by which one can set Variables.

### Variable Name / Value pair binding.

You can set the value of a variable name directly by using the method `addVariable(qualifiedVariableName, value)`. If the operation succeeds, a variable with the said name is set with the corresponding value. You must always provide a qualified name. Otherwise if a variable with the said name exist, its value will be overwritten. Nevertheless a warning message is added in the Variable Debugger message list.

See the class `Variables` in the *WIPS Rule Engine Documentation* for more Information.

**Variable names set from Objects.**

These are runtime Object property values. The **variable Terms** must be defined in the **Terms asset container**. Variable values are set by using the method `setVariables(objectNamePrefix, actualObject)`. The values of **variable Terms** defined in the **Terms asset container** are set to their corresponding string representation of the object property. Debugger messages can be found in the Variable Debugger message list.

**Variable Debugger message list.**

You can get Exception, error and warning messages that occurs when performing any of the variable operations by using the method `getDebugger()`.

See the classes `Term` and `Debugger` in the *WIPS Rule Engine Documentation* for more Information.

**Evaluating a Condition**

You can evaluate a condition by using the method `evaluate(evaluationContext)`. The condition evaluates to a boolean value `true` iff the attributes of the condition are set correctly and evaluates to `true` depending on the values of left and right terms and the operation performed. Otherwise the condition evaluates to a boolean value `false`. The parameter `evaluationContext` associates conditions with working memory. It keeps reference to objects necessary for condition evaluation. In particular, it keeps reference to `Variables` object.

See the classes `EvaluationContext` in the *WIPS Rule Engine Documentation* for more Information.



## Appendix B: SurveyItems Generation Language Grammar

RuleSet	:= Rule*
Rule	:= Condition Action
Action	:= LiteralAssignment   VariableAssignment   ObjectOperation
LiteralAssignment	:= Term = LiteralAssignValue
LiteralAssignValue	:= Value   Term
VariableAssignment	:= Term = Term
ObjectOperation	:= ObjectIdentifier.MethodName(Parameter [,Parameter]*)
NameToken	:= [A-z] {[A-z   0-9]*}
ObjectIdentifier	:= NameToken
MethodName	:= NameToken
Parameter	:= ElementTerm
Condition	:= { NameToken } BooleanExpression   NOT Condition   Condition LogicalOperator Condition
LogicalOperator	:= AND   OR
BooleanExpression	:= ElementTerm ComparisonOperator ElementTerm   ElementTerm InclusionOperator ElementTerm
ElementTerm	:= Literal   Variable
SetTerm	:= Enumeration   Range
ComparisonOperator	:= '='   '!='   '>'   '>='   '<'   '<='
InclusionOperator	:= 'IN'
Term	:= ElementTerm   SetTerm
Literal	:= Value
Variable	:= <i>String</i>
Enumeration	:= '('ElementTerm '['ElementTerm']*')
Range	:= '['LowerBound '..' UpperBound']
LowerBound	:= ElementTerm
UpperBound	:= ElementTerm
Value	:= <i>Integer</i>   <i>String</i>   <i>Boolean</i>   <i>Double</i>   <i>Date</i>



## Bibliography

- [Ah02] Ayaz Ahmed, Design of a Process Model for a Cooperative Information System, Student project, Technical University Hamburg-Harburg, February, 2002.
- [Ah03] Ayaz Ahmed, Prototype of Software Components for Process Support in an Enterprise Information Portal, Master Thesis, Technical University Hamburg-Harburg, February, 2003.
- [Blaze02] The Business Case; Technical white paper, issued by the Blaze Advisor From HNC <http://www.blazesoft.com>; September 2002.
- [BRG02] The Business Rules Group <http://www.businessrulesgroup.org>; December 2002.
- [CLIP97] CLIPS Application Abstracts, issued by the ILOG S.A, France, issued November 1997. <http://www.ghg.net/clips/CLIPS.html>; April 2002.
- [CLIP98] CLIPS User's Guide by Joseph C. Giarratano, Ph.D., August 1998 <http://www.ghg.net/clips/CLIPS.html>; April 2002.
- [Fedex02] The Business Rules Group <http://www.fedex.com>; December 2002.
- [Fo82] Charles L Forgy. Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem. Addison-Wesley, 1982.
- [GaFi02] Knowledge Management and knowledge Automation Systems, Technical white paper, issued by the Gallagher Financial Systems, Inc. FL, USA, issued 2002. <http://www.gogallagher.com>; April 2002.
- [Gi02] Gichahi, Henry Kamau, Analysis and Design of a Rule-based System for Process Support in a Maritime Information Portal, Student project, Technical University Hamburg-Harburg, May, 2002.
- [GiRi93] Giarratano, J and Riley, G. Expert Systems: Principles and Programming. Second Edition, PWS Publishing, Boston, 1993.

- [GL02] Germanischer Lloyd AG. [www.germanlloyd.org](http://www.germanlloyd.org); May 2002.
- [Halley01a] Agile Business Rule Processing, Technical white paper, by The Haley Enterprise, Inc, PA, USA, issue in 2001. <http://www.haley.com>; April 2001.
- [Halley01b] Eclipse:Knowledge Automation through Rule-Based Programming, Technical white paper, by The Haley Enterprise, Inc, PA, USA, 2001. <http://www.haley.com>; April 2001.
- [ICS02] School of Information & Computer Science, Design and Analysis of algorithms, University of California, Irvine. <http://www.ics.uci.edu/~eppstein/161/960312.html>; September 2002.
- [ILOG01] The ILOG JRules 3.5, User's, Manual issued by the ILOG S.A, France, issued June 2001. <http://www.ilog.com>; April 2002.
- [ILOG02a] Business Rules; Powering Business and e-Business, Technical white paper, issued by the ILOG S.A <http://www.ilog.com>; April 2002.
- [ILOG02b] The ILOG Business Rules; Development Tools, Technical white paper, issued by the ILOG S.A <http://www.ilog.com>; April 2002.
- [Info01] The InfoAsset Broker, Technical white paper, issued by the Info Asset AG, Hamburg, Germany, issue date: 7<sup>th</sup> April 2001. <http://www.infoasset.de>; April 2002.
- [McW01] Bob McWhirter. The Rete-OO Algorithm for inference Rules in Object-Oriented Language Systems, issued November 2001. <http://www.werken.com>; April 2002.
- [MüHu01] Reference Slides about the *Info Asset Broker [Info01]*, Provided by the Software System (STS), TUHH. Authors: Patrick Hupe and Rainer Müller, November 2001.
- [NoJo94] Expert Systems, Term project by John C. Norton, issued Dec. 1994, <http://www.eng.rpi.edu/dept/chem-eng/Biotech-Environ/EXPERT/project.html>; April 2002.
- [Prolog03] <http://cbl.leeds.ac.uk/~paul/prologbook/> (Leeds University); January 2003.
- [PUTSP01] <http://www.math.princeton.edu/tsp/index.html> (Princeton University Travelling Salesman Problem-TSP page); September 2002.
- [RuleML01] Rule Markup Language <http://www.dfki.uni-kl.de/ruleml>; September 2002.
- [SICS03] SICStus Prolog. <http://www.sics.se/SICS-reports/SICS-T--93-01--SE/>; February 2003.
- [W3C01] <http://www.w3.org/2001/sw/>; September 2002.

- [Wi02a] Wikipedia, Open Content WikiWiki free encyclopedia.  
[http://www.wikipedia.org/wiki/Complexity\\_classes\\_P\\_and\\_NP](http://www.wikipedia.org/wiki/Complexity_classes_P_and_NP); September 2002.
- [Wi02b] Wikipedia, Open Content WikiWiki free encyclopedia.  
[http://www.wikipedia.org/wiki/Chomsky\\_hierarchy](http://www.wikipedia.org/wiki/Chomsky_hierarchy); December 2002.
- [Wi03a] Wikipedia, Open Content WikiWiki free encyclopedia.  
<http://c2.com/cgi/wiki?HornClauses>; February 2003.
- [YASU02] QuickRules Datasheet: Business Rules Engine-Features, by The YASU Technologies, Inc, San Bruno, USA, 2002.<http://www.yasutech.com>; April 2002.