NTNU

Norwegian University of
Science and Technology

Integrated Case Based and Rule Based Reasoning for Decision Support

Azeb Bekele Eshete

Master in Information Systems

Submission date: July 2009

Supervisor: Agnar Aamodt, IDI

Co-supervisor: Pål Skalle, Verdande Technology As (previously Volve As)

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Knowledge based decision support systems can benefit from combining generalized experiences, e.g. in the form of rules, with situation-specific experiences, in the form of past cases. In this thesis architecture for an integrated system, combining CBR and RBR methods, shall be developed, a demo system that instantiates essential parts of the architecture shall be designed, implemented, and tested. The rule based tool Jess shall be used for the rule based part, and a freely available tool shall be chosen for the case based part.

The application domain in this study will be oil well drilling. The KBS group at IDI cooperates with the company Verdande Technology As, which builds software for the oil and gas industry in which case-based reasoning methods are used. For this thesis a sub-problem of oil drilling has to be selected. Cases should be constructed and represented within the chosen CBR tool, partly based on existing cases in Verdande Technology's case base. Rules should be acquired from documents that contain generalized experience in the form of best practice and lessons learned.

Assignment given: 21. January 2009
Supervisor: Agnar Aamodt, IDI

Abstract

This project is a continuation of my specialization project which was focused on studying theoretical concepts related to case based reasoning method, rule based reasoning method and integration of them. The integration of rule-based and case-based reasoning methods has shown a substantial improvement with regards to performance over the individual methods. Verdande Technology As wants to try integrating the rule based reasoning method with an existing case based system.

This project focuses on designing, implementing and testing of a demo system that demonstrates the development of a rule based reasoning component and integrating it with the existing case based system of Verdande Technology As.

Preface

This thesis is done for the course *TDT4900- Computer and Information Science, Master Thesis* as part of a Master degree in Information Systems at the Department of Computer and Information Science at NTNU. Its main goal is to design, implement, and test a demo system that illustrates a rule based reasoning component and integration of it with an existing case based system. The demo system belongs to Verdande Technology As.

In the first place, I would like to forward my gratitude to my supervisor, Professor Agnar Aamodt, for his advice, supervision, and guidance from the very early stage of this thesis until its successful accomplishment. Many thanks go in particular to Pål Skalle who worked a lot on making the rules for my demo system and who provided me valuable information about oil drilling operation. I would also like to thank Frode Sørmo, Tore Brede and all software developers at Verdande Technology AS who helped me by providing important information and resources. At last but not least I would like to thank my husband Abera Hailu who was encouraging and helping me by being beside me throughout my M.Sc. study.

Trondheim, July 8, 2009

Azeb Bekele

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 The Project

This section introduces about what the project is dealing with and the collaborating company. It provides the background and motivation why the collaborating company is interested on this project. It also includes the objective of the project and its scope to identify what will be covered to achieve the project's objective. The result and effect of the project is also mentioned in this section.

1.1.1 Project Title

This project is entitled *Integrated Case Based and Rule Based Reasoning for Decision Support*. It is about integrating the two knowledge based techniques, rule based reasoning and case based reasoning, in order to support a computer system that facilitates decision making activities.

1.1.2 Collaborating Company

This project is done in collaboration with *Verdande Technology AS*. The company was founded in 2004 by a group of students and professors from Norwegian University of Science and Technology. Though it was established with few people, it shows fast development; currently it has about 21 employees.

Verdande's ultimate goal is to be favored provider of knowledge based decision support systems all over the world. The company has developed a

software platform called *Edge* that is designed to reduce risk and cost of complex and critical operations across industries by enabling better and faster decision making. Edge is built on the principle of *case based reasoning technique*; a process of solving new problems based on the solutions of similar past problems. It guides producing of products and services for particular application domain in order to support users to make better decisions more easily. Currently, the company is serving the oil and gas industry with their case based software system which is known as *DrillEdge*.

1.1.3 Background and Motivation

Decision making is a mental process that identifies possible alternative actions and chooses one from the alternatives for the given situation. It requires ability of analyzing available data and information about the situation in order to compare and contrast the alternatives so that the best and right option can be selected. The quality of a decision depends on its rightness and being made in time. Capability of making the right decision is the result of the decision maker's knowledge about the domain, availability of necessary data, and experience & ability of applying the knowledge on the given data. Sometimes lacking of necessary data and the time it takes to analyze huge amount of data prevents from making the right decision.

Computer programs are being developed to analyze business data to provide correct set of relevant information at reasonable time to support decision making activities. Such computer programs are known as *decision support systems*. Their main goal is to provide the right information at the right time in a format that can help decision makers to make a quality decision easily. Making the right decision depends on the data and the ability to interpret and use the data; consequently decision support systems require data and knowledge that determines how the data can be processed to provide the right information. [44], [51]

Knowledge based techniques are playing great role for decision support systems to process raw data and to present the information in a way that can facilitate decision making process. The knowledge based technique provides knowledge about the domain and reasoning skill to produce the right information from the data. The information that is provided by a decision support system to a user may be an information that provides hint or evi-

dence on making a particular decision or an action that a user can follow to handle the problem. [45]

Integration of two or more knowledge based techniques is a very active research area in Artificial Intelligence. Case based reasoning and rule based reasoning are two examples of knowledge based techniques. Their integration has shown significant improvement on a system than it would have been achieved from a system with a single reasoning technique. Examples of integrated systems in [6], [7], [8], [12], [20], [18], [17], [15], [9], [13], [10], testify that the two methods are complement of each other. Each method serves to handle limitations of the other. Their integration increases the competence of the application in handling very complex and various problems and providing accurate solution.

Verdande's products and services are based on the principle of case based reasoning technique on the platform of Edge to facilitate decision making activities. The benefits that are gained from integrated reasoning techniques calls Verdande's attention and motivates to think about improving their product by integrating rule based reasoning technique with the existing systems.

1.1.4 Project Objective

The objective of this project is to research how a rule based reasoning method can be integrated with the existing case based system called DrillEdge in order to improve the system's performance in handling complex situations in simpler and accurate ways.

The research includes studying how they can be integrated, and designing, implementing and testing of a demo system that demonstrate the integration. Since rule based reasoning is new for the company studying and developing a rule based component is part of the project work.

1.1.5 Project Scope

The scope of this project includes designing architecture for the integration of rule based reasoning technique with the existing case based system and developing a demo system based on the architecture. Development of the

demo system should follow a selected software development methodology. The demo system should be tested and evaluated to reflect the role of the rule based reasoning on problem solving process of the existing system.

Researching about required methodologies, technologies and important concepts regarding to the achievement of the project objective is part of the scope. A report needs to be written so that what is required for the integration process, how the demo system is developed, and the result of the integration process will be documented.

1.1.6 Expected Project Result and Effect

The expected result from this project work is:

- A demo system that demonstrates how a rule based reasoning component can can be integrated with the existing case based reasoning system and how it plays the desired role in the integrated system.
- A report that documents the study, design , implementation and testing of the demo system and evaluation of their integration on the achievement of the desired goal.

The effect of this project will be on Verdande Technology As. The company will get tested and evaluated demo system and accompanying report that suggests and demonstrates one way of integrating approach and its result, which can be possibly implemented and expanded on their real product.

1.2 Project Plan

Planning is one of the ways to visualize the tasks that are required to achieve a desired goal. In a software development process, selecting the development methodology that will be followed throughout the software development process is the first step for planning. This project work has development of a demo system hence early selection of a software development methodology helps to plan the overall work of the project. Detailed discussion about software development methodologies are presented in Chapter 1.4: Section 2.6.

1.2.1 Selection of Software Development Methodology

There is no particular model that is best for every project. The complexity, how large the project is, the given time, desired output from the project work and other features of a project matters to select the model that fits best.

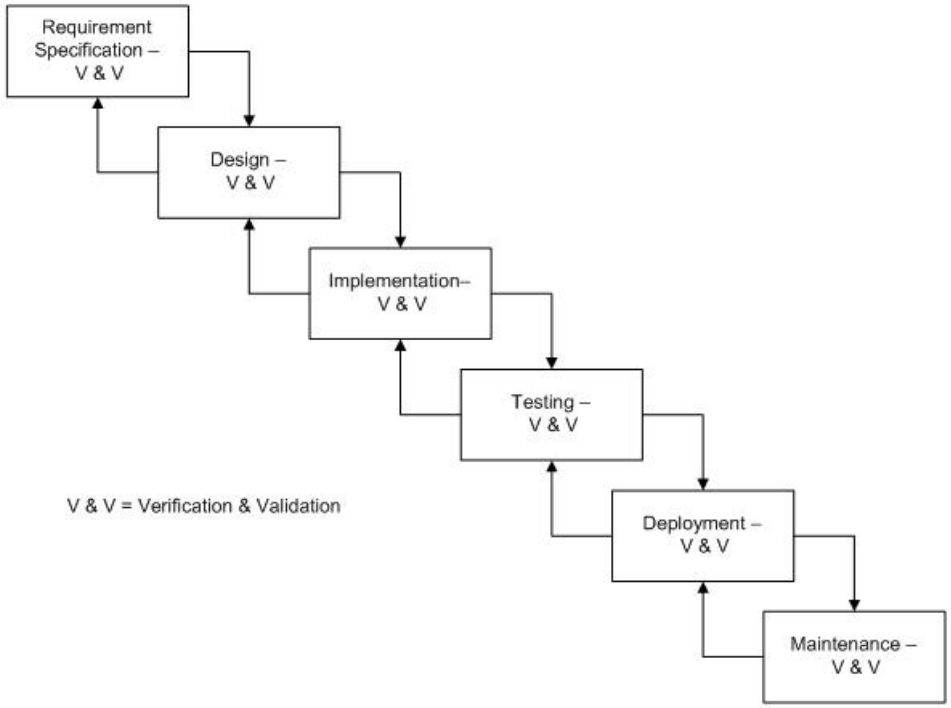

Figure 1.1: Modified Waterfall

From the candidate models discussed in section Chapter 1.4: 2.6.2, I choose waterfall model with some modification. The reason that I don't want to use prototyping or spiral is, the main goal of this project is to integrate a rule based reasoning technique with the existing case based system. Hence, documentation of the development process and a demo system that demonstrate the integration are enough. Producing perfect product is not expected from this project due to time limitation and its complexity to be handled by one person. In addition, it is a school project that focuses on studying their integration not mainly focused on producing the real product.

Some limitations of waterfall are, it's not being flexible in handling changes once the phase is completed. It is tested when the product is finished. To

solve these problems, waterfall model is modified by adding validation and verification in each phase and by letting overlapping of phases. This model is known as *Modified Waterfall Model*. It is depicted in Figure 1.1. Each phase is validated and verified; if there is need of some change, it is possible to make the change on previous phases as well.

Modified waterfall model provides flexibility while it preserves logical order of the phases and good documentation features of waterfall model.

1.2.2 Project Phases

Once the software development methodology is chosen, it is possible to break down the project work into smaller phases. Identifying the phases serve as a base for detailed planning and scheduling. The phases include not only activities that involve in the software development process but also other activities that are required for the accomplishment of the work. It also represents the structure of the report. They are shown in Table 1.1.

Project Phases
Project planning
Pre-study
Requirement Specification
Design
Implementation
Testing
Discussion and Recommendation
Conclusion

Table 1.1: High-Level phases of the project

1.2.3 Project Schedule

The Gantt Chart shown in Figure 1.2 shows the overall project schedule.[1]

[1]The last date of the project was June 17, 2009; however due to pregnancy sickness, i was not able to finish on the time and I postponed it by three weeks. The schedule is modified based on the new deadline.

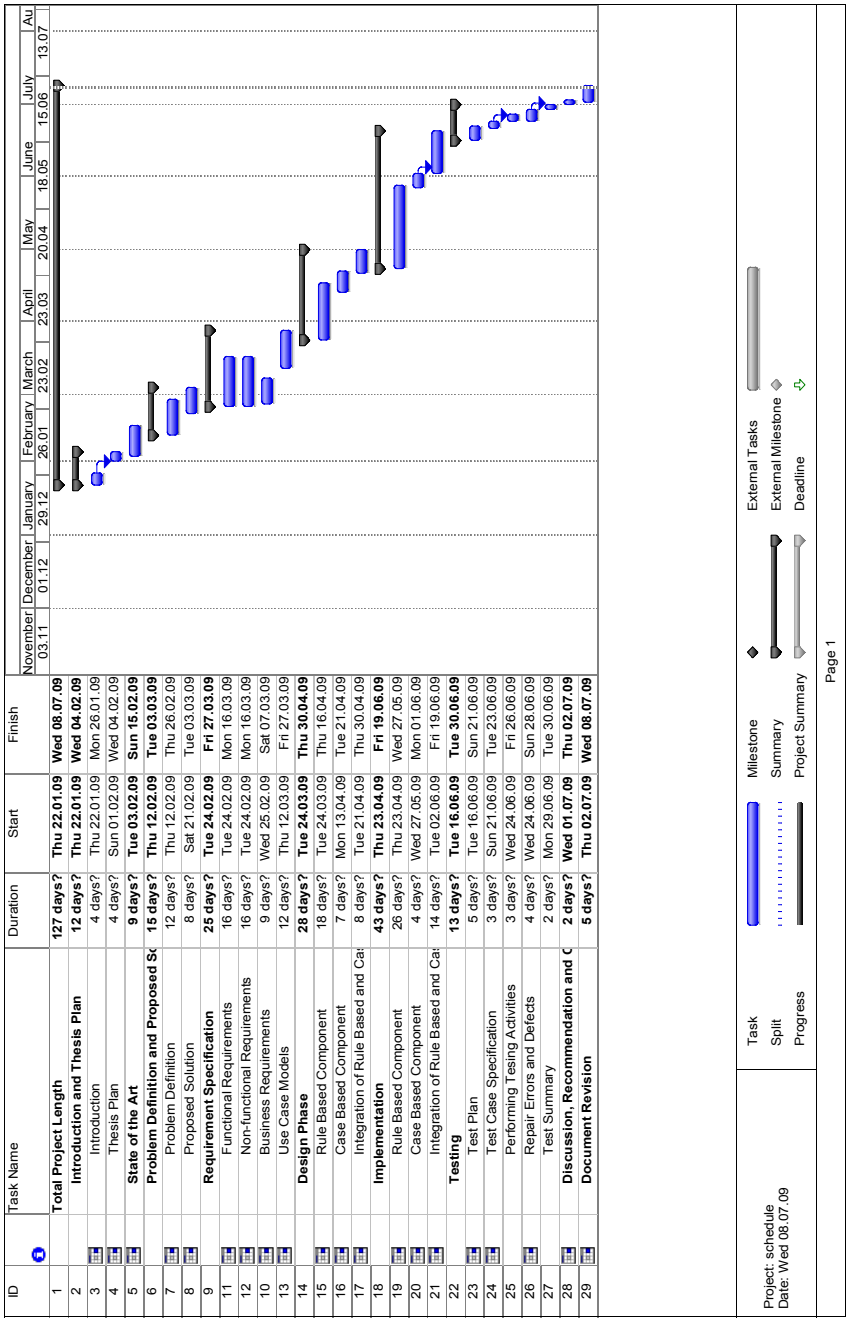

Figure 1.2: Project Schedule

1.3 Documentation Resource Requirements

I use the report format recommended for the customer- driven project course to structure and organize my report. I found the format is a good template to organize a report in a well ordered and logical format.

I am using Latex to write the report. Latex is capable of managing and keeping format consistency, especially for large documents like this report. All pictures except project schedule and those that are screen shots are drawn by using Microsoft Office Visio 2007. Microsoft Office Project 2007 is used to draw the project schedule. Pictures that are taken from screen shots are edited on Paint program.

1.4 Report Outline

This section provides an overview of how the report is organized.

Chapter 1 - Introduction

This chapter provides general information about the project. in terms of its background and motivation, objective and scope. It also provides the project plan.

Chapter 2 - Pre-study

This chapter explores theoretical concepts of knowledge based systems, case based systems, rule based systems, and integration of rule based and case based systems. It also presents about software development methodologies.

Chapter 3 - Problem Definition and Proposed Solution

This chapter presents the existing case based system and identifies the problem that is aimed to be solved in this project. It also proposes a solution for the problem.

Chapter 4 - Requirement Specification

This chapter identifies and presents what functionalities the demo system should have to demonstrate the proposed solution. The functional

and non-functional requirements are identified and presented by using use case models.

Chapter 5 - Design Phase

This chapter presents how the demo system will be developed to achieve the requirements that will be discussed on chapter 4.

Chapter 6 - Implementation Phase

This chapter deals about how the demo system is developed and illustrates the implementation with some examples.

Chapter 7 - Testing Phase

Testing phase presents a test plan to guide how the demo system will be tested and the testing results after it is tested based on the plan.

Chapter 8 - Discussion, Recommendation and Conclusion

This chapter discusses some important points and provides some highlights for further work. Finally, it provides conclusion of the project

Chapter 2

State of the Art

This chapter provides an explanation about the basic concepts that are touched for this project work.

2.1 Knowledge Based Systems

The concept of knowledge based systems is derived from the field of artificial intelligence (AI). AI intends understanding of human intelligence and building of computer programs that are capable of simulating or acting one or more of intelligent behaviors. Intelligent behaviors include cognitive skills like thinking, problem solving, learning, understanding, emotions, consciousness, intuition and creativity, language capacity, etc. These days some of the behaviors such as problem solving, learning and understanding are handled by computer programs.[41], [42], [14]

Computer programs that try to solve problems in a human expert-like fashion by using knowledge about the application domain and problem solving techniques are known as *Knowledge based system*. Human experts use the knowledge they have about the domain and techniques that lead how to use the knowledge to solve problems. Knowledge based systems handle problems in the same way. They represent the knowledge about the application domain and they use one or more techniques that guides on how to use the knowledge to solve problems. Every knowledge based system has two building blocks which are known as *knowledge base* and *inference engine*.[42], [50]

Knowledge base contains all necessary knowledge about the domain that is required to handle problems. The knowledge can be acquired from experts,

documents, books and/or other sources. It is formalized and organized with a technique called *knowledge representation*. There are several ways to represent knowledge in the knowledge base. Two examples of such techniques are *cases* and *rules* that will be introduced and discussed later.

The second component of a knowledge base system is inference engine. After the system gets the required knowledge, it needs to be instructed how to use the knowledge in solving problems. Inference engine represents the reasoning technique that manipulates, uses and controls the knowledge to solve the problems. Case based reasoning and rule based reasoning are two examples of reasoning techniques. They will be discussed later.

Case based systems and rule based systems are two examples of knowledge based systems that uses cases and rules for knowledge representation and case based reasoning and rule based reasoning for reasoning techniques respectively. They will be discussed briefly in the following sections.

2.2 Case Based Systems

Human beings handle situations by being reminded of the experiences we have on similar situations. If the situation is novel, we try to handle it by relating it with other experienced situations. We normally learn from our successful and wrong activities to handle future similar situations in the right way and not to repeat our mistakes. Remembering and reusing previously solved problems, and learning from experiences for future use, is natural and useful. [1], [24], [25] Case based systems are designed to work in the same way with the basic idea of similar problems have similar solutions.

Case based systems are knowledge based systems that solve problems by remembering similar past situation and reusing its solution and lesson learned from it. Case based systems combine problem solving and learning from new experiences for future use. [1], [24] The knowledge base of a case based system represents situations or domain knowledge in the form of cases and the inference engine uses case based reasoning method to solve new problems or to handle new situations.

2.2.1 Cases

Cases are used to represent domain knowledge of a case based system. A case refers to specific experience or knowledge tied to specific situation that is worth remembering for future use. So that cases in the knowledge base represent collection of specific experienced, captured and learned situations of the application domain.[1], [24] Each case is constituted with three main parts: [25]

- **Situation/Problem description**: describes specific circumstances, states of a situation, and state of the environment when this particular case is recorded.
- **Solution:** provides how the problem described in the problem description was solved or treated in a particular instance.
- **Outcome:** describes the final result or consequence and feedback gained from following the proposed solution.

2.2.2 Case Based Reasoning Technique

Case based reasoning, as its name indicates, uses cases to reason about a given problem. In its problem solving process, it reuses old similar cases to understand the problem, suggest a solution, and/or to keep it from failure. A case based reasoning technique follows four processes; retrieve, reuse, revise and retain, to accomplish its reasoning task. Figure 2.1 shows sequence of the processes and each process is described below. [1], [5], [24], [25]

2.2.2.1 Retrieve

when a new problem occurs, this process tries to identify the descriptive features of the new problem and searches previous cases that match with the new situation based on the identified features. Indentifying descriptive features involves tasks of identifying properties that describe the new problem, leave out those that don't describe it strongly and represent the descriptive features in a case format. There are algorithms that are capable of doing this task. Searching similar previous cases is performed by matching the new

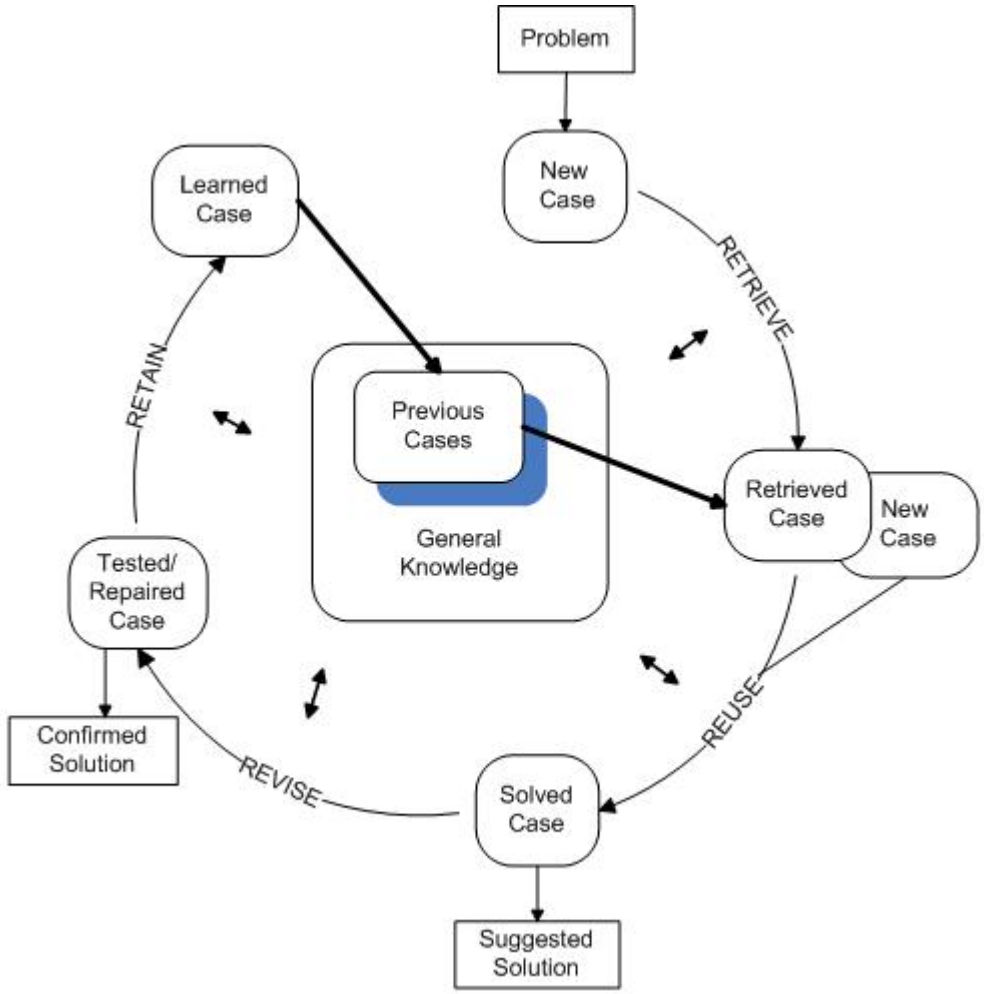

Figure 2.1: Case-Based Reasoning Cycle [1]

case with saved old cases from the knowledge base. It results in a collection of similar cases.

The final step of retrieval process is to select the best matched case or a set of cases from the collection of similar cases. The degree of similarity is measured by using similarity assessment methods.

Quality of the retrieval process depends on its descriptive feature identifying algorithm, searching algorithm and similarity assessment method.

2.2.2.2 Reuse

the selected case in the retrieval process can be used to understand the new situation when it is not clear by itself, to propose a solution based on the solution taken on the selected case, or to prevent from following a wrong way of solving the problem based on the outcome of the selected case. Proposing a solution can be performed into two ways: reusing the solution as it is or by adapting it. When the selected case and the new case do not have significant difference, the solution in the selected case will be proposed as it is for the new problem. Whereas, if there is a significant difference between them, the solution in the selected case is adapted based on the unique feature of the new case, this process is known as *adaptation*.

2.2.2.3 Revise

in case based systems proposing a solution is not the only goal, it also aims to learn from the consequence of applying the proposed solution. This process evaluates how good the proposed solution is for the given problem. The evaluation is performed by using simulator, by getting feedback from a human expert of the application domain or by applying it in the real world and see the result. This process may take hours, days or months until the result is being realized. The system learns from the result whatever it is: *success* or *failure*. If it is failure, the fault needs to be repaired and explanation of why the failure occurs should be given to prevent future similar problems from such kind of failures.

2.2.2.4 Retain

case based systems upgrade their domain knowledge by learning from new experiences obtained while problems are solving. After the proposed solution for the given problem is evaluated in the revise process, the retain process identifies useful and worth remembering new experiences and decides how to merge with existing knowledge. This type of learning is known as *incremental learning* because it always adds knowledge that is new and useful in addition to the existing knowledge.

The new experience may be success or failure. If it is success, the retain

process keeps how the problem is solved by modifying existing cases or by creating a new case if it has significant difference with the existing ones. The advantage of keeping failure processes is to prevent future similar problems from similar failure. The failure can be *task failure* where the solution is unsuccessful or *expectation failure* where the observed solution is different from the expected solution.

The bigger inner rounded rectangle in Figure 2.1 represents the knowledge base which is made up of cases and general knowledge. General knowledge of a case based system is domain-dependent knowledge that represents generalizations of cases, adaptation strategies and case matching procedures in order to support case based reasoning process. [1], [25]

2.3 Rule Based Systems

Rule based systems are knowledge based systems that represent the domain knowledge with set of rules and suggest a solution or conclusion of a problem by using rule based reasoning method. A rule based system has one more component, which is known as working memory, in addition to knowledge base and inference engine. As Figure 2.2 shows, the inference engine receives a problem from the working memory and provides the reasoning result to the working memory. The working memory contains the description of the problem and updates its content based on the reasoning results received from the inference engine. The rules in the knowledge base and the reasoning method used by the inference engine are discussed below.

2.3.1 Rules

Normally rules represent what to do or not to do while certain situations are met. Similarly, the application domain knowledge is represented with set of rules that represent the facts that would be true when some conditions are given true. A typical rule has a format of *If* <*conditions*> *then* <*conclusion*>; where conditions represent premises or facts, and conclusion represents associated actions for the premises. The condition might be a premise or set of premises that are connected with logical operators: AND & OR. The conclusion can be an action to be taken or facts that are inferred

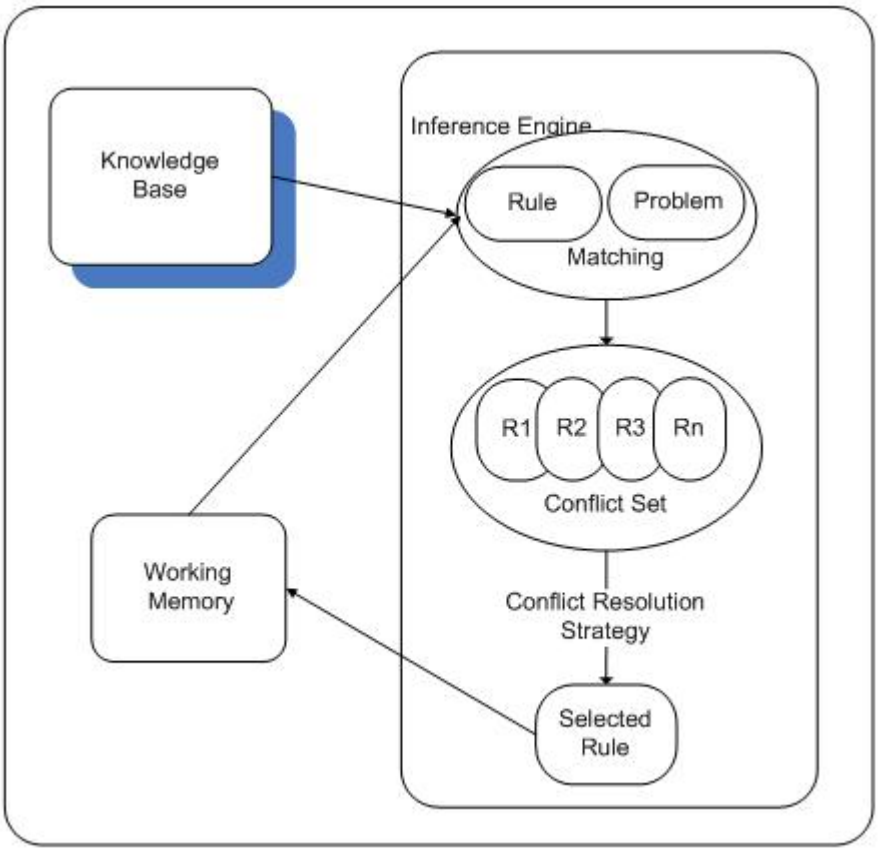

Figure 2.2: Rule Based Reasoning

from the given premises. [19], [23]

Frequently used means of acquiring rules is, interviewing of the domain experts. Rules represent general knowledge of the application domain. They preserve the naturalness, modularity and ease of explanation because they are used in a direct fashion as acquired from experts. Its shortcoming is its difficulty in acquiring complete and perfect knowledge in a complex domain due to the experts may be incapable of expressing their knowledge or unavailability of some experts. In addition, sometimes representing the domain with only general knowledge may not be enough. [19], [22], [23], [25]

2.3.2 Rule Based Reasoning Technique:

rule based reasoning technique represents how a system solves a problem by using knowledge of the application domain that is represented in form of rules. There are two ways of rule based reasoning methods: *forward chaining* and *backward chaining*. [19]

Forward Chaining: in this process, it receives a problem description from the working memory as a set of conditions and tries to derive conclusions as a solution. Once it receives the conditions, it searches all rules whose condition matches with part or all of the conditions in the working memory. The searching result provides a set of rules that are applicable to provide a conclusion about the problem, the set is known as a *conflict set*. Rule based reasoning technique uses conflict resolution strategy to select one rule at a time from the set. The selected rule is then applied to derive a conclusion about the problem. Content of the working memory is updated based on the derived conclusion. Searching applicable rules continue based on the updated working memory content and the reasoning process continues based on the new matched rules. This process continues until the desired solution is obtained or there is no rule whose condition matches with current description of the problem in the working memory.

Backward Chaining: it is similar with forward chaining in most process, the big difference is it receives the problem description as set of conclusions, instead of conditions, and tries to find the premises or causes of the conclusions. It searches the rules whose conclusion matches with part or all conclusions in the working memory. Like forward chaining, conflict resolution strategy is used to select one rule from the set of applicable rules. The selected rule is used to derive the premises that led to the given conclusion. The working memory is updated each time a premise(s) is derived and the reasoning process continues on the updated content of working memory until the desired solution is obtained or there is no a rule whose conclusion matches with the give conclusions in the working memory.

Rule based systems are more applicable for complete, narrow, limited and well understood application domain due to its difficulty of acquiring knowl-

edge. A problem is solved from the scratch in rule based systems; the reasoning process for a problem is performed again though the problem had been solved before by following the same reasoning process.

2.4 Integration of Case Based Reasoning and Rule Based Reasoning

The ultimate goal of artificial intelligence discipline is to develop systems that exhibit human-like, even better intelligence.[14] Most of current knowledge based systems represent some aspects of human beings intelligence. Integrating of two or more knowledge based techniques begets a better simulation of intelligence than it would have been gained from one technique. [16], [21], [22]

On the other hand, the reasoning power of a knowledge based system depends on the explicit representation and use of different kinds of knowledge about the domain. There is no one way of knowledge representation that can represent the domain knowledge as it is in the reality. The more knowledge based techniques are integrated, the more the domain knowledge is represented, which begets the more efficient system.[4]

Case based reasoning and rule based reasoning techniques are two alternative ways of problem solving in intelligent systems. Their knowledge representation and reasoning methods are naturally alternatives. [22] The following paragraphs compare them based on their knowledge representation and problem solving capability.

Cases represent knowledge that is accumulated from specific situations whereas rules represent general knowledge about the domain. Acquiring rules when it is compared to that of cases is really hard. Because of that maintaining or updating rules is harder than updating and maintaining cases. [19], [21], [22], [23]

In the problem solving process, case based reasoning uses solutions that was solved in similar past problems whereas rule based reasoning solves problems from scratch though similar problems had been solved previously. Case based reasoning method plays greater role in handling missing or unexpected features in the problem description and selected cases than that of rule

based reasoning in problem description and rules. The case base system tries to find the similarity between the problem and the cases though there are features that do not match between them. However, the rule based system tries to find rules that perfectly match with part or all of the problem description. Rule based reasoning method is better in providing explanation for the given solution than case based reasoning. [19], [22], [23], [25]

Due to their interchangeable nature, integrating them provides effective knowledge representation, effective problem solving power, and exceeding one's weakness with the other. [16], [22], [21]

2.5 Knowledge-Intensive Case Based Systems

In a case based system, cases represent experiences that bound to specific situations regarding to the application domain. New situations are handled based on similar past situations. The similarity is performed by checking the existence of similar situation descriptive features between the new case and past cases, and one factor to calculate similarity is the number of similar features. It is more of syntactical similarity; it doesn't consider the contextual meaning of the features that describe the problem. This limitation can be solved by integrating the specific cases with model of the general domain knowledge. The general domain knowledge enriches the cases by making it possible to interpret the features based on the context or the given situation. [2], [3], [4]

The general domain knowledge represents model of the application domain in the real world by providing the concepts and the different relationships among them. The model is network of inter-related concepts which is known as *semantic network*. The relations between concepts represent the meaning of the concepts at different situations. Hence each concept has many relations to other concepts. The reasoning method that is applied on semantic network is known as *model based reasoning*. [2]

Knowledge-intensive case based systems are systems that integrate case based technique with model based technique. In this case the domain knowledge is represented as specific cases and general domain knowledge, which increases the knowledge intensiveness of the system. The more the domain knowledge is represented, the more the system's capability in reasoning

about the problems. [2], [3], [4]

2.6 System Development Methodology

Software development methodology is a framework that structure, plan and control the activities involved in the development process. It divides the complex and big project into smaller and more easily manageable phases, which in turn makes resources allocation and project scheduling simpler.[46], [47], [34]

There are various Software development models that propose a specific approach to handle involved activities throughout the development process. The following section discusses common software development activities and the section followed presents candidate methodologies. Then based on the situation at hand the most appropriate methodology will be selected.

2.6.1 Software Development Activities

A software development is a sequence of activities that are aimed to perform specific tasks that need to be handled in the development process. Since the activities are taken into action in sequence, they are synonymously known as phases, steps and processes. I will use phases and activities interchangeably. Common software development activities are described in the sections below [32], [33], [38], [36].

2.6.1.1 Requirement Gathering phase

This activity is about defining what the customer wants from the software product. Understanding of the system is the first step to identify the requirements, so that close communication with the customer and end users is important. It should answer general questions like what the intended system should perform (functional requirements), what data the system does require (data requirements) and under what conditions the system should work (business requirements). In general, it is understanding and determining of what the system should perform without considering how. The requirements can be gathered by interviewing end users, observing existing system, and/or from documents & forms used in current system.

The requirements can be categorized into two general aspects:

- **Business Requirements:** it is about the business goals the customer wants to achieve from the new software product and the business logics that should be taken into consideration to determine how the software product should work or look like.
- **Functional Requirements:** it is about defining what capabilities the software product will have and what it will not have. It is also known as functional requirement specification.

2.6.1.2 System Analysis Phase

In this phase the gathered requirements are represented in well structured formats by using various tools, like use cases. The representation is independent of any implementation and technological issues. The output of this activity is documentation of the requirements that serves as a bridge to confirm that the people in the development group understands what the customer is really demanding. It is also the input of the design phase.

In some places requirement gathering and system analysis are considered as one activity with a name like *requirement specification, system analysis* and others. In my thesis I will use requirement specification to refer the combination of the two activities.

2.6.1.3 Design Phase

This phase includes the tasks of determining different components or elements of the intended system, communication among the components, hardware and software requirements for implementation and all the questions regarding to how the system will be implemented.

Output of design phase includes architecture, software and hardware requirements, and design of the intended system. It has to provide intuitive information about how the system should be implemented so that a person who had not been in the analysis and design phase can understand and use it for implementation phase.

2.6.1.4 Implementation Phase

This phase is where programming or writing code is performed based on the output of the design phase. While the programmers are writing the code, they make sure that each the smallest testable unit of the program is working right.

2.6.1.5 Testing phase

After implementation is completed, the system has to be evaluated against the requirements to make sure that it satisfies what was expected from it. In addition, acceptance testing is performed to make sure that the customer is satisfied and willing to accept the system.

2.6.1.6 Deployment phase

Deployment phase is installation of the system on the right machines and make it ready for use. It also includes providing training accompanied with manuals and training materials for end users.

2.6.1.7 Maintenance phase

The real testing of the software product is performed while it is being used by real end users. The end users may identify things that the system should do but doesn't, some faults or may suggest things that would improve the system. This activity fixes all the comments coming from the end user regarding the system while it is in use.

2.6.2 Candidate Software Development Methodologies for this Project

A software development methodology defines how the different activities of a software development are arranged and involved in the development process. This section presents alternative approaches that can be used in this project work.

2.6.2.1 Waterfall Model

Waterfall model is a software development method that starts from the requirement specification activity and continues with design, implementation, testing, deployment, and maintenance in sequence. One activity is taken into action when the preceding activity is completed satisfactorily. It is more suited for projects that have stable or unchanging requirements. Moving backward and forward to make some changes is expensive. Hence, it is important to make sure that the phase is as complete as possible before proceeding to the next activity. It provides structured and logical approach for documentation. Waterfall model is also known as Classic Life Cycle Model or Linear Sequential Model.[48], [36] The model is depicted in Figure 2.3.

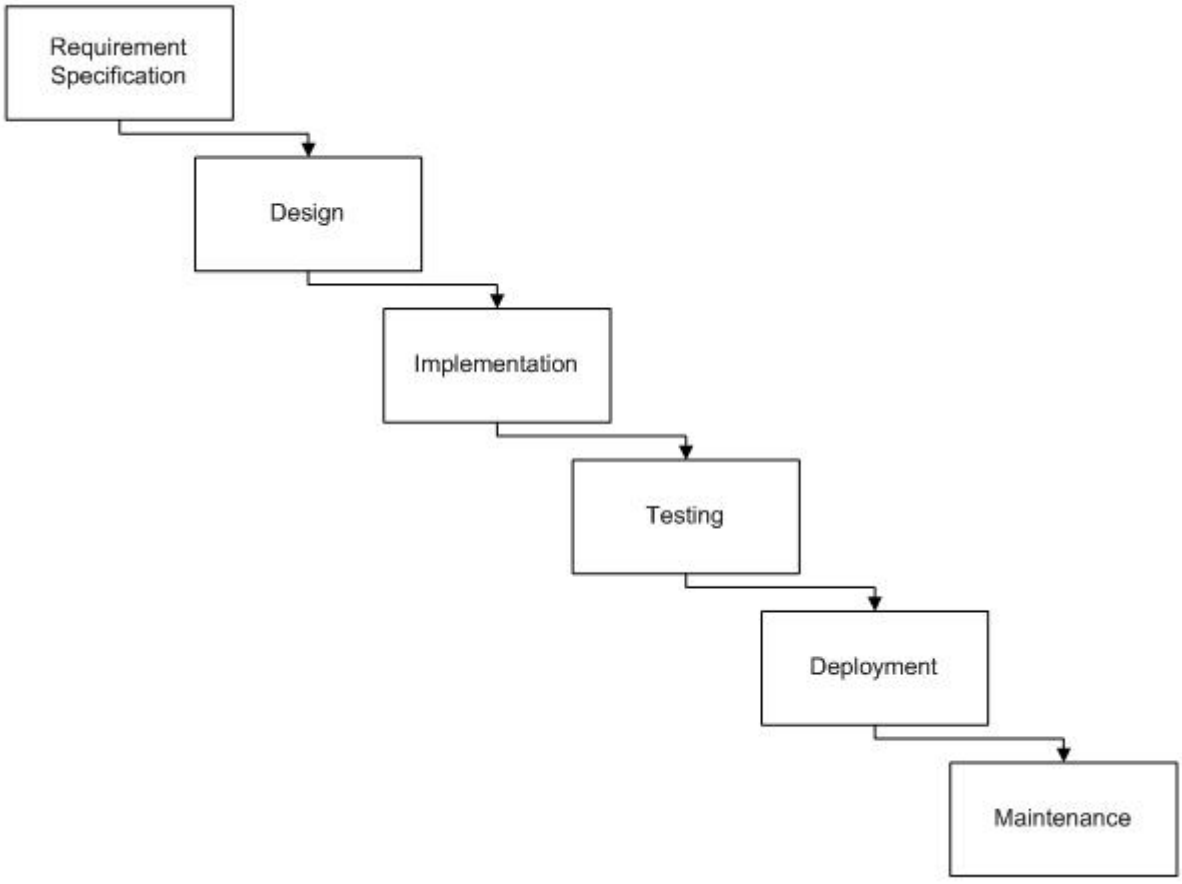

Figure 2.3: Water Fall or Linear Sequential Model

Though waterfall is simple and easily understandable to be used, it is criti-

cized for its being inflexible for accepting changes. Clients may change their requirements after they see a prototype. The designed solution may be really tough to be implemented; the developers should stick on trying it instead of going back and change the design. To overcome these limitations, various methodologies are proposed. Prototyping model and Spiral model are two examples that tried to solve waterfall's criticism.

2.6.2.2 Prototyping Model

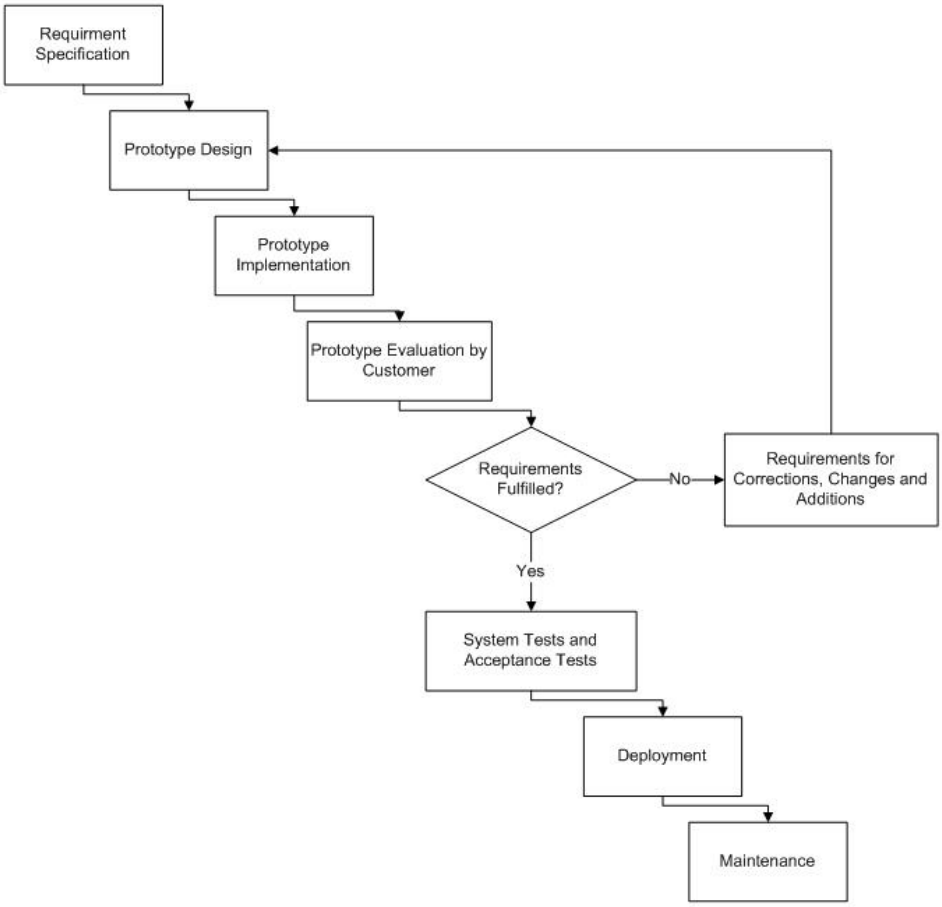

Figure 2.4: Prototyping Model

This model bases on creating prototypes. As Figure 2.4 shows, once the requirement specification is done, a design is proposed for a prototype of the specified requirements. After it is implemented, the prototype is given to the customers for feedback. Based on the feedback new requirements

are defined to refine the prototype and a new prototype design is proposed. The cyclic movement of the activities continues until the customers suggest that all the requirements are satisfied with the current prototype. Then the system as a whole is tested and deployed for final use.

The software is developed with continues users' feedback so that it reduces the risk of failure. Though, it provides more effort on developing the actual software which secures producing of the right product, documentation is difficult due to the prototype requirements and design are changing from time to time. In addition, the development process may take longer time or several iterations until the customer is satisfied with the prototype. [37], [38]

2.6.2.3 Spiral Model

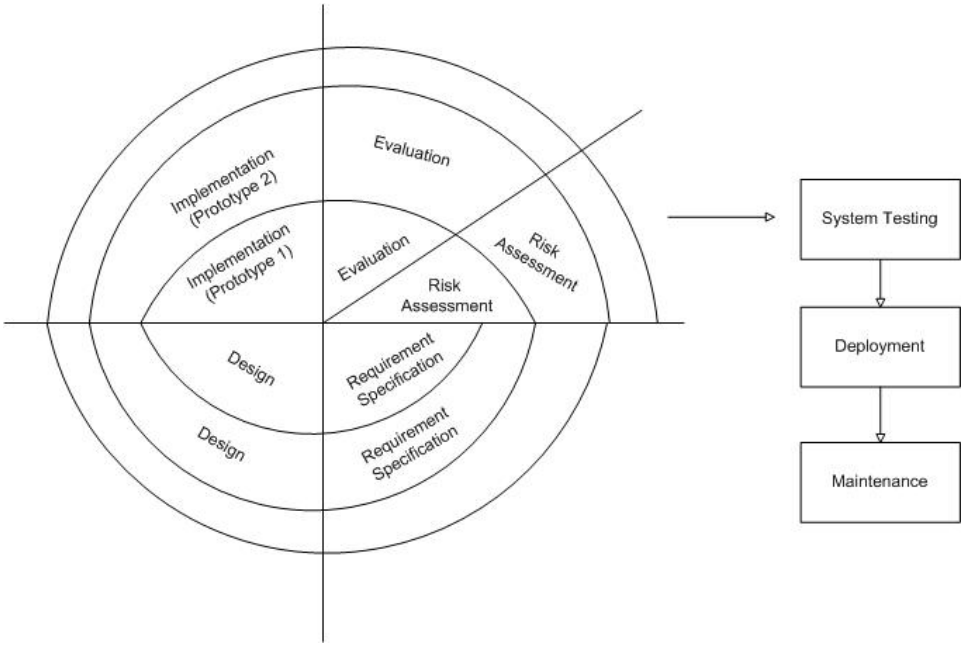

Figure 2.5: Spiral Model

It combines positive features of prototyping model and waterfall model; and it adds a new activity called *risk assessment*. Like prototyping model, it develops the first prototype and modifies it based on users' feedback or other additional requirements by iterating through the cyclic activities until the required requirements are satisfied or it is known that continuing on the

current prototype has high risk in terms of budget, time or other factors. However, unlike prototyping it carefully designs and documents each cycle by following the activities involved in waterfall model.

The risk assessment activity decides whether the iteration should continue or not. First it identifies possible risks and then it proposes alternative solutions to minimize or resolve the risks. If one or more risks are high and costly to be solved, the project may be halted at all or may take the current prototype as a final solution.

When the final prototype is taken as a solution, testing is performed to make sure that the software product achieves the required goals and the customer is satisfied. Then it will be installed on the machines where the customer wants to use it.

Spiral model is more applicable for large, expensive and complicated projects. It simplifies large project by breaking down into smaller requirement categories which can be handled in the different iterations. The weakness of spiral model is, it is complicated and requires a person with strong knowledge of risk assessment. [35], [36], [49]

The two chapters so far have provided the basic background of the project. The next chapters focus on identifying the specific problem that needs to be solved in this project and solving the problem by developing a demo system.

Chapter 3

Problem Definition and Proposed Solution

This chapter presents the problem that Verdande wants to be solved in this project and the proposed solution to solve the problem.

3.1 Problem Definition

The problem to be solved in this project is defined based on Verdande's desire to integrate rule based reasoning method with the existing case based system. Before going to the problem directly, understanding how the current system works is important. Hence the following section describes the current case based system called *DrillEdge*. [27], [28]

3.1.1 Current System

The existing case based system, DrillEdge, is developed based on the Edge platform in order to help decision making activities in oil drilling operations to avoid unwanted events. The main reasons that raise the desire of a computer tool to support decision making in oil drilling operation are mentioned in the following paragraph.

Drilling operation is data-intensive that is represented with vast amount of data. Processing this large amount of data to control the progress of the operation at real time is hard for human beings. On the other hand, many drilling engineers' experiences or knowledge may be well documented and kept in a database as a form of daily drilling reports, best practices, lessons

learned and other forms. However, it is unpractical for drilling operators to use these experiences from the database to process the real-time data to make sure that the operation is going on well and to handle problems before they occur or before they are going to the worst situation. Hence, difficulty of handling real-time data and separation of this data from the experiences in the database are the two reasons that make controlling and managing of the drilling operation at real time difficult. This difficulty begets the desire of supportive computer tools.

DrillEdge is developed to bridge the real-time data and experiences in the database in order to process the data and to make the experiences readily available so that interpreting progress of the drilling operation is simple and possible. It evaluates the state of the operation continuously to control and manage unwanted events. Evaluation of the state of the operation is performed by processing the real-time data and interpreting it by actively recalling relevant past experiences from the database. It identifies unwanted events that slower or prevents the expected drilling progress. When it faces unwanted events, it informs drilling engineers about the situation and suggests solution to avoid the unwanted events based on similar experiences from the database.

DrillEdge is designed to assist drilling engineers to re-use knowledge in order to diagnose and avoid costly problems before they escalate. It helps operators to easily recall experiences and use them to make the right decision about the situation in order to lower risks, increase well drilling operation, and to minimize non-productive time while drilling. The recalled experiences may be other operators' experiences, in which DrillEdge serves as experience sharing tool among experts that may not even known each other. Figure 3.1 shows the separation of drilling real-time data and experiences, and then their linkage by using DrillEdge software.

3.1.1.1 Knowledge Base of DrillEdge

DrillEdge system is a knowledge-intensive case based system. It combines case-specific and general knowledge of the application domain. The knowledge is represented as semantic network, which is a network of concepts and the multiple relations among the concepts. The concepts are represented as node, while the relations are represented as links. Each concept is defined

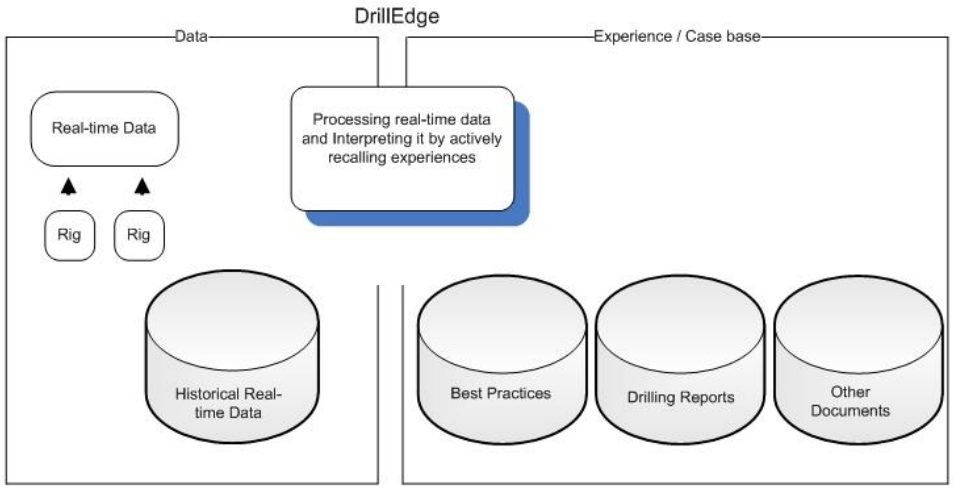

Figure 3.1: Semaration of real-time data and experiences, and their linkage by DrillEdge

by its relations with other concepts. The cases and the general knowledge are strongly coupled; the cases are submerged within the general domain model. Figure 3.2 shows the structure of the semantic network. It illustrates three main types of knowledge in DrillEdge. The *generic concepts* represent concepts that are general and domain independent but that are important to define other concepts in the application domain. For example, every object in this world is a thing. Hence, thing is the root node for all other concepts. The *general domain concepts* represent all concepts and relations that are specific and related to oil drilling operation, which is the application domain. *Cases* represent specific drilling experiences that are defined based on the concepts defined in the two higher concept groups. [2]

3.1.1.2 Case Based Reasoning of DrillEdge

Case based reasoning of DrillEdge is performed by following the four basic processes of the case based reasoning technique: retrieve, reuse, revise and retain. Figure 3.3 depicts the functionality of DrillEdge software in general. It receives the real-time data from the ongoing operation and interprets it to identify the important events and to describe the situation in the format that can be used by the case based reasoning process. Retrieval process of the case based reasoning method takes the description of the current situation

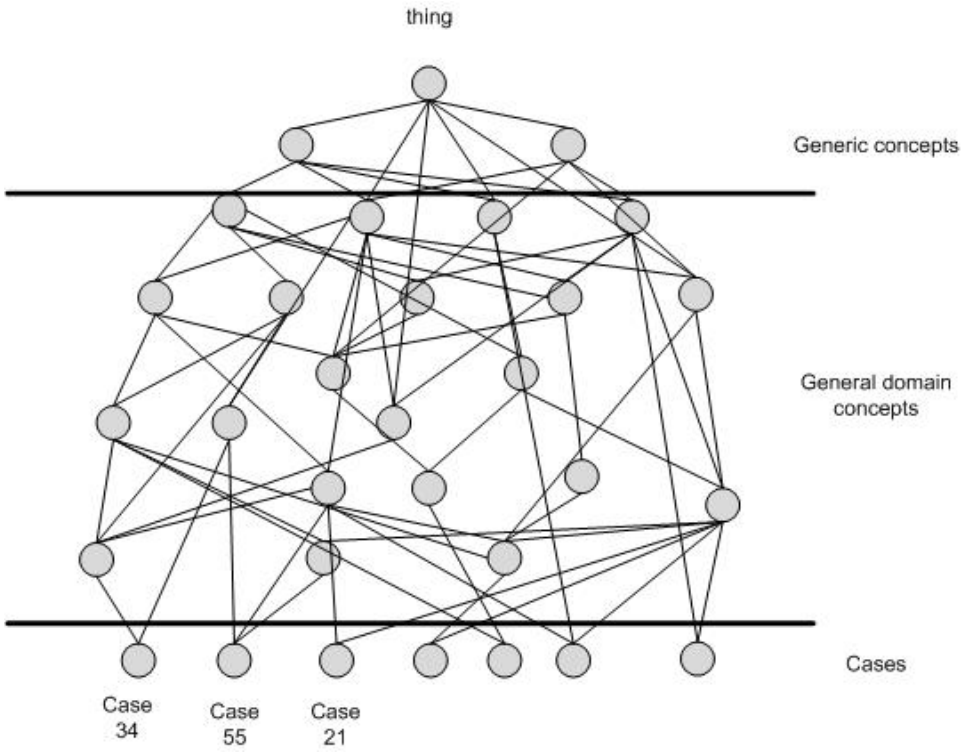

Figure 3.2: Structure of a Semantic Network

to find the most similar case from the case base. The selected most similar case is used to identify the state of the operation. This process is repeated for each new real-time data taken from the ongoing operation. The real-time data is taken in a given time interval.

If the current situation is found to be a problem based on its similar selected case, the reuse process is initiated to decide what solution should be taken to solve it. A solution is proposed based on a solution used in the selected case; the solution may be taken as-is in the selected case or with some modification, which depends on the situational description difference between the new situation and the selected case. After the solution is proposed, the next step is to revise the suggested solution. At this stage, drilling engineers are informed about the situation with the suggested solutions so that they can evaluate the solution and repair any faults that may have been uncovered. Revise process of DrillEdge is performed by the engineers.

If the problem is a bit different from the existing cases and/or is solved in a

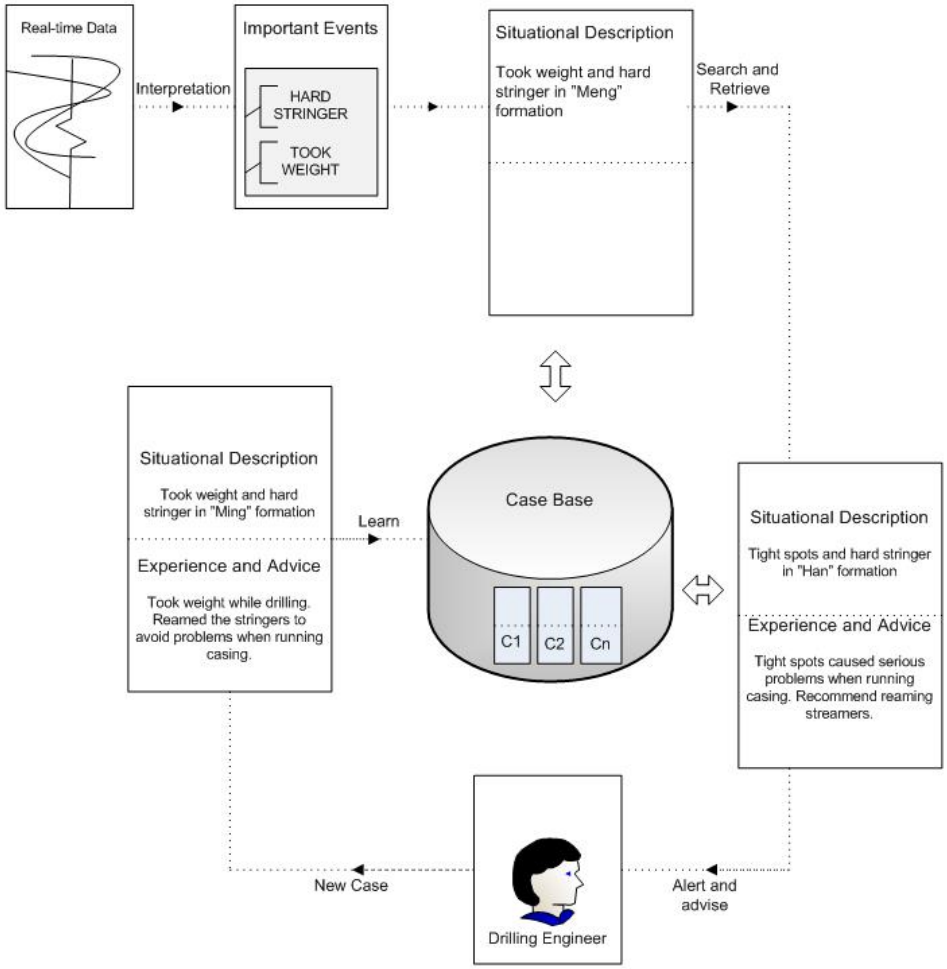

Figure 3.3: Reasoning process of DrillEdge

new way, the engineers create new case to represent its description with the way it is solved, or modify an existing case to add the lesson learned from the new experience. This process represents the retain phase of case based reasoning method. This phase also performs the integration of the new case in the case base and making it ready for subsequent extraction as needed for similar future problems. DrillEdge system learns/updates its knowledge base based on new experiences or lessons learned that are provided by the engineers.

3.1.1.3 General Domain Knowledge's Role

The general domain knowledge supports all phases of the case based reasoning process by providing contextual meaning and explanation of situation descriptive features. For example, features may represent the same concept while they are described with different terms. The semantic network helps to identify that they are representing the same concept by using the relation between the two semantically similar terms.

This project's interest is the role that the general domain knowledge plays in *elaborating cases*. Features of a case that are identified from the real-time data are known as *observed features*. They represent observable features of that particular drilling operation. However, there are several features that are not observable but can be inferred from the observable features. These type of features are known as *inferred features*. With more features the case is described, more likely for correct representation of the situation and more likely to get the right solution.

The relations between concepts in the general domain knowledge representation plays great role in elaborating cases by identifying hidden facts from the observed features. For example, if concepts A and B are related as A causes B and if we observe A in the current situation, we can conclude that B is also a feature of the current situation, where A is the observed feature and B is the inferred feature. Identifying hidden features of a case is known as *case transformation*. And the case that is described with both observed and inferred features is known as *transformed case*.

All cases in the case base are represented only with observed features. Every time each is used in case matching, the inferred features are derived by

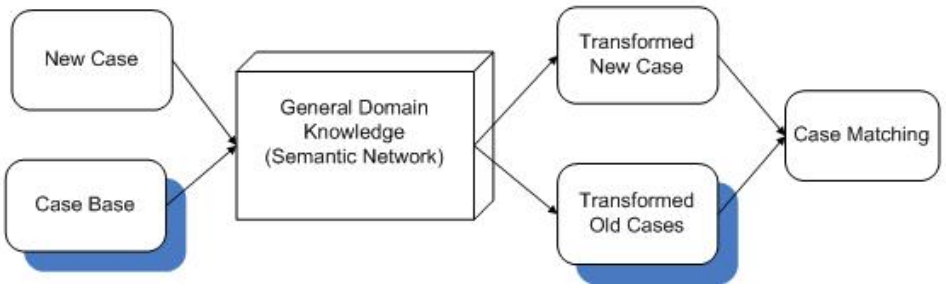

Figure 3.4: The process of case elaboration

using current version of the general domain knowledge. The reason that the inferred features are not saved in the knowledge base is to accommodate changes of the general domain knowledge on the cases while inferring hidden features of the case. As Figure 3.4 shows, the retrieval process of DrillEdge-case matching- uses transformed cases, where both the new and old cases are described with both observed and inferred features.

System developers at Verdande use a package called *knowledge Editor* to edit the knowledge base, and to visualize & test the performance of the case-based reasoning process. It is described in the following section.

3.1.1.4 Knowledge Editor

The knowledge editor simulates DrillEdge's case based reasoning process until case matching step. Instead of accepting real-time data it uses saved data that represents drilling operation. Figure 3.5 is a screen shot of the knowledge editor.

The knowledge editor provides facilities to deal with the knowledge model-semantic network, which comprises case-specific and general knowledge of the domain. The semantic network is one big hierarchically structured tree of concepts and relations with a root node of the concept thing. However, it is possible to view part of the network to increase its readability. The right middle box- *views*, contains some list of views that show the semantic network from different point of views. For example, the selected view- Top Level 061207 that is displayed at the left side- shows the concepts and relations at highest level in the semantic network. The right top box- *Frame View*, shows the description and definition of the selected node in the dia-

gram. It is possible to edit the definition of the selected node here. It is also possible to add and delete concepts and relations in the diagram.

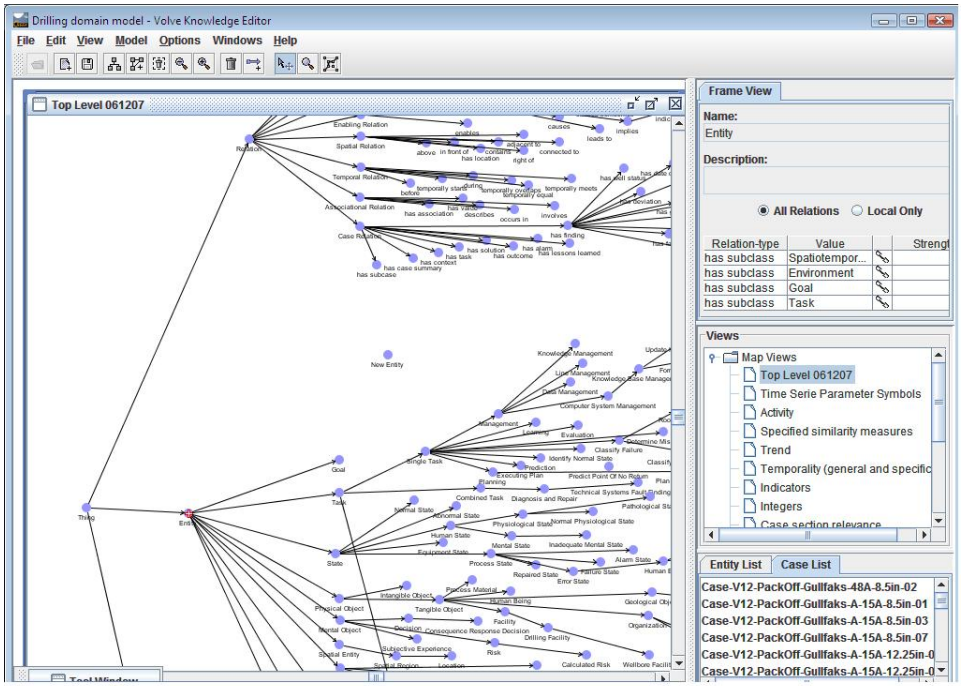

Figure 3.5: Screen shot of Knowledge Editor

The cases in the knowledge base are listed at the right bottom box- *Case List*, of the knowledge editor. The knowledge editor provides a *case view* in which the cases can be viewed from different point of view. The surrounded part of Figure 3.6 with bold rectangle, represents the case view window that is opened for the selected case from the case list.

When a case is double clicked at the case list, it is opened by using the case view and displays all observed features of the case under *View* tab. When the user clicks on the other tabs, the associated information regarding to this particular case is displayed. Each tab is introduced as follows.

Event tab presents each event that occurs in this particular case with respect to at what time and in which depth each event occurs. The case's features are categorized into groups based on the type of feature they represent. The *Tree Editor* tab helps to edit these features based on their group. *Text Editor* tab helps to view the case in XML format. It is possible to edit its content here. It also supports importing a new case from other place to add it into

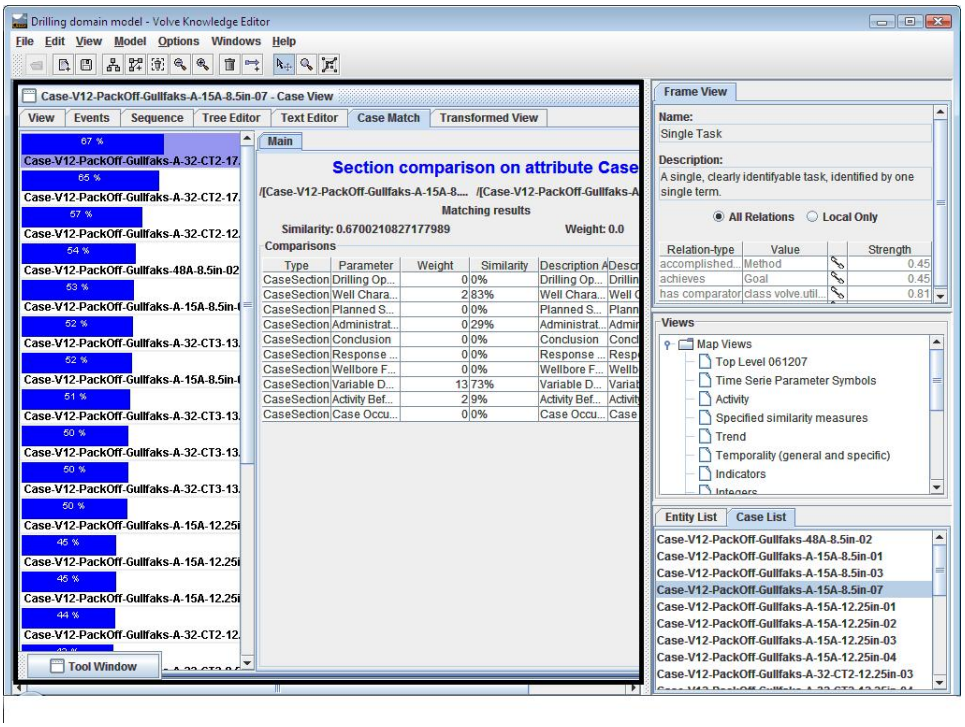

Figure 3.6: Case View of the Knowledge Editor

the case base. When *Case Match* tab is clicked the similarity between this particular case and each case in the list is performed and displayed as shown in Figure 3.6. The left side of the window shows its similarity in percentage with each case. As mentioned earlier, before case matching is done the new and all other cases are elaborated to infer the hidden facts. The elaborated or transformed case for this particular situation is displayed under *Transformed View* tab. It contains both observed and inferred features of this case.

Simulation of the case based reasoning process of DrillEdge includes steps until case matching. It doesn't involve suggesting solutions for the given situation.

3.1.2 Problem Description

The term problem normally connote negative sides however in this case it doesn't mean that the current system has a problem instead it refers the reasons that initiate this project idea. There are two main reasons that raise the need of integrating rule based reasoning with the case based system.

The first reason is the availability of best common drilling practices that can be formulated in form of rules better than other knowledge representations. The common practices represent general common facts about oil drilling operations. Though the case based system is performing well with current knowledge, integrating the best common drilling practices is thought to increase the accuracy of handling drilling operations.

The other reason is in the semantic network each relationship between concepts is weak, because it relates one concept with another. It is binary relation that relates only two features. Representing multi relation makes the semantic network complex. For example, representing a relation like *[(A and B and C) or D and (not E)]* on the semantic network is not simple. On the other hand, there are some drilling operations that need to be represented with more than two features for better presentation.

The availability of best drilling operations and situations that are representable with complex relation of features motivates the integration of rule based reasoning with the current system.

3.2 Proposed Solution

Integrating the rule based reasoning method with the case based system is the proposed solution for the aforementioned problems. The rule based reasoning component can be integrated in different ways to support the different activities of the case based reasoning process. In my specialization project, alternative ways of integrating case based and rule based reasoning methods are researched and exemplified with real world systems that integrate them.

In the proposed solution, the rule based reasoning method is suggested to replace the semantic network in the case elaboration process. It is assumed that the best common drilling operations and situations with complex relation can play a better role in identifying the hidden features than the weak relations in the semantic network. The overall structure of the suggested solution is depicted in Figure 3.7.

To demonstrate the suggested solution, a demo system will be developed. Development of the demo system involves two main activities; development

of a rule based component and integrating it with the existing case based system. The rule based component needs to represent the best common drilling operations and situations with multi-relation in form of rules. It uses rule based reasoning method to use the rules for the required purpose.

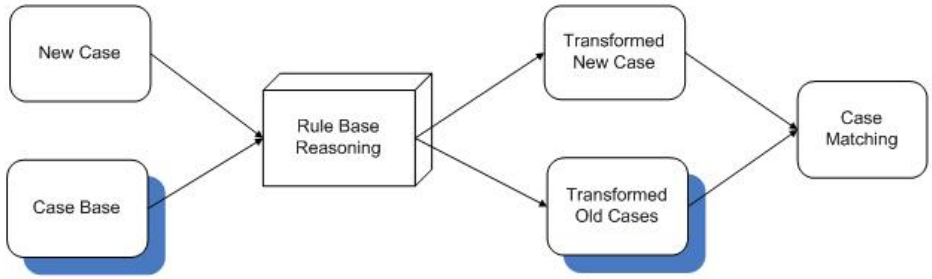

Figure 3.7: Structure of the Proposed Solution

As mentioned earlier, basic operations of DrillEdge are performed and tested on the knowledge editor package. It performs the case based reasoning process until case matching step. It doesn't include suggesting the best solution for the current situation. The demo system will be implemented on top of the existing knowledge editor package.

3.3 Technological Requirements

The demo system will use the existing case based system which is developed by using Java programming language. Hence, Java will be one of the programming languages that will be used on the implementation of the demo system.

The proposed rule based reasoning tool is Java Expert System Shell (Jess). Jess is a rule engine that is written in Java language and freely available for academic use. It provides the facility to develop a system that reasons by using knowledge represented in form of rules.

The reason that I choose Jess is embedding it in Java application and/or accessing all Java API's in Jess is possible, which simplifies integrating the rule based component with the existing case based system, which is developed by Java. It has licensed version for commercial use in case Verdande wants to use it in the future

Eclipse is an integrated development environment that provides full-fledged feature for writing java applications. It has Jess plug-in that provides Jess application development environment. Hence, all code on this project will be written on Eclipse.

Chapter 4

Requirement Specification

Development of the demo system starts by identifying what is required in the intended system from the project work. The requirements are defined based on the general discussions I had with developers at Verdande. Presentation of the requirements is categorized into business requirements, functional requirements and non- functional requirements. Use case models are also used to view the intended system from the user point of view in simple and intuitive way.

4.1 Business Requirements

Business requirements represent high level requirements that the customer wants to see or gain from the project. They are defined in broad terms and they serve as a base for identifying the specific functionalities of a system to be developed in the project.

- BR-01: The project shall propose, analyze, design and demonstrate a way that a rule based reasoning method can be integrated with the current case based system.
- BR-02: The change on the current system due to integrating the rule based reasoning method must be discussed and demonstrated.
- BR-03: The rule based reasoning method should replace role of the semantic network in the case elaboration process.
- BR-04: The tool that will be used to develop the rule based component must be able to integrate with java programs.

- BR-05: The demo system that will demonstrate the integration must be understandable and simple to use.
- BR-06: The project shall compare and contrast the use of rule based method and semantic network in the case elaboration.
- BR-07: The demo system that will be developed to demonstrate the integration should run on Windows operating system.

4.2 Functional Requirements

Functional requirements of the demo system represent the specific activities that the system is supposed to do to achieve the business requirements. They represent the intended behavior of the demo system. Importance of each requirement and interaction with the other requirements are presented in Table 4.1. The importance level is identified with terms high (H), medium (M) and low (L). Requirements with high importance level are essential and the most important functionalities that the customer wants to see on the demo system. Those with low importance level are requirements that are good to have in the demo system but they are not essential like those with high level. Requirements with medium level are preferred functionalities of the demo system.

- FR-01: The demo system shall have a rule based reasoning component.
- FR-02: The rule based component shall represent each case in a format that can be used by the rule based reasoning component.
- FR-03: The rule based component shall identify the inferred features of each case by using the rule based reasoning method.
- FR-04: Elaboration result of a case by using the rule based reasoning method shall be presented to users.
- FR-05: The rule based component shall be integrated with the case based reasoning component of the existing system.
- FR-06: The demo system shall perform the similarity of the new case with the other cases in the case base after each case is elaborated by using the rule based reasoning method.

- FR-07: The demo system shall present the similarity comparison result of the new case with the others where rule based reasoning method is used in the case elaboration process.

- FR-08: The demo system shall preserve identification of inferred features of a case by using the semantic network independent of the rule based component.

- FR-09: Presentation of transformed view of a case where the semantic network is used in the elaboration process must be kept.

- FR-10: The demo system shall maintain performing of similarity between a new case with the other cases in the case base where semantic network is used to elaborate the cases.

- FR-11: The demo system shall maintain presentation of the case matching result where semantic network is used in the case elaboration process.

- FR-12: The demo system shall provide intuitive and simple to use user interface.

- FR-13: The user shall be able to compare the use of rule based method and semantic network in the case elaboration process.

The functional requirements regarding with the rule based reasoning method and integration of this method with the existing case based system are given high importance. Because one of the goals of this project is to develop a rule based reasoning component that demonstrates the designed role and to integrate it with the case based system. Requirements that regard to user interfaces are given medium importance. It is advisable to have them to illustrate the functionalities of the system. Requirements that deal with semantic network are given low importance because the existing system has all these functionalities. They are included in the demo system simply to compare and contrast the role of semantic network and rule based reasoning.

Functional Requirements	Dependence	Importance
FR-01	-	H
FR-02	FR-01	
FR-03	FR-01, FR-02	H
FR-04	FR-01, FR-02, FR-03	M
FR-05	FR-01, FR-02, FR-03	H
FR-06	FR-05	H
FR-07	FR-06	M
FR-08	-	L
FR-09	FR-08	L
FR-10	FR-08	L
FR-11	FR-10	L
FR-12	FR-04, FR-07, FR-09, FR-11	M
FR-13	FR-04, FR-07, FR-09, FR-11	L

Table 4.1: Importance of and dependence among the functional requirements, where H = High, M = Medium and L = Low

4.3 Non-functional Requirements

Non-functional requirements are constraints or quality requirements that need to be considered during implementation of the system. They determine how the system to be rather than what the system to perform. They are intended to support the functional requirements concerning to user interface, modifiability, availability, performance and security. Security and availability issues of the demo system are out of the scope.

- NFR-01: The demo system shall be developed in modular bases.
- NFR-02: The inferred features in the elaborated case must be easily identifiable.
- NFR-03: The user shall be able to identify his request to the demo system with a single click.
- NFR-04: The user interfaces must use intuitive names.
- NFR-05: The demo system shall respond in less than 2 seconds for a user request.

- NFR-06: The demo system shall run on Windows operating system.

4.4 Use Case Models

Basic functions of a system that are identified in the functional requirements are described from the user point of view by using use case models. The model presents potential usage of the system by using use case diagrams with their descriptions. A use case diagram consists of use cases, actors and interaction among them. While a use case represents sequence of actions that give measurable value to an actor; an actor represents users or external systems that interact with the system to perform an action. [29]

The demo system has only one actor which is the developers. Only developers interact with it to test and evaluate its functionalities. The use cases are defined based on the requests the developers raise to the system. The developers may request the demo system to see transformed view of a new case by using semantic network or the rule based component, the case matching results, and to compare & contrast use of the two methods in the case transformation process. Name and identifiers of the use cases for the demo system are listed below:

- UC-01: Open a case
- UC-02: Transform a Case: Semantic Network
- UC-03: View Transformed Case: Semantic Network
- UC-04: View Case Matching result: Semantic Network
- UC-05: Transform a Case: Rule Based Reasoning Method
- UC-06: View Transformed Case: Rule Based Reasoning Method
- UC-07: View Case Matching result: Rule Based Reasoning Method
- UC-08: Compare Semantic Network and Rule Based Reasoning Method

The interfaces that will be used by developers to request the system or to view information from the system are known as *user interfaces*. The user interfaces that will be used in the demo system are identified and named below:

- UI-01: Case List
- UI-02: Case View
- UI-03: Transformed View: Semantic Network
- UI-04: Case Matching: Semantic Network
- UI-05: Transformed View: Rule Based Reasoning Method
- UI-06: Case Matching: Rule Based Reasoning Method

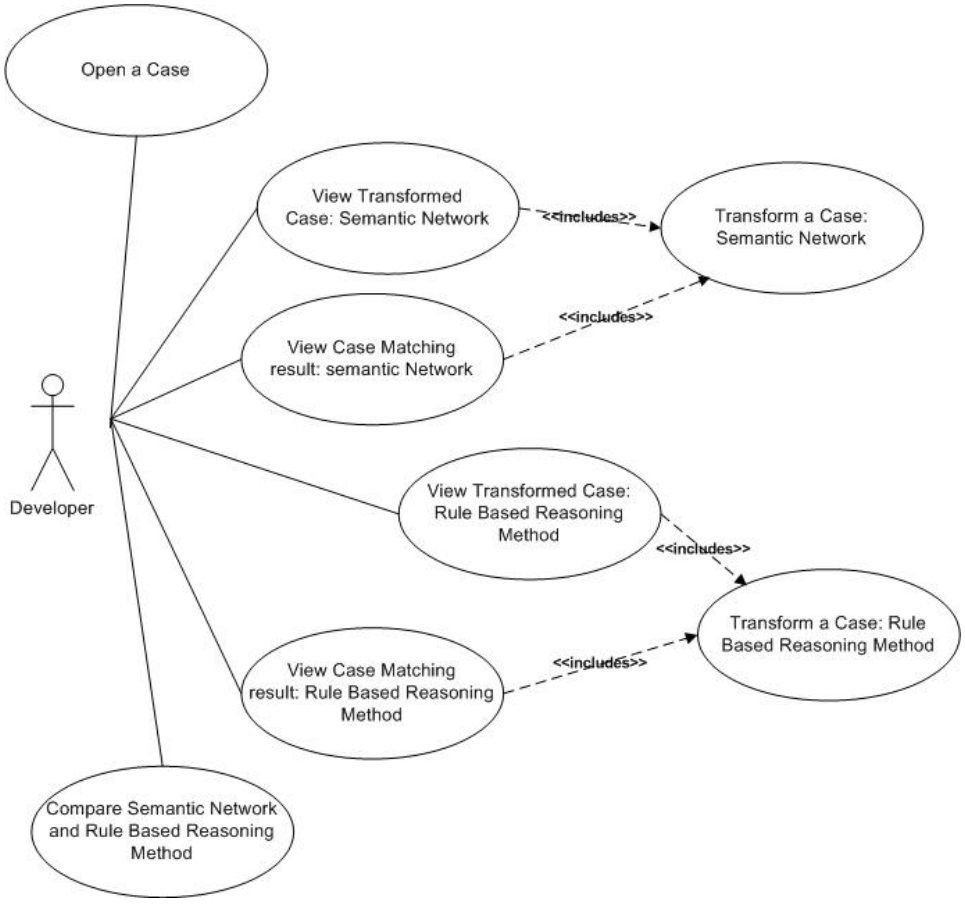

Figure 4.1: Use Case Diagram of the Demo System

4.4.1 Use Case Diagrams

A use case diagram in Figure 4.1 shows the interaction between the use cases and the actor of the system. The actor developer refers any person

who wants to see the demo system functions identified in the use cases. The use cases represent all the functions of the demo system that are concerns of this project. Some of the use cases are already developed and functional on the existing knowledge editor package. But they are included to show they are still part of the demo system.

4.4.2 Use Case Description

This section describes the use cases in the diagram.

Use-Case ID	UC-01
Use-Case Name	Open a Case
Description	A case shall be opened in the case view to perform the other actions.
Precondition	The knowledge editor is opened and it has list of cases in the case base (case list).
Post Condition	A case is opened in the case view from the case base.
Extends	
Includes	
Inherited From	
Actor	Developer
Basic Course of Action	1. The developer wants to open a case. 2. The demo system provides list of cases in the case base of the knowledge editor via *UI-01: Case List*. 3. The developer double clicks on the case that s/he wants to be opened. 4. The demo system opens the case via *UI-02: Case View* of the knowledge editor. 5. The use case ends.

Table 4.2: Use Case Description of UC-01.

Use-Case ID	UC-02
Use-Case Name	Transform a Case: Semantic Network
Description	It is to transform or elaborate a case by using semantic network.
Precondition	A case is opened in the case view through **UC-01.**
Post Condition	The case will be transformed.
Extends	
Includes	
Inherited From	
Actor	Developer
Basic Course of Action	1. The demo system wants to transform a case by using semantic network. 2. The demo system takes the case and identifies its hidden features by using the semantic network. 3. The demo system represents the case including the identified hidden features. 4. The demo system returns the transformed case. 5. The use case ends.

Table 4.3: Use Case Description of UC-02.

Use-Case ID	UC-03
Use-Case Name	View Transformed Case: Semantic Network
Description	It is to view transformed or elaborated form of the opened case while semantic network is used to identify the hidden features.
Precondition	A case is opened in the case view through **UC-01.**
Post Condition	The transformed case will be displayed.
Extends	
Includes	Transform a Case: Semantic Network
Inherited From	
Actor	Developer
Basic Course of Action	1. The developer wants to view the opened case after it is elaborated by using the semantic network. 2. The developer clicks on *UI-03 Transformed View: Semantic Network* which is tab of the case view. 3. The demo system transforms the case by using **UC-02 Transform a Case: Semantic Network**. 4. The demo system presents the transformed case via *UI-03 Transformed View: Semantic Network*. 5. The use case ends.

Table 4.4: Use Case Description of UC-03.

Use-Case ID	UC-04
Use-Case Name	View Case Matching result: semantic Network
Description	It is to view the similarity between the opened case with the other cases in the case base while semantic network is used in the case elaboration process.
Precondition	A case is opened in the case view through **UC-01.**
Post Condition	Similarity between the opened case and the other cases in the case base will be displayed.
Extends	
Includes	Transform a Case: Semantic Network
Inherited From	
Actor	Developer
Basic Course of Action	1. The developer wants to see similarity between the opened case with the other cases in the case base. 2. The developer clicks on *UI-04 Case Matching: Semantic Network* which is tab of the case view. 3. The demo system transforms the opened case by using **UC-02 Transform a Case: Semantic Network**. 4. The demo system takes one case at a time from the case base and transforms it by using **UC-02 Transform a Case: Semantic Network**. 5. The demo system compares and calculates the similarity between the two transformed cases. 6. The demo system repeats steps 4 and 5 for each case in the case base. 7. The demo system displays the similarity of the opened case with each case in the case base in percentage via *UI-04 Case Matching: Semantic Network*. 8. The use case ends.

Table 4.5: Use Case Description of UC-04.

Use-Case ID	UC-05
Use-Case Name	Transform a Case: Rule Based Reasoning Method
Description	It is to transform or elaborate a case by using rule based reasoning method.
Precondition	A case is opened in the case view through **UC-01.**
Post Condition	The case will be transformed by using the rule based reasoning method.
Extends	
Includes	
Inherited From	
Actor	Developer
Basic Course of Action	1. The demo system wants to transform a case by using rule based reasoning method. 2. The demo system takes the case and identifies its hidden features by using the rule based reasoning method. 3. The demo system represents the case including the hidden features. 4. The demo system returns the transformed case. 5. The use case ends.

Table 4.6: Use Case Description of UC-05.

Use-Case ID	UC-06
Use-Case Name	View Transformed Case: Rule Based Reasoning Method
Description	It is to view transformed or elaborated form of the opened case while rule based reasoning method is used to identify the hidden features.
Precondition	A case is opened in the case view through **UC-01.**
Post Condition	The transformed case will be displayed.
Extends	
Includes	Transform a Case: Rule Based Reasoning Method
Inherited From	
Actor	Developer
Basic Course of Action	1. The developer wants to view the opened case after it is elaborated by using the rule based reasoning method. 2. The developer clicks on *Transformed View: Rule Based Reasoning* tab of the case view. 3. The demo system transforms the case by using **UC-05 Transform a Case: Rule Based Reasoning Method**. 4. The demo system presents the transformed case via *UI-05 Transformed View: Rule Based Reasoning Method*. 5. The use case ends.

Table 4.7: Use Case Description of UC-06.

Use-Case ID	UC-07
Use-Case Name	View Case Matching result: Rule Based Reasoning Method
Description	It is to view the similarity between the opened case with the other cases in the case base while rule based reasoning method is used in the case elaboration process.
Precondition	A case is opened in the case view through **UC-01.**
Post Condition	Similarity between the opened case and the other cases in the case base will be displayed.
Extends	
Includes	Transform a Case: Rule Based Reasoning Method
Inherited From	
Actor	Developer
Basic Course of Action	1. The developer wants to see similarity between the opened case with the other cases in the case base. 2. The developer clicks on *UI-06 Case Matching: Rule Based Reasoning Method* which is tab of the case view. 3. The demo system transforms the opened case by using **UC-05 Transform a Case: Rule Based Reasoning**. 4. The demo system takes one case at a time from the case base and transforms it by using **UC-05 Transform a Case: Rule Based Reasoning**. 5. The demo system compares and calculates the similarity between the two transformed cases. 6. The demo system repeats steps 4 and 5 for each case in the case base. 7. The demo system displays the similarity of the opened case with each case in the case base in percentage via *UI-06 Case Matching: Rule Based Reasoning Method.* 8. The use case ends.

Table 4.8: Use Case Description of UC-07.

Use-Case ID	UC-08
Use-Case Name	Compare Semantic Network and Rule Based Reasoning Method
Description	It is to compare the two methods that can be used in the case elaboration.
Precondition	A case is opened in the case view through **UC-01.**
Post Condition	A developer will able to compare the role of semantic network and rule based reasoning method in the case elaboration process
Extends	
Includes	UC-04: View Case Matching result: Semantic Network, UC-07: View Case Matching result: Rule Based Reasoning Method
Inherited From	
Actor	Developer
Basic Course of Action	1. The developer wants to compare the role of semantic network and rule based reasoning method in the case elaboration process. 2. The developer wants to view the case matching result of the opened case with all other cases in the case base by using semantic network in case transformation process. 3. The demo system displays the case matching result by using **UC-04 View Case Matching result: Semantic Network**. 4. The developer records the name of three top similar cases with their similarity percentage from the case matching result. 5. The developer wants to view the case matching result of the opened case with all other cases in the case base by using rule based reasoning method in case transformation process. 6. The demo system displays the case matching result by using **UC-07 View Case Matching result: Rule Based Reasoning Method** . 7. The developer compares the top three similar cases from the case matching result with the record s/he has from the case matching result of the semantic network. 8. The use case ends.

Table 4.9: Use Case Description of UC-08.

This chapter has described what the demo system is intended to perform in general. Some of the use cases such as UC-01, UC-02, UC-03 and UC-04 are implemented and already functional in the existing case based system. The demo system will be developed on top of these functionalities. The reason they are included in this chapter is to show that they are still functionalities of the demo system. Hence the design and implementation phases of this project work will focus on the functionalities or use cases that need to be implemented.

Chapter 5

Design Phase

This chapter discusses how the demo system should be constructed to satisfy the requirements identified in Chapter 3. The overall system design is shown in Figure 5.1. As the figure shows the demo system has four basic components: *knowledge editor package, rule based component, case based component* and *user interfaces*.

The knowledge editor package provides the opened case in its case view component and the list of cases in the case base for the intended operation of the demo system.

The rule based component takes its input from the knowledge editor package which is the opened case or a case from the case base in XML format. The rule based component has sub components that process the case to identify the hidden features. Its output is transformed case that will be used by the case based component.

The case based component represents how similar the opened case is with the other cases in the case base after each case is transformed by using rule based reasoning method. Integration of the rule based reasoning method and the case based reasoning method is done here, in case similarity operation.

The user interface component represents the interfaces that will be used by users to interact with the demo system.

5.1 Knowledge Editor Package

The knowledge editor package is the base line of the demo system. Most of the functionalities in the existing package are taken as they are for the

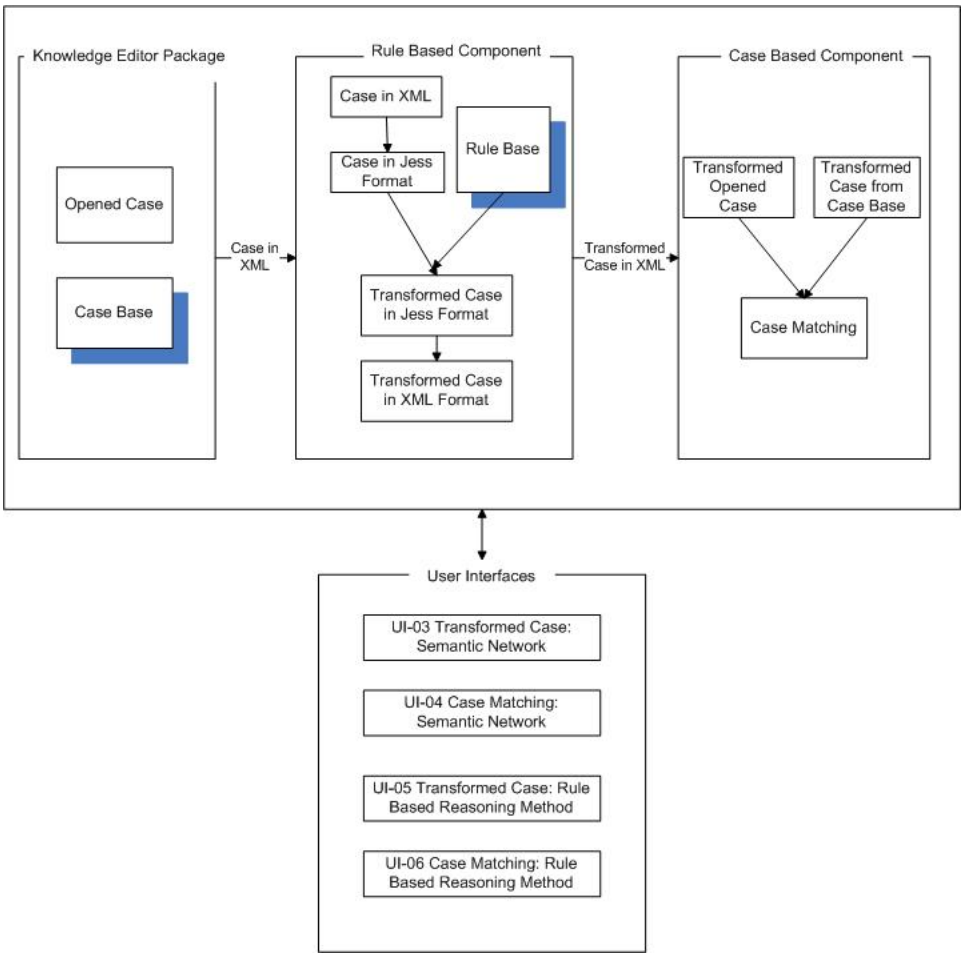

Figure 5.1: Overall Structure of the Demo System

demo system. Supplying cases for the demo system from the case base is part of these functionalities. The demo system provides list of cases so that the user can open a case to compare its similarity with the others.

5.2 Rule Based Component

The rule based component is designed and will be implemented based on Jess framework. The component involves sub tasks like *defining rules, representing the case in jess format, applying the rules to identify hidden features* and *representing the transformed case in XML format* so that it can be used by the case based component.

Rule based reasoning in Jess rule engine requires three main parts: *templates, rules* and *facts*. Templates define concepts or objects in the given domain by using set of properties. Facts are instantiation of objects or concepts defined in the templates. They represent the objects that are known. The rule engine returns an error if the rules or facts deals about an object which is not defined in the templates. Templates in Jess resemble with classes in Java programs. Each template has a name, which is like a class name, and set of slots, which are like properties of a class. Rules represent the domain knowledge in form of if ... then ... statements by using objects defined in the templates. [30]

Listing 5.1: Jess Syntax to Define a Template

```
(deftemplate template_Name
   [ Optional Comment about the template]
      (slot    slot_name_1)
      (slot    slot_name_2)
      (slot    slot_name_3)
      (slot    slot_name_n)
)
```

To define templates, Jess uses the syntax shown in Listing 5.1. The terms *deftemplate* and *slot* are Jess keywords that define the template and the slots, which describe the template, respectively.

The rule based component of the demo system is supposed to accept a case in XML format. A sample case is depicted in Appendix A.1. As shown in the sample case, cases are represented with concepts that are categorized into sections, sub-sections, and sequences. The concepts are described with set of properties which are tagged as entry. The associated value for each property is identified under them with symbolvalue or datavalue tag.

Listing 5.2: An Example for XML Representation of a Case

```
   <section name="Administrative Data">
      <entry parameter="Operator Company" source="Human">
         <symbolValue>Statoil</symbolValue>
      </entry>
      <entry parameter="Well Identification" source="Human">
         <symbolValue>Well 34/10-48A</symbolValue>
      </entry>
   <\section>
```

The example in Listing 5.2 shows some lines from XML representation of a case. It has one concept named as *Administrative Data* and the concept

is described with two properties named as *Operator Company* and *Well Identification*. The concept Administrative Data is instantiated with values of Statoil for its Operator Company property and Well 34/10-48A for its Well Identification property. This is a fact for the particular case from which this XML statements are taken.

In order Jess to understand about the concept Administrative Data and its two properties, they have to be defined as template and slots of the template respectively. Listing 5.3 shows the definition of template Administrative Data for Administrative Data concept.

Listing 5.3: An Example for a Template Definition of Listing 5.2

```
(deftemplate Administrative_Data
      Administrative data about a case
      (slot    Operator_Company)
      (slot    Well_Identification)
  )
```

Name of a template or a slot can not have a space in between words. If it consists more than a word, they have to be concatenated in some way.

For the demo system a script shall be written that will parse all cases to identify potential concepts and the associated properties. Categorization of the concepts in the case representation into sections, sub-sections and sequences helps to identify the concepts and properties easily. While the categories help to identify the concepts, the parameters for each entry are considered as properties of the associated concept.

It is also possible to parse the knowledge model to identify the concepts and properties instead of parsing every case in the case base. But, for this demo system parsing the knowledge model results in large number of templates which won't be used. In addition, the number of cases for the demo system is few and parsing them provides the important and precise templates. For the final system parsing the knowledge model is advisable. The algorithm in Listing 5.4 shows design of the script that will be used to parse the cases and to define the templates.

Listing 5.4: An Algorithm that Parses XML Format of a Case to define Templates

```
For each case
{
   Find the next entry and take its name from its parameter attribute
   Use the entry name to define a slot
      Find immediate parent category of the entry
      If the immediate parent category is sequence-section
      {
    Find name of the sequence which is parent of the sequence-section
      (Sequence-section doesn't have a name so that we have to use
       the sequence name under which the sequence-section is)
    Use the sequence name to define a template
       }

       If the immediate parent category is not sequence-section
    { Use the parent category to define a template }

   Check whether there is a defined template with the name of this parent
    category
   If there is a defined template with this parent category
   {
      Check whether there is a slot under this template with the name of
       this entry
      If there is no slot with this entry name under this template
    { Define the slot under this template with the name of this entry }
      If there is a slot with this entry name under this template
    { Go to the next entry and start from finding its name }
   }
   If there is no a defined template with this parent category
   {
      Define a template with the name of this parent category and define a
         slot under it with the name of the entry
   }
   Go to the next entry and start again from finding its name
}
```

A property can be under a sequence, where the sequence is under a section, and where the section may be under another section. The property describes only the immediate parent category. The hierarchical structure of the categories can be shown by using template inheritance like class inheritance in Java. However, the lower categories can't be described with properties of the super categories and there is no way of setting access control of properties in Jess unlike Java class inheritances. The templates need to be defined with properties which are immediate to them. Hence, templates inheritance is not applicable for this application domain.

5.2.1 Define Rules

Rules represent the domain knowledge in form of rules. The rules I have got from Verdande are written with if ... then ... format. They state that whenever the features in if part are known and they are true, then the features in then part are concluded to be true. All the rules are listed in Appendix B.1.

Jess has its own way of representing rules as shown in Listing 5.5. All rules in if ... then ... format has to be represented in Jess format in order to be used by the rule engine. The list of patterns in the syntax indicates the features that are in if part of a rule, the symbol => represents the term then, and the list of conclusions represents the features in then part of a rule.

Listing 5.5: Jess Syntax to Define a Rule

```
(defrule rule-name
     [ Optional Comment about the rule]
     (pattern 1)
     (pattern 2)

     (pattern n)
     =>
     (conclusion 1)
     (conclusion 2)

     (Conclusion n)
)
```

Listing 5.7 shows a simple example for defining a rule in Jess format for the rule that was represented in If ... then ... format as shown in Listing 5.6. The rule defines to print a message *The Company is Statoil* when there is a fact that instantiate the concept *Administrative Data* has its property *Operator Company* with a value of *Statoil*.

Listing 5.6: An Example of a Rule in IF ... THEN ... Format

```
IF      Operator Company is Statoil
   THEN      print the message "The Company is Statoil".
```

Listing 5.7: An example of a Rule in Jess Format

```
(defrule   simple_rule_example
   (Administrative Data     (Operator Company    "Statoil")
   =>
   (System.out.println("The Company is Statoil")
)
```

If part of a rule may have more than one features that are connected with simple/complex relations. Some features may need some kind of calculations. All should be considered and represented in the way that the Jess rule engine can understand them. The Jess manual in [31] provides strong background on defining rules in Jess.

5.2.2 Represent a Case in Jess Format

A situation to reason about is known as a fact. In the demo system the rule engine is supposed to reason about a case to identify its hidden features. Currently, the cases are represented in XML formats. They need to be represented in Jess format to be used by the jess engine. It can be done by writing a script that parses the XML file and generates a jess file.

The XML representation includes features and events that occur during the drilling operation at different depth and time. Normally, a case is a situation that occurs at specific time or situation. Though the XML file records drilling information at different depth and time, a case is captured at specific depth and time from the record. The drilling information for the case depends on the capturing depth and time. The case capturing depth and time are also recorded in the XML file. Though both capturing time and depth are important for the case, we agreed to use capturing depth for this demo system. Hence, to represent a case in Jess format from the XML file, the capturing depth will be the main factor to identify which drilling information is applicable and more relevant for the case.

The occurrence of some features and events is independent of the depth and time of the drilling operation. They have only one value. However, some features and events occur more than one time and/or vary their value depending on the depth and time of occurrence. For such kind of features and events, the captured case shall consider depth of occurrence.

Figure 5.2 depicts hierarchical structure of features and events for a sample

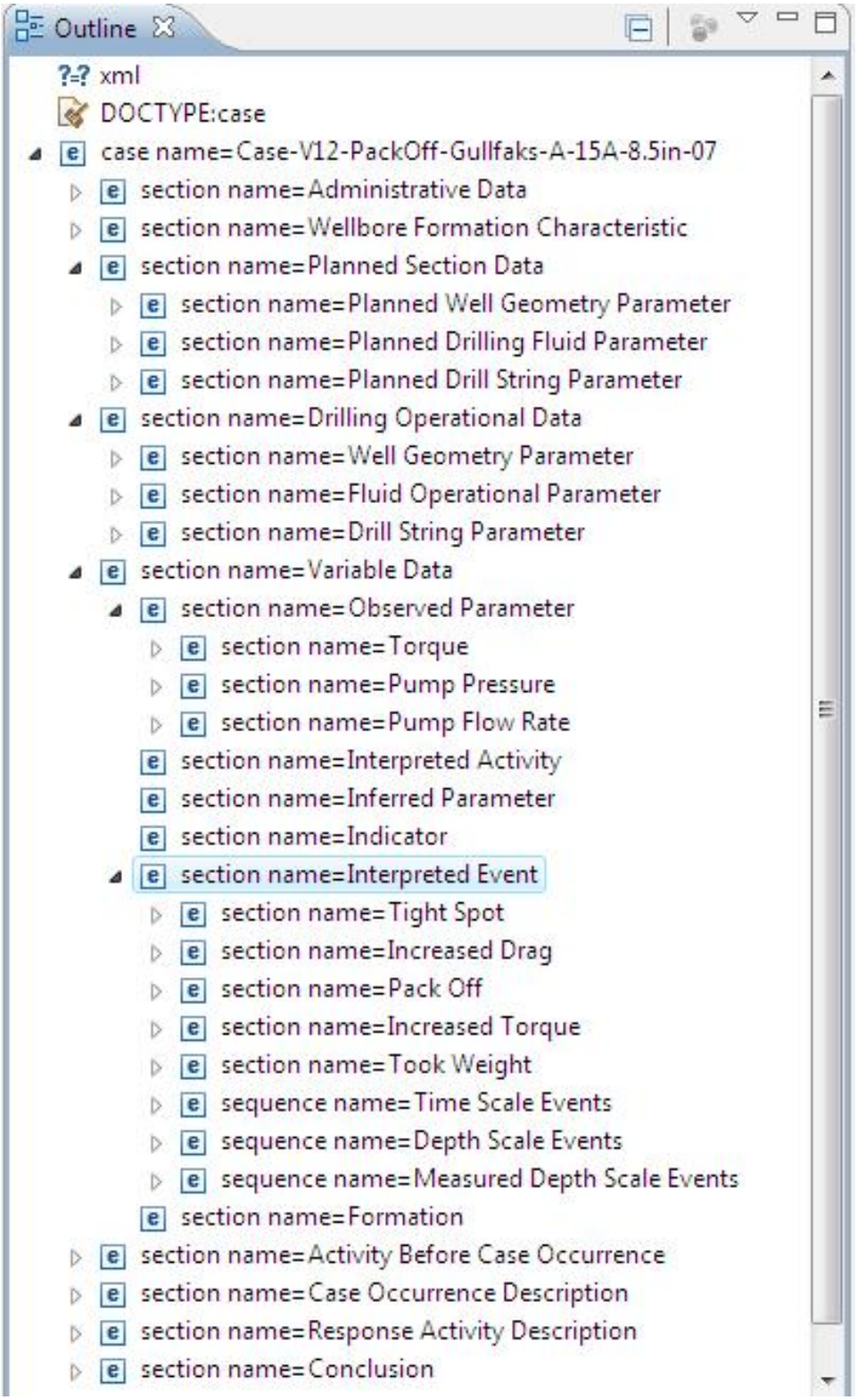

Figure 5.2: Hierarchical Structure of Features and Events in an XML Representation of a Case

case. Occurrence of all except *interpreted event* section is independent of the depth and time of drilling operation. The features and events under interpreted event section occurred more than one time and they have different values that are associated with different depth and/or time of the drilling operation. These different values shall be considered based on the depth where the case is captured. Two different techniques are proposed to handle the sections and sequences that are under the interpreted event section.

The sections *Tight Spot, Increased Drag, Pack Off, Increased Torque* and *Took Weight* have occurred at different depth and time of drilling operation. The important fact or information that is needed about these events is their occurrence and at what depth they occurred. If occurrence depth for an event is closer to the case capturing depth, it is more relevant for the captured case than the events that occur in far distance. Different weight is used to count the occurrence of such events based on their distance from the depth where the case is captured. The algorithm for weighting each event is shown in Listing 5.8.

Listing 5.8: An Algorithm for Weighting Occurrences of Events like Tight Spot Increased Drag Pack Off Increased Torque and Took Weight with regard to the Case Capturing Depth

```
If positive value of event_depth minus case_depth is less than or equal to
   100, the event occurrence will be counted as 1.
If positive value of event_depth minus case_depth is less than or equal to
   200 and greater than 100, the event occurrence will be counted as 0.5.
If positive value of event_depth minus case_depth is less than or equal to
   300 and greater than 200, the event occurrence will be counted as 0.33.
.
.
If positive value of event_depth minus case_depth is less than or equal to n
   and greater than n-100, the event occurrence will be counted as 1/n.
```

Time Scale Events and *Depth Scale Events* sequences repeat the same information that was recorded in the above sections by categorizing the events based on time of occurrence and depth of occurrence respectively. Hence considering these sequences for the captured case is not necessary if the sections are considered once.

In *Measured Depth Scale Events*, depth of the drilling operation is grouped in range of values with 10 meters difference. The events that occur in the range that includes the case capturing depth will be considered as events

of the captured case. For example, while the drilling depth is grouped into two like 5743.16 - 5753.16 and 5753.61 - 5763.61; if the case is captured at depth of 5751.87, all events that occur in the range of 5743.16 - 5753.16 will be evens of the captured case, since case capturing depth, 5751.87, is in this range.

Now the script that can parse the XML file to represent the captured case in Jess format can be designed. As discussed above, events and features with one value will be taken without considering depth or time of case capturing. Whereas, features and events that varies when the depth of drilling operation changes needs to be considered based on the case capturing depth. Parsing the XML file is similar with the one designed to parse cases and define the templates. The script for parsing the XML file and generating jess representation of the captured case is shown in Listing 5.9.

Listing 5.9: An Algorithm for Generating the Jess format of the Captured Case from the XML File

```
For each entry of a case
{
 Find its immediate parent
     If parent is description type [if it is simply to give description of
      a thing]
    Go to next entry
     If parent is not description type
     {
    If entry is independent of drilling depth
         Record it with the associated value and the immediate parent
        category.
    If entry is dependent with the drilling depth such as entries
        under interpreted events
    {
         If the entry parent is one of the events such as pack off events,
        tight spot events, increased torque events, increased drag
                  events or took weight events
         {
        Record the entry with the associated value and the immediate
           parent in a Jess file only one time though it occurs many times.
        Calculate the weight of the event occurrence and keep the sum of
           weights for each similar event occurrences.
        When similar events are finished, record the sum for the
           associated entry.
         }

         If the entry parent is measured depth scale events
         {
         Check if entry depth range includes the case capturing depth.
```

```
        If it is in the range
          Record the entry with the associated value and the parent
        If it is not in the range
          Ignore the entry and check the next entry depth range
         }
    }
     }
}
```

The script will be used to identify the template name , slot name and the associated values. When they are written in a jess file, they have to follow the syntax for defining facts. The syntax for defining facts in Jess is shown in Listing 5.10.

Listing 5.10: Jess Syntax to Define Facts

```
(template-name
  (slot-name   slot-value)
)
```

The example in Listing 5.11 shows Jess representation for the XML case representation example shown in Listing 5.2.

Listing 5.11: Jess Representation of the XML Case Representation Example in Listing 5.2

```
(Administrative_Data
   (Operator_Company    "Statoil")
   (Well_Identification     "Well 34/10-48A")
)
```

5.2.3 Applying the Rules to Identify Hidden Features

Once the knowledge is represented in form of rules, the inference engine decides which rule should be executed for a given fact. The inference engine uses an algorithm called Rete [1] to match the rules with the given facts in order to identify which rules are applicable for the facts. The reasoning process is taking place in the working memory. The rules react for the addition, deletion, and modification of facts in the working memory. [30]

[1] The Rete algorithm is an efficient pattern matching algorithm for implementing production rule systems.[52]

In the above sections we have seen how facts can be defined but not yet added into the working memory. To be used by the rule engine, they have to be added by using the Jess key word *assert or add*. Listing 5.12 shows the addition of a fact into the working memory. Now, the fact will be readily available in the working memory for the reasoning process.

Listing 5.12: An Example Using of Assert to Add a Fact into Working Memory

```
(assert   (Administrative Data
      (Operator Company    "Statoil")
                (Well Identification     "Well 34/10-48A")
      )
)
```

We have seen a definition of a template in Listing 5.3, definition of a rule in Listing 5.7 and definition of fact in Listing 5.12. Though all definitions are done, the reasoning process is not started yet. Because the inference engine needs an explicit request to start reasoning about facts in the working memory. The Jess key word *run*, is used to do so.

If we launch the reasoning process by using the key word run, the inference engine takes the fact from the working memory, which is the Operator Company of Administrative Data object has a value of Statoil. It then searches rules of which the if statements have matched with the fact at hand. Then the conclusion of the matched rules will be applied. In the example the conclusion of the matched rule is to print a message, hence the message *The Company is Statoil* will be printed on the console. The conclusion of a rule can be to print a message, to add another fact into the working memory, to modify some values of existing facts, etc ....

5.2.4 Representing the Transformed Case into XML Format

Output of the rule based component is an XML format of the case that includes the hidden or inferred features. After the reasoning process is finished, the inferred features should be added into the existing XML representation of the case which was the input of the rule based component.

Template name of an inferred feature will be the parent category as section in the XML representation, name of the slot will be the name of the entry and the value will be the associated value for the entry.

For example, if we assume the feature in Listing 5.13 is inferred from given facts, it can be added at the end of the XML representation of the input case as shown in Listing 5.14

Listing 5.13: An Example for an Inferred Feature in Jess Format

```
(Administrative_Data
   (Company_Owners   Private)
)
```

Listing 5.14: XML Representation of the Inferred Feature shown in Listing 5.13

```
   <section name="Administrative Data">
      <entry parameter=" Company Owners" source="System">
      <symbolValue>private</symbolValue>
      </entry>
   <\section>
```

5.3 Case Based Component

The main function of the case based component is to perform case matching. There is no big change on this funcitonality for the demo system from the existing case based system. The only difference is in its input. The demo system is supposed to receive a transformed case by semantic network or by rule based reasoning depending on the user request. However the existing system receives a transformed case by semantic network only. Though it uses different types of input types, the case matching process is independent of the method used to transform the case, hence the demo system uses the same case matching process.

5.4 User Interface Component

As shown in the use case model at Chapter 3, the users interact with the demo system by using user interfaces. Figure 5.1 shows the user interface component has four main user interfaces that are required for the demo system. UI-03 and UI-04 are to accept requests from a user to display a transformed case and case matching result by using the semantic network in the case transformation process. These case transformation and case

matching processes exist in the current system. The only task they require for the demo system is to change the name of the user interfaces. In the current system, the interfaces are named with *Transformed View* and *Case Match*, they don't intuitively indicate what method is used to transform the cases. Hence changing the names with *Transformed Case: Semantic Network* and *Case Matching: Semantic Network* will enable to identify what method has been used to transform the cases.

User interfaces UI-05 and UI-06 are totally new for the demo system. They need to be added as additional tabs into the case view component of the knowledge editor package. When a user clicks on either of these tabs, the clicked tab has to display the requested information. The user interfaces initiate transformation of a case by using rule based reasoning method and case matching processes of the demo system based on the user request.

5.5 Integration of the Rule Based Component with the Knowledge Editor Package

As mentioned before, the rule based component receives its input from the knowledge editor package when a user requests to view transformed view of a case or when the case matching process is to be performed. The package understands and uses the case throughout the system performance with its representation as semantic network in the knowledge model. The editor package has methods to convert a case representation from knowledge model to XML format and from XML format to representation in the knowledge model. These methods can be used to let interaction between the rule based component and the knowledge editor package.

The reason that XML format is chosen for the rule based component instead of using the case representation in the knowledge model, is the desire to make the rule based component independent of the semantic network.

The class diagram in Figure 5.3 shows how the rule based component can be integrated with the knowledge editor package. The two Java packages *volve.creek.gui.representation* and *volve.creek.gui.representation.casexml* are taken from the existing system while the package *volve.rules* is the rule based package that is added to the existing system. The *SeparatedCasePane* class

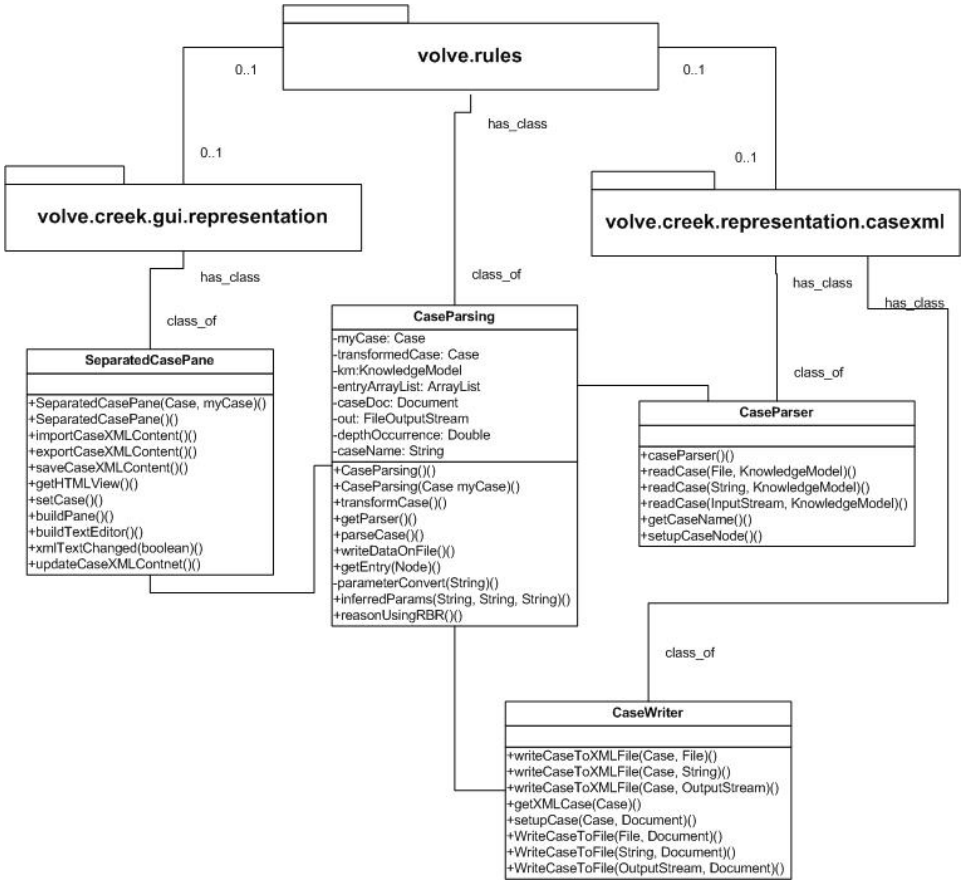

Figure 5.3: Class Diagram for Integration of the Rule Based Component with the Knowledge Editor Package

responds for requests that come from users via the knowledge editor package. *CaseParsing* class of the rule based component interacts with SeparatedCasePane class to access user request and the opened case. As mentioned earlier, since the case is represented as semantic network, the CaseParsing class interacts with *CaseWriter* class to convert it into XML format. After the rule based component has transformed the case, the CaseParsing class interacts with *CaseParser* class to convert the XML format into case representation on the knowledge model. Then the rule based component provides the converted case to the SeparatedCasePane class to display the transformed case to the user.

5.6 Integration of the Rule Based Component with the Case Based Component

The main goal to integrate the rule based reasoning method with the case based reasoning method in this project is, to support the case matching process of the case based reasoning method by using the rule based reasoning method to elaborate the cases.

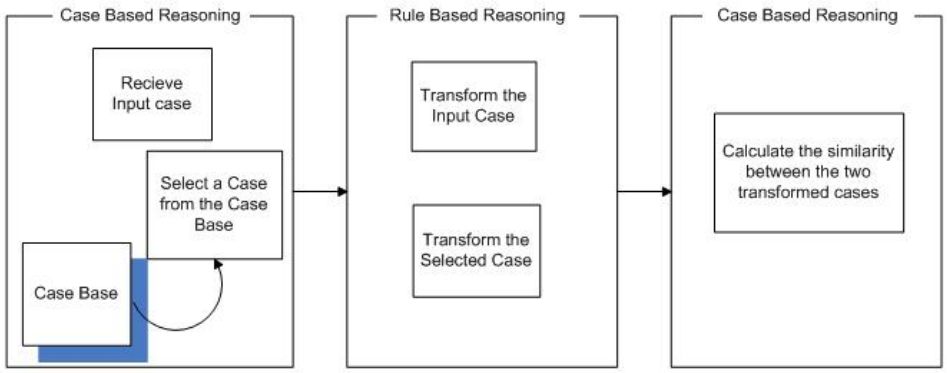

Figure 5.4: An architecture for integrating the rule based reasoning method with the existing case based system

As Figure 5.4 shows the case based reasoning method receives the input case and provides it to the rule based reasoning method for further elaboration. It does the same after it retrieves a case from the case base. The rule based reasoning method interferes between retrieving a case and case matching

processes of the case based reasoning method. The rule based component is designed independent of the case based component. The interaction between the two reasoning methods can be done by sending messages. The case based component can call the rule based reasoning method with a message of the case to be elaborated. When the rule based reasoning method has finished its transformation, it can call the case based reasoning method with a message of the transformed case.

The flow of logic in the integrated system to describe the behavior of the demo system is illustrated in Figure 5.5.

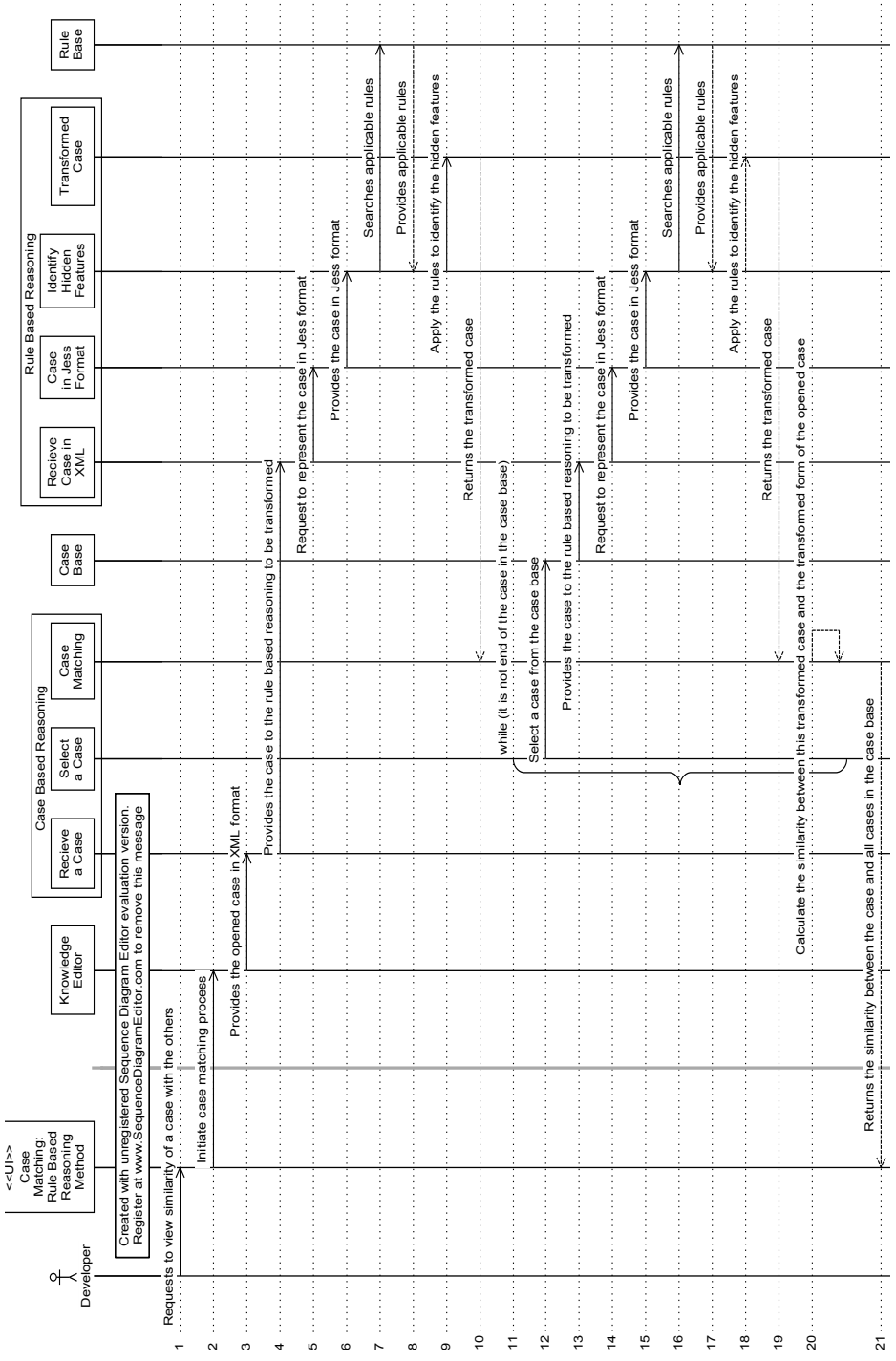

Figure 5.5: A sequence diagram that shows the sequence of actions that the integrated system performs to achieve case matching by using rule based reasoning method in the case elaboration. The message in box at the top is not part of the sequence diagram. It is displayed since I have used freely available tool to draw the diagram.

The following chapter will discuss the implementation of the demo system based on the designed scripts and algorithms proposed in this chapter.

Chapter 6

Implementation Phase

This chapter is concerned about documentation of how the demo system is implemented. As discussed in Chapter 3 and shown in Figure 5.1, the demo system has different components. Implementation of the demo system is divided based on the components.

6.1 Implementation of the Rule Based Reasoning Component

The rule based reasoning component is developed independent of the case based component. Making it independent is chosen because modifiability of the rule based component or the case based component can be done without affecting the other. In addition the source code for the case based component was not allowed to take and use it out of Verdande computers, which required going there to perform the implementation. Because of these reasons, making the rule based component independent is advantageous to keep quality of component modifiability and implementing the rule component in flexible times.

The two components interact by sending and receiving an XML representation of a case. The case based component sends the case that needs to be elaborated and the rule based reasoning component sends the transformed form of the case for which it has received.

Before the rule based reasoning component has started accepting and transforming cases, it needs to have the templates and the rules on which the transformation process is performed. The component uses Jess files to con-

tain the templates, the rules and the fact. The reasoning process will use the files to get the necessary data. The following sections discuss about implementation of defining templates, defining rules, defining facts and the transformation process.

6.1.1 Implementation of Defining Templates

Based on the proposed algorithm, the script is written in Java to parse the sample cases given for this demo system. The script accepts one case at a time and writes potential templates on a Jess file. When it parses another case, it checks availability of the concepts as templates and/or properties as slots of the associated template in the file, if the slot or combination of the template and the slot is new, it will be added to the file. The templates are written on the Jess file by following Jess syntax to define templates; however the code that defines the templates is written in Java.

Some of the rules have new concepts and properties which have not been used in any of the sample cases. I have added these concepts and properties into the template file by checking their parent object on the knowledge model. Some of the rules have also included concepts and properties which are new and have not been added in the knowledge model. Since the rule based component is supposed to be integrated with the case based component, all the concepts and properties that have to be used in the templates, rules, or facts, must exist in the knowledge model. This required updating the knowledge model by adding the new concepts and properties. Adding the new concepts and properties in the knowledge model is discussed in the following section. Figure 6.1 shows part of the knowledge model that includes most of newly added concepts and properties with some existing concepts.

The defined templates are shown in Appendix C, they are saved with a jess file called *template.clp* for the demo system.

6.1.2 Implementation of Defining Rules

The rules I received from Verdande are statements with if . . . then . . . format, as shown in Appendix B.1 , where the if part indicates features that are observed or known about the situation and the then part indicates features

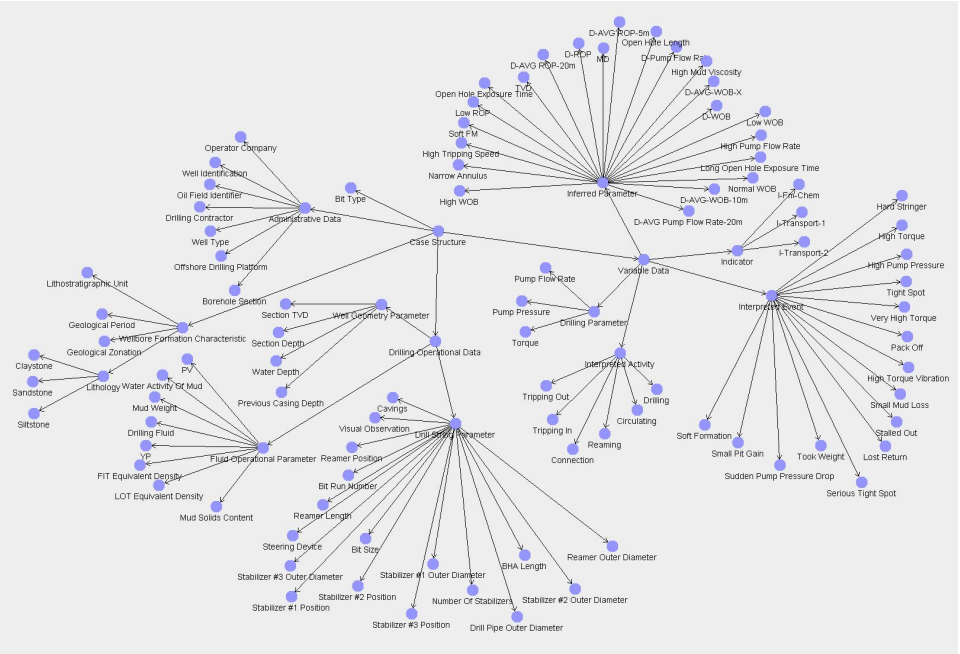

Figure 6.1: Concepts and Properties Representation in the Knowledge Model

that can be inferred from the known features. Based on the rule in Listing 6.1, for example, when a situation has greater than 2000 L/min of Instantaneous Pump Flow Rate, it means, the situation has High Pump Flow Rate. The feature High Pump Flow Rate is inferred from the feature Instantaneous Pump Flow Rate.

Listing 6.1: An Example of a Rule

```
IF Instantaneous Pump Flow Rate > 2000
   THEN High Pump Flow Rate
```

When I received the rules, the features used in the rules haven't been checked for their existence in the knowledge model. When the templates are defined all properties used in the features are checked for their existence on the knowledge model and those that are new are added to the knowledge model. Adding a property to the knowledge model required identifying its parent category or under which concept it belongs to. It is done by discussing it with people at Verdande.

The knowledge model uses different relations to relate concepts and properties with concepts. The relation that is main focus of the rule based component is the one that relates a property and its immediate parent category. This is indicated with a relation called *has case entry parameter* or *case entry parameter of* based on the direction from which the relation is. The new properties and concepts are added and related with this relation. The existing knowledge model didn't explicitly relate most of the properties with their immediate parent categories. Hence, updating the knowledge model to indicate the relation between them was part of the implementation task. This knowledge model updating will help to parse the knowledge model in order to define all templates for the big system by identifying this relation between the concepts and their properties.

For the rule example in Listing 6.1, the immediate parent category of Instantaneous Pump Flow Rate and High Pump Flow Rate is *Pump Flow Rate*. In the knowledge model, the relation is indicated as Pump Flow Rate has case entries of instantaneous pump flow rate and high pump flow rate; inverse of the relation is, instantaneous pump flow rate and high pump flow rate are case entry parameters of pump flow rate.

Once it is confirmed that all the parameters used in the rules exist in the knowledge model, defining the rules in Jess format to be used by the rule engine is possible. Jess representation of the rule example in Listing 6.1 is shown in Listing 6.2.

Listing 6.2: An Example of a Rule in Jess Format

```
(defrule rule_defination_example
    (_Pump_Flow_Rate {_Instantaneous_Pump_Flow_Rate > 2000})
    =>
    (assert (_Pump_Flow_Rate (_High_Pump_Flow_Rate true)))
    (InferredParam "_Pump_Flow_Rate" "_High_Pump_Flow_Rate" true)
)
```

The rule performs two tasks when instantaneous pump flow rate of pump flow rate object is greater than 2000. The first task is to add the inferred feature high pump flow rate into the working memory by using assert key word of Jess. Adding inferred features into the working memory helps to find other rules that are applicable for the situation by having the inferred features in their if statement.[1] For example, in Listing 6.3 the rule has the

[1]The rules react for the facts that are in the working memory.

inferred feature high pump flow rate in its if part. The rule will be active whenever high pump flow rate, soft fm and narrow annulus are known to be true and are available in the working memory. If they are not added explicitly into the working memory, any of them won't be there. Assert statement in Listing 6.2 adds the feature High Pump Flow Rate is true for the given situation in the working memory.

Listing 6.3: An Example of a Rule that uses an Inferred Feature in its If part

```
If    High Pump Flow Rate
   AND Soft FM
   AND Narrow Annulus
THEN  Hydraulic Erosion
```

The second task is to add the inferred feature into the case representation. The second statement after => calls a Jess function called InferredParam which accepts three parameters - the immediate parent category of the inferred feature, the name of the inferred feature and the associated value for the inferred feature. The Jess function InferredParam calls a java method that accepts the same parameters and writes the inferred feature into the XML representation of the case. The jess code in Listing 6.4 is the inferredParam function. It accesses a java class named *caseParsing* to instantiate an object named as *?pars*. The function uses the object to call the java method *inferredParams* which is member of class caseParsing.

Listing 6.4: A Jess Function named as InferredParam that calls a Java Method named as InferredParams

```
(deffunction InferredParam (?parent ?inferredFeature ?value)
   (bind ?pars (new caseParsing))
        (call ?pars inferredParams ?parent ?inferredFeature ?value)
 )
```

All other rules are defined in the same way; each rule performs the two aforementioned tasks for the associated inferred feature. Appendix B.2shows Jess representation of the rules that was accepted in IF ... THEN ... format from Verdande.

6.1.3 Implementation of Representing a Case in Jess Format

Every case that needs to be transformed requires to be represented in Jess format and to be added into the working memory of Jess. The algorithm proposed in Chapter 5 is used and written in Java to parse the XML representation of the case. The Java class DocumentbuilderFactory is used to obtain a parser that produces a DOM document of the case from the XML file.

Listing 6.5: Parsing an XML File to Generate a DOM Document.

```
DocumentBuilderFactory caseDocFac = DocumentBuilderFactory.newInstance();
DocumentBuilder newCase = caseDocFac.newDocumentBuilder();
caseDoc = newCase.parse(fileName);
```

The code snippet in Listing 6.5 shows instantiation of the Documentbuilder-Factory class with a name of caseDocFac, which is used to define an instance of DocumentBuilder class, named as newCase, to obtain the DOM document from the XML file. The newCase instance parses the XML file given with a name of fileName and generates the DOM document named as caseDoc by using parse method. The DOM document structures the content of the XML file in hierarchical, a tree like format.

Java provides application programming interfaces (API) to access different kinds of data from a DOM document. Such kinds of interfaces are used to identify properties, their immediate parent category, and the associated value for the parameters from the DOM document of the XML file. The following example illustrates how the different data are accessed and used from the DOM document.

Let's say the XML lines in Listing 6.6 are taken from an XML file that is parsed and represented in a DOM document named *caseDoc*.

Listing 6.6: Some Lines from an XML File

```
<case name="Case-V12-PackOff-Gullfaks-48A-8.5in-02" status="solved">
   <section name="Administrative Data">
      <entry parameter="Operator Company" source="Human">
         <symbolValue>Statoil</symbolValue>
      </entry>
      <entry parameter="Well Identification" source="Human">
         <symbolValue>Well 34/10-48A</symbolValue>
      </entry>
      <entry parameter="Oil Field Identifier" source="Human">
```

```
        <symbolValue>Gullfaks</symbolValue>
      </entry>
   </section>
   .
   .
   .

</case>
```

As discussed earlier, our interest in Jess representation of the case is to identify the properties or features about the case, the immediate parent category of the properties and their associated value in that particular case. The parameters are represented inside *<entry>* tag of the XML representation. The tags in the XML are represented as elements with the tag name in the DOM document. The code snippet in Listing 6.7 illustrates how the parameters can be accessed from the DOM document.

Listing 6.7: Accessing Elements from a DOM Document

```
Element rootnode = caseDoc.getDocumentElement();
NodeList sections = rootnode.getElementsByTagName("entry");
```

The method getDocumentElement() retrieves the document element, which is the root element of the DOM document. All XML statements in the file are enclosed with *<case>* and *</case>* tags. While the first represents the beginning of the case representation and the latter represents end of the case representation. All other tags are enclosed with these tags. Hence <case> is the root tag which is also the root element of the DOM document; when the DOM document is generated from the XML file, it keeps logical order of the file. Once the root element is obtained, it is possible to access all other elements which are under it. The first statement in the above code snippet returns the root element of the DOM document caseDoc.

Part of the code *rootnode.getElementsByTagName("entry")* indicates to find all elements whose tag name is *entry* from the rootnode. It returns array of elements named with entry which are under the rootnode. In this example, as shown in the XML representation in Listing 6.6, there are three elements whose tag name is entry; hence the NodeList *sections* will have three elements.

As shown in the XML representation the name of each entry is given for the attribute called *parameter*. To access attributes of elements in a DOM docu-

ment, *NamedNodeMap* Java interface can be used as shown in the following code snippet.

Listing 6.8: Identifying Properties Immediate Parent Categories and Associated Values for each property from a DOM Document

```
NamedNodeMap attributes;
for(int i=0; i<sections.getLength();i++)
{
   attributes = sections.item(i).getAttributes();

   // to find the attributes of elements which is the properties
   String param =attributes.getNamedItem("parameter").getNodeValue();

   //to find the immediate parent category of the parameter
   parentNode =  sections.item(i).getParentNode().getAttributes().
             getNamedItem("name").getNodeValue());

   //to find value of the parameter
   NodeList children = sections.item(i).getChildNodes();
   Element value = (Element)children.item(1);

   if(children.item(1).getNodeName().equals("dataValue"))
   {
      paramvalue = value.getFirstChild().getNodeValue() ;
   }
   else if (children.item(1).getNodeName().equals("symbolValue"))
   {
      paramvalue = "\"" + value.getFirstChild().getNodeValue() + "\"";
   }

}
```

For the example, the for loop is executed 3 times since the NodeList sections has 3 elements. Each element is accessed with *sections.item(i)* .The string variable param will have name of the property for the given element. The method getParentNode() returns an element which is the immediate parent category of the element that calls it. If the parent element is *sequence-section*, it needs other way of handling since it doesn't have any attribute called *name*. The associated value of the parameter is also obtained by accessing the child nodes.

Once the three values are identified, they are written on a jess file by following the Jess format to write a fact. The following code shows Jess representation of the XML example which is written by the program on Jess file. The key word asserts lets the data to be added on the working memory of Jess rule engine.

Listing 6.9: Jess Representation of the XML Statements in 6.6 that will be written on Jess File

```
(assert  (_Administrative_Data
   (_Operator_Company "Statoil")
   (_Well_Identification "Well 34/10-48A")
   (_Oil_Field_Identifier "Gullfaks")
))
```

The whole code that parses the XML file to generate and write its Jess representation on Jess file is attached in Appendix E.

Jess representation of the case in Appendix A is shown in Appendix D; after the case is parsed by the program discussed above.

6.1.4 Implementation of Applying the Rules to Identify Hidden Features

In order the rule engine to reason about the cases, it needs to have the templates, the rules which are knowledge of the domain, and the facts which are the engine to reason about. So far, we have seen how the templates are defined, the rules are represented and how a case can be represented in Jess and added in the working memory. All are saved in Jess files they are not given to the engine yet.

The templates and the rules are the same for all cases, hence they are saved in a file called *Template.clp* and every time when a cases is parsed its Jess representation is saved in a file called *newCase.clp*. The rule engine needs to access content of both files to reason about the case. The function in Listing 6.10 shows how the two files are given to the engine in order to start reasoning about the case.

Listing 6.10: A Java Function that Initiates Rule Based Reasoning by Embedding Jess Code

```
@SuppressWarnings("deprecation")
private void reasonUsingRBR()
{
   Rete engine = new Rete();
   try {
   engine.executeCommand("(batch \"Template.clp\")");
   engine.executeCommand("(batch \"newCase.clp\")");
   }
```

```
   catch(Exception e){
   e.printStackTrace();
   }
}
```

The function embeds Jess codes inside the Java function. The class *Rete* declares an object called *engine* that uses Rete algorithm to reason about the cases based on the files that are added into the rule engine by using Jess command of *batch*.

Since Jess representation of the case in *newCase.clp* have included the keyword assert, when the file is added, the facts goes to the working memory directly. The rule engine takes the facts from the working memory and searches applicable rules from *Template.clp*. When it gets the rules, it performs what is ordered on the conclusion of the applicable rules. In this demo system every rule performs two things: to add the inferred features into working memory for further reasoning process and to register the inferred features on the XML representation of the case.

6.1.5 Implementation of Representing the Transformed Case into XML Format

As discussed in Section 6.1.2, every rule has a statement in its conclusion to write the inferred features into the XML file when the rule is applicable for the given situation. Hence, the inferred features are added into the original XML representation of the case, when the applicable rule is executed.

6.2 Implementation of User Interface Component

As Figure 6.2 shows two new tabs are added and named as *Transfomed View: Rule Based Reasoning Method* and *Case Matching: Rule Based Reasoning Method* in the case view. The two existing tabs that were named as *Transformed View* and *Case Match* are renamed with *Transformed View: Semantic Network* and *Case Matching: Semantic Network* respectively. Implementation of the user interface is based on the discussion in user interface design section 5.4 in Chapter 5.

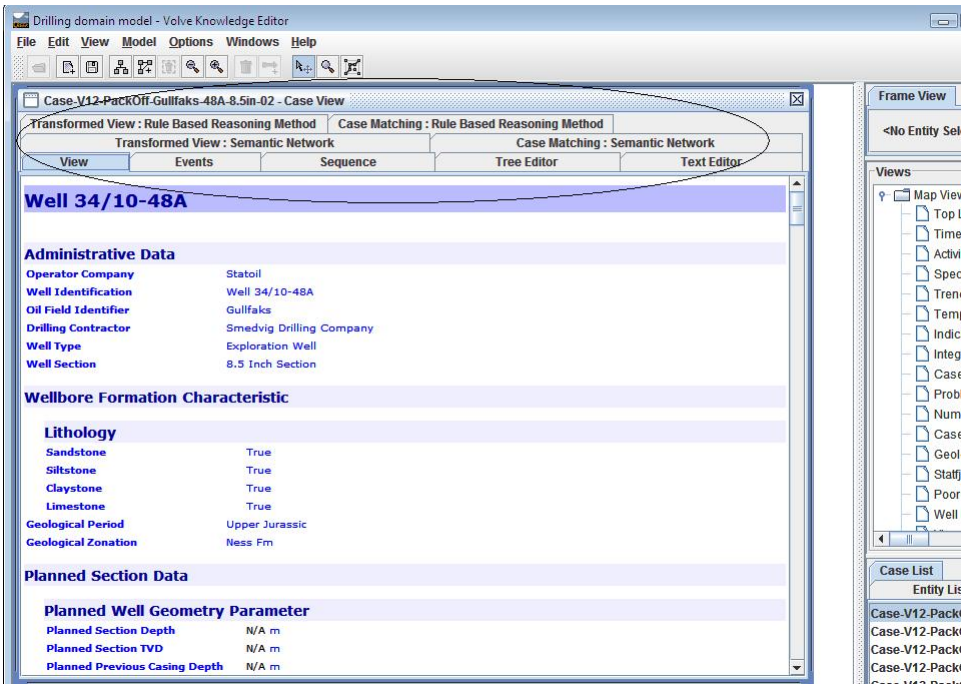

Figure 6.2: Added and Renamed Tabs in the Case View of the Demo System

6.3 Implementation of Integration of the Rule Based Component with the Knowledge Editor Package

The rule based component and the knowledge editor package are integrated to transform a case and to display the transformed case. Their interaction is by sending messages. A case is send to the rule based component from the knowledge editor package when a case is opened by a user. And a transformed case is send by the rule based component to the knowledge editor package to display it on the case view.

The integration of the two components is implemented based on the design discussed and proposed in 5.5. Figure 6.3 shows the transformed form of a case in the case view. As shown in the figure the feature *Low WOB*[2] was inferred for the given case.

[2]Only 3 rules were defined in Jess at this time; had it been using all the rules, more features would have been inferred.

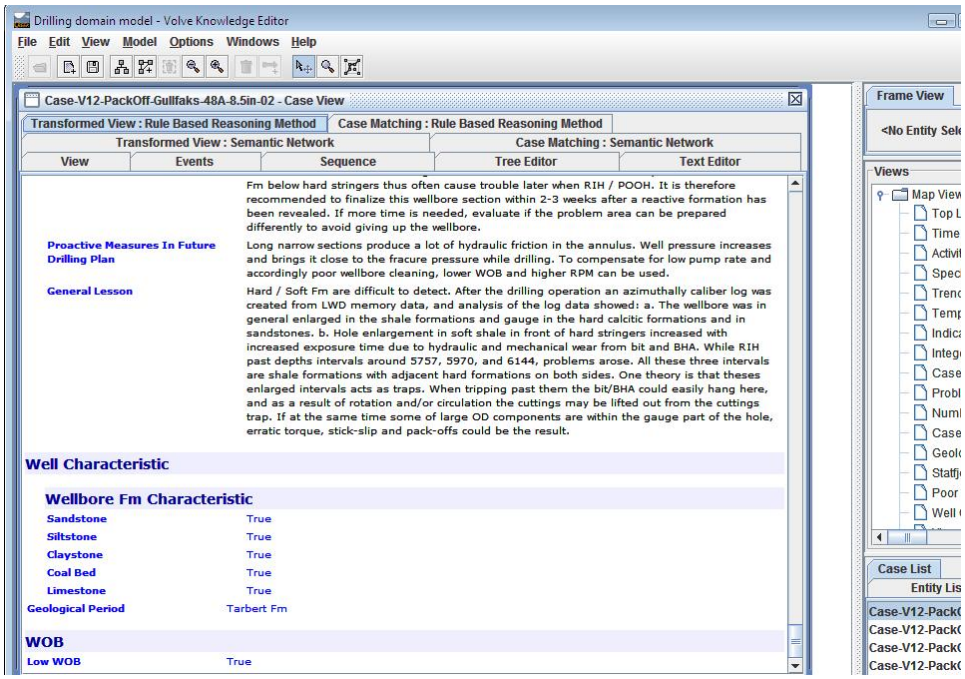

Figure 6.3: Tranformed Case by using Rule Based Reasoning Method in the Case Veiw

6.4 Implementation of Integration of the Rule Based Component with the Case Based Component

This section deals about case matching functionality of the demo system by using the rule based reasoning method in the case elaboration process. Functionality of elaborating a case by using rule based reasoning method is working as shown in Figure 6.3. However, implementation of using it in the case matching process is not finished. The main reason for not it to be completed was shortage of time, because implementation of the existing case based system used large number of packages and classes which are really challenging to understand them. It took longer time to identify which Java packages are relevant and which are not for the demo system. Even after that, while some of the classes were commented, some were not at all; which created difficulty on understanding of what class performs what. In addition, the source code was accessible only at Verdande which prevented me to work in flexible times.

Though I am not successful, I tried to integrate them with the time I had. The difficulty that I faced on my trial was accessing each case from the case base to compare them with the opened case.

In the next chapter the demo system will be tested if it achieves the desired objectives mentioned in previous chapters.

Chapter 7

Testing Phase

This chapter focuses on making sure that the objective of the project work is achieved. It prepares a test plan to facilitate testing activities to be carried out on the demo system and to document its results. Documentation of this testing phase is in accordance with the standard *IEEE Standard for Software Test Documentation [IEEE Std 829-1998]* [11]. The IEEE 829 is very detailed to this demo system; hence some sections of the standard are left out.

7.1 Test Plan

The purpose of the plan is to provide and record important information about the testing activities to be carried on the demo system. It identifies what to be and not to be tested based on the functionalities identified in Chapter 4. It also includes the approaches that will be followed to perform the testing activities.

7.1.1 Test Plan Identifier

The unique identifier for this test plan is **RBRCBRI-TP**, which stands for Rule Based Reasoning and Case Based Reasoning Integration Test Plan. This identifier will be used to refer anything from the test plan.

7.1.2 Reference

The test plan is also based on the IEEE 829 standard. Chapter 4 , Chapter 5 and Chapter 6 are the main sources to identify items and feature that will and will not be tested.

7.1.3 Test Items

The testing process focuses on testing of the rule based component of the demo system and any process that relies on this component. The reason for the focus on the rule based component is, the other components are available and functional on the existing system while the rule based component is new for the demo system. Hence, the items that will be tested are the rule based component and its integration with the knowledge editor package and with the case based component of the demo system

7.1.4 Features to be Tested

Features of the rule based reasoning component that will be included in the testing are the functional requirements with high and medium priorities. They are listed in tables from 7.1 to 7.6.

Test Feature Identifier	RBRCBRI-TP-01
Test Feature Name	Jess Representation of a Case
Test Feature Description	The rule based component should represent the XML case in Jess format.
Functional Requirement Reference	FR-01, FR-02

Table 7.1: Test Feature of Jess Representation of a Case

7.1.5 Features not to be Tested

Functionalities of the demo system that are regarding with the semantic network will not be tested in detail. They are already tested and functional in the existing case based system. Nothing is changed on their functionality

Test Feature Identifier	RBRCBRI-TP-02
Test Feature Name	Identifying Hidden Features of a Case
Test Feature Description	The rule based component should identify hidden features of a given case based on available rules.
Functional Requirement Reference	FR-01, FR-02, FR-03

Table 7.2: Test Feature of Identifying Hidden Features of a Case

Test Feature Identifier	RBRCBRI-TP-03
Test Feature Name	Presenting the Elaborated Case
Test Feature Description	The demo system should integrate the rule based component with the knowledge editor package and should present the case on the case View after it is elaborated by using rule based reasoning method.
Functional Requirement Reference	FR-01, FR-02, FR-03, FR-04, FR-05

Table 7.3: Test Feature of Presenting the Elaborated Case

Test Feature Identifier	RBRCBRI-TP-04
Test Feature Name	Case Matching: Rule Based Method
Test Feature Description	The demo system should integrate the rule based component with the case based component and should perform case similarity while the rule based reasoning method is used in case elaboration process.
Functional Requirement Reference	FR-01, FR-02, FR-03, FR-04, FR-05, FR-06, FR-07

Table 7.4: Test Feature of Case Matching: Rule Based Method

Test Feature Identifier	RBRCBRI-TP-05
Test Feature Name	Preserving Functionalities of the Existing System
Test Feature Description	The demo system should keep all functionalities of the existing case based system on the demo system while semantic network is used in the case elaboration process.
Functional Requirement Reference	FR-08, FR-09, FR-10, FR-11

Table 7.5: Test Feature of Preserving Functionalities of the Existing System

Test Feature Identifier	RBRCBRI-TP-06
Test Feature Name	User Interfaces
Test Feature Description	The demo system should use intuitive and simple to use user interfaces.
Functional Requirement Reference	FR-12, FR-13

Table 7.6: Test Feature of User Interfaces

for the purpose of the demo system. Testing their availability on the demo system is enough.

Non-functional requirements of the demo system are requirements that need to be considered when the system is developed to achieve the functional requirements. They are not necessary or directly related with the functionalities of the system. Hence, they will not be included on the testing activities.

7.1.6 Testing Approaches

Software testing methods are divided into two major types: white-box testing and black-box testing. In white-box testing, the tester has access to the source code and concerns about how the program is performing its tasks in detail. Alternatively, black-box testing concerns only the functionality of the system as seen from the outside. It provides an input and waits to see the output without dealing about how the program produces the output.

Testing of the demo system follows both white box and black box approaches. The white box method is chosen to test the individual components and integration of components before they make up the final system. The black box method is chosen to test the final system after all integrations are done. Testing activities of the demo system are categorized into *module testing, integration testing, system testing* and *usability testing.* Module and integration testing are performed while the demo system is implemented.

Module Testing: it is a white box testing that tests how the rule based component performs its task before it is integrated with the knowledge editor package and with the case based component. Test features RBRCBRI-TP-01 and RBRCBRI-TP-02 are module testing.

Integration Testing: it is also a white box testing that tests the integration of the rule based component with the knowledge editor package. The test feature RBRCBRI-TP-03 is an integration testing.

System Testing: this is a black box testing that tests the final product after the rule based reasoning process is integrated with the case based reasoning process to make the case matching process of the case based

component. Test features RBRCBRI-TP-04 and RBRCBRI-TP-05 are system testing that should be done on the completed demo system.

Usability Testing: It is testing of interfaces to be sure that the interfaces are user-friendly, understandable, and easy to use. Test feature RBRCBRI-TP-06 is usability testing.

Test cases are prepared and the testing activities will be performed based on the information given in the test cases to make sure that the output produced is the same as the output given in the test case.

7.1.7 Feature Pass/Fail Criteria

The pass and fail criterion of each test feature is presented in Section 7.2.2.

7.1.8 Suspension Criteria and Resumption Requirements

One of the main goals of testing is to identify errors and to correct them. When the testing is conducted, some errors might be noticed. If the error is minor and do not affect the process, testing activity will proceed and the error will be corrected after testing has been finished. But if the error halts the testing process, it has to be corrected and should be tested again to make sure the problem is corrected.

7.1.9 Test Deliverables

The deliverables after testing phase are the test plan, the test case specification and test summary. The test summary will provide testing results of each test feature.

7.1.10 Environmental Needs or Requirements

Environmental requirements to perform the testing activities are categorized into *software requirements* and *document requirements*.

Software Requirements: to perform the module testing a computer installed with Java Virtual Machine and an eclipse platform where jess rule engine is configured is required, whereas, to perform the other testing, the knowledge editor package is needed in addition to the above requirements.

Document Requirements: the primary documents that are required during testing are the test plan and test case specifications which will be used as guideline to conduct testing activities. Requirement specifications, design phase and implementation phase act as supportive documents, because they are base for preparation of the test plan and test case specifications.

7.1.11 Testing Tasks and Schedule

The activities that will be conducted during testing phase are summarized and scheduled as shown in Table 7.7.

Testing Task	**Predecessor Tasks**	**Deadline**
(t1) Prepare test plan	Requirement specification, design phase, and Implementation Phase	21.06.2009
(t2) Prepare test case specification	(t1), Implementation Phase	23.06.2009
(t3) Execute module testing	(t1), (t2)	24.06.2009
(t4) Execute integration testing	(t1), (t2), (t3)	25.06.2009
(t5) Execute system testing	(t1), (t2), (t3), (t4)	26.06.2009
(t6) Execute usability testing	(t1), (t2), (t3), (t4), (t5)	26.06.2009
(t7) Resolve errors	(t1), (t2), (t3), (t4), (t5), (t6)	28.06.2009
(t8) Write test summary	(t1), (t2), (t3), (t4), (t5), (t6), (t7)	30.06.2009

Table 7.7: Testing Tasks and Schedule

7.2 Test case specification

Preparing test cases for the features to be tested helps to make sure that the features are tested in the right way by providing the necessary inputs and the expected output.

7.2.1 Test case identifier

The unique identifier for this test case specification document is **RBRCBRI-TC**, which stands for Rule Based Reasoning and Case Based Reasoning Integration Test Case.

7.2.2 Test cases

This section will present test cases for each feature to be tested. Each test case has the necessary input and the expected output.

To perform the module test, RBRCBRI-TP-01 and RBRCBRI-TP-02, it is not necessary to have the knowledge editor package. They are tested by accessing the source code during implementation phase. It is to make sure that the smallest units are working well before they are integrated to make the bigger unit. They use a sample input case named as *Case-V12-PackOff-Gullfaks-48A-8.5in-02*. Some lines from the sample case is attached in Appendix A.

The other tests need the rule based component to be integrated with the knowledge editor package and with the case based component. In this case, the test requires the knowledge editor package and the input case will be from the case base.

Test	**Description**
Test Case Identifier	RBRCBRI-TC-01
Test Feature Identifier	RBRCBRI-TP-01
Test Feature Name	Jess Representation of a Case
Input Specification	Provide path of the XML file to the constructor *caseParsing(filePath)* in the main method of *caseParsing class* or *caseParsing.java*.
Output Specification	A Jess file with name of the case, *Case-V12-PackOff-Gullfaks-48A-8.5in-02.clp*, will be generated and it will represent the case in Jess format.
Environmental Specification	A computer installed with Java Virtual Machine and an eclipse platform where jess rule engine is configured.
Special procedural requirement	The tester opens caseParsing.java file and edit the parameter of caseParsing constructor with the path of the file that contains the XML representation of the case. And then run the program.
Inter-Test Case dependencies	None
Pass Criteria	The Jess file named with *Case-V12-PackOff-Gullfaks-48A-8.5in-02.clp* should be generated and it contains the case in Jess format like in Appendix D.
Fail Criteria	A file with the name of the case is not generated or/and the case is not represented in Jess format.

Table 7.8: Test Case for Jess Representation of a Case

Test	**Description**
Test Case Identifier	RBRCBRI-TC-02
Test Feature Identifier	RBRCBRI-TP-02
Test Feature Name	Identifying Hidden Features of a Case
Input Specification	Representing of a case in Jess format is tested and working. The templates and the rules are defined and available for the program.
Output Specification	Hidden features of the case will be generated and added in the original XML representation of the case.
Environmental Specification	A computer installed with Java Virtual Machine and an eclipse platform where jess rule engine is configured is required.
Special procedural requirement	There is no special procedural requirement; when the program runs to test RBRCBRI-TC-01, this also runs by default. They are inter related.
Inter-Test Case dependencies	RBRCBRI-TC-01
Pass Criteria	Hidden features of the case are identified and added in the original XML file, *Case-V12-PackOff-Gullfaks-48A-8.5in-02.xml*
Fail Criteria	The hidden features are not identified while the case has some hidden features based on the given rules. Or the hidden features are not added in the original XML file.

Table 7.9: Test Case for Identifying Hidden Features of a Case

Test	**Description**
Test Case Identifier	RBRCBRI-TC-03
Test Feature Identifier	RBRCBRI-TP-03
Test Feature Name	Presenting the Elaborated Case
Input Specification	The rule based component is integrated with the knowledge editor package and the package is running.
Output Specification	The case that contains the hidden features (the transformed case) will be presented in user friendly format on the case view of the package under *View Transformed Case: Rule Based reasoning Method*.
Environmental Specification	A computer installed with Java Virtual Machine and an eclipse platform where jess rule engine is configured is required. In addition, the knowledge editor package is required.
Special procedural requirement	The tester opens a case from the case list and clicks on *View Transformed Case: Rule Based reasoning Method* tab of the case view.
Inter-Test Case dependencies	RBRCBRI-TC-01, RBRCBRI-TC-02
Pass Criteria	The transformed case should be displayed in user friendly manner under *View Transformed Case: Rule Based reasoning Method* tab of the case view.
Fail Criteria	The transformed case is not displayed under the given tab.

Table 7.10: Test Case for Presenting the Elaborated Case

Test	**Description**
Test Case Identifier	RBRCBRI-TC-04
Test Feature Identifier	RBRCBRI-TP-04
Test Feature Name	Case Matching: Rule Based Method
Input Specification	The rule based component is integrated with the knowledge editor package and the case based component.
Output Specification	Similarity of the opened case will be compared with all other cases and the result will be displayed on the case view.
Environmental Specification	A computer installed with Java Virtual Machine and an eclipse platform where jess rule engine is configured is required. In addition, the knowledge editor package is required.
Special procedural requirement	The tester opens a case from the case list and clicks on *Case Matching: Rule Based Reasoning Method* tab of the case view.
Inter-Test Case dependencies	RBRCBRI-TC-01, RBRCBRI-TC-02, RBRCBRI-TC-03
Pass Criteria	Similarity result in percentage should be displayed under *Case Matching: Rule Based Reasoning Method* tab of the case view.
Fail Criteria	Nothing or something else is displayed under *Case Matching: Rule Based Reasoning Method* tab of the case view.

Table 7.11: Test Case for Case Matching: Rule Based Method

Test	**Description**
Test Case Identifier	RBRCBRI-TC-05
Test Feature Identifier	RBRCBRI-TP-05
Test Feature Name	Preserving Functionalities of the Existing System
Input Specification	The rule based component is integrated with the knowledge editor package and the case based component.
Output Specification	Both *Transformed View: Semantic Network* and *Case Matching: Semantic Network* are available and working.
Environmental Specification	A computer installed with Java Virtual Machine and an eclipse platform where jess rule engine is configured is required. In addition, the knowledge editor package is required.
Special procedural requirement	The tester opens a case from the case view and clicks in both tabs turn by turn to see they are working to display the transformed case and the case matching respectively while semantic network is used in the case transformation process.
Inter-Test Case dependencies	RBRCBRI-TC-03
Pass Criteria	Both *Transformed View: Semantic Network* and *Case Matching: Semantic Network* are working and displaying the required information.
Fail Criteria	Any of the two functionalities is not working.

Table 7.12: Test Case for Preserving Functionalities of the Existing System

Test	**Description**
Test Case Identifier	RBRCBRI-TC-06
Test Feature Identifier	RBRCBRI-TP-06
Test Feature Name	User Interfaces
Input Specification	The knowledge editor package is running
Output Specification	The name of the tabs in the case view is descriptive and they are responsive for a single click.
Environmental Specification	A computer installed with Java Virtual Machine and an eclipse platform where jess rule engine is configured is required. In addition, the knowledge editor package is required.
Special procedural requirement	The tester opens a case on the case view from the case list.
Inter-Test Case dependencies	None
Pass Criteria	The names of the tabs are simple and understandable
Fail Criteria	Names of the tabs are not simply understandable.

Table 7.13: Test Case for User Interfaces

7.3 Test Summary

The testes are performed based on the test cases. The result for each test feature is shown in Table 7.14. The result of the test will be marked as *PASSED* if the test result shows the expected result; or will be marked as *FAILED* if the test result shows unexpected and unacceptable result. If there is a failed test, the summary will provide a description why it fails.

Test Feature	**Achieved Functional Requirement**	**Test Result**	**Description**
RBRCBRI-TP-01	FR-01, FR-02	PASSED	-
RBRCBRI-TP-02	FR-01, FR-02, FR-03	PASSED	-
RBRCBRI-TP-03	FR-01, FR-02, FR-03, FR-04, FR-05	PASSED	-
RBRCBRI-TP-04	FR-01, FR-02, FR-03, FR-04, FR-05, FR-06, FR-07	FAILED	Implementation of case matching while rule based reasoning is used on case transformation is not completed. But most of the components that are required for this functionality are implemented and tested on the other features.
RBRCBRI-TP-05	FR-08, FR-09, FR-10, FR-11	PASSED	-
RBRCBRI-TP-06	FR-12, FR-13	PASSED	-

Table 7.14: Test Summary

Chapter 8

Discussion, Recommendation and Conclusion

8.1 Discussion

This project work studied and designed how a rule based reasoning component can be developed and integrated with the existing case based system. A demo system is developed to demonstrate the role of the rule based reasoning method on case transformation to support the existing case based decision support system on oil drilling operation domain.

On the demo system, the rule based component is implemented and tested to make sure that it has achieved the designed goal. The component performs its task successfully. Integration of the rule based component with the case based component is completed with 75%. It successfully works for transforming the opened case which is the one to be compared with all other cases in the case base. However, transforming cases from the case base for case matching process is not implemented because of time.

8.2 Recommendation

The demo system was supposed to help comparing and contrasting the result of using rule based reasoning method or semantic network on cases transformation. It would have been possible if the case matching process was working while the rule based reasoning method is used on the case transformation. The problem I faced on implementing this functionality is on accessing the cases from the case base since all cases are represented on the

knowledge model as semantic network. A person who is familiar with the source code of the existing case based system can fix the problem based on the design proposed to integrate the two components.

The relation between properties and immediate parent categories are updated on the knowledge model with a relation *has case entry parameter* or *case entry parameter of* which may help if parsing the knowledge model to identify all properties with their immediate parent category is needed.

The selected rule engine is Java Expert System Shell (Jess) which can be integrated with a java program as shown in the demo system. I used its freely available version. But, it has licensed version for commercial use if Verdande wants to use it for the big system.

8.3 Conclusion

The demo system illustrates how the rule based component transforms a case based on the available rules. Hidden features which were not able to be identified by the semantic network are now identifiable because of the rule based reasoning method. The rule based reasoning method uses complex relations among the features which were not possible by the semantic network. With more and unique features a situation is described, the more likely to get the right and most similar past case from the case base.

Appendix A

Sample for XML Format of a Case

Listing A.1: A sample for XML Representaiton of a Case

```
<? xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE case PUBLIC "Volve_caseSpec9" "VolveCase.dtd">
<case name="Case-V12-PackOff-Gullfaks-48A-8.5in-02" status="solved">
<section name="Administrative Data">
   <entry parameter="Operator Company" source="Human">
      <symbolValue>Statoil</symbolValue>
   </entry>
   <entry parameter="Well Identification" source="Human">
      <symbolValue>Well 34/10-48A</symbolValue>
   </entry>
</section>
<section name="Wellbore Formation Characteristic">
   <entry expertRelevance="0.2" parameter="Geological Period" source="DBR">
      <symbolValue>Upper Jurassic</symbolValue>
   </entry>
   <section name="Lithology">
      <entry expertRelevance="0.5" parameter="Sandstone" source="Human">
         <symbolValue>True</symbolValue>
      </entry>
   </section>
</section>
<section name="Planned Drilling Fluid Parameter">
   <entry dataConfidence="0.9" expertRelevance="0.2" parameter="Planned Mud Weight"
      source="DBR" statisticalWeight="0.0">
      <dataValue unit="kg/l" valueType="Double">null</dataValue>
   </entry>
</section>
<section name="Drilling Operational Data">
   <section name="Well Geometry Parameter">
      <entry parameter="Section Depth" source="DBR">
         <dataValue unit="m" valueType="Double">6621.0</dataValue>
      </entry>
   </section>
   <section name="Fluid Operational Parameter">
      <entry dataConfidence="0.9" expertRelevance="0.2" parameter="Mud Weight"
         source="DBR" statisticalWeight="0.0">
```

```
         <dataValue unit="kg/l" valueType="Double">1.57</dataValue>
      </entry>
   </section>
</section>
<sequence name="Depth Scale Events">
<scale datatype="Double" name="Depth"/>
   <sequence-section>
   <position end="5000.97" scale="Depth" start="5000.91"/>
      <entry parameter="Took Weight" source="System" statisticalWeight="0.3">
         <dataValue valueType="String">True</dataValue>
      </entry>
   </sequence-section>
   <sequence-section>
   <position end="5108.25" scale="Depth" start="5108.17"/>
      <entry parameter="Increased Torque" source="System" statisticalWeight="0.5">
         <dataValue valueType="String">True</dataValue>
      </entry>
   </sequence-section>
</sequence>
<sequence name="Measured Depth Scale Events">
<scale datatype="Double" name="Depth"/>
   <sequence-section>
   <position end="5721.16" scale="Depth" start="5711.16"/>
      <entry parameter="Stabilizer friction" source="system">
         <dataValue valueType="Double">181429.86324324342</dataValue>
      </entry>
      <entry parameter="PackOff" source="system">
         <dataValue valueType="Double">0.0</dataValue>
      </entry>
      <entry parameter="MFI Erosion" source="system">
         <dataValue valueType="Double">1640232.713233295</dataValue>
      </entry>
   </sequence-section>
</sequence>
</section>
.
.
.
</section>
</case>
```

Appendix B

List of Rules for the Demo System

B.1 IF ... THEN ... Format of the Rules

B.1.1 Deriveded parameters

The parameters in the if part are observable parameters, which can be read from the case representations.

- IF Bit Size - Stabilizer #1 Outer Diameter $<$ 2 in THEN Narrow Annulus
- IF ROP $<$ 5 m/h THEN Low ROP
- IF Instantaneous Pump Flow Rate $>$ 2000 l/min THEN High Pump Flow Rate
- IF WOB/ Bit Size $>$ 2 ton THEN High WOB
- IF WOB/ Bit Size =1-2 ton THEN Normal WOB
- IF WOB/ Bit Size $<$1 ton THEN Low WOB
- IF PV $>$ 25 cP THEN High Mud Viscosity
- IF Exposure Time $>$ 2 days THEN Long Open Hole Exposure Time
- IF Well Inclination $>$ 60 THEN High Inclination Change

B.1.2 Parameters Derived from Rules

Some of the parameters in the if parts are the hidden parameters that are derived from observable parameters in B.1.1.

- IF High Pump Flow Rate AND Soft Fm AND Narrow Annulus THEN Hydraulic Erosion
- IF Soft Fm AND High Side Stress AND Low ROP THEN Mechanical Erosion
- IF ROP $>$ 10 m/h AND Normal WOB THEN Soft Fm
- IF High WOB AND Narrow Annulus THEN High Side Stress
- IF $>$ 2 Increased Torque in one Case OR $>$ 2 Pack Off in one Case OR $>$ 2 Tight Spot in one Case OR $>$ 2 Increased Torque in one Case THEN Many Events

B.1.3 Root cause of Poor Hole Cleaning

- IF Soft Fm AND Many Events THEN Time Dependent Wellbore Instability
- IF Soft Fm AND Long Exposure Time THEN Time Dependent Wellbore Instability
- IF High Hydraulical Erosion OR High Mechanical Erosion OR Mechanical Erosion OR Hydraulic Erosion THEN Hole Enlargement
- IF High ROP AND High RPM AND Long Event Time AND Shoulder OR Ledges AND Pack Off THEN Cuttings Accumulation
- IF Tripping AND Pack Off AND Shoulder OR Ledges THEN Cuttings Accumulation
- IF Pack Off AND Ledges OR Shoulders OR Hole enlargement THEN Cuttings Accumulation

B.2 Jess Format of the Rules

Listing B.1: Jess Representation of some of the Rules

```
(defrule rule1
    "this is to test it"
    (_Measured_Depth_Scale_Events {_ROP > 5})
   =>
    (assert (_ROP (_Low_ROP True)))
    (InferredParam "ROP" "Low ROP" True)
)

(defrule rule2
    "rule about PV"
   (_Fluid_Operational_Parameter {_PV > 25})
    =>
    (assert (_Mud_Viscosity (_High_Mud_Viscosity true)))
    (InferredParam "Mud Viscosity" "High Mud Viscosity" True)
)

(defrule rule3
    "rule about PV"
   (_Pump_Flow_Rate {_Instantaneous_Pump_Flow_Rate > 2000})
    =>
    (assert (_Pump_Flow_Rate (_High_Pump_Flow_Rate true)))
    (InferredParam "Pump Flow Rate" "High Pump Flow Rate" True)
)

(defrule rule4
    "Rule for narrow annulus"
    (_Drill_String_Parameter
        (_Bit_Size ?X)
        (_Stabilizer_#1_Outer_Diameter ?Y))
        (test (< (- ?X ?Y) 2))
         =>
    ;(printout t "Narrow Annulus is true" crlf)
    (assert (_Annular_Flow_Parameter (_Narrow_Annulus true)))
    (InferredParam "Annular Flow Parameter" "Narrow Annulus" True)
)

(defrule rule5
    (_Drill_String_Parameter
        ;(test (> (- _Bit_Size _Stabilizer_#1_Outer_Diameter) 2))
        (_Bit_Size ?X))
    (_Measured_Depth_Scale_Events
        (_WOB ?Y))
    (test (> (/ ?Y ?X) 2))
         =>
    (assert (_WOB (_High_WOB true)))
```

```
    (InferredParam "WOB" "High WOB" True)
)

(defrule rule6
    (_Drill_String_Parameter
         (_Bit_Size ?X))
    (_Measured_Depth_Scale_Events
        (_WOB ?Y))
    (test ((< (/ ?X ?Y) 2) && (> (/ ?X ?Y) 1)))
         =>
    (assert (_WOB (_Normal_WOB true)))
    (InferredParam "WOB" "Normal WOB" True)
)

(defrule rule7
    (_Drill_String_Parameter
       (_Bit_Size ?X))
    (_Measured_Depth_Scale_Events
        (_WOB ?Y))
    (test (< (/ ?Y ?X) 2))
         =>
     (assert (_WOB (_Low_WOB true)))
    (InferredParam "WOB" "Low WOB" True)
)
```

Appendix C

Sample Templates Definitions for the Demo System

Listing C.1: Sample Templates for the Demo System

```
(deftemplate _Administrative_Data
   (slot  _Operator_Company)
   (slot _Well_Identification)
   (slot _Oil_Field_Identifier)
   (slot _Drilling_Contractor)
   (slot _Well_Type)
   (slot _Well_Section)
)
(deftemplate _Wellbore_Formation_Characteristic
   (slot  _Geological_Period)
   (slot _Geological_Zonation)
)
(deftemplate _Lithology
   (slot  _Sandstone)
   (slot _Siltstone)
   (slot _Claystone)
   (slot _Limestone)
   (slot _Marl)
)
(deftemplate _Planned_Well_Geometry_Parameter
   (slot  _Planned_Section_Depth)
   (slot _Planned_Section_TVD)
   (slot _Planned_Previous_Casing_Depth)
)
(deftemplate _Planned_Drilling_Fluid_Parameter
   (slot  _Planned_Mud_Weight)
   (slot _Planned_Drilling_Fluid)
)
(deftemplate _Planned_Drill_String_Parameter
   (slot  _BHA_Length)
   (slot _Number_Of_Stabilizers)
)
```

```
(deftemplate _Well_Geometry_Parameter
   (slot  _Section_Depth)
   (slot _Section_TVD)
   (slot _Previous_Casing_Depth)
   (slot _Water_Depth)
   (slot _Target_Depth)
   (slot _TVD)
)
(deftemplate _Fluid_Operational_Parameter
   (slot  _Mud_Weight)
   (slot _PV)
   (slot _Water_Activity_Of_Mud)
   (slot _YP)
   (slot _Drilling_Fluid)
   (slot _FIT_Equivalent_Density)
   (slot _LOT_Equivalent_Density)
   (slot _Mud_Solids_Content)
   (slot _Average_Mud_Weight)
   (slot _Average_PV)
   (slot _Average_YP)
   (slot _Number_Of_String_Wipers)
)
(deftemplate _Drill_String_Parameter
   (slot  _Bit_Run_Number)
   (slot _Bit_Size)
   (slot _Bit_Type)
   (slot _Drill_Pipe_Outer_Diameter)
   (slot _BHA_Length)
   (slot _Number_Of_Stabilizers)
   (slot _Stabilizer_#1_Outer_Diameter)
   (slot _Stabilizer_#2_Outer_Diameter)
   (slot _Stabilizer_#3_Outer_Diameter)
   (slot _Stabilizer_#1_Position)
   (slot _Stabilizer_#2_Position)
   (slot _Stabilizer_#3_Position)
   (slot _Reamer_#1_Outer_Diameter)
   (slot _Reamer_#1_Position)
   (slot _Reamer_Outer_Diameter)
   (slot _Reamer_Length)
   (slot _Reamer_Position)
   (slot _Steering_Device)
)
(deftemplate _Visual_Observation
   (slot  _ROP_Was_Limited_To_Clean_Well)
)
(deftemplate _Torque
   (slot  _Instantaneous_Torque)
   (slot _Torque_Trend)
)
(deftemplate _ECD
   (slot  _Instantaneous_ECD)
   (slot _ECD_Trend)
```

```
)
(deftemplate _Tight_Spot_Events
   (slot  _Tight_Spot)
   (slot count)
)
(deftemplate _Pump_Pressure
   (slot  _Instantaneous_Pump_Pressure)
   (slot _Pump_Pressure_Trend)
)
(deftemplate _Pump_Flow_Rate
   (slot  _Instantaneous_Pump_Flow_Rate)
   (slot _Pump_Flow_Rate_Trend)
   (slot _Normal_Pump_Flow_Rate)
   (slot _High_Pump_Flow_Rate)
   (slot _Recent_Pump_Flow_Rate)
   (slot _Low_Pump_Flow_Rate)
   (slot _Decreased_Pump_Flow_Rate)
   (slot _Very_High_Pump_Flow_Rate)
   (slot _Very_Low_Pump_Flow_Rate)

)
(deftemplate _Took_Weight_Events
   (slot  _Took_Weight)
   (slot count)
)
(deftemplate _Increased_Torque_Events
   (slot  _Increased_Torque)
   (slot count)
)
(deftemplate _Increased_Drag_Events
   (slot  _Increased_Drag)
   (slot count)
)
(deftemplate _Pack_Off_Events
   (slot  _Pack_Off)
   (slot count)
)
(deftemplate _Time_Scale_Events
   (slot  _Took_Weight)
   (slot  _Tight_Spot)
   (slot _Increased_Torque)
   (slot _Increased_Drag)
   (slot _Pack_Off)
)
(deftemplate _Depth_Scale_Events
   (slot  _Took_Weight)
   (slot  _Tight_Spot)
   (slot _Increased_Torque)
   (slot _Pack_Off)
   (slot _Increased_Drag)
)
(deftemplate _WOB
```

```
   (slot _Very_High_WOB)
   (slot _High_WOB)
   (slot _Normal_WOB)
   (slot _Low_WOB)
   (slot _Very_Low_WOB)
   (slot _Expected_WOB)
)
(deftemplate _Instantaneous_ROP
   (slot _Low_ROP)
   (slot _High_ROP)
   (slot _Very_High_ROP)
   (slot _Very_Low_ROP)
   (slot _Normal_ROP)
)
(deftemplate _Mud_Viscosity
   (slot _High_Mud_Viscosity)
   (slot _Normal_Mud_Viscosity)
   (slot _Increased_Mud_Viscosity)
   (slot _Very_High_Mud_Viscosity)
   (slot _Low_Mud_Viscosity)
)
(deftemplate _Changing_Inclination
   (slot _High_Inclination_Change)
)
(deftemplate _Directional_Parameter
   (slot _Well_Inclination)
)
(deftemplate _Measured_Depth_Scale_Events
   (slot  _Stabilizer_friction)
   (slot _PackOff)
   (slot _MFI_Erosion)
   (slot _Friction)
   (slot _WOB)
   (slot _Well_Inclination)
   (slot _Exposure_Time)
   (slot _rpm)
   (slot _RotationalFriction)
   (slot _Drag)
   (slot _ROP)
)
(deftemplate _Formation1_Sequence
   (slot  _Formation_name)
)
(deftemplate _Formation3_Sequence
   (slot  _Formation_name)
)
(deftemplate _Activity_Before_Case_Occurrence
   (slot  _Case_Series_Start_Time)
   (slot _Possible_Case_Start_Time)
   (slot _Drag_Recording_Quality)
   (slot _HKL-Down_Recording_Quality)
   (slot _Well_Cleaning_Performance)
```

```
)
(deftemplate _Activity_Manual_Int
   (slot  _Tripping_In)
   (slot _RIH)
   (slot  _Reaming_Up)
   (slot  _Backreaming)
   (slot _POOH)
   (slot  _Tripping_Out)
   (slot _Drilling)
   (slot  _Reaming_Down)
   (slot  _Reaming)
   (slot _Well_Cleaning_Performance)
   (slot _Worked_String)
)
(deftemplate _Activity
   (slot  _Tripping_In)
   (slot _Mud_Circulating)
   (slot  _Reaming)
   (slot _Drilling)
   (slot _Connection)
   (slot  _Tripping_Out)
   (slot _Undefined_Status)
)
(deftemplate _Predicted_Failure
   (slot  _Poor_Well_Cleaning)
)
(deftemplate _Well_Characteristic
   (slot  _Geological_Period)
)
(deftemplate _Wellbore_Fm_Characteristic
   (slot  _Sandstone)
   (slot _Siltstone)
   (slot _Claystone)
   (slot _Coal_Bed)
   (slot _Limestone)
   (slot _Textual_Description)
   (slot  _Unknown_Value)
)
(deftemplate _Annular_Flow_Parameter
   (slot _Narrow_Annulus)
)
```

Appendix D

Jess Representation of a Case and Adding to Working Memory

Listing D.1: Jess Representation of the XML Example in Appendix A. It is written on Jess File

```
(assert  (_Administrative_Data
   (_Operator_Company "Statoil")
   (_Well_Identification "Well 34/10-48A")
   (_Oil_Field_Identifier "Gullfaks")
   (_Drilling_Contractor "Smedvig Drilling Company")
   (_Well_Type "Exploration Well")
   (_Well_Section "8.5 Inch Section")
))

(assert  (_Wellbore_Formation_Characteristic
   (_Geological_Period "Upper Jurassic")
   (_Geological_Zonation "Ness Fm")
))

(assert  (_Lithology
   (_Sandstone "True")
   (_Siltstone "True")
   (_Claystone "True")
   (_Limestone "True")
))

(assert  (_Planned_Well_Geometry_Parameter
   (_Planned_Section_Depth null)
   (_Planned_Section_TVD null)
   (_Planned_Previous_Casing_Depth null)
))

(assert  (_Planned_Drilling_Fluid_Parameter
   (_Planned_Mud_Weight null)
   (_Planned_Drilling_Fluid "Unknown Value")
))
```

```
(assert  (_Planned_Drill_String_Parameter
   (_BHA_Length null)
   (_Number_Of_Stabilizers null)
))

(assert  (_Well_Geometry_Parameter
   (_Section_Depth 6621.0)
   (_Section_TVD 2869.0)
   (_Previous_Casing_Depth 5120)
   (_Water_Depth 134.0)
   (_Target_Depth 6621.0)
   (_TVD 2869.0)
))

(assert  (_Fluid_Operational_Parameter
   (_Mud_Weight 1.57)
   (_PV 35.0)
   (_Water_Activity_Of_Mud 0.91)
   (_YP 12.0)
   (_Drilling_Fluid "VersaVert OBM")
   (_FIT_Equivalent_Density 1.77)
   (_LOT_Equivalent_Density 1.85)
   (_Mud_Solids_Content 23.0)
   (_Average_Mud_Weight 1.57)
   (_Average_PV 35.0)
   (_Average_YP 12.0)
   (_Number_Of_String_Wipers 0.0)
))

(assert  (_Drill_String_Parameter
   (_Bit_Run_Number 1.0)
   (_Bit_Size 8.5)
   (_Bit_Type "PDC Bit")
   (_Drill_Pipe_Outer_Diameter 5.0)
   (_BHA_Length 39.9)
   (_Number_Of_Stabilizers 2.0)
   (_Stabilizer_#1_Outer_Diameter 8.25)
   (_Stabilizer_#2_Outer_Diameter 8.25)
   (_Stabilizer_#1_Position 0.26)
   (_Steering_Device "Xceed")
   (_Stabilizer_#2_Position 5.0)
))

(assert  (_Torque
   (_Instantaneous_Torque 0.0)
   (_Torque_Trend -0.03322222222222222)
))

(assert  (_ECD
   (_Instantaneous_ECD 1.57)
   (_ECD_Trend -0.02531111111109796)
```

```
))

(assert  (_Pump_Pressure
   (_Instantaneous_Pump_Pressure 6.99)
   (_Pump_Pressure_Trend -1.1255555555555823)
))

(assert  (_Pump_Flow_Rate
   (_Instantaneous_Pump_Flow_Rate 0.0)
   (_Pump_Flow_Rate_Trend -153.73468888888897)
))

(assert  (_Took_Weight_Events
   (_Took_Weight True)
   (count 1.4583333333333333)
))

(assert  (_Increased_Torque_Events
   (_Increased_Torque True)
   (count 2.7857142857142856)
))

(assert  (_Increased_Drag_Events
   (_Increased_Drag True)
   (count 2.0)
))

(assert  (_Pack_Off_Events
   (_Pack_Off True)
   (count 1.0)
))

(assert  (_Time_Scale_Events
   (_Took_Weight True)
   (_Increased_Torque True)
   (_Took_Weight True)
   (_Increased_Torque True)
   (_Increased_Drag True)
   (_Pack_Off True)
   (_Increased_Torque True)
   (_Took_Weight True)
))

(assert  (_Depth_Scale_Events
   (_Took_Weight True)
   (_Increased_Torque True)
   (_Took_Weight True)
   (_Increased_Torque True)
   (_Took_Weight True)
   (_Pack_Off True)
   (_Increased_Drag True)
))
```

```
(assert  (_Measured_Depth_Scale_Events
   (_Stabilizer_friction 25509.02717999972)
   (_PackOff 0.06203840535373137)
   (_MFI_Erosion 402434.87006651505)
   (_Friction 0.5751181618701464)
   (_WOB 2.5098999999998606)
   (_Well_Inclination 92.09080801853182)
   (_Exposure_Time 3.3648462499999803)
   (_rpm 179.92845000000005)
   (_RotationalFriction 0.04073513468678385)
   (_Drag 0.04249177654036431)
))

(assert  (_Formation1_Sequence
   (_Formation_name "Tarbert Fm")
   (_Formation_name "Ness Fm")
))

(assert  (_Formation3_Sequence
   (_Formation_name "Upper Jurassic")
))

(assert  (_Activity_Manual_Int
   (_Tripping_In "True")
   (_RIH "True")
))

(assert  (_Activity
   (_Tripping_In 75.0)
   (_Mud_Circulating 7.0)
   (_Connection 10.0)
   (_Undefined_Status 8.0)
))

(assert  (_Predicted_Failure
   (_Poor_Well_Cleaning "True")
))

(assert  (_Well_Characteristic
   (_Geological_Period "Tarbert Fm")
))

(assert  (_Wellbore_Fm_Characteristic
   (_Sandstone "True")
   (_Siltstone "True")
   (_Claystone "True")
   (_Coal_Bed "True")
   (_Limestone "True")
))(run)
```

Appendix E

Source Code for the Demo System

Listing E.1: Source Code for CaseParsing Class

```
import java.io.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;
import java.util.NoSuchElementException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.xml.sax.SAXException;

import jess.Rete;



public class caseParsing {



   private  ArrayList entryArrayList;
   private Document caseDoc;

   private FileOutputStream out;
   private PrintStream print;
   private FileOutputStream out3;
   private PrintStream print3;

   private static double depth_occurrence;
   private static String caseName ="";
```

```
private String fileName;

public caseParsing()
{}

 public caseParsing(String fName) throws FileNotFoundException
 {
    entryArrayList = new ArrayList();
    fileName = fName;

 }

public static void main(String[] args) throws FileNotFoundException
{
   caseParsing caseparsing = new caseParsing("src/case.xml");

   caseparsing.getParser();

   caseparsing.parseCase();

// caseParsing.getTemplate();

   caseparsing.writeDataOnFile();

   caseparsing.reasonUsingRBR();


}
public void getParser()
{

   DocumentBuilderFactory caseDocFac = DocumentBuilderFactory.newInstance();
   try {
      DocumentBuilder newCase = caseDocFac.newDocumentBuilder();

      caseDoc = newCase.parse("src/case.xml");


   }catch(ParserConfigurationException pce) {
      pce.printStackTrace();
   }catch(SAXException se) {
      se.printStackTrace();
   }catch(IOException ioe) {
      ioe.printStackTrace();
   }
}

public void parseCase()
{
   /* Now the file is parsed, the next code should read the elements
    * and put them in the appropriate format for the needed purpose.
    *
```

```
     * The following line reads the root node of the document: Case*/
    NamedNodeMap attributes;
    Element rootnode = caseDoc.getDocumentElement();
    try{

    attributes = rootnode.getAttributes();
    caseName = attributes.getNamedItem("name").getNodeValue();
    }
    catch(Exception e)
      {
          System.err.println (e + "Error writing to file caseName");
      }
    try{
    NodeList sections = rootnode.getElementsByTagName("entry");
// System.out.println(sections.getLength());

    if(sections.getLength()> 0 && sections != null)
    {

       for(int i=0; i<sections.getLength();i++)
       {
          attributes = sections.item(i).getAttributes();
          String param = attributes.getNamedItem("parameter").getNodeValue();
          if(param.equals("Depth Of Occurrence"))
          {
             NodeList children = sections.item(i).getChildNodes();
             Element value = (Element)children.item(1);
             depth_occurrence = Double.parseDouble(value.getFirstChild().getNodeValue());
             break;
          }
       }
       fact ee = new fact();
       ee.getdepth(depth_occurrence);
       //System.out.println(depth_occurrence);

       for(int i=0; i<sections.getLength();i++)
       {
          fact e = getEntry(sections.item(i));
          entryArrayList.add(e);
       }
    }
    }
    catch(Exception e)
      {
          System.err.println (e + "Error writing to file ****");
      }

}


private void writeDataOnFile()
{
```

```
      try
      {
         out = new FileOutputStream("src/newCase.clp");
         print = new PrintStream(out);

      // print.append("( assert  " );

         Iterator it = entryArrayList.iterator();
         while(it.hasNext()) {

            print.append(it.next().toString());
         }

         print.append("))");
         print.append("(run)");
         //**************defineTemplate(")");
         print.close();

      }
      catch(Exception e)
        {
            System.err.println (" Error writing to file case");
        }


   }


private fact getEntry(Node elmt)
{
   NamedNodeMap attributes;

   attributes = elmt.getAttributes();
   String param = parameterConvert(attributes.getNamedItem("parameter").
      getNodeValue());
   String paramvalue = " ";
   Double start=0.0;
   Double end=0.0;
   //System.out.println(param);
   NodeList children = elmt.getChildNodes();

   String parentNode;
   if(elmt.getParentNode().getNodeName().equals("sequence-section"))
   {
      Node sequenceParent = elmt.getParentNode();
      parentNode = parameterConvert(sequenceParent.getParentNode().
         getAttributes().getNamedItem("name").getNodeValue());

      NodeList seqChildren = sequenceParent.getChildNodes();
      NamedNodeMap attrs;

      for(int i=0; i < seqChildren.getLength(); i++ )
```

```
      {
         attrs = seqChildren.item(i).getAttributes();
         if(seqChildren.item(i).getNodeName().equals("position") && attrs.
            getNamedItem("scale").getNodeValue().equals("Depth"))
         {
            start= Double.parseDouble(attrs.getNamedItem("start").getNodeValue());
            end= Double.parseDouble(attrs.getNamedItem("end").getNodeValue());

            break;
         }
      }

   }
   else
      parentNode = parameterConvert(elmt.getParentNode().getAttributes().
         getNamedItem("name").getNodeValue());


   Element value = (Element)children.item(1);

   if(children.item(1).getNodeName().equals("dataValue"))
   {
      //Element value = (Element)children.item(1);
       paramvalue = value.getFirstChild().getNodeValue() ;
   }
   else if (children.item(1).getNodeName().equals("symbolValue"))
   {
      //Element value = (Element)children.item(1);
       paramvalue = "\"" + value.getFirstChild().getNodeValue() + "\"";
   }


   fact newEntry =new fact(param, paramvalue, parentNode, start, end);

   return newEntry;

}

private String parameterConvert(String name)
{

   String name2 = "";
   StringTokenizer st = new StringTokenizer(name, " ");
   String name3 = "";
   while (st.hasMoreTokens()) {
      name3 = st.nextToken();
      if(name3.equalsIgnoreCase("(Manual"))
         name3 = "Manual";

      if(name3.equalsIgnoreCase("Int)"))
         name3 = "Int";
      //if(st.nextToken().contentEquals("(Manual"))
```

```
         //name2= name2.concat("_Manual");
      //else if(st.nextToken().contentEquals("Int"))
         //name2= name2.concat("_Int");
      //else
            name2= name2.concat("_" + name3);

      };

   return name2;

}
/**
 *
 * public static void defineTemplate(String value)
{
   try
   {
      print2.append(value);
      //print2.close();

   }
   catch(Exception e)
    {
        System.err.println ("Error writing to file Template");
    }
}
 */
public void inferredParams(String a, String b, String c)
{
   try
   {

      removeLineFromFile("src/case.xml", "</case>");

      out3 = new FileOutputStream("src/case.xml", true);
      print3 = new PrintStream(out3);


      print3.append("<section name=\"" + a + "\"> \n <entry parameter=\"" +
         b + "\" source=\"System\">\n <dataValue>" + c + "</dataValue> \n
         </entry>\n </section> \n </case> \n");
   }
   catch(Exception e)
   {
       System.err.println (" Error writing to file case");
   }
}
@SuppressWarnings("deprecation")
private void reasonUsingRBR()
{
   Rete engine = new Rete();
   try {
```

```
   engine.executeCommand("(batch \"src/Template.clp\")");
   engine.executeCommand("(batch \"src/newCase.clp\")");
   }
   catch(Exception e){
   e.printStackTrace();
   }
}


private void removeLineFromFile(String file, String lineToRemove) {

    try {

      File inFile = new File(file);

      if (!inFile.isFile()) {
        System.out.println("File Name is not an existing file");
        return;
      }

      //Construct the new file that will later be renamed to the original filename.
      File tempFile = new File(inFile.getAbsolutePath() + ".tmp");

      BufferedReader br = new BufferedReader(new FileReader(file));
      PrintWriter pw = new PrintWriter(new FileWriter(tempFile));

      String line = null;

      //Read from the original file and write to the new
      //unless content matches data to be removed.
      while ((line = br.readLine()) != null) {

        if (!line.trim().equals(lineToRemove)) {

          pw.println(line);
          pw.flush();
        }
      }
      pw.close();
      br.close();

      //Delete the original file
      if (!inFile.delete()) {
        System.out.println("Could not delete file");
        return;
      }

      //Rename the new file to the filename the original file had.
      if (!tempFile.renameTo(inFile))
        System.out.println("Could not rename file");

    }
```

```
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
  }

}
```

Listing E.2: Source Code for Entry Class

```
package volve.rules;

public class Entry {

   private String parameter;
   private static String previousParam;
   private String value;
   private static String previousValue;
   private  String parentName;
   private static String parent;
   private Double start;
   private Double end;

   private static Double depth;
   private static Double count = 0.0;



   Entry()
   {}
   Entry(String par, String parvalue, String parent, Double startt, Double endd)
   {

      this.parameter = par;
      this.value = parvalue;
      this.parentName = parent;
      this.start = startt;
      this.end = endd;
   }

   void getdepth(Double depthOccurrence)
   {
      depth = depthOccurrence;
      parent = "";
   }

   public String toString()
    {
      StringBuffer sb = new StringBuffer();
      //**********************StringBuffer sb2 = new StringBuffer();
```

```
      /*sb.append("Entry Details - \n");
      sb.append("Parameter Name:" + parameter);
      sb.append("\n ");
      sb.append("Parameter Value:" + value);
      sb.append("\n  \n");*/
      if(parent.equals(parentName))
      {
         if(!parent.equals("_Case_Occurrence_Description") && !parent.equals
            ("_Response_Activity_Description") && !parent.equals("_Poor_Well_
            Cleaning_Case_Class") && !parent.equals("_Risk_Assessment") &&
            !parent.equals("_Risk_Calculation") && !parent.equals("_Final_
            Section_Consequence") && !parent.equals("_Lesson_Learned") &&
            !parent.equals("_Activity_Before_Case_Occurrence"))
         {
            if(previousParam.equals(parameter) && previousValue.equals(value))
            {
               if(parent.equals("_Pack_Off_Events") || parent.equals("_
               Increased_Drag_Events") || parent.equals("_Increased_Torque
               _Events") || parent.equals("_Took_Weight_Events") ||
               parent.equals("_Tight_Spot_Events"))
               {
                  int x = (int)((Math.abs(depth - end)) + 100)/ 100;
                  count = count + ((100.0 / (x * 100.0)));
               }
               return "";
            }
            else
            {
               if(parent.equals("_Measured_Depth_Scale_Events"))
               {
                  if(start <= depth && depth <= end)
                  {
                     previousParam = parameter;
                     previousValue = value;
                     sb.append("\t(" + parameter + " " + value + ")\n");
                     //System.out.println(start + "to " + end);
                  }
                  else
                     return "";
               }
               /*if(parent.equals("_Measured_Depth_Scale_Events") &&
                  previousParam.equals("_Drag") && parameter.equals("
                  _Stabilizer_friction"))
               {
                  sb.append(")) \n");
                  sb.append( "\n(assert \t(" + parentName + "\n \t("
                     + parameter + " " + value + ")\n");
               }*/
               else
               {
                  previousParam = parameter;
                  previousValue = value;
```

```
                sb.append("\t(" + parameter + " " + value + ")\n");
             }


          }
       }
       else
          return "";
    }
    else
    {

          if(!parent.equals(""))
          {
             if(parent.equals("_Pack_Off_Events") || parent.equals("_
                Increased_Drag_Events") || parent.equals("_
                Increased_Torque_Events") || parent.equals("_Took_
                Weight_Events") || parent.equals("_Tight_Spot_Events"))
             {
                sb.append("\t(count " + count + ")\n");
                count = 0.0;
             }
             sb.append(")) \n");
             //******sb2.append(") \n");
          }
          parent = parentName;
          previousParam = parameter;
          previousValue = value;
       if(!parent.equals("_Case_Occurrence_Description") && !parent.equals("_
          Response_Activity_Description") && !parent.equals("_Poor_Well_
          Cleaning_Case_Class") && !parent.equals("_Risk_Assessment") &&
          !parent.equals("_Risk_Calculation") && !parent.equals("_Final_Section
          _Consequence") && !parent.equals("_Lesson_Learned") &&               !parent.equals
       {
          if(parent.equals("_Pack_Off_Events") || parent.equals("_Increased_
             Drag_Events") || parent.equals("_Increased_Torque_Events")
             || parent.equals("_Took_Weight_Events") || parent.equals("_
             Tight_Spot_Events"))
          {
             int x = (int)((Math.abs(depth - end)) + 100)/ 100;
             count = count + (100.0 / (x * 100.0));
             sb.append( "\n(assert \t(" + parentName + "\n \t(" +
                parameter + " " + value + ")\n");
          }
          else if(parent.equals("_Measured_Depth_Scale_Events"))
          {
             sb.append( "\n(assert \t(" + parentName + "\n ");
             if(start <= depth && depth <= end)
                sb.append( "\t(" + parameter + " " + value + ")\n");
             else
                return sb.toString();
          }
```

```
            else
               sb.append( "\n(assert \t(" + parentName + "\n \t(" +
                  parameter + " " + value + ")\n");

         }
         else
            return "";
      }

      return sb.toString();
   }
}
```

Appendix F

Glossary

AI
Artificial Intelligence

API
Application Programming Interface

BR
Business Requirement

CBR
Case Based Reasoning

CBS
Case Based System

DOM
Document Object Model

FR
Functional Requirement

IEEE
Institute of Electrical and Electronics Engineers

JESS
Java Expert System Shell

KBS
Knowledge-Based System

NFR
Non-functional Requirement

NTNU
Norwegian University of science and technology

RBRCBRI-TC
Rule Based Reasoning and Case Based Reasoning Integration Test Case

RBRCBRI-TP
Rule Based Reasoning and Case Based Reasoning Integration Test Plan

RBR
Rule Based Reasoning

RBS
Rule Based System

UC
Use Case

UI
User Interface

WOB
Weight On Bit

XML
EXtensible Markup Language

References

[1] A.Aamodt and E. Plaza, "Foundational Issues, Methodological Variations and System Approaches," *Artificial Intelligence Communications,7(1)*, pp. 39-59, March 1994.

[2] Agnar Aamodt,"Knowledge-Intensive Case-Based Reasoning in CREEK", *Norwegian niversity of Science and Technology.*

[3] Agnar Aamodt,"Explanation-Driven Case-Based Reasoning", *Norwegian niversity of Science and Technology.*

[4] Belen Diaz-Agudo and Pedro A. Gonzalez-Calero,"An Architecture for Knowledge Intensive CBR Systems", *Universidad Computense de Madrid, Spain.*

[5] R. Lopez,D. Mcsherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. Maher, M. Cox, K. Forbus, M. Keane, A. Aamodt, and I. Watson, "Retrieval , reuse, revision, and retention in case-based reasoning", *The Knowledge Engineering Review,Vol 20:3, 215-240*, Cambridge University Press 2006.

[6] Mal Rey Lee, "An exeption Handling of Rule-Based Reasoning Using Case-Based Reasoning", *Journal of Intelligent and Robotic Systems 35: 327-338*, Kluwer Academic Publishers 2002.

[7] R. Bellazzi, S. Montani, L. Portinale, and A. Riva, "Integrating Rule-Based and Case-based Decision Making in Diabetic Patient Management", *ICCBR-99, LNAI 1650, pp 386-400*,SpringerVerlag Berlin Heidelberg 1999.

[8] R. Golding and S. Rosenbloom "Improving Accuracy by combining Rule-based and Case-based Reasoning", *Artificial Intelligence 87, 215-254*,Mitsubishi Electric Research Laboratories 1995.

[9] Kamalendu Pal and John A Campbell,"An Application of Rule-Based and Case-Based Reasoning within a Single Legal Knowledge-Based System",*The DATA BASE for Advances in Information Systems Vol 28, No. 4*,1997.

[10] M.Fathi -Torbaghan and D. Meyer "ICARUS: Integrating rule-based and case-based reasoning on the base of unsharp symptoms",IEEE 1995

[11] Software Engineering Technical Committee of the IEEE Computer Society "IEEE Standard for Software Test Documentation",*IEEE Std 829-1998 (Revision of IEEE Std 829-1983)* The Institute of Electrical and Electronics Engineers, Inc. 1998.

[12] N. Phuong, N.Prasad, D. Hung, and J. Drake "Approach to Combining Case Based Reasoning with Rule Based Reasoning for Lung Disease Diagnosis", IEEE 2001.

[13] D. Shimin, S. Huizhang, L. Hong "Research on Case-Based Reasoning Combined with Rule-Based Reasoning for Emergency", IEEE 2007.

[14] Alex J. Champandard , AI depot , *Artificial Intelligence* , October 23, 2008, http://ai-depot.com/Intro.html.

[15] Branting, K., and Porter, B. , "Rules and precedents as complementary warrants", in *Proc. 1991 The Ninth National Conference on Artificial Intelligence*, pp. 3-9.

[16] Cindy Marling , Edwina Rissland , Agnar Aamodt, "Integrations with case-based reasoning", in *The Knowledge Engineering Review, v.20 n.3*,p.241-245, September 2005.

[17] C.R. Marling, G.J. Petot, and L.S. Sterling "Integrating Case-Based and Rule-Based Reasoning to Meet Multiple Design Constraints", in *Computational Intelligence*, vol. 15, pp. 308-332, 1999.

[18] E.L. Rissland and D.B. Skalak, "Combining Case-Based and Rule-Based Reasoning: A Heuristic Approach", in *Proc. 1989 International Joint Conference on Artificial Intelligence*,pp. 524 -530.

[19] George F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*,Addison-Wesley, Pearson Education Limited 2002.

[20] I. Bichindaritz, E. Kansu, and K.M. Sullivan, "Case-Based Reasoning in Care-partner: Gathering Evidence for Evidence-Based Medical Practice", in *Proc. 1998 European Conference on Case-Based Reasoning*, pp. 334-345.

[21] J. Prentzas and I. Hatzilygeroudis, "Integrating Hybrid Rule-Based with Case-Based Reasoning", in *S. Craw and A. Preece (Eds), Advances in Case-Based Reasoning, Procs 2002 European Conference on Case-Based Reasoning*, LNAI 2416, Springer-Verlag, pp. 336-349.

[22] J. Prentzas and I. Hatzilygeroudis,"Integrations of Rule-Based and Case-Based Reasoning",in *Proc. International Conference on Computer*, 2003.

[23] J. Prentzas and I. Hatzilygeroudis,Categorizing Approaches Combining Rule-Based and Case-Based Reasoning, *Expert Systems 24(2)* 2007, 97-122.

[24] J. Kolodner, *Case-Based Reasoning*,San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[25] David B. Leake, *Case-Based Reasoning Experiences, Lessons, and Future Directions*,Menlo Park, CA: American Association for Artificial Intelligence, 1996.

[26] Home page of Verdande Technology As, February 12, 2009, http://www.verdandetechnology.com/.

[27] Lars Olrik, *Harvesting Knowledge to improve drilling performance in real time*, Verdande Technolgoy As.

[28] DrillEdge brochure, Verdande Technology As,*DrillEdge* February 12, 2009, http://www.verdandetechnology.com/images/verdande/drilledge_brochure_web.pdf.

[29] UML Training Courses from CRaG Systems, *Specifying Functional Requirements With Use Cases* May 12, 2009, http://www.cragsystems.co.uk/SFRWUC/index.htm.

[30] Jess, *Jess, The Rule Engine for the Java Platform* May 21, 2009, http://www.jessrules.com/jess/docs/70/.

[31] Making Your Own Rules, *Jess, The Rule Engine for the Java Platform* May 27, 2009, http://www.jessrules.com/jess/docs/70/rules.html.

[32] Robert V. Stumpf and Lavette C. Teague, *Object-oriented systems analysis and design with UML*, Upper Saddle River, NJ : Pearson Prentice Hall, 2005, pp 1-38.

[33] Code Better, *Software Development Life Cycle Models*, February 24, 2009, http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/13/129114.aspx.

[34] FFIEC IT Handbook InfoBase, *Systems Development Life Cycle*, February 21, 2009, http://www.ffiec.gov/ffiecinfobase/booklets/d_a/08.html.

[35] IT Knowledge Exchange, *Spiral Model*, March 07, 2009, http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci755347,00.html.

[36] Human Base India Inc., An Information Technology Company, *Software Development Life Cycle Models*, March 10, 2009, http://www.humanbaseindia.com/training/SDLC.htm.

[37] GeekInterview, *SDLC - Prototype Model*, March 03, 2009, http://www.learn.geekinterview.com/it/sdlc/prototype-model.html.

[38] Stylusinc Increasing your NET value, *Software Development Life Cycle (SDLC)*, March 07, 2009, http://www.stylusinc.com/Common/Concerns/SoftwareDevtPhilosophy.php.

[39] Business eSolutions, *Project Lifecycle Models: How They Differ and When to Use Them*, March 16, 2009, http://www.business-esolutions.com/islm.htm.

[40] Ann Gordon, Bright Hub, *What is a Work Breakdown Structure?*, March 16, 2009, http://www.brighthub.com/office/project-management/articles/2645.aspx.

[41] Virgil Andronache, University of Notre Dame: Computer Science and Engineering, *Behavior-Based Robotics*, March 21, 2009, http://www.cse.nd.edu/courses/cse498f/www/ln1.html.

[42] Robert S. Engelmore and Edward Feigenbaum, Japanese Technology Evaluation Center, *Knowledge-Based Systems in Japan*, March 21, 2009, http://www.wtec.org/loyola/kb/toc.htm.

[43] NetMBA Business Knowledge Center, *Work Breakdown Structure*, March 16, 2009, http://www.netmba.com/operations/project/wbs/.

[44] CIO defination, *Decision support system*, March 27, 2009, http://searchcio.techtarget.com/sDefinition/0,,sid182_gci213888,00.html.

[45] CIO defination, *What is a Decision Support System?*, March 27, 2009, http://www.tech-faq.com/decision-support-system.shtml.

[46] Wikipedia The Free Encyclopedia, *Software development methodology*, February 21, 2009, http://en.wikipedia.org/wiki/Software_development_methodology.

[47] Wikipedia The Free Encyclopedia, *Software development process* , February 21, 2009, http://en.wikipedia.org/wiki/Software_development_process.

[48] Wikipedia The Free Encyclopedia, *Waterfall model* , March 03, 2009, http://en.wikipedia.org/wiki/Waterfall_model.

[49] Wikipedia The Free Encyclopedia, *Spiral model* , March 05, 2009, http://en.wikipedia.org/wiki/Spiral_model.

[50] Wikipedia The Free Encyclopedia, *Knowledge-based systems* , March 22, 2009, http://en.wikipedia.org/wiki/Knowledge-based_systems.

[51] Wikipedia The Free Encyclopedia, *Decision support system* , March 27, 2009, http://en.wikipedia.org/wiki/Decision_support_systems.

[52] Wikipedia The Free Encyclopedia, *Rete Algorithm* , April 8, 2009, http://en.wikipedia.org/wiki/Production_rule_system.