ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Marcel Strittmatter

Network Troubleshooting Expert System

Diploma Thesis DA-2003.21
Winter Term 2002/2003

Tutors:
Michael Hausding (Open Systems AG),
Stefan Lampart (Open Systems AG),
Marc Rennhard

Supervisor:
Prof. Dr. Bernhard Plattner

3.3.2003

TIK Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Abstract

The claim on availability, functionality, and security of todays networks is continuously rising. Many companies depend on the availability of their networks. On the other hand the process of managing these growing networks gets always more difficult. Efficient and fast methods are needed to handle network problems.

This project tried to find ways how the fault management in IP networks can be handled with the help of artificial intelligence. Different expert system approaches have been surveyed on their usability for handling network problems. One expert system approach, namely case-based reasoning, was chosen to build a prototype of a network troubleshooting expert system. In order not to re-implement an already existing approach, the most known network troubleshooting and management applications have been studied. After implementing the prototype, some tests were made to discover if the used approach really helps to solve network problems. In these tests it emerged that the concept of case-based reasoning seems to be a useful approach to build a network troubleshooting expert system. But for the usage in a real environment, more and much longer tests are required.

Zusammenfassung

In der heutigen Zeit steigen die Anforderungen an die Verfügbarkeit, Funktionalität und Sicherheit von modernen Netzwerken stetig an. Viele Firmen hängen stark von der Verfügbarkeit ihrer (und anderer) Netzwerke ab. Auf der anderen Seite gestaltet sich das Verwalten dieser Netze mit wachsender Komplexität immer schwieriger. Effiziente und schnelle Methoden sind gefragt um bei Netzwerkproblemen schnell eine Lösung zu finden.

In dieser Arbeit wurde versucht Wege zu finden, wie Fehler in IP Netzwerken mit Hilfe von künstlicher Intelligenz verarbeitet und gelöst werden können. Verschiedene Ansätze von Expertensystemen wurden verglichen und die Tauglichkeit für die Behandlung von Netzwerk Problemen überprüft. Ein Ansatz, nämlich das Fallbasierte Schliessen, wurde ausgewählt, um ein Konzept und einen Prototypen eines "Network Troubleshooting Expert Systems" zu entwickeln. Diverse existierende Systeme wurden evaluiert, um nicht ein bestehendes System neu zu implementieren. Nach der Implementation des Prototypen wurden einige Tests durchgeführt, um die Tauglichkeit des benutzten Ansatzes für die Behandlung von Netzwerkproblemen zu untersuchen. Aus diesen Tests ging hervor, dass das fallbasierte Schliessen durchaus ein vernünftiger Ansatz für ein "Network Troubleshooting Expert System" sein kann. Allerdings müsste die Tauglichkeit in einer reellen Umgebung über noch längere Zeit getestet werden.

Many people helped me during this diploma thesis. I would especially like to thank Michael Hausding, Stefan Lampart, and Marc Rennhard for the supervision and the proof reading. Also I like to thank Nick Hocking who did the english corrections of the report and Svetlana Domnitcheva for her time to talk about expert systems.
The following people did help me also during the time of the diploma thesis: Mirjam Saltuari, Margrith and Pierre Strittmatter.

Marcel Strittmatter, marcel.strittmatter@dinoware.com
Zürich, 03.03.2003

Contents

List of Figures

Chapter 1

Introduction

Computer Networks are today a fundamental part of almost every company. It is important to have methods which allow an efficient management of these networks to ensure that the networks can operate continuously. To achieve this, a network operator needs several tools to manage such a network. One important task of a network operator is to react quickly to faults which may occur at anytime and to solve them as fast as possible. In order to do this it is essential that the operator knows how to solve a specific problem. But there are a huge number of different problems and problem solving procedures. For a company it is necessary to have the knowledge about such solving methods available even when the staff is changing over time. The question is how to store this knowledge and how to make this knowledge available to someone who needs to solve a problem which already has been solved one time.

Expert systems uses a human expert's knowledge in a way that such an expert system application can apply this knowledge where needed, run sometimes unattended by using the human expert's knowledge and solve problems or assist an other human expert in solving the same problem at another time. Different approaches to expert systems have been studied over a long time. This thesis gives a short introduction into a bunch of such expert system approaches.

Focus The term "Network" can be used for different kinds of networks such as telecommunication networks and computer networks. Because it is not possible to cover all different kinds of networks in one thesis, the focus of this thesis is on todays TCP/IP networks. Many network management systems can handle troubles in a local network, so this thesis looks particularly at issues in TCP/IP networks that are not entirely under the same control as the network troubleshooting system.

Goal The goal of this thesis is to find some ways as to, how knowledge about solving network problems can be conserved for future use, and how this knowledge can later be found. Any network administrator should be able to solve a newly occurred problem with the help of this expert system in less time than without the help of this application. The overall concept, design, and implementation of such a network troubleshooting expert system are discussed and tested against feasibility and usability.

Outline In chapter 2 the topic of expert systems in general is introduced as well as the different approaches of knowledge representation. Chapter 3 takes account of a general view of network elements, possible faults, and the difference between fault, failure and error. After a short overview of existing network management and troubleshooting systems in chapter 4, the concept, design, and implementation for a network troubleshooting expert system is presented (chapters 5 and 6). Chapter 7 contains the conclusions and the results of the work as well as some ideas for further work.

Chapter 2

Expert Systems

This chapter is an introduction to different approaches of building expert systems. The expert system field is very large and this introduction cannot take account of every expert system approach. The approaches for building expert systems presented in this chapter are the most familiar ones and the ones that seems to fit best for the task of a network troubleshooting expert system. If you like to read more about expert systems in general you probably want to read [2], [1], or [5]. Most of the information in this chapter was gathered from these books.

2.1 Introduction into Expert Systems

An expert system is an application which contains the knowledge of a human expert about solving problems in a bounded environment. The field of a such system area is really large. There are applications in almost every discipline for instance in manufacturing, medicine, business procedures, or engineering. The most important goals of an expert system are:

- to conserve the knowledge of human experts,
- to provide help to laymen and experts in solving problems,

- or to make information available that is difficult to recall.

Some of the expert systems may have capabilities to learn from experience or to adjust the systems behavior to new situations, others may describe and make use of models of complex procedures. Not every expert system approach is suited to a specific problem. It is important to clarify first what the intention of the expert system should be, and then choose the most appropriate approach to implement the system.

An expert system is not a replacement of human experts with a computer system. An expert system doesn't have the demand to be 100% precise, so it is still essential to have humans who work with the expert system and control its functionality. The human expert remains important in the solving process, but the expert system may help the human expert in order to find a solution. An expert system is best used for special problems in a narrow field. They can mostly not be used in a broad field. History has shown that expert systems failed to work correctly if the operational field for the system was to large.[2]

There are different definitions of expert systems. Some are based on the function, e.g. Control, Planning, Diagnosis, or Interpretation, some on the structure, e.g. Rule-Based, Case-Based, or Probabilistic, of the expert system. Many early definitions assumed a rule-based reasoning, which is today extended with object-oriented, procedural, model-based, case-based reasoning and other reasoning methods. The most important approaches for this thesis are described in the sections 2.5.1 to 2.5.6.

An expert system is some kind of a storage for human memory. It is important for a company to ensure that the knowledge of the experts working for the company is still accessible, even when the person is not working for this company anymore. Because of the relatively rapid changes of people in a company, an expert system can be very useful to conserve some of the company's knowledge.

The expert system has several advantages over a human expert: Time availability, perishability, constant performance, usually faster speed, and low running costs. But in many cases human experts are very important for the successful operation of the whole

system. It is – fortunately – not always possible to replace the human experts with an expert system. Assisting a human expert is the most commonly found application of expert systems. Some principal reasons why expert systems should be developed to aid a human expert are:

- Aiding expert in some routine task to improve productivity.
- Aiding expert in some difficult task to effectively manage the complexities.
- Making information that is difficult to recall available to the expert.

There are different problem-solving paradigms where expert systems can be deployed. The following paradigms are the most important ones for this thesis:

- **Diagnosis** systems infer system malfunctions of faults from observable information.
- **Interpretation** systems produce an understanding of a situation from available information sources like data from sensors, instruments, and test results.
- **Monitoring** systems compare observable information on the behavior of a system with system states that are considered crucial to its operation.
- **Prescription** systems recommend solutions to a given system malfunction.

The term knowledge-based system is often used as a synonym for an expert system. In principle a knowledge-based system is a software where the knowledge about the solution of a problem is separated from the logic.

The next section explains the components of the expert system more detailed.

2.2 Components of an Expert System

An expert system consists of three major modules, the knowledge base, the inference engine, and the working memory. This partitioning is deduced from how the human

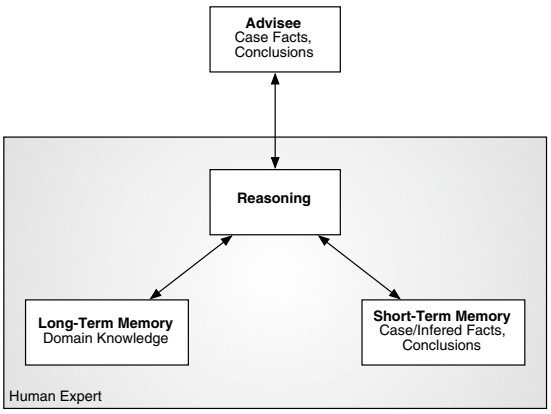

Figure 2.1: Human Problem Solving

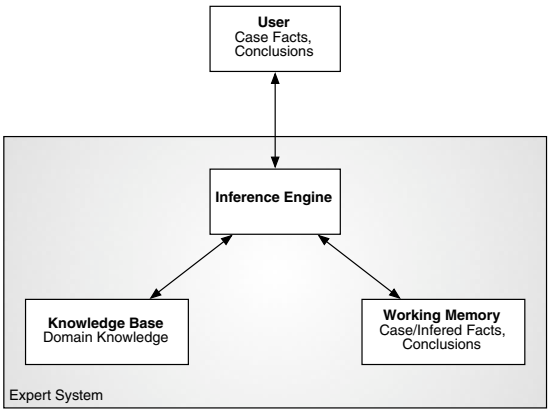

Figure 2.2: Expert System Problem Solving

expert problem solving works. Figure 2.1[1] shows the main concept of human expert problem solving and in figure 2.2[2] this is adapted to expert system problem solving. The long-term memory – like the knowledge base – contains e.g. the knowledge of a auto mechanic, collected in years of experience whereas the short-term memory – or the working memory – only contains information about an actual problem, e.g. "the car of customer x won't start". The mechanic then makes several steps (reasoning) to combine the knowledge in the long-term memory with the short-time memory. That may for

[1]Figure taken from [2]
[2]Figure taken from [2]

example lead him to the conclusion that "the problem may be a broken fuse".

2.2.1 Knowledge Base

```
Rule 1:
 IF    the sun is shining
 AND   today is sunday
 THEN  it's a nice day to do something outside

Rule 2:
 IF    it's a nice day to do something outside
 AND   season is winter
 AND   there is snow
 THEN  go skiing

Rule 3:
 IF    it's a nice day to do something outside
 AND   season is summer
 AND   temperature is high
 THEN  go to an open-air swimming pool
```

Figure 2.3: Sample rule-based knowledge representation

The highly specialized knowledge of the problem area is located in the knowledge base. This module contains the problem facts, rules, concepts, and relationships. The first step to build the knowledge base is to gather the knowledge from human experts, which should be stored in the knowledge base. After this step the knowledge must be coded in a form which is useful for automatic processing, which is subject of the knowledge representation. There are several techniques as to how to store the knowledge. As an example we could look at a rule-based representation. A rule is a if/then structure that logically relates information contained in the 'if' part of the rule to information in the 'then' part. Figure 2.3 shows a sample rule-based knowledge representation.

If Rule 1 matches the current conditions, it is executed. If "*the sun is shining*" is added to the working memory (described in the next subsection) nothing happens for the time being. But when "*today is sunday*" is also added to the working memory, rule 1 matches, and the 'then' part is added to the working memory. Every time the working memory gets updated, there are potentially new rules which may match. This process stops when

there is no change in the working memory anymore.

2.2.2 Working Memory

The working memory contains the facts about a problem that are collected during one consultation of the expert system. When a new problem has to be solved, the user enters information about the problem into the working memory. The expert system then uses this information together with the facts in the knowledge base to infer new facts. These new facts are also stored in the working memory. The content of the working memory can also consist of facts that have been collected from external storage like databases, spreadsheets, or sensors, beside the information that is taken from the knowledge base.

2.2.3 Inference Engine

The reasoning of an expert is modeled in the knowledge processor, usually named the inference engine. The engine uses the knowledge base as the source of information for the reasoning. Together with the other available information[3] it draws conclusions and/or gives recommendations as to how a specific problem can be solved.

[3] e.g. additional information from databases

2.3 Characteristics of Expert Systems

The most important characteristic of any expert system is that the knowledge is separated from control logic. This distinguishes an expert systems from a conventional program, where it is often not possible to add knowledge without changing the source code of the program at the same time.[4] The tasks of modifying and maintaining the system are easier with this separation. Another important feature of an expert system is that it embodies the expertise of a human expert. The knowledge of such an expert is used as well as his problem-solving skills. Because expert systems focuses on knowledge of a clearly defined domain they poorly perform outside their area of expertise.[2] This means that it is better to divide the problem space into smaller areas if the whole area of the subject is too big. While most conventional computer programs cannot handle inaccurate information, expert systems can. They allow inexact knowledge – e.g. "add *some* memory to a machine which is running *slow*" – and inexact information – e.g. "the computer is running *slow*" or "i *don't have* results of the tests".

But an important fact to remember is, that expert systems cannot handle unsolved problems. If a problem is not solvable, it doesn't make sense to try to build an expert system that should solve this problem. Expert systems can only solve problems where there is a human expert who is able to describe in detail the problem solution methods. Also problems should not be to easy or to difficult to solve. They should be of reasonable complexity and it should be possible to solve them in reasonable time.

Like his human counterpart who sometimes makes mistakes, the expert system can not guarantee to always suggest the optimal solution for a problem.

The ability of expert systems to explain their reasoning is another important feature of such a system. The two main explanation facilities are *how* and *why*.

At each time in an interaction between the system and the user it should be possible for the user to ask the system *how* it has reached the actual conclusion. This is important because an expert system does not have a well-defined structure like a conventional

[4] Note that there are also procedural and object-oriented expert system where this separation from knowledge and control logic is not made.

program that makes it easy to understand what the system is doing. Lets look at an example[5]:

```
Expert System: The battery is bad.
Person:        How?
Expert System: Since your car won't start,
               I assumed there was a problem with the electrical system.
               Once I found the battery voltage was below 10 volts,
               I knew that the battery was bad.
```

An expert system can also explain *why* it is asking a specific question. The explanation of the system makes the user feel more comfortable because he can reproduce why the system is asking this question. Lets look at an example[6]:

```
Expert System: Will the car not start?
Person:        Why?
Expert System: If I know that the car won't start,
               then I usually assume the problem is in the electrical system.
```

2.4 Building Expert Systems

This section describes the most important steps in building an expert system. The development of an expert system is a highly iterative process. Most of the time it is not possible to design first the whole system and then just implement the design. Instead it is needed to build a first prototype, then use this prototype to get more knowledge from the human experts and re-implement the prototype. This process is repeated until there is a final system that is able to do what the users are expecting it to do.

[5] taken from [2]
[6] also taken from [2]

2.4.1 Knowledge Engineering

One major part is the *Knowledge Engineering*. Usually there are human experts who know much about their specific domain, and application developers who have a lot of knowledge in programming but don't know much about the domain where the expert system is used in the future. In the process of *Knowledge Engineering* both groups of people are involved and try to find an efficient way as to how to encode the human experts knowledge in the expert system.

2.4.1.1 Assessment

During the assessment phase, studies are made to determine the feasibility and justification of the candidate problem. After this process the overall goals of the project have to be defined. These goals together with the information gathered by the process of *Knowledge Engineering* are then used to identify the sources of needed knowledge.

2.4.1.2 Knowledge Acquisition

Knowledge Acquisition is the process of acquiring, organizing, and studying the knowledge from human experts. In the early stage of expert system development the objective of this step is to uncover key concepts and general problem-solving methods used by the expert. In a later stage the results of the testing are used to explore for more detailed information. *Knowledge Acquisition* hast long been recognized as the bottleneck in expert system development.[2]

2.4.1.3 Design

After the knowledge acquisition, the design phase is used to define the overall structure and organization of the system's knowledge. Methods are defined for processing the knowledge and the appropriate software tool is chosen. After this step an initial prototype system is built, which serves as the focal point for further interviews with the human

experts. As mentioned above this task is repeated several times until a final design is reached.

2.4.1.4 Testing

The *Testing* phase is not a separate task. It is rather a continuous process throughout the whole project. After every step in building the expert system, it is tested and new knowledge is added to it. These tests should not only be done by the programmers, but also by the end-users because it is very important that the end-users are able to use the system and that the system is well adapted to the user's needs.

2.4.1.5 Documentation

The *Documentation* addresses the need to compile all of the project's information into a document. The Documentation must also support the knowledge engineer during the development of the system. It should contain a knowledge dictionary that provides a well organized representation of the system's knowledge and the included problem solving procedures.

2.4.1.6 Maintenance

Because of the highly iterative process in developing an expert system, it is also important to periodically maintain the system. The system's knowledge may need to be refined or updated to adapt the system to the actual circumstances. If the knowledge is separated from the control, this process is more flexible and easier as if the whole knowledge is included in the logic.

2.4.2 Knowledge Representation

Knowledge is represented in some symbolic form that can be manipulated by the expert system and by the users of the expert system. There exist many different forms of knowl-

edge representation. The knowledge engineer must choose the knowledge representation technique best suited for the given application. Some of the most important types of knowledge are presented in the following subsection.

2.4.2.1 Types of Knowledge

Knowledge can be divide into the following types:

- **Procedural knowledge** describes how a problem is solved, e.g. rules, strategies, and procedures.
- **Declarative knowledge** describes what is known about a problem, e.g. concepts and objects.
- **Meta-knowledge** describes knowledge about the knowledge.
- **Heuristic knowledge** describes a rule-of-thumb.
- **Structural knowledge** describes knowledge structures, e.g. rule sets and relationships of concept.

It is important to use an appropriate type of knowledge for a specific expert system application, because only a well adjusted structure is able to support effective problem solving. The next subsection describes different knowledge representation techniques.

2.4.2.2 Knowledge Representation Techniques

The most used knowledge representation techniques are object-attribute-value triplets, rules, semantic networks, and frames. These four techniques are described here in more detail.

Object-attribute-value triplets are facts (also referred to as propositions) like *"the ball's color is blue"* or *"the host's IP address is 139.23.44.21"*. The object 'Ball' has a property (attribute) 'color' which is 'blue'. The object can be a physical or an abstract

item, such as 'love' or 'mortage'. In most expert systems one object has more than one attribute. Not each attribute has only one value. It is possible to have attributes with multiple values, such as an attribute 'name of the childrens' can have the values 'Alice' and 'Bob'. As an addition, a fact can be extended with certainty factor which defines in a numeric value how certain it is, that the fact is true.

Rules are a kind of procedural knowledge. They associate given information to some action. A rule consists of one or more antecedents (if part), and one or more consequents (then part). They can also have a certainty factor, like object-attribute-value triplets. In a rule-based expert system the domain knowledge is captured as a set of rules. For more information about rules please see subsection 2.5.1.

Semantic networks were one of the earliest attempts in artificial intelligence to represent knowledge in a computer.[2] They provide a graphical view of a problem's important objects, properties, and relationships. Figure 2.4 shows a very simple semantic network. A powerful feature of semantic networks is inheritance, but it also causes problems. The exception handling is thus very important in semantic networks.

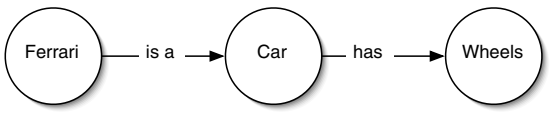

Figure 2.4: Simple example of a semantic network

Frames are based on the same idea like objects in object-oriented programming. They contain declarative knowledge (properties) as well as procedural knowledge (methods). Frames can be inherited like classes in object-oriented programming. Frame-based systems also offer a feature called facets that provide additional control over property values. With facets it is possible to restrict a numeric property to some range or a string property to a set of values.

2.5 Different types of expert systems

There are many different approaches for building expert systems. Early expert systems have been built with *rule-based* or *procedural* reasoning. Later approaches have been using *object-oriented*, *model-based*, *case-based* or *probabilistic* ways for the reasoning and the representation of the knowledge.[7] These different approaches are explained in the next subsections.

2.5.1 Rule-Based Systems

Most rule-based systems are built like depicted in figure 2.5. The main parts are the knowledge base, the working memory, and the inference engine, as explained in section 2.2. The used subsystems are a user interface to interact with the system, a developer interface to update the knowledge base, an explanation facility, and sometimes external programs that gather additional information or run external tools.

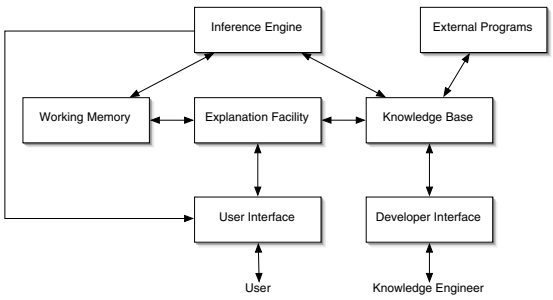

Figure 2.5: System Architecture of a rule-based system

The popularity of rule-based expert systems has grown out of a large number of successful rule-base systems built[8] and because there is a lot of rule-based expert system development software and languages[9]. The basis of a rule-based system is a so called production system. A production rule[10] consists of an antecedent (if) part describing

[7] See [2] and [1].
[8] e.g. MYCIN or STRIPS, see[1]
[9] e.g. CLIPS
[10] productions are sometimes also called canonical systems

the situation and a consequent (then) part describing an action to perform. Figure 2.3 on page 2.3 shows an example of some production rules. Production rules are intended as generative rules of behavior. Given some set of inputs, the rules determine what the output should be.

In expert systems we are interested in taking a representation of some problem and transforming it – using the production rules – until it satisfies some criterion that says: "This is a solution to the problem".

Advantages of a rule-based system

- The form of the rules (if - then) is very close to natural problem solving expressions of humans.
- The separation of control from knowledge makes it easier to maintain the system[11].
- The knowledge is modularly structured.
- It is easy to expand the system with new rules.
- The system only takes account of the rules that are relevant for the actual problem.
- Some expert system shells[12] allow a consistency check over the knowledge base (rules) to ensure that the same situation does not lead to multiple actions.
- With the addition of a *certainty factor* it is possible to include uncertain knowledge.
- To enhance the efficiency of the system, variables can be used that make rules more general.

Disadvantages of a rule-based system

- The match for a rule's 'if' part must be exact. It is not possible to do inexact matching of rules.

[11] This is true for all expert systems, except for most procedural and object-oriented systems
[12] An expert system shell is an easy to use software development environment to build expert systems.

- The performance can be bad if there is a large set of rules, because the inference engine has to search the whole rule set to decide which rule to apply.
- Although a simple rule may be easy to understand, it can be very difficult to determine how rules are logically related to the whole inference chain.
- The design concept of rule-based systems can be inappropriate for some problems. "Program designers have found that production systems easily model problems in some domains, but are awkward for others" (Davis and King, 1977) in [2].

2.5.2 Procedural Systems

Many expert systems are just an extension over a normal procedural programming language. The knowledge in procedural expert systems is implemented as a bunch of procedures that are executed step-by-step if necessary. The difference between a conventional computer program and such an expert system is just the role the system plays.

Advantages of a procedural system

- Use of 'known' procedural programming languages like C or also script languages like perl.

Disadvantages of a procedural system

- No separation of knowledge and control logic.
- Not suitable for bigger systems.
- Can not learn from experience.

2.5.3 Frame-Based / Object-Oriented Systems

A Frame-Based expert system uses techniques that are borrowed from object-oriented programming. Therefore this kind of systems is also called object-oriented expert sys-

tems. The terms object and frame are often used interchangeably.

All components of a rule-based system can be found in a frame-based system. There is just one little but important difference. In frame-based systems frames are used to represent knowledge rather than rules. One advantage of this kind of knowledge representation is the fact that an object contains not only information about itself, but defines also how the object behaves. In a rule-based system the entire problem is viewed as being neatly represented in the form of rules, where in a frame-based system everything is thought as of an object.

A typical field of this kind of expert systems can be found where knowledge can be represented in a hierarchical way (e.g. Human, Men, Jack or Animal, Bird, Penguin). There are many characteristics of object-oriented programming that can be used like subclassing (generalization, aggregation, or association), inheritance, and other known principles of object-oriented programming. An additional concept in frame-based expert systems are facets, which provides additional control over a frames property value, e.g. to constrain a field value to a defined value range. Another advantage of frame-based expert systems is the ability for a systems engineer to use message passing techniques to accomplish inter-object communications. The usage of such a system is indicated if you must work with several similar types of objects so that you can take advantage of the concept of instances.

Advantages of a object-oriented system

- Use of 'known' object-oriented programming languages like Java or C++.
- Usage of object-oriented programming features like inheritance.
- An Object can not only contain information about itself, but defines also how the object behaves.

Disadvantages of a object-oriented system

- No separation of knowledge and control logic.
- Can not learn from experience.

2.5.4 Model-Based Systems

Model-based reasoning systems attempt to solve problems by representing and reasoning with a deeper understanding of the application domain, which is typically embodied in a physical system. To be able to work with a deeper understanding it is necessary to create a model of the whole domain. If the domain is large it is quite difficult to create such a model. In particular, if there are many dependences between several components of the model, the creation of an overall model can be practically impossible.

One of the biggest advantages of a model-based system is the reduction of the burden of knowledge acquisition. It is not needed to acquire all possible symptomatic behaviors and knowledge from an expert. Also it is not necessary to identify every possible malfunction ahead of time, as it is required in systems that are rule-based. Instead it is required to develop a general model of the system. Therefore the system is able to diagnose faults that have never occurred before.

The drawback one has to accept with model-based systems is that this kind of system is not able to manipulate heuristic knowledge, cannot handle uncertainty and that it can not be applied to domains where models either don't exist or cannot be easily described.

Advantages of a model-based system

- Exact because of exact models of the reality.
- Acquire all possible symptomatic behaviors not needed.
- Not necessary to identify every possible malfunction.
- System is able to diagnose faults that have never occurred before.

Disadvantages of a model-based system

- Required to develop a general model of the system.
- System is unable to manipulate heuristic knowledge and cannot handle uncertainty.
- Very complex to build.

2.5.5 Probabilistic / Bayesian Approach

In many applications the facts and the knowledge are uncertain. Sensors, for example, are often not perfect and because of this it is uncertain, whether the value of the sensor is correct or not. This can be handled by different probabilistic methods like the Bayesian theory[13], certainty factors[14], the Dempster-Shafer theory, or fuzzy probability[15].

The conditional probability $p(A|B)$ permits us to obtain the probability of event A given that event B has occurred. But many times we are concerned with the reverse situation. This is often referred to as the *a posteriori* probability. A typical application of the *a posteriori* probability can be found in machine diagnostics.

Bayes solved the problem that the conditional probability is forward in time while the a posteriori probability is backward in time. Let's take now a closer look on the Bayesian Approach which is the basic idea for Bayesian Belief Networks. According to Bayes it is true that

$$p(h|e) = \frac{p(e|h)p(h)}{p(e)}\,[16].$$

The following example illustrates the usage of this formula[17].

[13] method of conditional probabilities
[14] for example: the patient has disease x with certainty 0.7.
[15] for example: the Porsche 944 is a *fast* car.
[16] Known as the Bayes' inversion
[17] Taken from [8]

```
h      = ‘‘Someone has malaria’’
e      = ‘‘Someone has calenture’’

p(h)   = Probability that a person fall ill with malaria
p(e)   = Probability that a person has calenture
p(e|h) = Probability that a person has calenture under the condition
         that this person has malaria

What we like to know is the conditional probability p(h|e) i.e. the probability
that someone has malaria when we know that this person has calenture.
```

The drawback of such an approach for handling uncertainty is that:

- Prior probabilities of an event must be known[18].
- Probabilities must always be updated.
- The sum of the probabilities for and against a hypothesis must equal one.
- Conditional independence of the data is required.

But if the above requirements or assumptions can be met this approach offers the benefit of a well-founded and statistically correct method for handling inexact reasoning.

Advantages of a probabilistic system

- May learn from experience.
- For some applications very efficient, e.g. for spam mail filtering.

Disadvantages of a model-based system

- Difficult to get the probabilities if there are not enough past data.
- Only usable if all the dependencies between all the components are known.

[18]This requires that there are enough past data on the events of models available to make accurate probability statements.

2.5.6 Case-Based Systems

The main idea behind case-based reasoning is, that new problems can be solved by remembering a previous similar situation (a case[19]) and by reusing information and knowledge of that situation. The goals of such a system are to learn from experience, to offer solutions to novel problems based on past experience, and to avoid extensive maintenance. Case-based reasoning is a cyclic and integrated process of solving a problem. Figure 2.6 shows the four main processes involved in CBR[20] systems: Retrieve, Reuse, Revise and Retain.

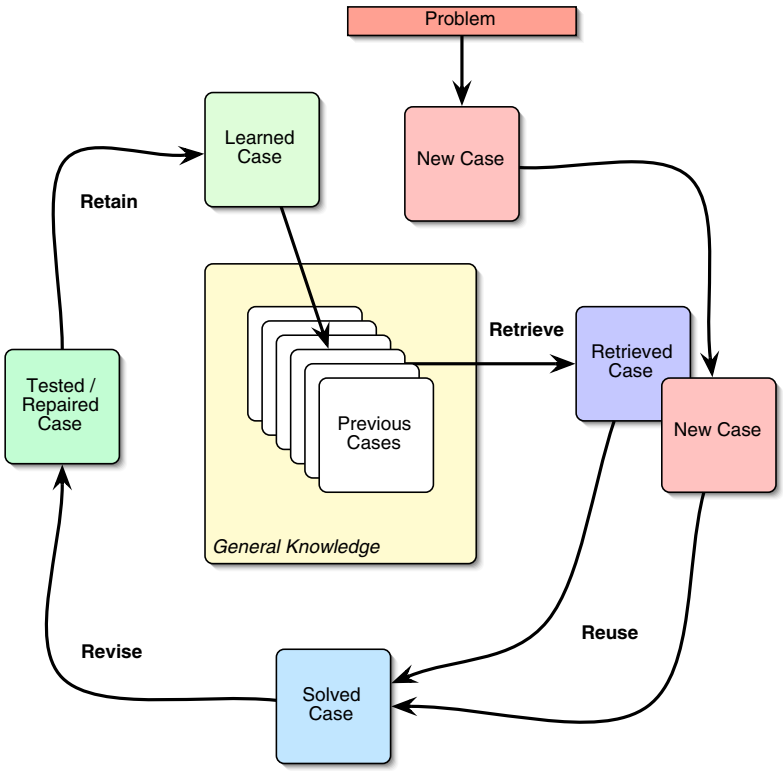

Figure 2.6: The CBR Cycle

Retrieve is the task of finding the most similar case in the knowledge base (in case based systems this is the case library). The new case is compared to all cases in the case

[19] A case usually denotes a problem situation
[20] Case-Based Reasoning

library, and the best matching case is returned (or the best matching cases).

The process of **reuse** means the use of information and knowledge in the retrieved case to solve the new case. This task is often referred to as *adaption*. The resulting case of this operation is the *solved case*, which contains the proposed solution.

Through the **revise** process the solution obtained by *reuse* is tested for success, e.g. by applying it to the actual situation. If the test fails, the system learns that this solution was not successful and remembers this for a next time.

During **retain**, useful experience is retained for future reuse and the case base is updated by the new *learned case* or by modifying an old case.

The retain process illustrates an important concept of CBR. Sustained learning, by updating the case base after solving a problem has been solved, can improve the efficiency of the system. And a CBR system is able to learn from success, but also from failure. When an attempt to solve a problem fails, the reason for this failure is identified and remembered in order to avoid the same mistake in the future.

When we compare the case-based to the rule-based approach we see that rules in a rule base are patterns, whereas cases in a case base are constants. Rules contain variables, they do not describe solutions directly. Rules are selected on the basis of an exact match between data in working memory and its antecedent (if-part), whereas a case is retrieved on the basis of a partial match, facilitated by knowledge about its features. While rules are applied in an iterative cycle – a series of steps towards a solution – cases can be viewed as approximations to an entire solution, although it is also possible to draw analogies to different cases iteratively to solve different parts of a problem.

The biggest challenge in building a case-based expert system is to design a clever case language and to find effective ways to compute the similarity between cases.

Advantages of a case-based system

- May learn from experience.

- No data needed (in contrast to the probabilistic approach).
- Can extend easily an existing system.
- Solution centered view, not symptom centered.

Disadvantages of a case-based system

- Dependencies are difficult to describe (if needed).
- Can be slow.
- Not applicable for every problem.

2.6 Summary

In this chapter you where introduced to the broad field of expert systems and the technologies used for building such systems. It is easy to see that every approach has its own advantages and disadvantages. The most important things to remember are:

- The domain for an expert system should be of reasonable complexity.
- It is important to choose the most appropriate approach for each domain.
- The expert system development is a highly iterative process.
- Expert systems can not solve problems that human experts are not able to solve.
- Expert systems don't have to be 100% precise.

Chapter 3

Computer Networks

This chapter depicts some basic concepts of computer networks in conjunction with an expert system based network troubleshooting system. It is not meant as an introduction to computer networks. Please see [4] or [3] for a detailed view of computer networks.

The term *Computer Network* encompasses many technologies which can not all be covered by this thesis. Because of the growth today of the internet this thesis focuses on TCP/IP networks. The first section presents some of the basic elements in a network, the second section explains the difference between fault, failure and error. Three examples of possible failures and malfunctions in a TCP/IP network are presented in the third section. The fourth section tries to show you the difficulties of mapping faults to root causes.

3.1 Elements of a network

A computer network consists of a lot of individual elements, such as routers, switches, servers, workstations, and of course all of the cabling. Each of this element can cause a degradation of performance, interruption of service, or unreliable communications between two nodes. Today, security issues are getting also more and more important. Denial of service attacks have heavy impact on the network availability and affect many users on the network. For a network operator it is important to find as fast as possible

a solution for a network problem, but many times the search for the root-cause of a problem or to find a solution is a very time consuming process. The next sections should clarify this.

3.2 Degrees of failings

For a network troubleshooting system it is crucial to think about the possible faults and failures in a network. There have been extensive studies about the differences between failure, error, and fault. In [9] the different degrees of failings are discussed. The most important are:

- **Fault:** A fault is a condition which leads to a malfunction or or outage of a functional unit. An example of a fault is a defective ethernet card.
- **Error:** An error is a sign for the difference between an observed, calculated, or measured value or condition and the true, specified, or theoretical correct value or condition.
- **Failure:** A failure is the deviation of a adduced service from the specified service.

These three degrees are related to each other as follows: Every fault can generate one or more errors. These errors indicate that there was a state change in a network's system. The error is then the product of the state change which cannot be observed directly. Normally these errors causes one or more failures which are often noticed by the users of the network. The symptoms can then be used as a starting point to find out the root cause of the failure.

An example of such a chain of fault, error and failure is:

```
Hardware Adapter defect
-> No connectivity on physical link from a to b
-> No connection between server and client
```

The root cause analysis of a failure is probably one of the most difficult tasks when troubleshooting a network. The troubleshooting process is normally the reverse counterpart of the fault escalation. Unfortunately it is often the case that one has no access to every layer of the source and target network as well as the networks between them. This makes identifying the root cause a difficult business. Before talking more about root cause analysis in detail you probably want to look at the three examples of network failures and malfunctions presented in the next section.

3.3 Possible Failures and Malfunctions

Because of the heterogeneous structure of computer networks it is not possible to create a list of every possible failure or malfunction. But often it is possible to create some categories and classify the failures, for example into connectivity, security, and performance related problems. The examples in the next subsections should clarify this. The setup for all three Examples remains the same. Figure 3.1 shows this setup. There is a client in a local network and a DNS server on the same segment. This local subnet is connected to the internet through a gateway. Anywhere on the internet is a server (acting as a webserver), some hops away from the local subnet.

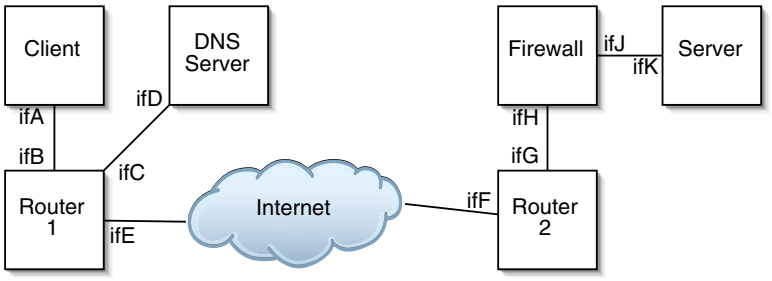

Figure 3.1: Setup for the samples

3.3.1 Connectivity Issues

This is a simple example of a connectivity problem. Assume that a user on the client likes to connect to the webserver to see a specific webpage.[1] The first thing the client has to do, after the user has entered the URL of the page, is to lookup the corresponding IP address of the webserver. Because the DNS server is on the same subnet, the client sends directly an ARP request to find out the hardware address of the DNS server. The searched address is then returned by the DNS server within an ARP response. The client now is able to send the DNS request to the DNS server, which answers with a DNS response containing the IP address of the webserver (together with other information about the query). Finally the client is now able to send the HTTP request via the default gateway to the webserver. Please note that in this case the ARP request to find out the hardware ethernet address of the default gateway is not necessary because this information should now be in the cache[2]. If everything works as desired the webserver sends back the corresponding page (in one or more TCP packets) and closes the connection.

There are many different possibilities for faults that results in failure of the communication between client and server. The following list presents you just a few root-causes for possible faults:

- The clients subnet mask is defined wrong.
- The clients default gateway is not correct.
- The DNS server returns the wrong address because of a misconfiguration.
- The router is out of service.
- The cabling is incorrect.
- One node has a blackout.

[1] This description assumes that the page consists only of one file, that the ARP and DNS caches of the client machine are empty at the beginning, that the client has only one interface (beneath the loopback interface), and that the client, gateway and DNS server are connected over a hub.

[2] The DNS server already asked for the hardware ethernet address of the default gateway, and the client machine stored the answer in his ARP cache.

- The webserver is offline.

The question now is whether it is possible to find the root-cause of the fault or not. This topic is discussed in the subsection 3.4.

3.3.2 Security Issues

This example of a security problem describes a sample scenario of an attack to the webserver. In this case the client machine is used to attack the server. The user on the client begins probably with a port scan. The result may look like this:

```
22/tcp     open        ssh
25/tcp     open        smtp
80/tcp     open        http
143/tcp    open        imap2
631/tcp    open        ipp
993/tcp    open        imaps
6000/tcp   open        X11
```

The attacker can use the information provided above to start the search for possible security holes.[3] As a next step he can try to find out which daemons are active behind the open ports. In this concrete case there are the following interesting daemons running[4]:

```
Port 22:
SSH-2.0-OpenSSH_3.4p1

Port 25:
220 foo.bar.com ESMTP Sendmail 8.11.2/8.9.3/SuSE Linux 8.9.3-0.1;
  Tue, 4 Feb 2003 16:27:17 +0100

Port 80:
Server: Apache/1.3.27 (Unix) mod_perl/1.27 PHP/4.2.0 mod_fastcgi/2.2.12
  mod_ssl/2.8.11 OpenSSL/0.9.6d
```

[3] It is assumed that the webserver is a single machine that is not hided behind NAT.
[4] The data is fictive

```
Port 143:
* OK [CAPABILITY IMAP4REV1 LOGIN-REFERRALS STARTTLS LOGINDISABLED] localhost
  IMAP4rev1 2002.336 at Tue, 4 Feb 2003 16:30:38 +0100 (CET)
```

At the time of writing there is a known vulnerability in PHP versions 4.2.0 and 4.2.1, which could be used by a remote attacker to execute arbitrary code or crash PHP and/or the web server[5]. Suppose that the user on the client knows about this vulnerability and that he knows how to shortchange this. Then he sends a POST request with the malicious code to crash the webserver or to execute code that gives him for example root access on the server. Another vulnerability of the server is that there is version 0.9.6d of OpenSSL installed together with mod_ssl and apache[6]. If the attacker knows that this vulnerability exists, he can try to send the following invalid HTTP GET request to the server[7]:

```
GET /mod\_ssl:error:HTTP-request HTTP/1.0
```

In this case the malicious code is placed on the victim server, where the attacking system tries to compile and run it. Once infected, the victim server begins scanning for additional hosts to continue the worm's propagation.

Intrusion detection systems (IDS) are often able to detect such attacks. The IDS then informs the administrator of the potential attack, who has to interpret the message from the IDS to see if the attack was just a false alarm or if it was successful.

3.3.3 Performance Issues

The most difficult thing in a packet switching best-effort network is to make a proposition about the performance of a connection between two nodes. Because every packet can go it's own way through the network there are a lot of possibilities why a part of the network

[5]CERT Advisory CA-2002-21
[6]CERT Advisory CA-2002-27
[7]In fact this is what the Apache/mod_ssl worm does.

seems slow. Not every applications in the internet is equally vulnerable to performance problems. Mail transfer for example can have relatively long delays without disturbing anyone, where as applications like telnet or ssh react very sensitive to bad performance, because the user gets a feedback for almost every character he enters in the terminal.

The following example should give you an idea of a performance issue in a computer network. Assume that the client wants to access the webserver by SSH. The connection is setup as desired and the user on the client begins to type. Suddenly the response of the server gets slower and slower. This could have various causes like:

- Heavy utilization of the client or the server.
- Huge packet loss on the connection that causes many retransmits.
- Too much traffic on any of the links between server and client.
- DoS attack which causes a total disruption of service.[8]
- Too many collisions in one collision domain.

Some of the listed reasons can affect the communication for just some seconds or also for a much longer time. Another possible reason for performance problems is of course the wrong dimensioning of components and cables.

3.4 Mapping the Faults to Root Causes

The mapping of faults to root causes is not always possible. Sometimes one needs physical access to a node or to a cable, sometimes it is required to have access to some nodes which can cause problems via a terminal (for example ssh), in other cases one should have the possibility to debug a subnet which is far away. An other thing which makes the mapping difficult is that there may be many faults which leads to the same error. Take as an example the setup from above (connectivity issue). Now suppose that the client is

[8]This is of course also a security problem.

not able to see a webpage on the webserver. The user on the client may try to ping and traceroute the webserver. If both fails it is really difficult to find the root cause of the failure. There are many possible faults like:

- The server is down due to a blackout.
- The ethernet cable is not connected to the server.
- There is a firewall in front of the server which doesn't permit packets sent to this server.
- NAT is used and there is a misconfiguration on the NAT gateway.

Without physical access to the subnet of the webserver it is not possible to find the root-cause in this case. Most problems that have their root-cause on the physical or the data link layer require that one has physical access to a single subnet or even more segments of the network. Even for some problems that are located in higher layers it is impossible to find the root-cause without access to the system or to the network.

3.5 Summary

This chapter introduced you to some of the most important network problems. It also described the difference between fault, error, and failure. Three examples of different network problems have been presented which introduced three categories of network problem such as 'Connectivity Issues', 'Security Issues', and 'Performance Issues'.

Because one major aspect of this thesis is to help also on problems that are located in the parts of the network that is not managed by the same operator than the management network, a more detailed look at the mapping of faults to root-causes has been dropped in advantage of a more detailed look to other expert system approaches which don't need such a root-cause analysis.

The next chapter presents some existing network management and troubleshooting systems that are interesting for this thesis.

Chapter 4

Existing Network Troubleshooting Systems

There are several different commercial products on the market that help administrators to manage and troubleshoot computer networks. The most popular systems that are used to troubleshoot a network are discussed in this chapter with a special focus on those who use expert systems. Unfortunately it is not easy to see if such a system is using expert system technology or not. To get a more detailed view on these systems, an email was sent to every company who is developing such network troubleshooting and management systems, but only Computer Associates and Rocket Software where able to send back cooperative email. For all other products as much information as possible had to be extracted out of the datasheets. The answers and information of the emails returned and datasheets are listed in the next sections.

4.1 Computer Associates: UniCenter Network and Systems Management

All of the following information has been taken from an eMail from Rocket Software and from the datasheet[14].

"UniCenter is Computer Associates's family of integrated eBusiness Infrastructure Management solutions that uniquely bridge past, present and future technology with the expanding infrastructure of the digital economy."

The scalable and modular architecture gives the customer a good decision what he like to manage and which features he like to have. Each UniCenter module can be used as stand-alone, in combination with other UniCenter modules, or even with third-party solutions. There are modules for planning, managing, and troubleshooting todays computer networks[1].

The most interesting question for this thesis is whether they use expert system technology for the root-cause analysis or not. According to the email received from Lukas Ringwald, a Senior Consultant of Computer Associates, they do not use expert systems for the root-cause analysis.

The software gets the information about the network status and failures from SNMP, ICMP messages, or dedicated agents on the target operating system. It identifies the root cause of network problems, automatically generates alarms that notify network administrators and executes automated corrective actions (if requested). The topology of the network can be automatically discovered and the objects will be placed in the correct IP subnet or segment on the UniCenter Explorer map. Part of the product is also an advanced correlation engine which filters and correlates incoming alarms and events. Events from Intrusion Detection Systems can be forwarded to the event management console. With the module *UniCenter Performance Management* it is also possible to use performance monitoring to locate the place of a network problem. By customizing the system it should be possible to propose a detailed procedure as to how to solve a specific network problem.

[1]There are many other modules available like *Application Management* or *Database Management* which are not important for this thesis.

4.2 Rocket Software: NetCure

All of the following information has been taken from an eMail from Rocket Software and from the datasheet[16].

NetCure from Rocket Software uses as a primary source for information about the network status and failures SNMP MIB's. But they get as well access from some proprietary sources. This information is used along with the discovered topology to determine the health of the network. The network topology can be discovered automatically (layer 2 and layer 3). NetCure handles all of the filtering of incoming events and then determines if there are any correlations to be made. There is also some kind of trouble ticketing system which is included in NetCure which can be configured to execute a response to any event automatically as necessary. As long as information from intrusion detection systems are captured as SNMP traps, it is possible to use this information too for network troubleshooting. The knowledge in the system – user written correlations – can be customized by the proprietary XML based language KBML. NetCure also uses performance monitoring to determine if a potential or observed performance problem is the source of other network anomalies. Rocket Software also included a root cause analysis into this system but didn't tell if this is done by an expert system. NetCure does not imply how to fix most problems. However, the descriptions of a number of the Event Management scenarios do include suggestions for what to change in order to fix a known problem.

4.3 Network Associated: Sniffer Distributed

All of the following information has been taken from the datasheet[10].

Sniffer Distributed is a complete set of network protocol decodes and unique, real-time Expert diagnose. This application uses information provided by SNMP, RMON1, and RMON2 for longterm monitoring and trend analysis. It delivers "the unequaled ability to anticipate, isolate, and diagnose network faults and performance problems through a combination of unique, real-time Sniffer Technologies Expert diagnoses and extensive

protocol decodes". In the datasheet they highlight the expert system used for Sniffer Distributed. It is able to diagnose the following failures:

- WINS no response
- Excessive retransmissions
- DB slow server response
- Slow HTTP server
- Protocol negotiation failure
- Slow file transfer
- IP NetBIOS session reject
- TA list resources exhausted (ATM)
- LANE configuration phase failure (ATM)

They also included continuous monitoring for real-time network status updates and reporting functionality like Expert analysis reports that should help to isolate problems quickly. In addition Sniffer Technologies Expert diagnoses complement Intrusion Detection System capabilities to provide incident analysis, identify log-in breaches, and detect unauthorized database login attempts.

4.4 HP: OpenView TeMIP expert

All of the following information has been taken from the datasheet[13].

HP OpenView TeMIP Expert provides expert system rules that capture the expertise and knowledge within network operations and automate operations processes to solve a problem. With this expert system it is possible to process a variety of data like events, alarms, equipment states, route availability and usage. TeMIP Expert is an add-on

module to the HP OpenView TeMIP modules. The filtering and correlation of the data is done by TeMIP Expert too. The expert system is based on ILOG Rules[2] – one of the most efficient rules algorithms available today. A root-cause analysis is included in TeMIP Expert as well as a customized trouble ticket processing and a superior inference engine.

4.5 Concord: eHealth

All of the following information has been taken from the datasheet[11].

"eHealth for Network Management Solution delivers integrated and proactive fault, performance, and availability management across complex, heterogeneous, IT environments." This system is able to – beside other powerful features – perform continuous tests to identify faults and response degradation for network circuits and to identify which resources are at risk of failure with the help of embedded intelligence to prevent outages and ensure availability. It is not sure if Concord uses expert system technology for the *intelligence*. In the eHealth Suite also included is a fault management which is used to correlate the incoming alarms and events.

4.6 Lucent: Navis Network Fault Management Software

All of the following information has been taken from the datasheet[15].

The Navis Network Fault Management Software – which is part of the Navis iAssure Suite – "detects and links network faults with their sources in real time, determining root causes so you can resolve them quickly". This software is able to collect alarms from virtually any network element that exists today. As a main information base they use SNMP MIB's. With the help of user-defined triggered responses (UDTR) the system

[2]More information on ILOG rules can be found on the ILOG website: `http://www.ilog.com/products/rules/`

executes remote programs whenever certain conditions are met. Navis offers a Web GUI to access all of Navis functionality from web browsers under Windows and a command line interface to allow users and administrators to create and execute scripts. They also implemented a graphical map display which uses color coded elements based on the criticality of the alarms. In addition there are four types of thresholding and up to 10 escalation levels to prevent important alarms from being disregarded and to involve management when needed. An other important part of the Network Fault Management Software is the event correlation. This feature uses the rules-based look and feel of an artificial intelligence platform, without forcing users to learn a complicated syntax. As an optional feature there is a trouble ticketing system.

4.7 Micromuse: The Netcool Suite

All of the following information has been taken from the datasheet[12].

The three most interesting parts of the Netcool Suite are Netcool/Visionary, Netcool/OMNIbus, and Netcool/Service Monitors. The Visionary module is an application which analyzes information from SNMP devices and applications and provides real-time diagnosis. “Its patent-pending MicroCorrelation technology analyzes the SNMP Management Information Base (MIB) information to pinpoint the underlying cause of hundreds of unique problems that pertain to specific devices.” This technology searches for deviations from normal behavior and provides an instant explanation for the condition. Netcool/OMNIbus collects and consolidates events and alarms from more than 300 networking environments like for example servers, mainframes, applications, routers, among many others in real-time. The Netcool/ObjectServer, the core component of the Netcool/OMNIbus application gathers fault messages from the network for filtering. The Netcool/Service Monitors is capable of monitoring ecommerce applications, email services, wireless services, service infrastructure, and bandwidth.

4.8 Summary

All the described management systems have a lot of features that makes them useful for managing computer networks. An email has been sent to every company to ask for more detailed information about their network management solutions, but only a few companies were able to respond to the email. Because of this there is too little information to compare all described systems. The marketing information in the datasheet doesn't tell enough, for example if the companies are using expert system technology. Most of them presumably use some kind of artificial intelligence to take over a couple of complex task like for example the root-cause analysis, the alarm and event filtering, or other problem solving assistive parts of their software. Most of the systems are also too complex to install a demo version. The basic design of all network troubleshooting systems remains the same. Events and alarms are generated from monitor devices or readout of SNMP MIB's, are then filtered to reduce the count of alarms and correlate events which depend on each other. The filtered data then is stored in a database or similar structure and can be viewed from a user interface. This interface gives the user the possibility to interact with the system and execute maintenance procedures. There is often a ticket system integrated in such systems to keep track of changes during the troubleshooting process.

The next section describes the basic network troubleshooting expert system design with respect to the fact that there is an existing system, such as a trouble ticketing system, that should be extended with expert system technology.

Chapter 5

The Network Troubleshooting Expert System

Network troubleshooting systems are often closely related to network management systems. Most troubleshooting systems are sold as an extension to existing management solutions, albeit it is sometimes possible to buy only the troubleshooting part. The main idea of the work done in this thesis is to have a troubleshooting system which extends an already existing system, such as a trouble ticketing system.

Open Systems AG already implemented some part of the whole system, namely the ticketing system with the needed components to run it (input sources, alarm filtering). This and the reasons described above are the main reasons why this thesis tries to use a trouble ticketing system as a starting point for further development to build a more 'intelligent' system.

This chapter describes the basic concept of a network troubleshooting expert system that is based on a trouble ticketing system. This includes an overview and a short description of each component.

5.1 Overview

Figure 5.1 shows the whole system including input sources, a trouble ticketing system, event and alarm filtering and the core of the expert system. The next section explains every component in detail.

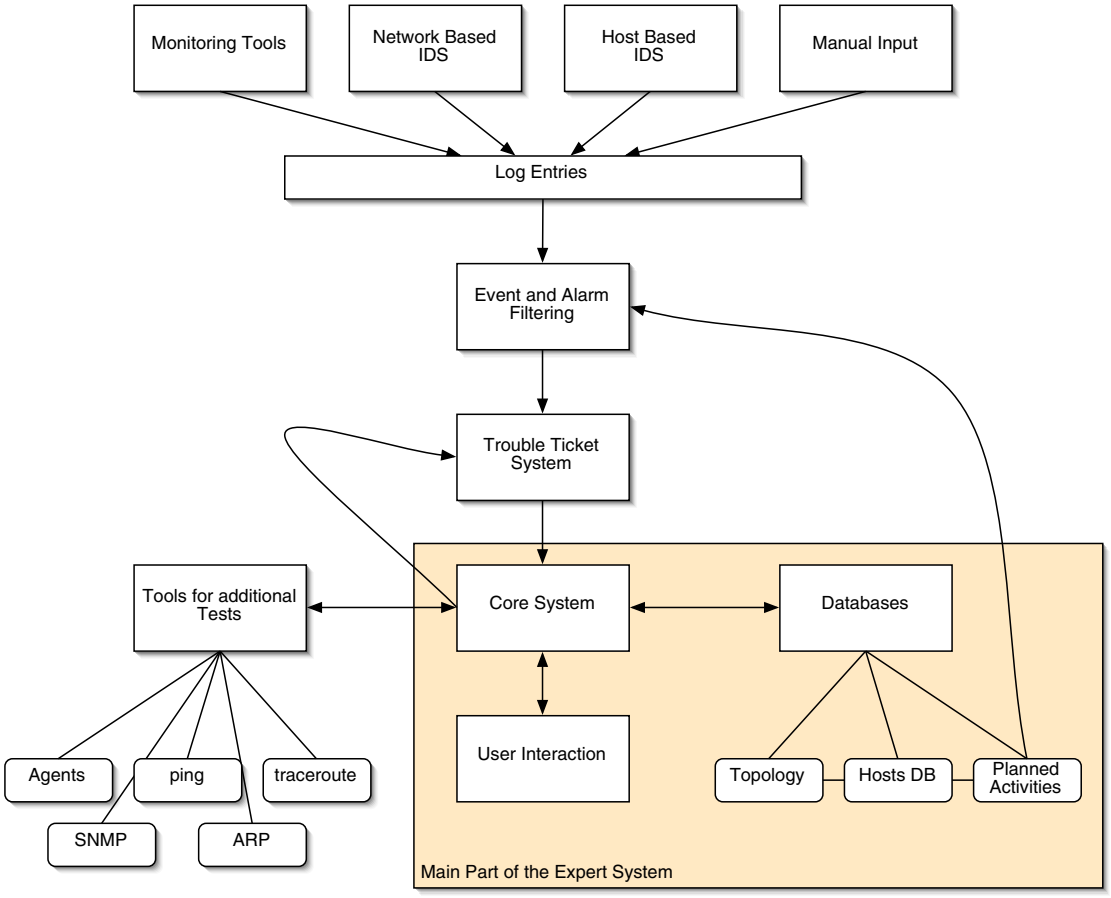

Figure 5.1: Overview of the whole System

5.2 Components of the System

5.2.1 Input Sources

Input sources are the main resource for information about the status of the network. These sources can include any kind of monitoring tools (host or network based), network and host based intrusion detection systems (IDS), and manual input from users of the network. All these input sources generate log entries that are collected in one or more log repositories.

5.2.2 Event and Alarm Filtering

The event and alarm filtering process is responsible for filtering and correlating the events provided by all the input sources. This step is important because one fault in the network can generate many log entries. These log entries can all come from different input sources. And it is also very likely that these entries are different regarding the log message as well as regarding the time they occur. Take as an example the situation where a router has a power breakdown. Some monitoring tools will report that the router is not reachable while others announce that they cannot connect to the server behind the router. In addition it is possible that a monitoring tool behind the router tries to deliver a message to the log entry database that reaches the log entry database as soon as the router is working again. This example should account for the need of such an event and alarm filtering. The process should take all the events and alarms and fade out the less important ones. It should also be able to block events and alarms if there is a known service interruption because of a maintenance task going on. Another important function of the event and alarm filtering is to prevent alarm-flooding to ensure a reliable operation of the trouble ticketing system.

5.2.3 Ticketing System

All the relevant events and alarms are administrated in the trouble ticketing system by the network operators. It assists the operators with the fault recovery. The ticketing system guarantees that every alarm and every event is processed by the operators. For each action the operator executes he enters the information into the ticket database. Each ticket is generated automatically by the event and alarm filtering process or manually by a network operator. A ticket contains one or more ticket events and some properties such as a ticket number, state, owner, and date information.

5.2.4 Core of the Expert System

The core of the expert system is the second part of the whole system that is 'intelligent', beside the event and alarm filtering. It is responsible for helping the users of the expert system to troubleshoot a network problem in a most efficient way. It should either help to find the root cause for the problem and/or propose solutions for solving the problem. The core system communicates with the user, internal and external databases, the ticketing system, and different helper applications.

5.2.5 Databases

The databases represent partly the knowledge of the system. These databases can contain information about the topology of the network, about the nodes in the network, about known activities, as well as information about network protocols, port numbers, contact persons, etc. Other kinds of databases are publicly available sources such as information from RIPE, CERT, or other organizations.

5.2.6 External Information Sources

As external information sources you can think of UNIX tools like 'ping' or 'traceroute', scripts or applications that use the MIB's of SNMP enabled nodes, or agents that can

provide more detailed information about the state of a node in the network. The data of the trouble ticketing system and the databases may not be enough for the core system. It may be necessary to get additional data from external information sources to help the core system to provide reasonable help to the experts who use the system.

5.3 Summary

This chapter introduced you to the main components of a network troubleshooting expert system. One part of this system has already been implemented at Open Systems AG, namely the ticketing system and the components used for the ticketing system, such as the input sources and the event and alarm filtering. The other parts are subject of the next chapter which describes the concept and the implementation of the expert system's core and the additional components used to build the expert system part of the whole system.

Chapter 6

Concept and Implementation

One of the tasks in this thesis was to design and implement a prototype of a network troubleshooting expert system. In order to do this it was required to choose an expert system approach which is best suited for this task. As mentioned earlier the field of all computer networks is too big to fit in one expert system. This thesis focuses on todays IP networks, with respect to the fact that is is often the case that one doesn't have access to every part of the network. Another demand for the system was that it should be possible to integrate it into existing systems. Many companies already have some kind of trouble ticketing system. Therefore a system should be designed which can extend an existing trouble ticketing system, in particular because there is already a trouble ticketing system at Open Systems AG. This chapter describes the basic ideas for the concept of the core of the network troubleshooting expert system. The first section presents the used approach for the expert system and explains why this approach has been used. In the second section you will find a detailed explanation of the concept. The way how a problem is described is the subject of the third section, while the fourth and fifth section deal with the similarity calculation and some implementation specific details.

6.1 Used Approach

The basic idea for building this system was to design a very flexible system that is able to help laymen and experts in troubleshooting network problems in IP networks. The system should be easy to manage so that any expert, who is working with the system, can change the systems behavior without the need of changing the source code of the system. For a real world use it is necessary that there are access restrictions for users who shouldn't be able to change some data, but this aspect was ignored for the prototype built for this thesis. There exists already many commercial systems that have an integrated root-cause analysis. Thus an approach should be selected that doesn't need to re-implement such a root-cause analysis. Another question is where to get the knowledge for the system and which expert system building approach should be used. As seen in chapter 2 on page 3ff. there exist many approaches for building expert systems. Each approach has its strength. The most interesting approaches for the use for a network troubleshooting expert system are the rule-based, the probabilistic, and the case-based approach. Procedural and object-oriented expert systems do not fit the requirements because the separation of knowledge and control logic is too difficult to handle. The model-based approach was not considered because the creation of a detailed model of the network, including all layers, seemed to be too complex. The rule-based approach is usable for some network problems, such as some connectivity issues. Rules can be used for example to describe the dependencies between the network protocols. But if the expert system should be used for any kind of network problem, the knowledge-base (set of rules) will get very complex. The management of such a complex rule-base can be very time expensive. In addition the rule-based approach qualifies for a root-cause analysis, but many existing systems do such an analysis, and it is not the idea to re-implement existing things in this thesis. The probabilistic and the case-based are the most interesting approaches for this thesis, because both of them are able to learn from experience. While the probabilistic approach needs a lot of past data and it is difficult to define every probability for each event in the network, the case-based approach seemed to be the most practical approach for this thesis. The main reasons why case-based

reasoning was chosen are:

- The system is able to learn from experience.
- There is no need for past data to begin working with the system.
- Different problem categories can be handled in one system.
- The system can be integrated into other existing systems, especially into a trouble ticketing system.
- When the system is configured once, there is no extensive maintenance needed.

The most difficult tasks in building a case-based expert system are:

- To find a well adjusted case representation.
- To calculate the similarity between two cases.
- To find a solution for a given case.
- To build a fast system.

There are two possibilities for finding a solution for a case. The first possibility is to calculate a solution out of the detailed description of the case. The second possibility is to have a pool of solutions where all past solutions are stored and to link one or more solutions to the current case. To implement the first possibility it is needed to know all potential problem types, in order that a solution can be calculated. The second possibility is more flexible, but the user of the system has to do more work, because he has to choose one solution from the pool of solutions. The effect of this difficulty can be minimized by presenting the used solutions of a similar case and let the user choose one of these. For a real world usage of the case-based system it is required to create a very efficient system. First of all the similarity calculation has to be very fast. This was not considered for this thesis because the prototype does not have to be fast.

A detailed description of the used case representation and of the similarity is presented in the next section. Near the end of the concept phase of this thesis, a similar approach has been found in [7]. The basic idea of a case-based network troubleshooting system is the same, but in this book there is not much written about the concrete implementation of the important stuff, such as the similarity calculation. Enquiries made have showed, that the idea presented in this book has been used to implement a commercial product (SprectroRX from Cabletron Systems). Lundy Lewis[1] was asked why SpectroRX is not available anymore. He answered that they stopped selling SpectroRX mainly because of the economy and bad marketing. But he thinks that the case-based reasoning approach is still a good approach for building a network troubleshooting expert system.

6.2 Detailed Description

This section describes the main parts in a case-based reasoning network troubleshooting system. First the case representation is discussed and the entity relation diagram of the case representation design is presented. Then the similarity calculation between two cases is described in detail. At the end of the section the use of variables, that can be used in the value of a case field, is explained.

6.2.1 The Case

In the case-based reasoning approach, one of the most important parts is the representation of a case. A case consists of the following three components:

- A primary slot for the main problem description and some additional informative fields, for example the creation date.
- Many different slots that hold the case description.

[1] He is the author of the book [7], an Adjunct Professor at the University of New Hampshire and Director of Research in a high-tech company.

- One or many solutions that are related to the case. This solution(s) can be computed by the values provided by the slots or entered manually by the user.

There exist two possibilities of representing a case. One approach is to define all the possible slots (fields) in advance. This is much easier but only possible if all the fields used to describe a case are known beforehand. The other possibility is more flexible because the slots are not explicitly named when implementing the system. The big challenge for this approach is, that the calculation of the similarity between two cases is much more difficult. This calculation is one of the most important things in a case-based reasoning system. If the comparison of two cases does not return results as expected by the user, the system will be unusable. In the network environment there are many changes over the time. There may be new protocols or new technologies that demand for new slots. If the first way of describing a case is used, the addition of new slots requires the system developer to change the way the similarity of two cases is calculated. In addition, the system developer most likely has to change the database structures and some code in the implementation. To avoid such maintenance intensive tasks, the second approach was used for the concept and implementation. It should provide an as flexible as possible case description as well as an adaptable behavior of the system to new situations. Such new situations can be changes in the network protocols, new categories, or other changes that results in a change of the case description.

6.2.2 Entity Relation Diagram

Figure 6.1 shows the entity relation diagram of the case representation, the associated similar cases and the connection to the trouble ticketing system. The central item is the case entity. Each case consists of some basic attributes such as a note, the warning time, and the warning source. The values of all other slots are not modeled as attributes to the case entity. Instead there is a pool of all fields (Key entity) used to describe a network problem. The used fields for a specific case are then related to it. The category entity is used to classify all network problems in about 3-5 classes.[2] Each case is related to such a

[2]Three categories where used for the prototype: Connectivity, Security, and Performance

category which is also used for the weighting. With the weighting relation it is possible to define a weight for each combination of a field (key) and a category. This weight is used to calculate the similarity of two cases. In this concept every case can have one or more solutions related to it and is related to other (similar) cases.

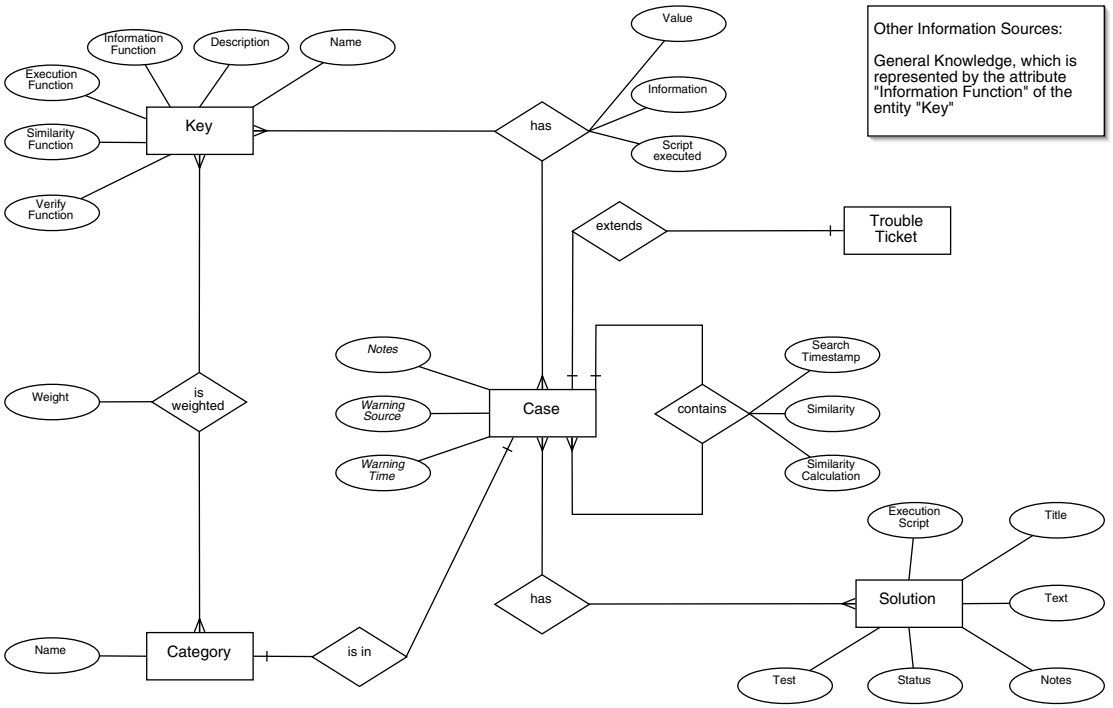

Figure 6.1: Case Entity Relation Diagram

6.3 Case Representation

This section describes the case representation in detail. The following conventions are used in this section:

- Tables are written emphasized, for example *classes*.
- Fields are written in single quotes, for example 'dest_ip'.
- Values are written in double quotes, for example "129.132.3.11".

6.3.1 Database Scheme

The database scheme according to the ERD[3] used is shown in figure 6.2. Each table has a primary key (id) and a modification date (md). Let's take a more detailed look on the tables and fields of the database.

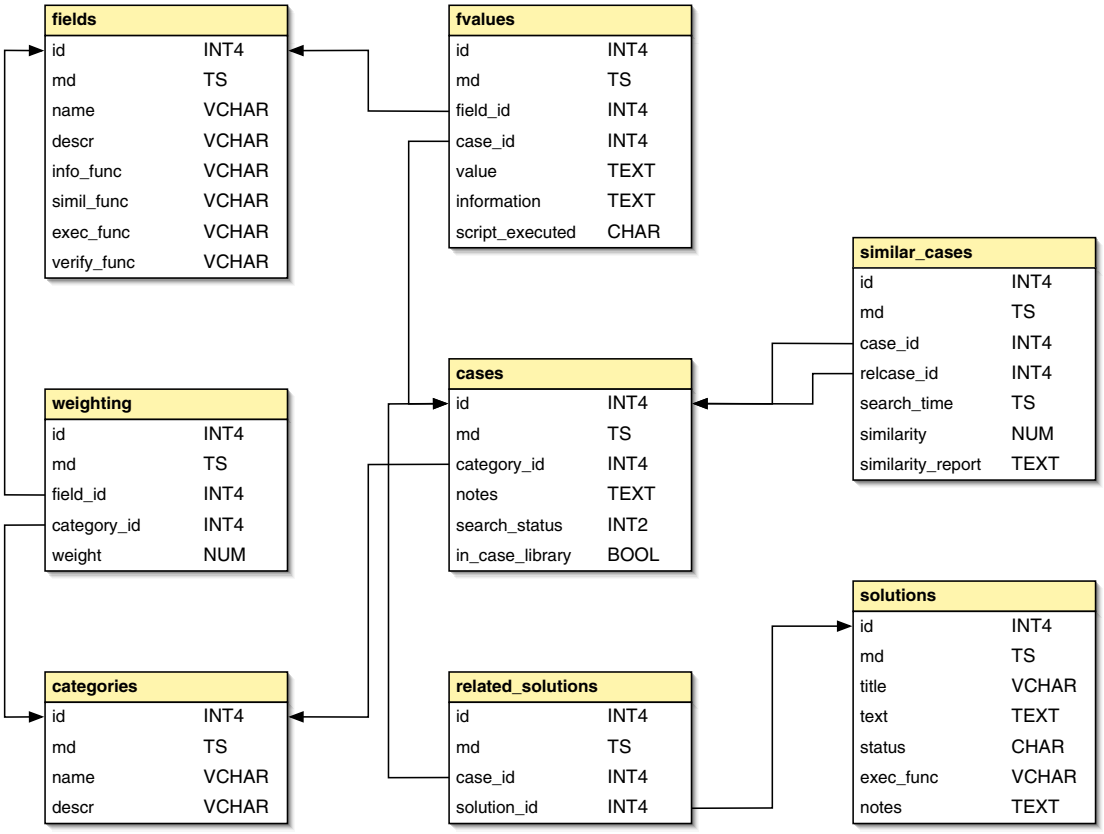

Figure 6.2: Database Scheme

6.3.1.1 Table *cases*

The table *cases* only contains information that is not relevant for searching a similar case, except the category_id, which defines the weighting set used for the similarity calculation. All values used to calculate the similarity between two cases are stored in the table *fvalues*

[3]Entity Relationship Diagram

(described in 6.3.1.3). The table *cases* contains the following interesting fields:

- notes: Holds general notes about a case. These notes will not contribute to the similarity computation.
- in_search_library: This boolean field defines if the case will be considered as part of the case library. A case will never be deleted. When a case is not used anymore this field is set to false.
- search_status: Because the search for similar cases can take a long time, this field is used to define what percent of the case library has been searched for similar cases. This field should tell the user about the progress of the actual search.

6.3.1.2 Table *fields*

This table represents the pool of field that can be used to describe a case. Because of the very flexible design, it is needed that there is a place where all the possible fields are stored. The table *fields* takes over this functionality. Every field that is used to describe a network problem is stored here, for example 'dest_ip' for the destination IP address or 'problem_desc' for a problem description. The field 'description' allows a more detailed explanation of the meaning of the field 'name'. This should help the users to enter the information in a field in a homogeneous way. There are also four special fields in this table, that represent each a path to a script or to an application[4]:

- 'info_func': This application is used to get more information about the value of a field. As an example you can think of the field 'dest_ip' and the value "129.132.3.11". There is more information behind this number, such as information to whom this IP address belongs, contact information, DNS name, etc. The function referenced by the field 'info_func' receives the value as an argument and returns more information about it, which is then stored in the *fvalues*'s information field. An example of an information function can be given for the field 'dest_port',

[4]Please see also table *fvalues* to understand the relation of the function fields to the values stored in the *fvalues* table.

where this function may return “http” for the value “80”, or even more complex things like the dependencies of the protocols.

- 'simil_func': This is the most important function in the whole system. It is responsible for calculating the similarity of two different values of corresponding fields. For the field 'dest_port' the values “80” (http) and “443” (https) are most likely more similar than the values “80” and “25” (smtp). The similarity function receives the two values and returns the similarity between them as a number between 0 and 1.

- 'exec_func': This function defines another script that is executed when a field is added to the table *fvalues* (described in 6.3.1.3). One difference to the field 'info_func' field is that the function referenced by the 'exec_func' field is able to create additional entries in the *fvalues* table. Another difference is the way the two functions are executed. While the information function is called by a database trigger, the execution function is called by a daemon which periodically checks if there are any functions that need to be executed. Please note that the triggered function should be fast because the database blocks until the function returns. An example for such an execution function is explained for the field 'dest_ip' the function for an IP address. This function can automatically call the tools ping and traceroute and add the results to a new row in the *fvalues* table like 'traceroutable: no' and 'pingable: 0% packet loss'.

- 'verify_func': The field 'values' of the table *fvalues* is a text field because it is unknown what information this field has to store. This is the reason why it is necessary to have the possibility to check the entered values of this field for validity. The field 'verify_func' represents a script that checks a given value for correctness and returns an error message when the input does not fit the needs for the specific field. This function can use external databases or regular functions to check the value for correctness. For the field 'dest_ip' for example the verify function should check if the entered value is a valid IP address. The input “129.132.9.333” for example should be rejected for the field 'dest_ip', because it is not a valid IP address.

6.3.1.3 Table *fvalues*

The table *fvalues* contains the necessary keys for the relation between the cases and the fields (keys), as well as the value (for example "129.132.98.12" for the field 'dest_ip'), additional information as described above in 6.3.1.2, and the field 'script_executed' which is used for the daemon to know if the function represented by the 'exec_func' of the given relation has to be executed or not.

6.3.1.4 Table *categories*

This table holds just the names of the categories together with an optional description. The categories are used to specify multiple parts of the case library in a way that the search for a similar case only has to executed within one category.

6.3.1.5 Table *similar_cases*

The search for similar cases can take some time. Once some similar cases are found, these cases should be linked to the actual case. This makes it possible to browse through the similar cases of a similar case of the actual case. The intention of the table *similar_cases* is to store the found cases together with the information how the cases have been found. The most important fields of the table *similar_cases* are:

- 'case_id': This field is one of the foreign keys for the relation between the actual case and the related similar cases.
- 'relcase_id': The other foreign key for the relation described above.
- 'similarity': This field holds the result of the similarity calculation between the case in the field 'case_id' and the case in the field 'relcase_id'.
- 'similarity_report': To give the user the possibility to understand why the system thinks that two cases are similar, this field should explain the details of the similarity calculation. This should include the weighting used to calculate the similarity

because the weighting table may not represent the state at the time the similarity was calculated.

- 'search_time': This field contains the date and time of the last search for similar cases. It is important because the relation to the similar cases just reflects a state at a specified time.

6.3.1.6 Tables *solutions* and *related_solutions*

These two tables are used to store the solution(s) for the problem described by the case. A solution consists of a title which describes the solution in a few words and a text that contains a detailed description of each step used to solve the problem. As already mentioned the solutions are not calculated on the basis of the case fields. Instead there is a pool of solutions that can be related to a case. That means that one solution can be used for more than one case. Because learning from unsuccessful solutions is also an important issue in case-based reasoning systems, the status field memorizes if the solution was successful or not. The most important fields of the table *solutions* are:

- 'title': A short description of the solution, for example "Restarted the webserver process".

- 'text': This field holds a detailed description of the solution, including every executed step.

- 'status'[5]: This field can hold the following values: "successful", "failed", or "informational". The value "informational" is intended to be used when there is not a real solution described, but more a information such as that the case describes a false alarm[6].

[5]This field should really be in the table *related_solutions*, because the status defines if a solution was successful for a specific case. The same solution may be successful for one case, but may fail for another one.

[6]A false alarm doesn't have a solution. The information that it is a false alarm is enough to 'solve' the problem.

The field 'exec_func' is used to execute some automatic procedures. As an example you can think of the trivial task of restarting a webserver. If the access to the webserver is granted, the script can just execute the procedure to restart the webserver. But the execution function can also prepare the environment for a better debugging like opening some terminals that automatically connect to different hosts.

6.4 Calculation of the similarity between two cases

Calculating the similarity between two cases is the most important issue for this system, because when the system doesn't deliver appropriate results, it will be unusable. One thing to remember is always that it is not important that a case that is dissimilar to another has a low similarity value – as long as a more similar case has a higher value.

The similarity of two cases is calculated in two steps. In a first step the number of matching fields is calculated. This predicates something about the similarity of the problem type. When two cases can be described with the use of the same fields, at least the type of the problem is similar. Only fields that are in the new case are considered for this comparison.

Figure 6.3 shows a small sample case of a connectivity problem. The first column contains the field names, the second and third column holds the current and one old case, while the last three columns displays information about the similarity calculation. The next subsection describes the first step of the similarity calculation in detail.

6.4.1 Similarity of the fields

The first steps to calculate the similarity between two cases is to compare the fields they contain. Because of the very flexible approach in describing the cases, it is needed to look which fields are used to describe a case. The current case in figure 6.3 contains 8 fields, but 2 of them are not shown, because they are not used in the old case. The first calculation is done by dividing the number of fields in both cases by the number of fields

Field	Current Case	Old Case	Similarity	Weight	Total
Destination IP	194.158.230.78	129.132.3.15	0.00	2	0.00
Source IP	217.162.81.90	217.162.81.96	0.75	1	0.75
Problem Description	connection problem	connection problem	1.00	3	3.00
Problem Text	the webserver on $dest_ip is not reachable	port $dest_port on $dest_ip is not reachable	0.79	1	0.79
Pingable	10% packet loss	0% packet loss	0.90	1	0.90
Traceroutable	no	no	1.00	1	1.00
Total				**9**	**6.44**
Similarity of the values				**6.44 / 9**	**71.56 %**
Similarity of the fields (6 of 8)					**75.00 %**
Total similarity between current and old case					**53.67 %**

Figure 6.3: Sample Case

in the current case. Additional fields, that are not in the old case, are not considered for this step of the calculation, because they can falsify the result when additional fields are added to the case, such as contact information or search time. The calculation in figure 6.3 shows that there is a 75% correspondence of the fields (6 of 8 fields of the current case can be found in the old case). The next subsection illustrates the second step in calculating the similarity.

6.4.2 Similarity of the values

This step is the more complex part of calculating the similarity between two cases. Each field can have its own similarity calculation function, as described on page 55. If this function is not defined for a specific field, a general comparison function[7] is used. Every similarity function takes two values as arguments and returns one value between 0 and 1, the similarity of the two values. The function can use other slots in the case to calculate

[7]See [6] for a detailed description of this general comparison function.

the similarity. Lets look at two examples of such a function. The field 'dest_port' for example should return a higher value for the comparison between the values "80" (http) and "443" (https) than for the values "80" (http) and "427" (svrloc). The field 'dest_ip' for example should compare the two values with respect to the netmask (if available) to give IP addresses in the same subnet a higher similarity. IP addresses that are not in the same subnet, nor in the same AS (autonomous system) should get a low similarity value. The calculated value is then weighted by the value provided by the weighting table. The similarity of the values is then calculated as the weighted average of all fields. The next subsection describes the combination of both steps.

6.4.3 Total similarity

The total similarity between two cases is calculated by multiplying the values obtained in the two steps described above.

6.4.4 Variables

It is necessary for this very flexible approach, that variables can be used in the field's values, which has the following advantages:

- Better partitioning of values that don't belong together.
- Similarity calculation is more exact.
- Ability to reuse values and solutions without the need of extensive parsing of strings.

The following example should clarify the use of variables. Suppose that a webserver is not reachable, and the fields 'dest_ip' and 'dest_port' are already defined or will be defined. Some slots of the case looks then like this if there is no use of variables:

```
problem_desc : Port 80 on 129.132.3.11 could not be reached
dest_ip      : 129.132.3.11
dest_port    : 80
```

If variables are used the slots looks like this:

```
problem_desc : Port $dest_port on $dest_ip could not be reached
dest_ip      : 129.132.3.11
dest_port    : 80
```

The probability to find a similar case with the same problem description is much higher if variables are used. It makes it also possible to use popup lists with predefined values, where the user can choose one if a case is created manually. In the user interface the variables will be replaced by its corresponding values for better readability.

6.5 Implementation

This section describes some aspects of the implementation as well as some ideas about the user interface. The user interface is more important in the chosen expert system approach than in other expert systems because the user can affect the systems behaviour in some way. It is desirable that every user describes a specific problem type in a constant manner. This problem is not as delicate when most of the problem fields are entered by the ticketing system or through the scripts mentioned in 6.3.1.2. The user interface should assist the user in entering as homogeneous data as possible. The way the prototype has been built is described in the next subsections. A list of all files used to implement the network troubleshooting expert system can be found in appendix A.

6.5.1 Prototype Overview

The prototype has been built as a web-based solution. Although this may not be the optimal solution, it was chosen mainly because the existing ticketing system at Open Systems AG is also web-based, and the programming for a standalone solution would have taken much more time. What was not considered for building the prototype is an authentication for multiple users and locking functions, that more than one user can work

with one case at the same time. The tools and languages used to build the prototype are:

- PostgreSQL
- Perl
- The Perl Modules:
  - DBI
  - CGI
  - Text::Template
  - Number::Format
  - Net::DNS::Resolver
  - Proc::Daemon
  - POSIX
- C
- Additional UNIX tools:
  - traceroute
  - ping
  - ifconfig
  - whois

The main part of the prototype consists of a PostgreSQL database (as described in 6.3.1) and a perl script that handles all the requests from the webbrowser. This perl script allows the following actions to be executed:

- Creation of a case.
- Search for a case by its ID.

- Adding and deleting fields of a case.
- Start the search for similar cases.
- Look at the details of the similarity calculation.

The handling of the solutions related to a case has not been implemented because it does not affect the similarity calculation. The focus of the prototype was to see if the similarity calculation works as expected, and if the problems can be described with the used case representation.

The user can create a new case by defining a category, the short problem description, and the problem text in a form. After submitting this form, the case layout is shown where the user can add more fields to the case. A sample case layout is shown in appendix B. To add a field, the user has to click on the button '+'. A new window is opened, where the field can be chosen from a popup menu. This menu tells the user also something about other cases, namely how many times a field has been used in another case of the same category. The string "dest_ip (324)" for example means that the field 'dest_ip' has been used in 324 other cases to describe these cases. This information is important because the user can see which field may help best to find a similar case. A field that has never been used to describe a case of the same category doesn't help, whereas a field that is used in many other cases do. After adding all the necessary fields to the case, the user can start the search for similar cases. Until the search is not finished, the webbrowser will reload the case every second to inform the user about the progress. How the search for similar cases has been implemented is described in the next subsection. As soon as one or more similar cases have been found, the user can look at the details of the calculation by moving the mouse over the text "report". To see the detailed layout of one similar case, the user can click on the ID of the similar case. A click on the solution uses then this solution for the actual case and the user has to try out the proposed solution and to define if the solution was successful or not.[8]

[8]This feature has not been implemented as described above.

The information function described on page 54 has been implemented with the help of a trigger on the *fvalues* table in the database. This trigger calls the stored procedure `run_info_func`, which is implemented as follows:

```
DECLARE
    info_script VARCHAR;
    ret TEXT;
BEGIN
    IF NEW.information != '''' THEN
        RAISE NOTICE ''information field not empty ... nothing to do.'';
    ELSE
        info_script := get_info_script(NEW.field_id);
        NEW.information := run_info_script(info_script, NEW.value, NEW.id);
    END IF;
    RETURN NEW;
END
```

The `get_info_script` function gets the path to the information script of the related field, while the `run_info_script` executes this script with the given arguments as parameters and returns the output of the script.

The execution function described on page 55 is used by the daemon 'exec_func_daemon.pl' described below.

The verify function has not been implemented because it does not affect the systems behaviour, but for a productive system the function should be implemented, in particular if several users are using the system.

6.5.2 Components

This subsection describes the component 'search.pl', which is used to search for similar cases, and 'exec_func_daemon.pl', the script used to execute the scripts in the 'exec_func' field of the *fields* table. Both have been realized as a perl script.

6.5.2.1 search.pl

This script is called with one argument, which represents the primary key of the current case. It first checks if the ID of the current case is valid. Then it generates a list with the case ID's of the old cases and the number of fields that are in both cases (the current case and the old case). This list is sorted by the descending number of conformable fields, to find the cases first where the fields match best (not the values). The script then deletes all the similar cases, that are related to the current case, from the *similar_cases* table. It then sets the field 'search_status' from the *cases* table to 1% to give a feedback that the search has started. After this each case in the case library is compared to the current case and the similarity is calculated as described in 6.4. To speed up the calculation, two values that have to be compared are first compared for equality. If they are not equal, the similarity function in the corresponding 'simil_func' field is executed. The script also keeps track of the whole calculation and stores this report in the 'similarity_report' field of the *similar_cases* table. The script uses the DEBUG environment variable, that can be set to a value between 0 and 4 to define how verbose the output of the script should be.[9]

There is one threshold that can affect the behaviour of the script. To speedup the calculation one can define how many fields that have to match. If this value is set to a value greater than 1, the script may be faster, but if the value is set to a to high value, it may not find some similar cases.

6.5.2.2 exec_func_daemon.pl

The 'exec_func_daemon.pl' is responsible to execute the scripts in the 'exec_func' field of the *fields* table. It is executed in a separate background thread because some scripts may take some time to execute, for example such that use traceroute or ping. It loops until it gets a HUP, INT, or TERM signal. For each loop it searches for *fvalues* entries where the 'script_executed' field is set to "f" (false) and executes the execution function for every

[9] This is only useful for debugging in a terminal.

entry. Before calling the external script (exec_func), it sets the 'script_executed' field to "p" (in progress). The user interface then knows that the script is executing until the value is set to "t" (true), which is done by the called script.

6.5.2.3 Example Scripts

In this part some of the implemented scripts are explained. This should give you an idea what one can do with this system.

- **exec_dest_ip.pl:** This script checks if the fields 'pingable' and 'traceroutable' are related to the case. If this is not the case, it executes *ping* and *traceroute* to the destination IP address and adds the results to the corresponding fields. In addition it looks if a destination port is defined and executes *nmap* to test if the port accepts a connection or not.
- **exec_port.pl:** This script executes *nmap* to test if the port is accessible if the destination IP is defined. Otherwise it doesn't execute anything.
- **info_ipaddr.pl:** This script gathers more information about an IP address. It first executes a reverse DNS lookup to find the corresponding hostname of the IP address. Then it connects to the whois database of RIPE to lookup more information about the IP address.
- **info_macaddr.pl:** If a vendor corresponding to the hardware ethernet address can be found, this script returns it.
- **info_port.pl:** This script returns more information about a specific port number, for example that port 22 is used for "ssh: Secure Shell Login".
- **simil_dest_ip:** In the implementation of the prototype this function returns 1 if the ip addresses are equal, 0.75 if only the first 3 parts of the IP address are equal, 0.5 if only the first 2 parts are equal, 0.25 if only the first part is equal and 0 if they are not equal at all. For a real world usage it would be better calculate the similarity with respect to the value in the netmask field, if available.

- **simil_dest_port:** This script arranges the known ports into classes, so that the comparison of two values that are in the same class (for example “80” and “443”) gives a higher value than two values that are not in the same class (for example “22” and “80”).
- **simil_general:** This is a general script that can be used to calculate the similarity between two strings. See [6] for a detailed explanation of this calculation.

6.6 Summary

This chapter explained the concept and the implementation of the network troubleshooting expert system. The used approach, namely case-based, for building the expert system has been justified. The case-based approach then lead to the case representation and the concept for the database where the cases are stored. One of the most important issues in a case-based reasoning system, the similarity calculation between two cases, has been described in detail. As a last section in this chapter the implementation of the prototype has been explained.

The next chapter discusses the tests that have been executed with the prototype, as well as the results obtained. The section ’Further Work’ then gives some ideas how the concept and implementation of this system can be improved.

Chapter 7

Conclusions and Results

This chapter describes the tests that were run with the prototype of the presented system, the conclusions and results of this tests, and some ideas for further work.

7.1 Comparison to existing systems

As mentioned in chapter 4 it was not easy to get detailed information about existing systems. The datasheets of the systems don't tell much about the technologies the companies are using, in particular not much about expert systems they are possibly using. This makes an extensive comparison between existing systems and the system designed in this thesis very hard. Another difficulty in comparing the different systems with the one presented in this thesis is the fact that the case-based approach could not be found in existing systems, except in SpectroRX, which is not available anymore. SpectroRX uses also the case-based approach, but there is no information about how they implemented the case language and how the similarity calculation works.

What makes the system in this thesis differ from others is the following:

- It uses the case-based expert system approach.
- It learns from experience.

- It is very flexible, so new problem types are easy to handle.
- No extensive maintenance for building complex rule bases is needed.
- There is no need to have a lot of past data (as for the model-based approach).
- It can be used in conjunction with existing trouble ticketing systems.
- It uses cost free software.

7.2 Tests

Two different tests have been made with the prototype. While the first test used fictitious data, the second test used real world data which had been extracted from the ticketing system of Open Systems AG. The next two subsections describe the tests in more detail.

7.2.1 Fictitious Data

The test with fictitious data has been made with about 500 cases in the case library in a early development stage. The intention of this test was to get an idea about the speed of the similarity calculation. The cases used the following fields with the according values:

- dest_ip: 20 different IP addresses.
- source_ip: the same IP's as used for the dest_ip field.
- dest_port: 10 different port numbers.
- source_port: the same port numbers as used for the dest_port field.
- dest_mac: 30 different hardware ethernet addresses.
- source_mac: the same mac addresses as used for the field dest_mac.
- traceroutable: either 'yes' or 'no'.
- pingable: A value between 0 and 100 % packet loss.

- problem_desc: One of 10 descriptive problem description like “host xy unreachable”.

- problem_desc: One of 52 characters (each character in upper and lower case).

The cases have been created by a perl script that creates a random number of fields (between 6 and 10) for each case and gets for each field a random value as described above. After the cases have been created, some tests has been made to see how long it takes to calculate the similarity between one case and all other cases. In the test environment (iBook 700MHz with MacOS 10.2.2), the prototype was able to compare about 10 to 20 cases per second. This seems a reasonable number if you consider that:

1. The iBook is not a very fast machine.

2. All the scripts have not been optimized for speed.

3. Real world cases may have only a bounded number of field in one case.

In a productive environment it may be possible to compare about 50 to 100 cases in one second. For a search library with 3000 cases (reasonable size for a company like Open Systems AG), this leads to a search time under one minute, which should not be a problem. The test also showed that there is some value for the similarity that has to be reached that one seems to see a really similar case. For the fictitious tests this value was at about 60%. But please note that this value depends on the number of fields describing a case, on the weighting function, and of course it depends on the person who is using the system. The similarity between two cases is a subjective value that can be different for every person who is using the system. So this value may not be very significant. A thing that is much more interesting is to see if the solution of a similar case could contribute to the solution of the actual case, regardless if the found case(s) seamed very similar or not. But for the test with fictitious data there are no solutions that can be used to evaluate this. The test with real world data should give more information about this.

7.2.2 Real World Data

The intention of the real world data tests was to use existing data from the trouble ticketing system at Open Systems AG to fill the case library, but there were two major issues why an automatic conversion of the data was not possible:

1. Most of the tickets did not contain enough data to describe the problem in a useful way.

2. It was not possible to anonymize the data automatically, so that there was no customer data left.

Because of the above reasons only about 100 cases have been entered manually into the network troubleshooting expert system. About 50 cases concerned security or hardware problems, about 50 concerned connectivity problems.

These test showed that it is possible to handle most of the problems with this case-based system, although it sometimes can be difficult to describe a problem only with key/value pairs. Unfortunately most of the solutions were not available or had not been described n enough detailed to make a proposition about the usability of past solutions as wanted. The threshold where a case looked really similar was raised to about 80%. This was mainly because there were other weightings defined for this test and because of the general similarity calculation function used (not used for the fictitious data tests).

7.3 Conclusions

Although the tests could not really help to see if the implemented system will help to find a useful solution in a past case, there are some conclusions that can be made:

- There is a minimum similarity of 80% needed in the implemented configuration of the weighting in order to consider a case as subjectively similar.

- It would be optimal if there is for each problem type at least one case in the case library. Otherwise the system will work but is most likely not able to find a similar case. Also the adding of fields to a case is easier if there are already some cases of the same problem category available (cp. 6.5.1 on page 63).

- The similarity calculation is the most critical part in the whole system. If the similarity calculation fails or doesn't return the results as expected by the user, the system will be unusable. In order to explain to the user why the system thinks a case is similar to another, it is required that the user is able to look at every detail of the similarity calculation (the similarity report). Another important thing is the use of variables. If variables (described on page 60) are used, the system will work better than without variables.

- The weighting function is also very important and in this context also the classification of network problems in different categories. Because the weighting is defined separately for each category, it is required that different problem types are also in a different category. This is the only way to define meaningful weightings.

- It turned out that it is very possible to begin with an empty case library, but the field structure and the weightings should be defined when the users begin to work with the system. It may not be possible to define every field in advance, but it will help the user to define the problem more in detail if he knows which fields can be used.

As every computer program has its disadvantages, the network troubleshooting expert system has them to. Because a case can only be described by key/value pairs, there is no possibility to describe problems with complex dependencies between the fields, or problems that can not be reduced to a bunch of key/value pairs.

7.4 Further Work

This section presents some ideas for further work to extend the network troubleshooting expert system or to make a more useful proposition about the system's use in a real world environment.

The most important thing is to make extensive tests with real world cases and with different users. These tests should last at least half a year. The period for testing depends of course on the number of cases that occur. The more cases that are in the system, the better the system will behave. It is also thinkable that the weighting is adapted after a while by a script that calculates new weightings on the basis of the old ones and the status of the solutions.

Another extension to the implemented system would be to create a more intelligent case adaption process. For some types of network problems it may be possible to calculate the solution directly with the given fields and values, without the need to search for past cases. This may require other expert system approaches, such as the rule-based or the model-based approach. For very specialized problems these approaches could be successful, but not for every imaginable network problem.

Another useful upgrade of the system would be if the user gets the possibility to search for concrete values in any case fields. This makes it for example possible to search for every issue that affected a specific IP address or a specific company (supposing that there is a field that holds the company information). The system then is not used anymore as an 'intelligent' system, instead it is used as a simple knowledge base where information can be searched like in any normal database.

The weightings used in this implementation are based on the category. Each category has its own weighting. It should be proven if the separation into categories is valuable and if the weighting is separated enough by problem category or if it should be related to the problem description.

It would also be interesting to see if it is helpful when the weighting can be changed

temporarily. This change should not be stored in the database, instead it should be stored on a session basis in the web application. The advantage of this idea is to let the user define which fields are important for a specific case.

For a real usage the implementation should of course take care of user authentication, locking of cases or case fields, and appropriate permissions per user. The permissions may be needed when not everyone is allowed to access every case or every function, such as the changing of the weights should be in the responsibility of one administrator.

7.5 Summary

This chapter described why comparing existing systems to this system is difficult, and the main differences were presented. Two tests were made with the implemented prototype, one with fictitious data and one with real world data. The most important of the conclusions as a result of the test are, that the system seems to work well – at least for the test cases – but for using the system in a real environment one should do more extensive tests. The section about further work then presented some interesting additions to extend the system, such as to calculate the solutions out of the case's field values.

Appendix A

Files

These are all the files used for the implementation:

```
/www/ntes.lan/
|-- bin
|   |-- exec_func_daemon.pl
|   `-- search.pl
|-- docs
|   |-- blank.html
|   |-- css
|   |   `-- design1.css
|   |-- img
|   |   |-- add.gif
|   |   |-- button.psd
|   |   |-- del.gif
|   |   |-- failed.gif
|   |   |-- head2.jpg
|   |   |-- processed.gif
|   |   |-- processing.gif
|   |   |-- reload.gif
|   |   |-- success.gif
|   |   `-- waiting.gif
|   |-- index.cgi
|   |-- index.html
|   `-- libs
|       |-- overlib.js
|       `-- overlib_mini.js
|-- lib
|   `-- case-lib.pl
|-- scripts
|   |-- exec_dest_ip.pl
```

```
|   |-- exec_port.pl
|   |-- info_hostname.pl
|   |-- info_ipaddr.pl
|   |-- info_macaddr.pl
|   |-- info_ping.pl
|   |-- info_port.pl
|   |-- info_traceroute.pl
|   |-- simil_dest_ip
|   |-- simil_dest_port
|   |-- simil_general
|   |-- simil_general.src
|   |   |-- Makefile
|   |   |-- fstrcmp.c
|   |   `-- fstrcmp.h
|   |-- simil_pingable
|   |-- simil_port_info
|   `-- simil_source_ip
|-- setup
|   |-- database
|   |   |-- fill_tables_with_random_data.pl
|   |   |-- insert_categories.dump
|   |   |-- insert_fields.dump
|   |   |-- ntes.setup
|   |   |-- ntes_exec_func.setup
|   |   |-- ntes_info_func.setup
|   |   |-- ntes_triggers.setup
|   |   `-- ntes_triggers_cases.setup
|   `-- setup_database
`-- templates
    |-- addfield.html
    |-- case.html
    |-- error.html
    |-- main.html
    `-- showinfo.html
```

Appendix B

Screenshots

The following screenshots have been taken from the web-based prototype of the network troubleshooting expert systems. The first screenshot shows the main case view, the second one displays the dialog for adding fields and values to a case. The third one shows how the system displays a more detailed information about a field, and the last one shows the report of a similarity calculation between two cases.

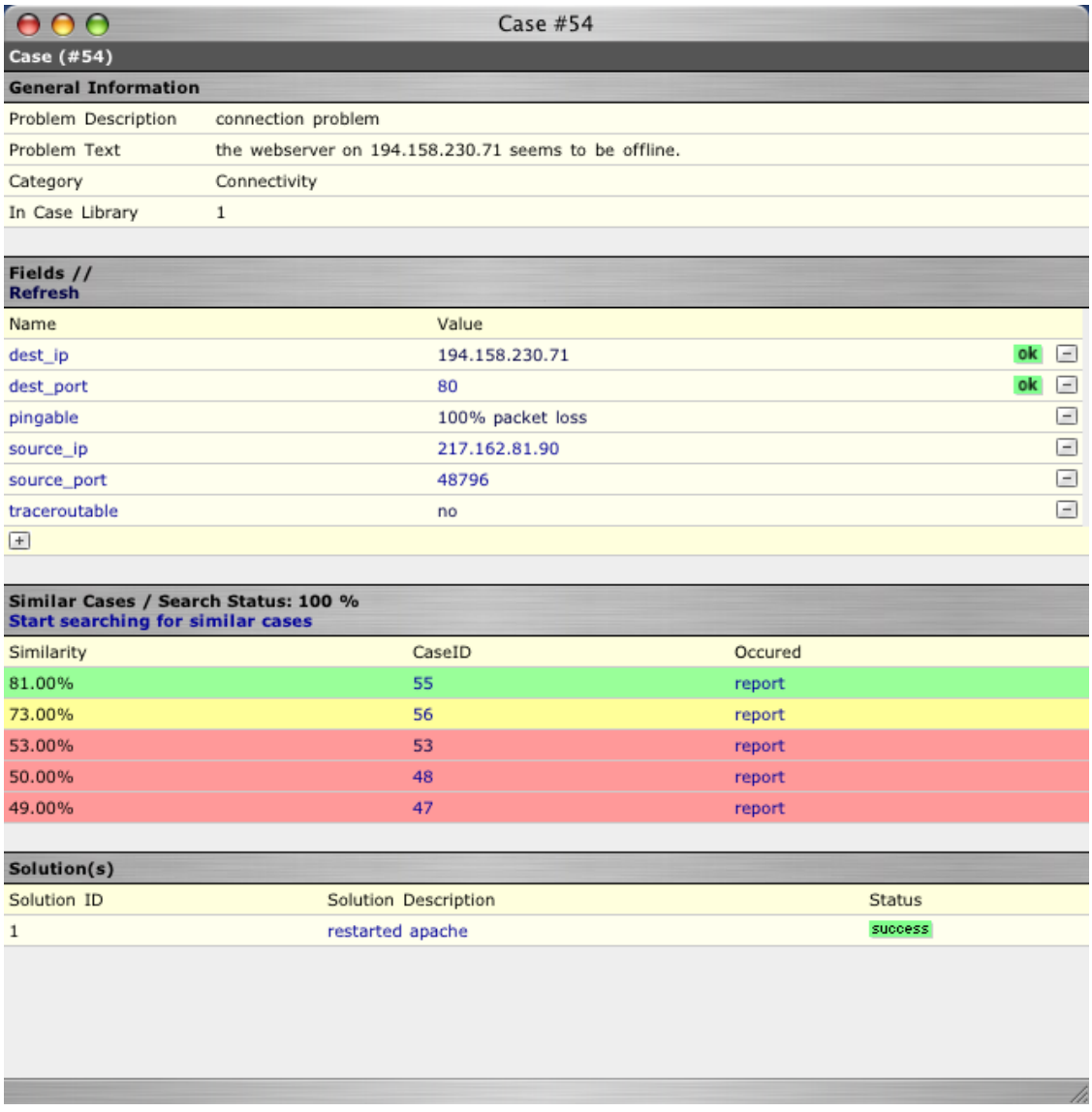

Figure B.1: A Case

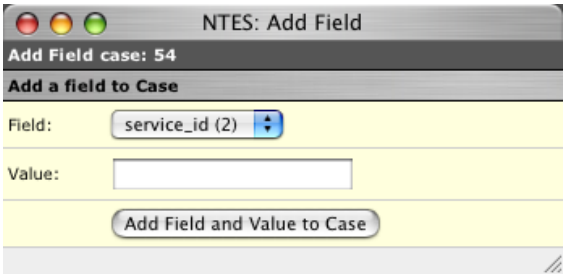

Figure B.2: The Add Field Dialog

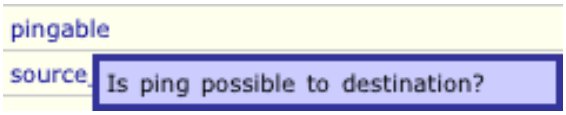

Figure B.3: Detailed Description of a Field

Comparing to case 55

Field	Current	Old	Script	Weight	Similarity	Total
source_ip	217.162.81.90	217.162.81.90		1	1.00	1.00
dest_port	80	631		2	0.00	0.00
source_port	48796	46875		0	0.00	0.00
dest_ip	194.158.230.71	194.158.230.71		2	1.00	2.00
traceroutable	no	no		1	1.00	1.00
problem_desc	connection problem	connection problem		3	1.00	3.00
problem_text	the webserver on $dest_ip seem	the printserver on $dest_ip se		1	0.90	0.90
pingable	100% packet loss	100% packet loss		1	1.00	1.00
Total Values (tot weight: 11)					80.89%	
Total Fields (8 / 8)					100.00%	
Overall					**80.89%**	

Figure B.4: Comparing to an Old Case

Appendix C

Assignment

Institut für
Technische Informatik und
Kommunikationsnetze

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

Communications Systems Research Group

Winter 2002/03

Diploma Thesis

for

Marcel Strittmatter (D-INFK)

Main Reader: Michael Hausding (Open Systems AG)
Alternate Reader: Marc Rennhard

Issue Date: 4th November 2002
Submission Date: 3rd March 2003

Network Troubleshooting Expert System

1 Introduction

Network troubleshooting is a field dominated by manual problem analysis and intervention. Typically, highly specialized engineers deliver 7x24 hours support to keep a network and its devices up and running and react properly in case of failure.

Failure detection on the first hand is a field very well covered as simple scripts are already able to detect malfunction based on specified conditions. Periodic checks can deliver status information about services, connectivity, and performance very easily and thus inform about service level agreement violations.

Typically, incidents or malfunctions are reported to a central data-store where a ticketing system escalates if needed. The ticket models from there on the incident and provides an access point to its history. Analysis, comments, and 3^{rd} party interactions may all be logged together with the ticket.

An expert system would be able to deliver automated analysis based on various status information and provide the engineer with a suggestion of further steps. Some problems may be solved without manual intervention (e.g. the system can deduce that a 3^{rd} party is to be blamed and triggers escalation to this party). A useful expert system should be able to decrease response time, support the engineer, and enhance the problem resolution in most cases.

A possible use-case is the detection of complex ISP failures or routing misconfigurations in the case when only some protocols are routed. If a system is not reachable using a specific protocol, it usually triggers an alert. In most cases, such a loss of connectivity is caused by network reconfigurations and the monitored system is still running. An expert system can suggest connection and network tests that helped to find the problem in the past.

Traditional expert systems are usually constructed to help with systems where the engineer has full control of the whole system and has access to the state of the system at failure time, or at least can

reconstruct the data and reproduce the error. With IP networks, this is not the case. Most problems arise when connections over the Internet are used, and measurements can only take place at the end-points of a connection. Routing in the Internet is very dynamic and already may have changed once debugging starts, which means that reconstruction of the state at the moment of failure is not always possible.

1.1 Typical Incidents

- ISP (Internet Service Provider) problem
- Network partition through device malfunction
- Server software problem
- Performance break down due to link congestion
- Routing problems through either malfunction or malconfiguration

1.2 Typical Information Sources

- Daemon and software monitors on nodes
- Host based intrusion detection agents
- Network connectivity and performance monitors
- SNMP servers

2 Assignment

2.1 Objectives

The objective of this thesis is to study and classify network problems, analyze the potential for an expert system and provide a design and prototype of such a system. This will include a complete description of the data flow (sources, input into the system, processing, output), the strategy (modeling, logic), and the interfaces needed and provided.

The student can rely on having a single access point for status information from the sources mentioned above, which is typically a relational database. His system may also trigger additional checks when following a strategy to gather specific information about the network and the devices attached to it.

2.2 Tasks

The following tasks have to be fulfilled:

- Study the anatomy of a network environment (servers, clients, connections, devices, protocols) and classify the possible failures and malfunctions.
- Study, evaluate and compare existing expert system approaches (rule-based, procedural, object-oriented) and map them to a network environment.
- Propose an expert system handling network failure conditions and in turn proposing automated solutions, problem analysis and general engineer support and define its interfaces.
- Define a data store the expert system can use as a knowledge base and propose mechanisms how this store is updated through external sources (ticketing system, manual interaction) or the expert system itself.

- Create a prototype of your solution
- Evaluate the prototype by replaying events from the existing database and by using real-time events from network operation center.

2.3 Deliverables

- At the end of the second week, a detailed time schedule of the semester thesis must be given and discussed with the advisor.
- At half time of the semester thesis, a short discussion of 15 minutes with Prof. B. Plattner and the advisors will take place. The student has to talk about the major aspects of the ongoing work. At this point, the student should already have a preliminary version of the written report, including a table of contents. This preliminary version should be brought along to the short discussion.
- At the end of the semester thesis, a presentation of 20 minutes must be given during the TIK or the communication systems group meeting. It should give an overview as well as the most important details of the work.
- The final report may be written in English or German. It must contain a summary written in both English and German, the assignment and the time schedule. Its structure should include an introduction, an analysis of related work, and a complete documentation of all used software tools. Two copies of the final report must be delivered to TIK.

3 Literature

- Introduction to Expert Systems
  Peter Jackson, Addison Wesley 1998, ISBN 0201876868
- Expert Systems: Design and Development
  John Durkin, Prentice Hall 1994, ISBN 0023309709
- CLIPS, a tool for building expert systems
  (http://www.ghg.net/clips/CLIPS.html)
- JESS, a rule engine for Java
  (http://herzberg.ca.sandia.gov/jess/)

4th November 2002

Prof. B. Plattner

Appendix D

Time Schedule

Zeitplan Diplomarbeit
Marcel Strittmatter
4. Nov. 2002 – 3. März 2003

	So 1	Mi 1	Mo 1
	Mo 2	Do 2	Di 2
	Di 3	Fr 3	Mi 3
Mo 4 **Einarbeitungsphase**	Mi 4	Sa 4	Do 4
Di 5	Do 5	So 5	Fr 5
Mi 6	Fr 6	Mo 6	Sa 6
Do 7	Sa 7	Di 7	So 7
Fr 8	So 8	Mi 8	Mo 8
Sa 9	Mo 9	Do 9	Di 9
So 10	Di 10	Fr 10	Mi 10
Mo 11	Mi 11	Sa 11	Do 11
Di 12	Do 12	So 12	Fr 12
Mi 13	Fr 13	Mo 13 **Integration OSAG**	Sa 13
Do 14	Sa 14	Di 14	So 14
Fr 15	So 15	Mi 15	Mo 15 **Report and Presentation**
Sa 16	Mo 16 **Knowledge Representation**	Do 16	Di 16
So 17	Di 17	Fr 17	Mi 17
Mo 18	Mi 18	Sa 18	Do 18
Di 19	Do 19	So 19	Fr 19
Mi 20 **Knowledge Acquisition**	Fr 20	Mo 20	Sa 20
Do 21	Sa 21	Di 21	So 21
Fr 22	So 22	Mi 22	Mo 22
Sa 23	Mo 23	Do 23	Di 23
So 24	Di 24	Fr 24 **Design and Test**	Mi 24
Mo 25	Mi 25	Sa 25	Do 25
Di 26	Do 26	So 26	Fr 26
Mi 27	Fr 27	Mo 27	Sa 27
Do 28 **Expert Systems and Tools**	Sa 28	Di 28	So 28
Fr 29	So 29	Do 29	Mo 1
Sa 30	Mo 30	Fr 30	Di 2
	Di 31	Sa 31	Mi 3

Appendix E

Expert System Glossary

Most of these definitions are taken from [2].

Artificial Intelligence

A field of study in computer science that pursues the goal of making a computer reason in a manner similar to humans.

Expert System

A computer program designed to model the problem-solving ability of a human expert.

Expert

An expert is a person who has reasoning abilities in a domain that are superior to others in their profession.

Knowledge Base

Part of an expert system that contains the domain knowledge

Working Memory

Part of an expert system that contains the problem facts that are discovered during the session.

Inference Engine

Processor in an expert system that matches the facts contained in the working memory

with the domain knowledge contained in the knowledge base, to draw conclusions about the problem.

Knowledge Engineering

The process of building an expert system.

Knowledge Acquisition

The process of acquiring, organizing, and studying knowledge.

Knowledge

Understanding of a subject area.

Domain

A well-focused subject area.

Knowledge Representation

The method used to encode knowledge in an expert system's knowledge base.

Rule

A knowledge structure that relates some known information to other information that can be concluded or inferred to be known.

Semantic Network

A method of knowledge representation using a graph made up of nodes and arcs where the nodes represent objects and the arcs the relationships between the objects.

Frame

A data structure for representing stereotypical knowledge of some concept or object.

Reasoning

The process of working with knowledge, facts, and problem solving strategies to draw conclusions.

Inference

The process used in an expert system of deriving new information from known information.

Conflict Resolution

Strategy used for choosing a rule-firing sequence when more than one rule can fire.

Rule-Based Expert System

A computer program that processes problem-specific information contained in the working memory with a set of rules contained in the knowledge base, using an inference engine to infer new information.

Frame-Based System

A computer program that processes problem-specific information contained in the working memory with a set of frames contained in the knowledge base, using an inference engine to infer new information.

Inheritance

Process by which the characteristics of a parent frame are assumed by its child frame.

Facet

Extended knowledge about a frame's property.

Method

A procedure attached to an object, that will be executed whenever requested.

Bibliography

Books and Papers

[1] Peter Jackson.
Introduction to Expert Systems.
Addison-Wesley, Harlow, England, 1998.
ISBN: 0-02-33970-9.

[2] John Durkin.
Expert System Design and Development.
Macmillan Publishing Company, New York, 1994.
ISBN: 0-201-87686-8.

[3] Larry L. Peterson and Bruce S. Davie.
Computer Networks. A Systems Approach.
Morgan Kaufmann, 1999.
ISBN: 1-55860-577-0.

[4] Andrew S. Tanenbaum.
Computer Networks.
Prentice-Hall International, Inc., 1996.
ISBN: 0-13394-248-1

[5] Reasoning, case-based. In Stuart C. Shapiro, editor.
Encyclopedia of Artificial Intelligence, volume 2.
John Wiley and Sons, Inc., New York, second edition, 1992.
ISBN: 0-471-50306-1.

[6] Eugene Myers. In Algorithmica, volume 1.
An O(ND) Difference Algorithm and its Variations.
1986.

[7] Lundy Lewis.
Managing Computer Networks: A case-based reasoning approach.
Artech House, Inc, Boston, 1995.
ISBN: 0-89006-799-6.

[8] Dr. Johannes Steinmueller, 2001.
Expertensysteme.

[9] American National Standard for Information Technology.
Fault Isolation - Information Characterization X3T8-1994, Draft.

Datasheets

[10] Network Associates. Sniffer Distributed.
`http://www.sniffer.com/products/literature/default.asp`

[11] Concord. eHealth – Network.
`http://www.concord.com/download/Brochure_Network.pdf`

[12] Micromuse. The NETCOOL Suite.
`http://www.micromuse.com/downloads/int/de/NETCOOL_DE.pdf`

[13] HP. hp OpenView TeMIP expert.
`http://openview.hp.com/sso/getdoc?doc=/solutions/TeMIP/PB/temip_expert_pb_dec02.pdf`

[14] Computer Associates. Unicenter Network and System Managment.
`http://www3.ca.com/Files/BrochuresAndDescriptions/Unicenter_NSM_Solution_Brochure.pdf`

[15] Navis. Network Fault Managment (NFM) Software.
`http://www.lucent.com/products/solution/0,,CTID+2020-STID+10439-SOID+786-LOCL+1,00.html`

[16] Rocket Software. NetCure.
`http://www.rocketsoftware.com/products/netcure.htm`

User Manuals

[17] Cabletron Systems, Inc, Rochester, 1996.
SpectroRx (SPECTRUM Resolution Expert) Users Guide.

Index