Case-Based Reasoning in a Support system

Johan Nordlund and Henrik Schäfer

April 19, 2006
Master's Thesis in Computing Science, 2*20 credits
Supervisor at CS-UmU: Michael J Minock
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

This report describes a master thesis project done at TietoEnator in Umeå. The main purpose of the project has been to try to create an embryo of a knowledge base that with help of underlying computer communication eases the systems´ users error-reporting to the support department. The knowledge base also supplies an automatical web-based helpsection - based on previous problems - to the users. This report also contains an in-depth study of case-based reasoning and how it could be implemented on the knowledge base system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most office workers, at least in the United States and Europe, spend most of their time at work in front of a computer. This computer is most certainly connected to Internet. Since many of these workers not are experts in computer science, many questions, wonderings and problems arises around their computers. Every bigger enterprise should therefore have a properly working help-desk division that quick and easy can handle all the common problems that the workers in the enterprise runs into when working with their computers. However the systems for error reporting and support is rarely working smooth and this often results in a lot of time consuming telephone calls, even for very simple problems/questions.

Since the users computers holds a lot of systems and instruments for remote information retrieval, a support system that took advantage of these would be very appropriate to develop. Such a system could automatically determine the problem on a specific computer and in the same time use this to build statistics over the most common problem areas for its users.

1.1 TietoEnator

The Company TietoEnator is one of many international architects behind the building of a more effective information society. About fifteen thousands of employees and a yearly turnover of 1.5 billion Euro makes TietoEnator the biggest company in information technology in Norden/Scandinavia.

The company's hallmark is in developing innovative information technology solutions that realizes and digitalizes the customer's visions.

TietoEnator has chosen to focus in the areas where they consider themselves to have the most widespread competence in their field. The most important areas are: banking, foresting and telecom. In these areas, the company has a close cooperation with many of the leading companies and organizations of the world. Together with them TietoEnator is growing and has, while this is being written, activity in more than 20 different countries.

1.2 This Paper

This paper is a part of a Master Thesis project that is performed at the support department at TietoEnator in Umeå. The main purpose of the project is to try to create an embryo of a Knowledge base that with help of underlying computer communication eases the users' error reporting to the support department. The Knowledge base should also offer the users to find solutions to their problems by themselves. This by searching among, by the support, published solutions. Another purpose of the master thesis project is to study Case-based Reasoning (CBR) and examine the possibility to implement this paradigm of problem solving into the Knowledge base.

This paper describes the master thesis in its entirety.

1.3 Background

TietoEnator has for the moment a web based support system that easily described works as follows.

When a problem arises at a client computer the user can send a trouble ticket to the support. The support is then supposed to identify the problem and solve it. To be able to send the ticket the user first has to fill out a quite complicated web form giving the ticket some information about the user and the problem he or she has encountered. An alternative, even more complicated web form is also available. In both cases the ticket is mailed to a database (KIRS). Since the users rarely find themselves in time of filling out any of the web forms it most often ends up with a phone call to the support that in turn has to fill out the web form. The KIRS database holds a large amount of information about different reported problems, finding the information or statistic that could be interesting in it is very complex

A schematic view of the system as it is today can be found on top of the next page.

1.4 Problem Description

The support department at TietoEnator in Umeå is now looking for an application that depending on the client computers software and hardware guides and asks suitable questions to the user of the client. The application shall also store statistics that later on easily is retrieved from an underlying database. The help-section of the system shall only be open for updates by those persons in the support department that has special knowledge in the specific area. The help/solutions from the support shall be stored in a well designed database that is easily updated and read.

The purpose is to create a first version (an embryo) of this help-desk application that will make it easier for the users to find solutions to their problems and that will give the support interesting information.

Some important demands/requirements:

- Scalability, flexibility and modular design.

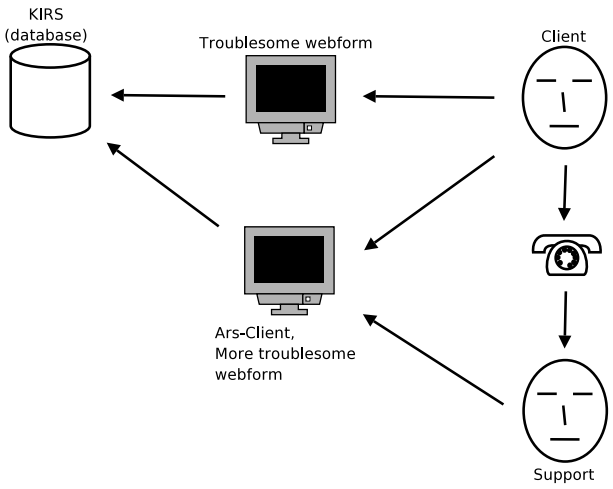

Figure 1.1: The system as it is today.

– Security: Possibility to make information user- or group-specific.

– The information shall be easy to administrate and to publish.

– Statistical features.

– The information regarding what software any client holds shall be fetched from the System Management Server (SMS) database.

– The information regarding the hardware on any client shall be fetched with Windows Management Instrumentation (WMI) queries against the client.

– Continued reporting to the KIRS database.

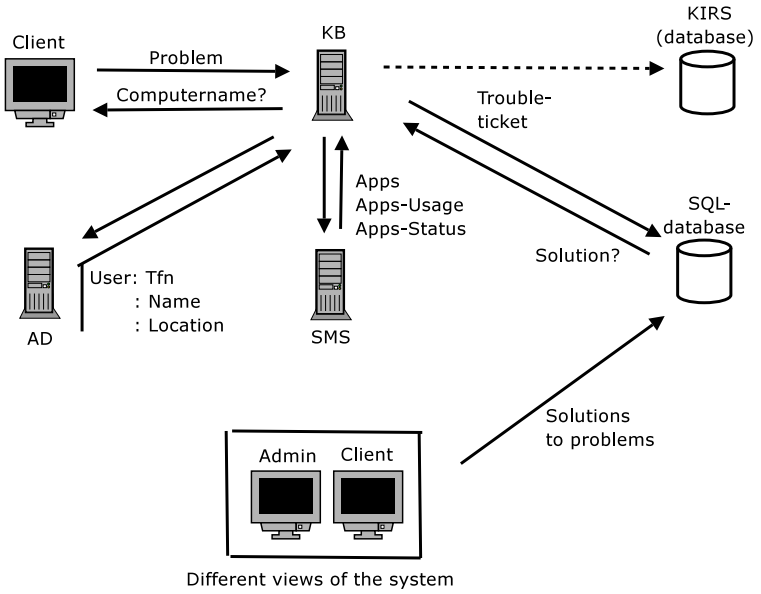

Figure 1.2: The becoming system.

1.5 The Layout of the Report

Chapter 2: Describes the cornerstones in the problem solving paradigm Case-based Reasoning.

Chapter 3: Holds an argumentation concerning why a system with Case-based Reasoning should be used instead of just a Knowledge base system and deals with the question of how Case-based Reasoning could be implemented in the system developed in this project.

Chapter 4: Describes the implementation of the projects system.

Chapter 5: Is about what conclusions that can be drawn from this project.

Chapter 6: Has some acknowledgements.

Chapter 2

Case-based Reasoning

What is Case-based Reasoning?
In the early 90-th the interest and the attention towards a reasoning paradigm/computational problem solving method increased. The method that seemed to solve many of the problems addressed in Knowledge-based (KB) systems was called Case-based Reasoning (CBR). The idea behind it was that it solved new problems by adapting existing successful solutions to similar problems[13]. An important advantage against KB system is that CBR also is an approach to a step-by-step increasing knowledge by learning, this because new experiences is captured every time a problem is solved. The new solution is immediately accessible for future problems[1]. Some features of CBR that address the most familiar problems of KB:s are[1]:

- CBR doesn't need an explicit model of the problem domain. This makes elicitation (the process of extracting expert knowledge about some unknown quantity of interest) only a question of collecting history from earlier cases.

- The implementation can in difference from a KB system be reduced to identify significant features that describes a case, an easier task than creating an explicit model.

- By applying techniques of database handling, large volumes of information can be handled.

- Since the CBR systems can learn by assimilate new knowledge in the form of cases the maintenance of the systems becomes an easier task[13].

The area of CBR has grown fast since the beginning of the 90-th at least regarding the increasing amount of papers in larger conferences, available commercial tools and successfully working applications in everyday usage[1].

This chapter covers the history behind Case-based reasoning an also the techniques behind the method.

[1]The beginning of chapter 3 holds more informaiton about the most severe problems in knowledge-based systems with an explicit model of their problem domain.

2.1 History

The work by Schank and Abelson in 1977 is considered the origin of Case-based Reasoning. They proposed the idea that our general knowledge about different situations is stored as *scripts* that allow us to specify expectations and draw conclusions. Scripts were proposed as a data structure for conceptual memory describing information about stereotypical events like going to a restaurant or visiting a doctor[6]. Experiments on scripts unfortunately showed that they not were complete representations of memory. People often mixed up different event with similar scripts. For example, a person could mix up room-scenarios like visiting a doctor and visiting a dentist.

However, Schank continued to examine what part the memory of earlier situations (cases) and situation patterns played in both problem solving and learning[6]. In the same time Gentner developed a theoretical framework for analogies that also was relevant to CBR. Also, but maybe only with the help of hindsight is it possible to find references of CBR significance in Wittgenstein's observations that *natural concepts* like tables and chairs is polymorph and can not be classified by a set of necessary features, but they can be defined with a set of instances (cases) with family resemblances. This is anyway quoted by Aamodth and Plaza in 1994 as a philosophical basis for CBR [13][Wittgenstein, 53].

While the philosophical roots of CBR can be claimed by many there is no doubt about that it was the work accomplished by Roger Schank's group at Yale University (USA) in the early 80-th that led to both a cognitive model of CBR and to the first CBR-applications based on this model[6]. The one to first develop a working CBR-system was Janet Kolodner. The system that was designed in 1983 at Yale University was called CYRUS after the ex-US Secretary-of-State Cyrus Vance. It was a planning agent that could plan the travels and meetings for Vance. The system had an implementation of Schank's memorymodel[11]. Its case-memory model later served as basis for several CBR-systems like MEDIATOR [Simpson, 85], CHEF [Hammond, 86], PERSUADER [Sycara, 87], CASEY [Koton, 89] och JULIA [Hinrichs, 92][6].

The research of CBR is not concentrated to the US only although it took a few years before the Europeans got started. One of the first works in Europe came from Derek Sleemans group in Aberdeen, Scotland. They studied the usage of cases in knowledge gathering while they developed the system REFINDER [Sharma & Sleeman 88]. Looking even more in the eastern direction there are CBR groups also in Israel, India and Japan. The amount of CBR-articles in Artificial Intelligence publications is however increasing and also is the amount of successful CBR-applications. This can ensure that more countries will increase their interest in CBR in the future.

2.2 Techniques of Case-based Reasoning

A case-based reasoner solves new problems by adapting solutions that were used to solve old problems [Riesbeck & Schank, 89]

If described in a simple way, CBR is a method based on the observation that when humans solve a problem, the solution is often based on a solution that was successful for an earlier problem. Some examples of this phenomenon follows:

- A man is using his bicycle to get to work. When he gets on his bike he does not explicitly plan his route, he takes the route he always takes. If a roadwork is blocking this route he might remember the last time he avoided a similar situation and does the same. However, if he uses at totally new route to avoid the roadwork this will probably be remembered in the future when a similar circumstance pops up.

- After examining a patient in his office a doctor reminds himself of a patient he treated two weeks ago. If the reminder is caused by the fact that the new patient has similar symptoms (not by that the patients have the same hair-color) the doctor will use the same diagnose and examination as on the earlier patient to establish the decease and the treatment for the new patient.

- A financial consultant working with a hard-to-decide credit inquiry uses reminders from earlier cases with companies in somewhat similar situations to decide if to grant or decline the credit.

So far it seems like CBR is a very simple problem solving paradigm that is about matching present problems against earlier that has successful solutions. The process can be widened by for example adapting the solutions for a better match with the needs of the current problem[13].

The main tasks that all Case-based Reasoning applications must handle is to identify the actual problem situation, find a previous case similar to the new one, use that case to create a proposal to a solution to the new problem and finally to update the system by learning from and storing the new experience. How this is done, what part of the process that gets most of the focus, what type of problem that forms the methods and so on varies a lot between different system[2].

In this chapter the different techniques used in CBR-systems are covered.

2.2.1 The CBR-cycle

The process that involves all the states of CBR can be described in a schematic picture (Figure 3). Aamodt and Plaza [1] describes CBR as a cyclic process containing the four RE:s:

1. RETRIEVE. In this phase the actual problem is compared to earlier ones. Similar problems with their solutions (cases) are retrieved out of the memory holding earlier cases.

2. REUSE. The information about the solution from the old cases is extracted and suggested as a proposal to a solution to the new problem.

3. REVISE. The proposed, not working, old solution is worked up/adapted so that it fits and hopefully solves the new problem

4. RETAIN. In the last phase the new problem and its solution (the new case) is added to the memory. In this way the system has improved and learned by its experiences[11].

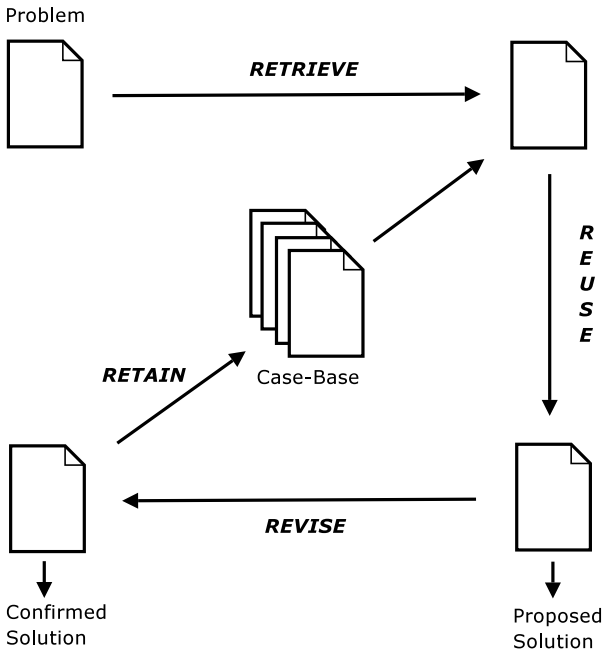

Figure 2.1: The CBR-cycle [1]

Shortly, a new problem is matched against old cases in the case-base and similar cases are retrieved. The solution proposed by the retrieved cases is tested. If not working it is revised and so on a new case is generated and put in the case-base. The cycle above rarely occurs without any human interference. Many CBR tools are primarily serving as case- retrieval and reusing systems. The revision of cases is mostly taken care of by the staff responsible of the case-base. It must not be seen as a weakness of CBR that the method encourages human collaboration when taking decisions[6].

The general knowledge that most often plays a part in the cycle above can vary a lot from very little (or no) help to very strong help, all depending on the specific CBR-application. General knowledge means general problem-domain dependent knowledge as an opposite to specific knowledge encapsulated in each individual case. For example, when diagnosing a patient by retrieving and reusing a case with an earlier patient a model over the anatomy of the human body can together with known effects of pathological treatment constitute the general knowledge used in the CBR system[1].

In this chapter should also be added that in [14] is Watson talking about not the four but the six RE:s. These are then:

1. RETRIEVE
2. REUSE
3. REVISE
4. REVIEW the new problem-solution pair to see if they can be used as a new case.

5. RETAIN
6. REFINE the indexes in the case-base and the feature weights if this is needed.

This gives the CBR-cycle a slightly different look:

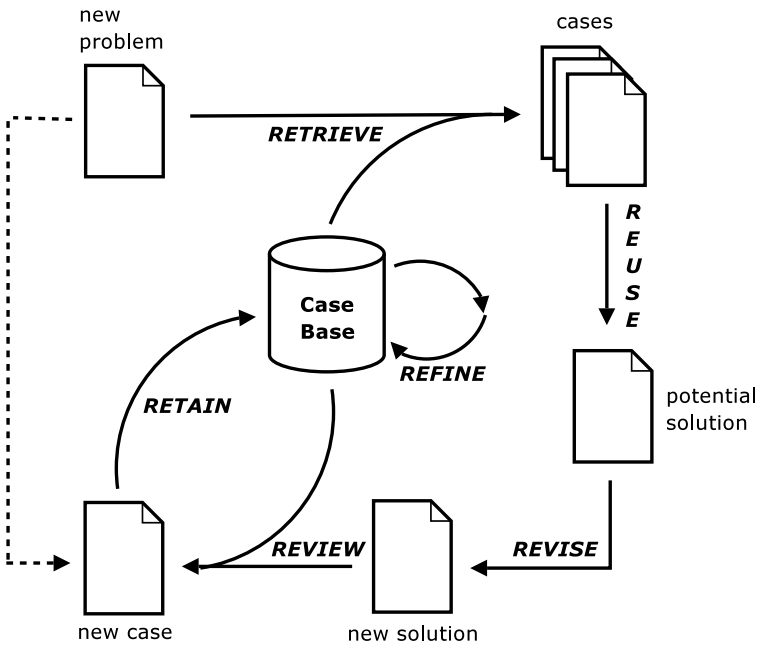

Figure 2.2: A CBR-cycle containing six RE:s [14]

2.2.2 The Case-base

A working CBR-system takes more than just well working methods, knowledge in the form of cases is also needed. This knowledge must in some way be represented. The question of how - the knowledge representation question - is and has always been one of the great problems for developers of CBR systems. A great part of the functionality rely on how the cases are chosen to be represented and organized. The most important part of this problem is the structure of the case-base. There are two models for this. The dynamic memory model and the category and exemplar model[1].

The dynamic memory model.

The most common model of case representation is based on a theory by Shank about the human dynamic memory. The idea of the model is to build a hierarchy of the cases where similar cases are categorized under the same unit, a *Generalized Episode* (GE). A GE consist of three objects: norms, cases and indexes. The norms are constituted by all parts in the cases that are common for all cases in a GE. The indexes consists of a value and a name that separates different objects and features. These can index both individual cases or many cases or GE:s. This model generates a structure somewhat

like a network or a tree where every node is either a GE, an index, a value or a case. When a new case is being inserted into the hierarchy its features is compared with the features of the earlier cases. As the features are matching the case is being pushed down in the structure until it cant go deeper. The case is then indexed on that level. If a new case stops in the same place as an old their common features are extracted and a new GE with these are created as a parent to the two cases. One of the advantages of

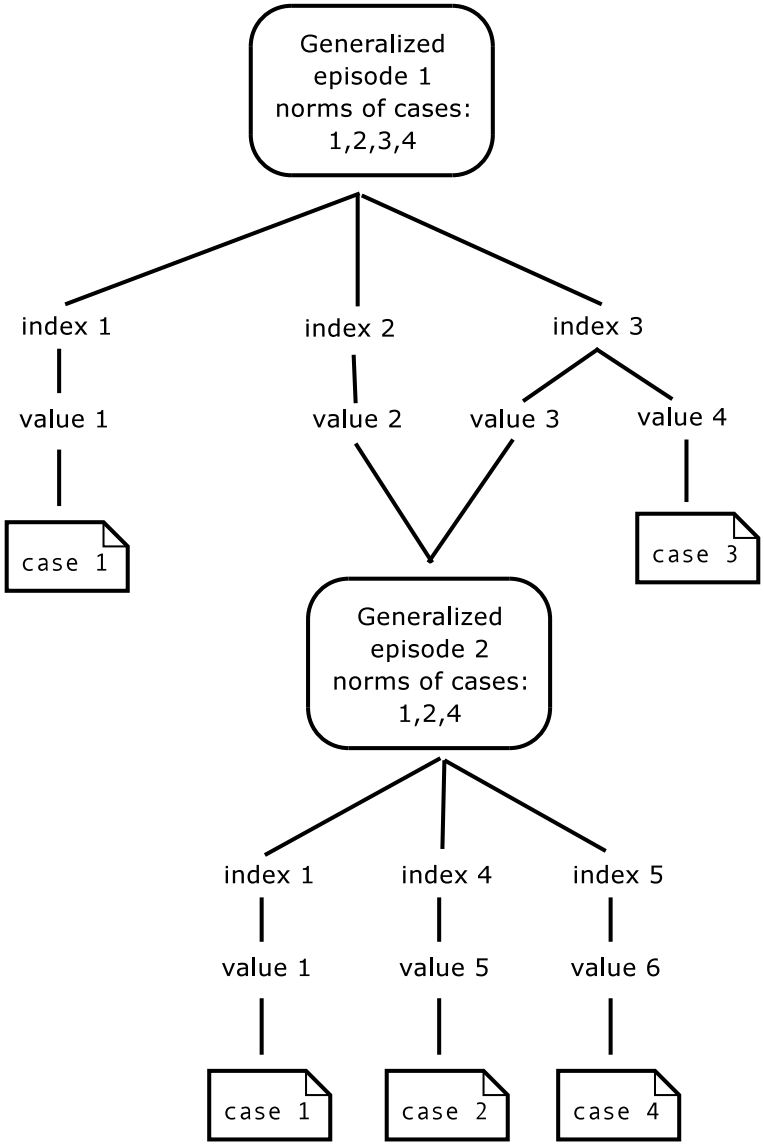

Figure 2.3: A schematic representation of the dynamic memory model.

representing cases in this way is that the search of old cases works pretty much the same as the storing of new cases. Matching is made by queering down through the indexes. For example, an index in a cooking planner can have a name like "Main Course". If

then there is two different Main Courses they each represent a value to the index and the node "Main Course" get two outgoing links. The system has to traverse down in the hierarchy until a GE containing the norms it is looking for is found. When the GE is found the system searches below the indexes existing, now to find the cases that can be useful for creating a solution to the problem.

A disadvantage with this representation is that it sometimes turns out to be a very large amount of indexes to store in the memory. This because several index can reference to the same case. In some systems is therefore a limitation of the amount of index introduced. An other problem is to the decide how the generalization is to be done.

The category and exemplar model.

An other model of case representation avoids generalization. Instead it uses an analogy to the exemplar theory of knowledge representation. The exemplar theory says that a concept is stored as an exemplar and other objects can then be compared with exemplars to see how well they agree. This to categorize under a concept.(Yule 1996).

Also this model has a network structure with three different objects: categories, cases and index. Every case can be associated to a category and index can refer to either a case or a category. Every case is defined by its features that each consists of a name and a value. In each category is then the cases ranked depending on how well their features agree with the category's exemplar features and also depending on which features even exist in the case.

To find a case, the features that the new case holds is searched for. In the category under which the found case belongs is the most suitable cases retrieved. In the same way can a solved case be inserted into the case-base. It must just be sorted in under the category were it fits the most.

Common factors

Both models above is based on a categorization and an indexation of the cases. To achieve these a case could manually be investigated to find its properties and features. Much more elegant though is to feed the system with cases in a given format and let it analyze the structure automatically. In both models it could in some applications be suitable to store larger structures where not only the solution but also a description of the procedure, the methods and the operations that has been used to find it is stored.

2.2.3 Case representation

In the terminology of CBR a case is a description of a problem situation. An earlier experienced situation that has been conquered in a way that it can be used for future solutions to a problem is called a *past case*, *previous case* or a *retained case*[1]. A case is in other words a contextual piece of knowledge that represents an experience. Cases most often consists of:

- *The problem* that more or less describes the state of the world when the case occurred.
- *The solution* which simply is the solution to the problem, and/or

– *The outcome* that describes the state of the world after the case.

Cases that consists of problems and their solutions can be used to construct solutions to new problems. In the same time can cases that hold the problem and the outcome be useful to evaluate new situations. If such a case also holds a solution it can be used to evaluate the result of a proposed solution and thereby prevent eventual problem. Cases can be represented in many different ways by using the full AI-representations formalism containing frames, object, predicates, semantical nets and rules. The frame/object representation is the one for the moment used by most of the CBR software.

There is no standard at the CBR community about what information should be in a case. But there are in fact two, for practical use, measures that can be brought in when to decide what to represent in a case: the functionality of the information and a measure of how easy the information represented in the case can be retrieved.

An example of an implementation.

The implementation of case representation is very separated between different systems. The most convenient way for a user would be to be able to describe the case in free text then let a dialog handler convert it to a correct representation form.

An early system, CASEY, that could diagnose heart deceases represented the cases as lists where every index was represented by a list of two elements, name and value. In another system, JULIA the cooking planner, the cases was represented in a kind of STRIPS[2]-looking frames construction where the name of the index is a field name and the value an object in the field.

case101
(isa: main-course)
(cost: cheap.meal)
(time-to-prep: short)
(ease-of-prep: easy)
(ingredients: (tomato cheese))
(service: buffet)

Searching and matching was done by comparisons between the values in the fields[11].

2.2.4 Indexing

The process of indexing a case is about assigning an index to it, this to facilitate the retrieval of it. There has been many guiding principles proposed by CBR researchers and here are some of them:

– Index shall be predictive

– Index shall indicate the purpose if the case

– Index shall be so abstract that they allow future widened usage of the case-base

[2]STRIPS (Stanford Research Institute Problem Solver) is an automated planner invented by Richard Fikes and Nils Nilsson in 1971[15].

– Index shall be so concrete that they can be recognized in a latter situation.

Both manual and automatic methods have been used for choosing index. Choosing index manually is about determining the purpose of the case from the reasoner's objective and also about to decide under what circumstances the case can be useful. The automatic indexing methods are constantly increasing and consist amongst others of:

– Indexing cases on features and dimensions that can be seen over the entire domain. In this method the domain is analyzed and the dimensions that seem important are calculated. These are put in a checklist and all cases are indexed regarding to their values along these dimensions. For example, CHEF(a case-based planner which can output new recipes given particular ingredients and tastes) indexes on texture and taste. The technique is called *checklist based indexing*.

– *Differential based indexing* chooses index in a way that they separates a case from other cases. During the process the system discovers the features that separates a case from similar and then chooses the most differentiating features as index.

– *Explaining techniques* chooses relevant features for every case. This method analyzes every case to find out which of its features that are predictable. The cases are then indexed on these features.

Janet Kolodner consider that although much success for automatic indexing methods, the human tends so chose index better than any algorithms. Index should therefore be chosen manually in applications that shall be used in practice[6].

2.2.5 Case storage

The storage of the cases is also an important aspect in the design of an effective CBR-system. It must give a conceptual view of what is represented in a case and also consider the index that characterize the cases. The case base should be organized into a easy-to-handle structure that supports effective search- and retrieval methods. There must be a balance between storage methods that retain the case semantics and the methods that eases the retrieval of cases.

The cases must of course also be stored in a physical media. Most systems use a standard relational database but other alternatives like XML-databases can also be useful for this task.

2.2.6 Retrieval

An algorithm for retrieval should, given a description of a problem, use the indexes in the case memory and retrieve the cases that is most similar to the current case. The algorithm must be able to trust the indexes and the structure of the memory to guide the search to potentially useful cases[13].

The problem of choosing the best matching case has been addressed to the research of analogy. A lot of different algorithms has been tested. Some of these are *serialized search*, *hierarchical search* and *simulated parallel search*[3].

[3]These algoritms are not explained in this report but the reader is free to search for more information about them on his own.

Case-based Reasoning will be ready for large scale problems when the retrieval algorithms are effective enough to handle more than a thousand cases. In difference from databases that aim for a special value in a record, the retrieval of case from the case-base must be equipped with a heuristic match. This because there is no case in the case base that is identical with the new case. Among the most known methods for case-retrieval is: *nearest neighbor, induction, knowledge-based induction* and *template retrieval*. All these methods can be used for themselves or combined to hybrid retrieval strategies. The purpose of the retrieval should be considered before the retrieval strategy is decided. If the purpose for example is to retrieve cases that should be adapted or reused, this should be taken into count in the retrieval method[2].

Nearest neighbor.

This retrieval strategy estimates the similarity of stored cases and the input case. The similarity estimation is based on a match of weighted sums of features. The limitation of nearest neighbor includes the problem of converging to the right solution and the retrieval time. Most of the time the usage of this method leads to that the retrieval time increases linearly with the amount of cases in the case-base. This strategy is therefore most effective when the case-base is relatively small. The method has among other systems been used in BRODWAY[Skalk, 92] to choose car models and in Compaqs SMART system [Acorn & Walden, 92], a product help-desk for customers.

A typical algorithm for deciding nearest matching neighbor is the one used in the cognitive system ReMind, reported in Kolodner[93].

$$\frac{\sum_{i=1}^{n} w_i \cdot sim(f_i^I, f_i^R)}{\sum_{i=1}^{n} w_i} \tag{2.1}$$

W is a weight of the feature i:s importance, sim is the similarity function, and f_i^I and f_i^R is the value of feature i for the input case and the retrieved case.

Induction

The induction algorithms' decide what features that best separates cases from each other. After that, a decision tree that organizes the cases is generated. This method is very useful when a single case feature is needed in a solution and when this feature is dependent on other features.

Knowledge-based induction.

In this method cases that are known or thought to affect the primary case feature are manually identified. This applies knowledge to the induction process. Since the explanatory knowledge not always is ready for large case bases the approach is often used in conjunction with other techniques.

Template retrieval

By using queries similar to SQL-queries this method returns all cases that fits some chosen parameters. This technique is often used combined with other techniques as for example nearest neighbor. In such combinations it is often used before some other technique to restrict the search space to a more relevant part of the case-base.

2.2.7 Adaption

After the retrieval of a matching case a CBR system should adapt the solution of the retrieved case to the needs of the present case. Prominent differences between the retrieved and the present case is looked for. These differences are then applied on roles or formulas, this to create/propose a new solution. In general there is two main kinds of adaption in CBR:

- *Structural adaption* where adaption rules are applied directly on the solution stored in a case.
- *Deriving adaption* that by reusing the algorithms, method and rules that generated the original solution, produces a new solution to the present problem. The method demands that the planning sequence that produces the original solution is stored together with the solution. Deriving adaption should only be used on cases that are well understood.

The ideal is that the rules of adaption is strong enough to generate complete solutions from scratch. An effective CBR system sometimes need to use both structural adaption rules for less understood solutions and deriving mechanisms to adapt solutions of well understood cases.

Many techniques, varying from quite simple to more advanced have been used in the CBR adaption stage:

- *Null adaption*, a very easy technique that regardless to the retrieved solution applies this directly to the problem, without any kind of adaption. This technique can be useful on problems that involve complex conclusions with simple solutions. For example: Somebody wants to loan money from a bank. After answering many analyzing questions the final answer is very easy: grant the loan or reject it.
- *Parameter modification*, a technique of adaption that uses structural adaption by comparing specified parameters on the retrieved and present case. Then it modifies the case in an appropriate direction. The technique is among others used in JUDGE[Bain, 86], that recommends a shorter punishment for a criminal when the crime is less violent.
- *Abstraction and respecialisation*, a classical structural adaption technique that is used in a simple way to achieve simple adaptions and in a complex way to generate former unknown solutions.
- *Critic-based adaption.* In this technique critics searches for combinations of features that can cause problems in a solution. It is important that the critic is aware of ways around these problems. The technique is used together with other ones.
- *Re-instantiation* is used to replace features from an old solution with new features. For example can CHEF replace chicken and peas in a Chinese recipe with beef and broccoli and thereby create a new recipe.
- *Derivational replay* is a process that uses a method that derives an old solution or part of a solution to generate a new solution to the new situation. Old blueprints of buildings can be used to find solutions to new design problems.[6]

2.2.8 CBR in a hierarchial structure.

To further describe the top-level steps of Case-based Reasoning, a task oriented view can be used. In such a view each step, or subprocess, can be viewed as task that the CBR reasoner must achieve. The top-level task is *problem solving and learning from experience*, in other words *Case-based Reasoning*. The top-level task is split into the four major CBR tasks. All these four are necessary to perform the top-level task. In the same manner are then the four major ones partitioned. The tree can be built deeper and display more levels of subtask but that will not be done here.

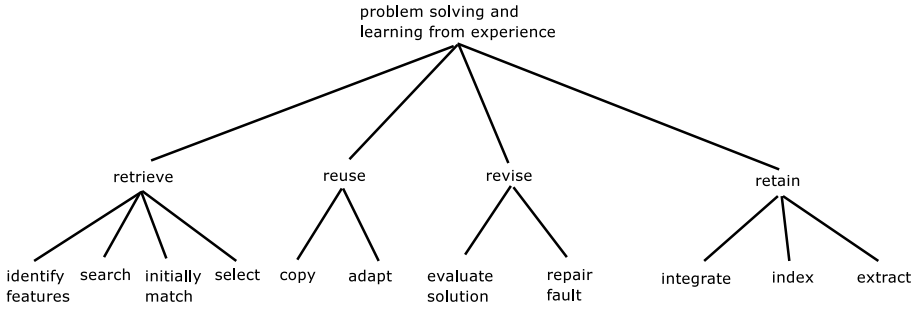

Figure 2.4: A simple model of Aamodt and Plazas task-method decomposition of CBR[1].

Chapter 3

Case-based Reasoning in our KB system.

3.1 Why use Case-based Reasoning?

During the last forty years many Knowledge-based Systems (KBS) that has an explicit model of their problem domain have been developed. In a lot of such systems, the model is implemented with rules and in the latter with objects. There is no doubt that such systems can be successful. There are however five severe problems with the approach.

1. Knowledge elicitation is difficult.
2. Knowledge-based decision support system can take many man-years to develop and can often be very complex.
3. The systems are often very slow and needs expensive specialized hardware.
4. The systems often have got big problems with handling large volumes of information.
5. After the development and installation the systems are both hard and difficult to maintain.

The first problem was discovered in the beginning of the KBS and was often called "*the knowledge elicitation bottleneck*" [Hayes-Roth et al., 83]. All KBS developers are also aware of the second problem that partially is responsible for the increasing interest in the last years development of KBS techniques. The third problem has gradually been defeated by the constantly decreasing costs for processor speed and physical memory. Regarding the fourth problem solutions have been searched through integration of artificial intelligence techniques and database technology. Considering the fifth problem where KBS in many years claimed to be easy to maintain. All early books about KBS development held quotes like "maintenance of a rule-base is easy, it is mostly about adding or removing rules from the KB". Experiences from later years has unfortunately shown that it is not the case. While a knowledge-base is growing it becomes more and more difficult to debug[13].

An other overlooked fundamental problem with KBS is that they not are possibly to

build when a problem model not exists. The early KB-systems did all operate in areas with robust models from principles or statistics under them. Outside the safe walls of the university's many people have to take decision without referring to principles or underlying causal or statistical models. These people solve problems by using their experience. There is no secret that expertise and experience comes hand in hand. The KBS community was most certainly seduced by rules and therefore neglected the matter of course that experts solve their problems by applying their experience on them. The usage of experience in problem solving is the hallmark of CBR. That is why CBR by some has been proposed as a psychological theory on human cognition that provides a cognitive model of how people solve problems.

Since they have many potential advantages to KB systems there is many strong argument for CBR systems:

- CBR systems can be built without passing "*the elicitation bottleneck*" since elicitation becomes the simple task of retrieving old cases.
- CBR systems can be built where no model exists.
- The implementation becomes a not to hard task consisting of identifying relevant case features. The system can be started with only a small case base since this never becomes complete, it grows and improves the system all the time. This eliminates one of the worries from the KB systems, how to know when a KB is complete. These first three points were all clearly demonstrated by the CLAVIER[1] system.
- CBR systems can quickly suggest a solution by avoiding the need to develop solutions via first principles every time. This is important when a decision has to be taken quickly.
- Generalized or individual cases can be used for supplying explanations that are more satisfying than explanations generated by chains of rules, this can be very useful in for example problems in the judicial area.
- By acquiring new cases a CBR system learns by itself and thereby makes maintenance an easier task.
- Finally, by acquiring new temporary cases the CBR systems grows and can in the long run reflect the experience and history of the organization its working in. If a rule-based KB system was delivered to six different organizations and used in six month, all six of them would be identical if no maintenance were done. If the same thing were done with six CBR systems they would all differ since they would have acquired different episodical cases.

Besides the bullet-list above it can be mentioned that the statement claiming that CBR systems are faster to implement than model-based systems is supported by a study made by the organisation Cognitive Systems. They established that it took two weeks to develop a case-base version of a system that took four month to build in a rule-based form[Goodman 89]. Statements like these should however be treated with cautiousness since CBR now is hyped in the same way that KB systems were 25 years ago. It should

[1] CLAVIER: a CBR system that assists in determining efficient loads of composite material parts to be cured in an autoclave.

also not be overlooked that model-based systems can be very effective and is a mature and well understood technique.[13]

3.2 Case-based Reasoning in our KB.

What would it look like if Case-based Reasoning was implemented in the knowledge base in this project developed at TietoEnator. Would the system have to be totally rebuilt or would it be enough to only add some modules and methods? Such system could be built to work totally automatic or with great manual influence on indexing and case matching from the support department. These and other aspects are treated in this section.

3.2.1 Is the system developed only a KB or is it a CBR system?

What is the difference between a Knowledge-base system and a Case-based Reasoning system? Can a CBR system also be a KB and vice versa. From *Case-based reasoning Technology: from Foundations to Application*[4] comes the following quote:

"It has become clear that CBR is a generic methodology for building knowledge-based systems, rather than an isolated technique that is capable of solving only very specific tasks."

The meaning of the quote is apparently that a CBR system is a further developed/ a variation of a Knowledge-based system. The next question is then how the project system should be classified. There is no doubt that it has not filled all the criteria's to be classed as a fully developed CBR system. But as it is developed it is not just a simple KB either.

The method that a user of the system uses when he is making a search down the tree structure of problem groups to find similar problems is clearly a degeneration of one of the CBR cycles main tasks, namely RETRIEVE. Also the support-user does searches of earlier problems and their solutions. This search is not only done to find old solutions that might work but also to get inspiration for new solutions. This retrieval of earlier "cases" is not performed with any automatic methods or by any structured pattern. It is this far simply made with manual brute-force but it still has some obvious tendencies of RETRIEVE and REUSE.

If the system shall fill the whole CBR-cycle the two remaining main tasks REVISE and RETAIN also must be involved into the system. Except for these, of course indexing and retrieval methods must be developed and a well designed case representation must also be involved. The current representation of a problem-report is good start of a case representation but a more detailed representation of the general problems created by the support is needed.

The system is thus more than just a simple Knowledge-base but still only just a part of the way to a complete CBR system. The following parts of this chapter gives proposes to

and discusses ways of upgrading the system to a fully developed Case-based Reasoning system.

3.2.2 Case representation

A, rich of contents, representation of every problem-report is already implemented in the system developed during the project. To manage an upgrade to CBR level a case representation must be created. A case would in the system consist of a generalized problem together with or without a solution. The generalized problem does need some further information to be called a case. This case would then preferably be created manually by the support that could fetch a lot of the information needed from his own knowledge and also from the problem-report that was the origin of the case/the general problem. A case should have a structured and rich-of-content representation with a structure holding number of features. The features should all have exact values except the one called the "wildcard" that should be a short description of the problem. A proposal of which features that should be represented in the case is found below:

Feature	Value
Year	2006-∞
Week	1-53
Weekday	Monday-Sunday
Hour	1-24
Location	All office locations and the alternative "off-site".
Model	All models handed out
Laptop	True or False
Operative system	Windows, Unix or Linux
Applications	All applications distributed by the SMS (represented in tuples)
Problem-group	Anyone of those in the problem-group table in the database
Subnet	Any subnet in the domain
Description	A freely designed description containing formatted text and if needed pictures and links.
Solution	The ID number of a solution or 0 if none exists.

Table 3.1: A proposal of features in a case.

All of the time-fields could of course be represented in one compound feature but in this proposal they are separated since they can be of different interest in some retrieval.

3.2.3 The Case-base

With the previously proposed representation of a case it remains to build a fast and smooth structure for the systems storing of cases. One way to store cases would be to put them all in a matrix with as many dimensions as there are features in the cases. Every dimension would then represent a feature. For example would the dimension of week have 53 different positions for a case. Finding similar cases to a new case would then easy be done by calculating the vector distance between the new case and all previously stored cases and select those with the shortest distance. As the case-base grew, these calculations would become more and more time consuming and the system become slow. Another problem would be that many cases would end up in a small cluster inside the matrix. This because many of the features does not have more than 2-4 possible

values.

It would probably be better to design the case-base with Shanks dynamical memory model (chapter 2) as a template. Using that model, every case would end up in a node in a tree structure and this would make the task of finding close and similar cases a very basic task. Below is a proposal of a start of such a memory model for the system.

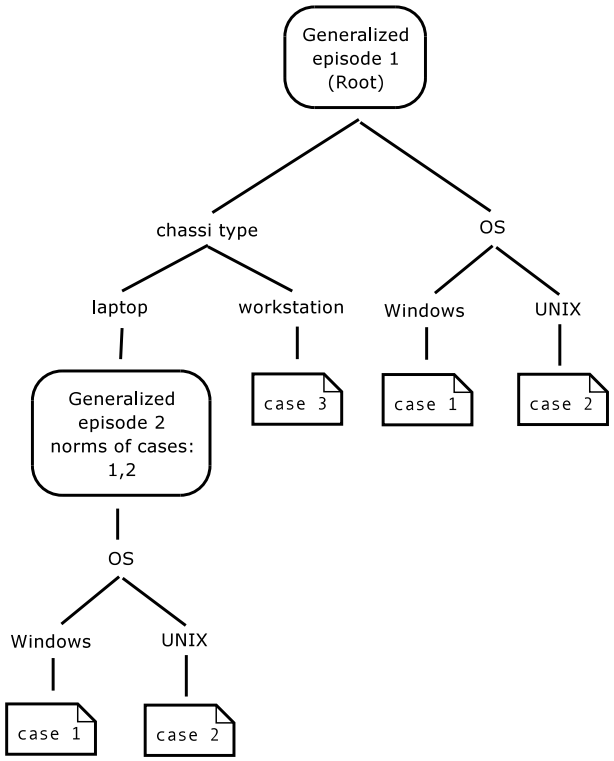

Figure 3.1: Possible start of a dynamic memory.

If this memory model is used and another case with chassi-type, laptop and operational system Unix is to be added a new Generalized Episode must be added under the Unix-node. This tree-structure would grow very fast and another drawback is that it is redundant. That is clearly seen in the example above: case 2 is in two different places in the tree. This effect could partially be opposed with a ranking system for the indexes. With a ranking system, some indexes should be decided to be of the highest rank and therefore have a high position in the tree. The choice of what indexes to rank highest could effect the equilibrium of the tree which is important if a fast memory structure is desired.

A third way of organizing the case-base is to simply set every case as a record in a table of an ordinary database. This would be a very fast and easy implementation but unfortunately it would radically increase the difficulty of finding matching cases.

After this discussion about three different ways of designing the case-base it is clearly seen that the third one is easiest to implement. But since both the first and the third design carries problems with case-matching and case-retrieval it would be wisest to use the second alternative, the dynamic memory model.

3.2.4 Retrieval of cases.

To retrieve matching cases from the dynamic memory model is at the beginning the same procedure as to insert a new case. The tree structure is traversed as far as possible and after this the cases referred to from the present node is retrieved. Depending on the number of cases in the case-base the amount of cases retrieved vary. If to many cases are found, some of them must be filtered. The filtering can be done manually but that is in most systems to time-consuming. Instead it can be appropriate to run a nearest neighbor algorithm (se chapter 2.2.6.1). It is an advantage if the features used in the algorithm has not been used for indexing in the memory structure.

For this system there is also a third interesting alternative for filtering cases. This one, that also most likely is the best one, is about using the free text (the wildcard) that describes a problem. With the help of this text the user of the system can search for specific keywords in a case. The cases that has the least amount of the specified keywords are filtered. If the amount of cases remaining after this filtering is still to big the last alternative for filtering is to do another keyword filtering but this time on the text that belongs to the solution.

When a suitable amount of cases is retrieved it is time to study them manually. It is then a good idea to sort the cases according to their date and study the cases that has the latest date of creation. If then a satisfying case is found it is time to REUSE this case, or at least its solution, to create a solution to the new problem/case.

3.2.5 Adaption of cases

As section 2.2.7 describes there are two main categories of case adaption: Structural adaption and Deriving adaption. In this system none of those with the help of an automatic mechanism, will be able to generate a working solution to a problem. If the cases retrieved from the memory not by themselves can offer a directly working solution it

is the users task to by himself investigate if any of the retrieved cases can give an idea about how to solve the new problem.

Many of the adaption techniques studied in chapter two can probably be useful for adaption in this system. The easiest technique to use called *null adaption* will only work if the new problem is an exact copy of an older one. The *parameter adjustment* technique is not suitable for this system since the parameters in the cases are not very comparable.

The best techniques to use for adaption of cases would be critic-based *adaption*, *reinstantiation* and *derivational replay*. These three combined in a new adaption method or followed by each other could in many situations generate suitable adaption of cases. If only one technique for adaption would be used, the best one for this system would be *derivational replay* since this technique gives the user a lot of free space to work with.

Chapter 4

Accomplishment

TietoEnator has for some time struggled with their existing method for error reports. The forms are many to fill in and most times only partial information is sent about the problem and user to the support. To solve this problem, TietoEnator requested a knowledgebase that could automatically collect all the necessary data and then save all information in a database.

A way to create public solutions and categorize these was also requested, so that redundant help would become less frequent.

4.1 Project Knowledge Base

Before doing anything else, some education about how the computer system at TietoEnator works was necessary. After obtaining the needed information an in depth study about the different parts needed to retrieve all requested information had to be done.

Since no testing on the original TietoEnator system could be done a test environment had to be set up. For this purpose, VMware was installed on two machines, one running as a Windows 2003 Server and the other as an ordinary Windows XP client.

The 2003 server, besides from imitating the TietoEnator Active Directory, also served as MSSQL, IIS and SMS server. This setup allowed extensive testing to be done without compromising the security on the real TietoEnator system.

All parts were then evaluated and tested one by one. What they could be used for, how to use them and what security issues that came with using them.

After evaluating all different parts, modules for each one was created and outlines for tying them all together were discussed.

4.1.1 Tools

This sections describes the different tools used in this project. First, hardware tools are presentet and then software tools.

Hardware tools

Access to two stationary Dell computers were given. Both of them equipped with extra network cards so that direct communication between the two could easily be set up without using TietoEnator's network.

Software tools

1. Both workstations were equipped with TietoEnators corporate baseline software, this included Microsoft Windows XP Professional and Microsoft Office 2003.
2. To generate html-code and code behind pages for Active Server Pages in .NET written in C#, Microsoft Visual Studio .NET 2003 have been used as editor during the whole project.
3. To run the Active Server Pages a webserver have been used. The webserver chosen was Microsofts Internet Information Services (IIS) 6.0 and was running on a Virtual machine with the operating system: Microsoft Server 2003.
4. The database was set up in a Microsoft SQL Server, this as well on the virtual machine running Microsoft Server 2003.
5. To send mail with SMTP a POP3 Service on the Microsoft Server 2003 have been used.
6. To generate a graphical view of what information the LDAP server could supply, Softerras LDAP Browser Version 2.6 was used.
7. File transfers between the server and the workstations have been made over FTP using the client application FileZilla. As FTP server, the FileZilla FTP server have been used.
8. Because of safety reasons, no testing could be done on TietoEnator's actual system during development. Therefore a small copy of their system was created with two workstations each having a virtual machine running on them. For creation of these virtual machines VMWare GSX Server Version 3.2.0 was used.

4.2 Modules and security

4.2.1 Microsoft Windows Management Instrumentation (WMI)[10]

Effective maintenance of personal computers and server systems in an organization benefits from well defined software and hardware that allows components to be supervised and controlled both locally and remotely. For this purpose WMI has been developed.

WMI, earlier called WBEM[1] was started on initiative of a number of large companies as Cisco System Inc., Compaq Computer Corporation, Intel Corporation and Microsoft Corporation. The idea behind the project was an unpatented, industrial standard with a well defined set of specifications not dependent on environment. These specifications would allow management information to be shared between applications on both similar and different operating systems.

[1] Web Based Enterprise Management

Using WMI it's possible to supervise, manage and control a unit using a common interface. Developers can expand the control over the information model so that it contains new hardware and applications by writing modules called "WMI-providers". WMI has an event architecture that makes it possible to identify information changes and compare them with other information. WMI also has a query language (WQL) that makes it possible to ask questions to the model. There's also an API that developers can use to create management applications, for this purpose a number of programming languages can be used like for example, Visual Basic, JScript, C++ and C#.

In a more simple way, WMI can be seen as a database that holds information about all hard and software in a PC. Aside from this, WMI also makes it possible to supervise and control services and hardware. The database is built by so called namespaces that are categorized from hard and software. The namespace that has been of interest in this report is the CIMV2 one that contains information about the computer hardware.

WMI in the project

To make it easier for the user traversing the problem categories there are rules hard coded that examines the client computers hardware configuration and depending on installed hardware it automatically makes choices for the user.

The greatest part WMI plays is gathering information regarding the client computer when the user sends in an error report. When this happens, some hardware information is gathered using WMI and this information is then included in the report. This simplifies for the user when a report is to be sent since the user does not have to enter all this information self. As a bonus the support gets all the information needed and the possibility to extract statistics from the reports regarding what hardware configurations are causing the most failures.

WMI security

There are four main issues when it comes to WMI and security that needs to be addressed. All of these can, if not taken seriously enough, compromise the security of the whole system. Because of this reason a lot of time has been spent on securing the system while using WMI and the most important tasks are presented below.

Impersonation

To authenticate when doing a remote call a method called impersonation is used. This means that a "fake" clients credentials, username and password, is sent instead of the credentials the current user is logged on with. Since the requesting part is a server and the credentials can be used on all computers in the enterprise, using the server credentials is a huge risk and should be avoided. So, instead of using the server credentials, a user with only the necessary rights for retrieving WMI data is used for impersonation doing the remote calls.

Distributed Component Object Model (DCOM)[8]

DCOM is a Microsoft concept and programming interface created to give objects in a client application access to services on a remote computer in a network. DCOM is based

on COM (Component Object Model) which provides an interface that allows clients and servers to communicate locally. DCOM uses RPC (Remote Procedure Calls) for access to the distributed services and is Microsoft's equivalence to CORBA (Common Object Request Broker Architecture).

As WMI runs as a service it also has a DCOM object for remote requests. This DCOM service has to be configured so that the client that makes the request has rights to start and use the service remotely. If this is done only for the user/group that is impersonated instead of the common way, using a local administrator account credentials, the possibility to access WMI remotely is restricted to only this users/group and the harm done if these credentials are obtained by some third part is pretty much none since the only possible thing to do with the client is reading WMI remotely.

WMI namespaces

For the WMI service, access rights can be modified for each part (namespace) in its database. In each of these namespace's a user can have the rights to read, write, execute and call remotely. As the only thing needed for retrieving information about the hardware is reading and remote connection (if writing and execution rights were to be given, the user could, in principle, get full access to the computer) only these rights are granted to the impersonated user. This user is the same user mentioned earlier that has the remote access to the DCOM object associated with WMI.

Since only one user with these rights is needed it is easy to create a security policy for this in the system AD and run that policy on all clients. The only security issue this reveals is the possibility to read data from WMI.

Firewall

All computers running a Microsoft operating system at TietoEnator are using Microsoft Windows own internal firewall. To access WMI remotely this firewall must be opened for remote administration. This means that TCP ports 135 and 444 are open. Since all client computers at TE are already opened for remote administration nothing further needs to be opened in the firewall for remote WMI calls to work properly.

Traffic security

Since all WMI request are remote the traffic needs to be protected from tapping. To enforce this, a method called PacketPrivacy is used for all remote WMI requests. This technique makes all traffic sent between client and server encrypted and authenticated. This way, all communication between client and server is sure to be confidential.

Conclusion

When a remote WMI call is made, these are the steps that it goes through.

1. The client in computer A makes an asynchronous call with impersonated credentials.

2. The call passes through the Windows Firewall and is authenticated in computer B's DCOM module.

3. The request is carried out and answered in the WMI service on computer B if the credentials are ok.

4. The answer is returned to the DCOM module and from there returned to computer A where the answer can be used.

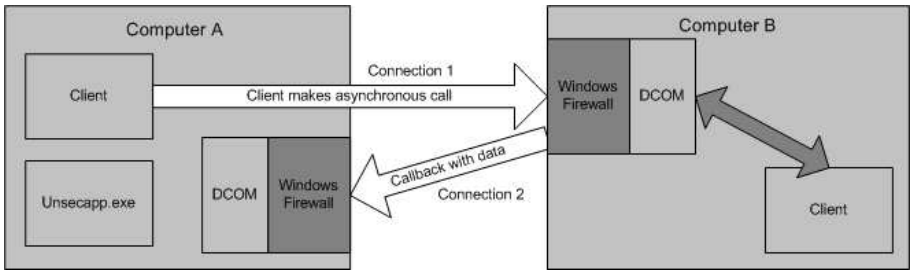

Figure 4.1: Connecting through firewall with WMI[7]

4.2.2 System Management Server (SMS)[9]

SMS offers a solution to remotely administer changes and configuration of windows based platforms. It renders it possible for example organizations to offer software and software updates to their clients using the network.

The main features in SMS are in short as follows:

- Application deployment
  - Software distribution and other maintenance points can be done with specific computers or users as targets. Applications can also be published in the "Add or remove programs" interface on the clients computer.
- Asset Management
  - Summaries or a detailed report that specifies which applications and how much a client uses them can be generated. Software usage can be traced via users and computers and reports can be created that compares usage with licenses. The computers and the software distribution status can also be monitored with reports.
- Security path management
  - Tools for inventorying the system after applicable patches and vulnerabilities are available. Missing security patches identified during individual scans are posted to a central database.
- Mobility
  - When a new software package has been distributed to a client it will stay in the client cache until the package has been executed according to schedule.

- Windows Management Services integration
    - SMS 2003 can automatically discover Active Directory properties of both users and systems. Distribution of software packages can be based on these properties.

4.2.3 SMS database rights

The SMS database is an ordinary MSSQL relational database. Since it contains a lot of information that is sensitive and the SMS service uses for its procedures it is of great importance that the user reading from the SMS database included in this project doesn't have more than read rights.

To further ensure that no vulnerable information is extracted from the database a user with only the right to read certain tables, important to the project, is used. If more tables would be interesting to read, it's simple to extend the read rights to these as well.

4.2.4 Traffic security

Since no inserts or updates needs to be done in the database this traffic is allowed to run with no security or encryption at all. Neither are any transactions used since data is only extracted from and never inserted into the database. If this fails, the call can simply be run again and the database will not need a rollback to any earlier state.

4.3 SupportDB

This database is the foundation of the whole project. All data concerning: from clients sent reports, created problems, created solutions, bindings and revisions of these is stored here. The database runs on a MSSQL 2000 server and handles all requests with stored procedures.

It's a simple relational database in BCNF[2] which contains only a few tables. A structure over the database can be seen in the section concerning the database construction and which stored procedures are used. This section will only deal with information concerning the security in and around this database.

4.3.1 Rights

Since there is need for both reading from and writing to the database it requires a user with complete rights to run all queries. At the moment only one account is used for accessing the database. This could be a security issue, but since all traffic is local, it should be hard to fetch the user credentials.

To make things a bit more secure using two different users for the database access could be a solution. Having one user that makes all the database reads and writes when a client is browsing the knowledgebase and sending in error reports. This users only rights would be reading the problem and solution tables and writing to the report table. This would only make the report table vulnerable if the credentials were acquired by a

[2]BCNF: every determinant is a candidate key.

third part. The other user would have full read and write rights and would only be used under the support side of the knowledgebase.

4.3.2 Stored Procedures

No direct SQL queries are executed towards the database. Instead, stored procedures are used for all calls. The stored procedures are often precompiled which gives some advantage in performance. Since they are strongly typed no injections will be allowed either and attempting such a feature will only lead to a failure in the stored procedure.

In some of the stored procedures several following SQL calls are made, like inserts and reads. These are placed in a transaction that locks the database and prevents so called "dirty reads"[3]. Since transactions inside the stored procedure are used the database will also be able to go back to its previous, untouched state if a call in the stored procedure fails. If one call fails there will be a complete rollback so that all calls carried out earlier in the stored procedure will be made undone and a failure message will be sent back to the user regarding this.

4.3.3 Transactions

In the module used for calls to the stored procedures in the database, transactions are also used. At times, more than one stored procedure is required to run. This means that if one of the stored procedures fails the database can end up containing erroneous information. Because of this, a transaction surrounding the stored procedures is done so that the database can be completely recovered if one fails.

4.3.4 Traffic security

All traffic between the database and the application runs unencrypted. Since the web-server and the MSSQL server most likely will run on the same machine, and so generating only local traffic, no security is needed for the traffic.

4.4 Lightweight Directory Access Protocol (LDAP)[3]

LDAP is an internet protocol that programs, for example email applications, can use to retrieve information from a server.

An email programs often has an address book for the user. In most cases, this address book contains only email addresses the user added by own hand. This has lead to companies like IBM, Lotus and Netscape implementing support for the standardized LDAP. Clients with knowledge about LDAP can ask a LDAP server for different entries holding facts. Through filters in LDAP, the applications can then find a specific person or group in the company and return the wanted information. A search for a person can look something like: "Look for all persons living in Stockholm with the name Lars" and the resulting answer could then be their full name, email address and telephone number.

The protocols also contains more services than only contact information. LDAP can

[3]Dirty read: When a read occurs on a table that has been updated by a not yet commited transaction.

also be used to find encryption certificates, pointers to writers and other network services. It can also offer a point of "single signon" so that the password for one user is shared between many different services.

There are three different levels of LDAP servers. The mid public servers, large servers for organizations like universities and smaller servers for workgroups. Most public servers though has disappeared and the idea of listing persons email addresses in public has been destroyed by all the spam it generates. Aside from that, LDAP continues to be a popular standard for sending data based entries between applications.

Since the LDAP database is public information, no security measures has been taken regarding traffic or authentication.

4.5 Simple Mail Transfer Protocol (SMTP)[12]

SMTP is today's standard for email transactions over the Internet.

SMTP is a relatively easy text based protocol where one or many receivers of a message is specified (and in most cases also verified that they do exist) and then the message text is sent.

It is easy to test a SMTP server by using a telnet application. SMTP uses port 25 to communicate. SMTP started to get used in public in the early 80'ies and was then used as a complement to another protocol (UUCP).

Since SMTP doesn't have any standards for authentication it's a pretty insecure protocol where a sender can easily be faked. This also makes SMTP very useful for sending automated messages where an "authentic" return address can be specified as sender so that possible answers wont just be returned to the server that sent them but instead to the department responsible to handle the issue.

4.5.1 Traffic security

The traffic sent over SMTP is unencrypted. This makes it quite vulnerable for sniffing as everything is sent in clear text. For example a password can easily be subtracted from reading the ip-packages. This however is not an issue since only notifications and no confidential material is sent via email in this project.

4.6 Project Knowledgebase

All earlier security analyzes has only touched the external parts this project uses to retrieve and save information. The project in itself also has some specific points where good security is needed, especially the administrator interface.

The project is mainly based on two pages. One user interface where published solutions can be retrieved, and error reports can be created and sent. On this interface, no immediate security is needed since everyone will have the rights both to read solutions and send error reports. The possible security issue is attempts to make injections into

the database.

The other interface is the administrator part where the support logs in to get information about error reports, create common problem and solutions to these problems. Some statistics can also be retrieved there.

On this interface, there are a number of security issues. To be able to log onto the support interface the user must, apart from authenticating with a username and password, also be a member of the specified support group in the Active Directory. When the support is logged in, there is no difference made between the administrators, all of them have the same rights to insert and change information.

To make it sure that no important information can get lost if someone manages to break in there are no possibilities to remove anything else then reported errors. Problems and solutions created by the support can only be changed and when a change is made, the user doing this change is logged. The reason it works that way is the revision system. All changes made are stored as a revision so that it is possible to step back to an earlier revision if a mistake is done or someone has tried to sabotage the created solutions.

As mentioned earlier, the only thing possible to remove by the support is by clients sent error reports. Since these reports are of no importance for problems and solutions, except for statistic purposes and the fact that they can be generated by mistake, this isn't taken as an immediate threat.

4.7 Implementation

Implementing the whole system was a matter of tying all the stand alone modules above together. An interface for the support, the database storing all data and a user interface had to be produced as well.

A database holding all necessary information was also needed.

4.7.1 The design of the database

The database is the heart of this Knowledge-base system. It stores all the information that the KB collects. Chapter 4.2.1-4.2.7 is all about the database and its design. To this project were a Microsoft SQL Server installed to serve as database server. The administration - creation of tables, creation of stored procedures, creation of dependencies with foreign keys and query testing - of this server has been done with Microsoft Enterprise Manager. Figure 4.2 displays the design of the database. Primary keys has a symbol of a key in front of them and foreign key relations are symbolised with grey lines, ending with a key, running between the tables.

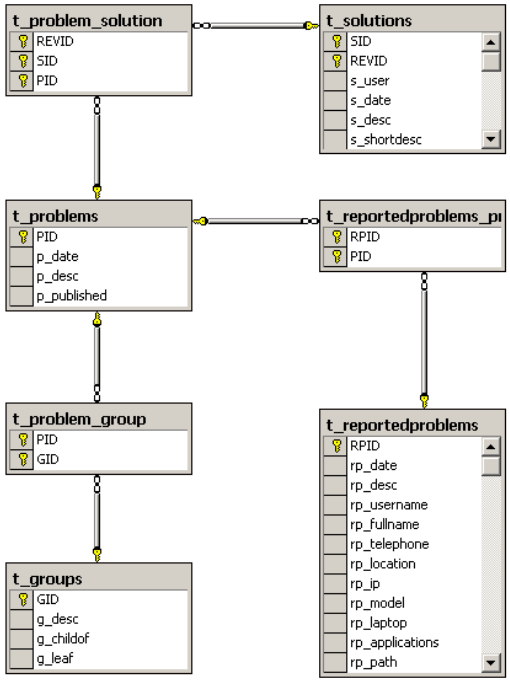

Figure 4.2: The knowledgebase project database

The table t_reportedproblems

This table is the largest one and the one with the most fields in the database. The table stores all the problem-reports that arrives to the Knowledge-base. The primary key of the table is RPID that simply describes the id-number of a report in the table. This number is automatically assigned by the database. The rest of the fields each contain some specific information to every report. An explanation for every field follows here:

- rp_date - the date the report was created
- rp_desc - a short description of the problem that the user has encountered
- rp_username - the username of the user that has created the report
- rp_fullname - the whole personal name of the user
- rp_telephone - the office-telephone-number to the user
- rp_location - the location of the users office
- rp_ip - the ip number of the client from which the user created the report
- rp_model - the brand of the client computer from where the report came
- rp_laptop - indicates by true or false if the client computer that created the report is a laptop

- rp_applications - the software that has been distributed and installed on the client computer via the SMS.
- rp_path - a description of the path that the user took through the KB menu-system before he or she created the report.

4.7.2 The table t_problems

When a user of the support team has viewed a report he may decide to create a general problem on the basis of the report. The general problem that the support user then creates is stored in t_problems. The field PID holds the id-number of the problem and is the primary key of the table. p_date gives the time when the general problem was created and p_desc is a description of the problem that the support user has formulated. The last field, p_published indicates if the problem is published. A published problem is viewable by ordinary users of the KB. A problem can only be published if it has a solution bound to it, also stored in the database.

4.7.3 The table t_reportedproblems_problems

The table binds reports and problems. When a problem is created on the basis of a report both the id-number of the report and the problem is inserted in a record in the table. RPID and PID is foreign keys to the field with the same names in the tables *t_reportedproblems* and *t_problems*. The main purpose of this table is to help the support to keep track of what reports that has been taken care of and to keep track of what general problem that takes care of a certain report.

4.7.4 The table t_solutions

A general problem that a user of the support team has created can have a solution. These solutions are stored in the table *t_solutions*. Every solution in the table is automatically assigned a id-number, SID. Every solution can also have several revisions, therefore every record, is also assigned revision-id REVID. Together these two fields are creating the primary key of the table. The four other fields of the table are: s_user - the user of the support team that created the solution. s_date - the time of the creation of the solution. s_desc - the solution. This field holds text, describing the solution, formatted with an editor. The text can include pictures and web links and so forth. s_shortdesc - a shorter description of the revision of the solution. It should describe how the revision separates itself from the earlier revisions.

4.7.5 The table t_problem_solution

In the same way as *t_reportedproblems_problems* binds reports and problems, this table binds problems and solutions. It links the correct problems to the correct solutions.

4.7.6 The table t_groups

This table describes the problem groups. Every problem that is created by a user from the support team must belong to a problem group. These groups are the structure behind the menu-system in the Knowledge-base. The problem groups are together built up as a tree structure that can look like in figure 4.3 below. If a user of the KB clicks on for

example Browers, the next selection should be some of its children namely: IExplorer, Mozilla or Opera. If any problems created by the support is bound to a problem group, these will then show.

A problem groups id-number is GID that also is the primary key of the table. g_desc is the name of the grop, g_childof can hold a GID to an eventual parent of the group. Finally g_leaf indicates by true or false if a problem group is a leaf, meaning it has no children in the structure.

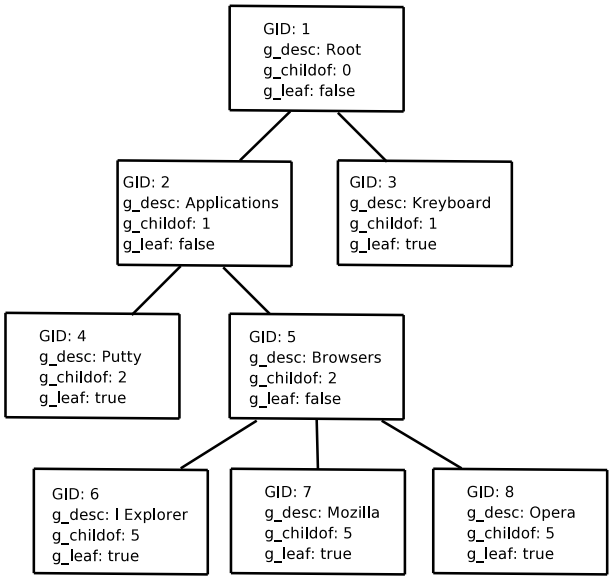

Figure 4.3: Every node in the tree can have problems bound to it. These bindings are inserted in the table t_problem_group and explained in the next section.

4.7.7 The table t_problem_group

Also this table has a similar function as *t_reportedproblems_problems*. It binds problems and problem groups meaning that it controls that all problems are linked to a suitable problem group.

4.8 Results

4.8.1 Support interface

The support interface, like all other parts was developed using ASP.NET and Visual Studio. It is mainly a web interface that presents reports and allows the support to create problems and solutions to these problems. This part retrieves and stores all information in the database described above. The design was not highly prioritized and is mainly constructed only to present data and letting the support design solutions

in a simple way. To design these solutions, an external open source project was used, FCKEditor [1].

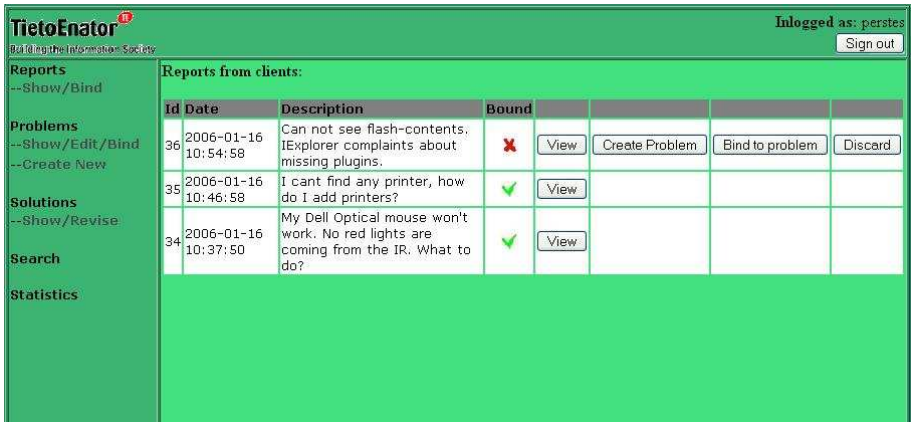

Figure 4.4: A typical view of the support interface.

FCKEditor

To present solutions in a good way, TietoEnator requested the possibility to edit the solution layout in a way similar to a text editor like Microsoft Word. Using such an editor would require documents to be downloaded in order to be modified and then uploaded again. Also, not all users have the possibility to open a document created in Word. The solution was the FCKEditor. It's an open source project mainly done in Javascript that looks like and acts like a normal text editor [5]. The output however is strict xhtml. The editor also handles image and document uploads and links to these. The possibility

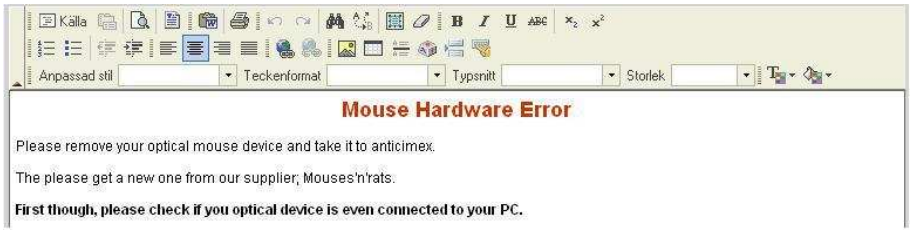

Figure 4.5: FCKEditor

to create own templates and styles are also there. This makes it a powerful tool when it comes to writing solutions with a good layout and design.

User Interface

The user interface is very simple. It show's the problem groups the support has created and links to subgroups or defined problems. If a user cannot find a solution to their

[1] FCKEditor: A javascript text editor, www.fckeditor.net

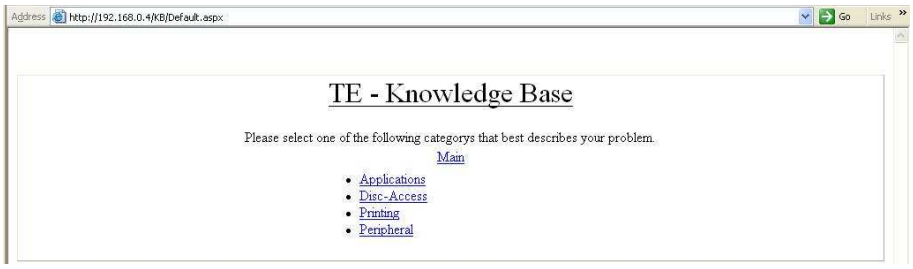

Figure 4.6: Default view for users entering the Knowledge Base

problem, they can send a trouble ticket. The picture below shows when a user has traversed the problem groups Applications and Printing and chosen a specific problem relating to printing. The solution provided to the problem did not help and therefore, a trouble ticket is being written.

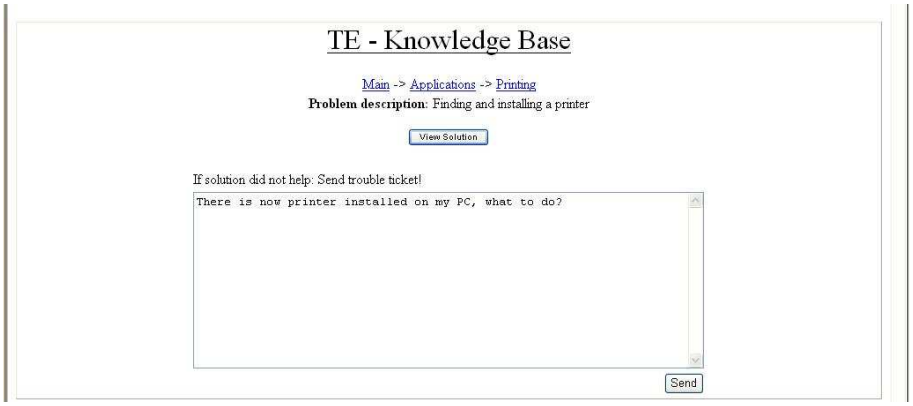

Figure 4.7: The solution to finding and installing a printer did not help so a trouble ticket is created.

The user interface was created to be as simple as possible to follow and understand.

4.8.2 Problems

Lots of small problems were encountered during the work. Most of these were quite easily solved after searching the net and reading books. One of the problems that took the most time to solve was the security issue with WMI.

Debugging the project has been one of the main issues as well. Many times the errors generated by .NET has been hard to understand and difficult to trace to the erroneous code.

4.8.3 Functionality

This chapter describes all that happens in a problem scenario in the Knowledge-base system. Beginning with a client encountering a problem it continues with him creating a report, the support sees the report and creates a problem and a solution to this report and finally publishes them.

A user encounters a problem and reports it

When a user in the domain encounters a problem with his client computer he should open the internal web page that supplies a graphical interface of the KB for the users. If the user after traversing the KB not has found any solution to its problem he sends a problem report. The KB stores this report in a database. To create the report R1

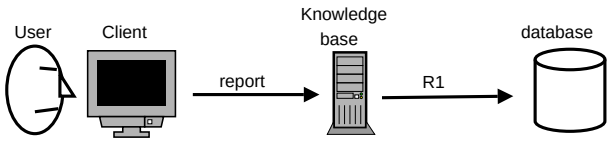

Figure 4.8: A user creates a report.

the KB needs some information. A small description of the problem is received in text directly from the user. Information about the user is fetched by the KB with help of the ldap-module and the ldap-server. With communication via the WMI-module is information about the client computer fetched and information about what software that the SMS has distributed to the client computer is fetched via the smsDB-module. All information fetched is then included in the report R1.

A user of the support team deals with the report

When the support user discovers the report R1 through its view of the KB he has many ways of treating it. If he thinks that there already exist published solutions to the report or that the report describes a problem that the support team is not responsible of solving he can make the decision to delete the report from the KB. In this case is a mail, explaining why, is automatically sent to the user that created the report. Another way for him to treat the report is to create a general problem on the basics of it. The report is then automatically bound to this problem and the user must also bind the problem to a problemgroup. If not it will not be displayed in the right place in the KB:s graphical interface. If the support user also knows a solution to the problem he can create this

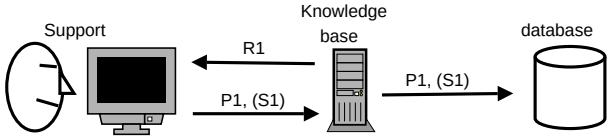

Figure 4.9: A user of the support team deals with the report R1.

solution to it. This solution is then automatically bound to the problem. However, it is also possible to create a solution to the problem at an other time.

A third way to treat the report is to just bind it to an already existing general problem and thereby showing other support users that the report is dealt with.

The problem and its solution are published

When the support feels that the problem and its belonging solution is correctly formulated and well working he can make the choice of publishing them both. A publishing makes the problem visible for all users of the KB. If another user in the future encounters a similar problem with his client computer he can now find the general problem and its solution in the KB. If then the solution is written/created in an understanding way the user will only with the KB:s help be able to solve his problem.

The user that originally created a report that lead to the creation of a general problem and solution does by the time of publishing automatically receive a mail that tells him that there now exists a solution to his problem in the KB.

4.8.4 Restrictions and Limitations

Logic

First off, the project is only developed as a "proof of concept". Only the most vital parts have been implemented and therefore, there are quite a few restrictions. Some of these are listed in the section *Future Work*. Most noticeable is the lack of support for other operating systems than Microsoft Windows. Users of other operating systems can of course traverse the Knowledge-base for facts and help but cannot send a report containing the wanted data about user and machine.

The project has not yet been tested on the TietoEnator enterprise. It has only been tested on a virtual network enterprise simulating the TietoEnator system. Because of this, there might be errors occurring that have been impossible to discover on the test system.

The system should work on most browsers. Support has only been tested on Internet Explorer and Mozilla though so there might be compatibility problems running the administrator application on other browsers.

GUI

The graphical user interface has not been a part of the project in more then a "show data in the simplest possible way" fashion.

If a huge amount of reports and problems exists in the database, it might be hard to overview the information. Since the system has not been tested in the real environment yet, there is no test run on this yet. There should probably be different ways of sorting reports, problems and solutions in a sharp version of the system.

4.8.5 Future work

Since the specification given by TietoEnator requested the application to function on a "proof of concept" basis, only the most fundamental parts has been implemented. For a more robust application there are a lot of features that's possible to implement. Below follows a list over the some of these features and a short explanation why they should be included.

Modules for Unix and Linux interaction is probably the extension with the highest priority. As of now, only client machines running a Microsoft Windows operating system is supported when it comes to gathering information about the client computer automatically.

Since the system is modularized and already is able to find out what operating system that is trying to contact it, modules for information gathering from Unix/Linux machines can easily be added. The only requirement is that they return information in the same form as the already existing modules.

As it is now, the number of problem categories are static, predefined and can not be changed unless changes is made directly in the database. The need for adding and removing these categories will, with most certainty, exist in the future. Because of this, a module where the support themselves can administer which categories should exist should be useful.

Asking the SMS server about software, under this category, there are two different features that could be implemented. Since all client computers has a specific set of applications distributed to them the Knowledge Base shouldn't have to show help for any applications other than those the client has installed. This filtering should exist in a complete version of the system.

When the client traverses a specific programs category the system should examine, via the SMS server, if the application has been properly installed. If it is not a standardized answer regarding this could be given.

In the same way the SMS server is questioned about software, so should the WMI on the client machine be questioned about hardware. Depending on the client machines hardware, different choices could be made traversing the knowledge base without user interaction.

This is already implemented on test and in very small scale. The system retrieves information on whether the client machine is a laptop or a stationary. Depending on the answer the user is forwarded to a category that fits. A stand alone module would be best for this and it is implemented in the KB code right now for testing only.

The way the error reporting currently works at TietoEnator, a error report is sent as an OMT to the helpdesk. A module to send reports from the system developed as an OMT as well would make the developed system possible to use by the current helpdesk also.

Since readers for the PDF format is supported on pretty much every existing platform

and the format itself is designed to produce good looking printouts, offering solutions as a PDF would be nice. At the moment, all solutions are presented as an html-page, this gives ok looking printouts but they can at times, look strange.

Implementing a module that converts between html and pdf would be appreciated and would present the solutions in a better way.

If a problem with solution is published, there is no way for the support to unpublish them so that the problem and solution is no longer available on the client side. Since it is possible to publish by mistake this functionality is quite important and should if possible be implemented before the first version of the system is taken in use.

More statistics would also probably be a good idea. This depends on what kind of statistics the support wants access to. A module for showing statistics as graphs is already implemented but right now it only shows statistics over number of reports per month. Lots of more information would probably be interesting in a future version.

To see if any new reports have been sent by clients the support has to log into the support interface. To avoid this, an rss feed could be generated from new error reports. Using a web browser or rss reader the support could then easily get notifications when new reports arrive.

When a error report is discarded or a solution bound to a report is published the client that sent the report will receive feedback about this. This feedback, however, is completely static and could with beneficiations include a free text part that the support can fill in.

Chapter 5

Conclusions

5.1 About the project

In whole, we are pleased with the result. Since we had no idea what to expect when we started with this Master Thesis, we had no idea how the whole thing would evolve. Of course, there are points where we could have made different choices and where more work could have been done. This section will cover most of our thoughts regarding our choices, the resulting application, what's good, bad and what could have been done differently.

5.2 Initial choices

Since there are windows, unix and linux clients running on TietoEnator and all these should be able to access the knowledge-base we chose to run the application as a internet application. This choice was made mainly because only one application had to be made and all these platforms have access to web browsers.

The request from TietoEnator though, was to make an "embryo" for the KB, not a fully functional solution and since most of the clients runs Microsoft Windows as an operating system the KB would focus on these users as a start. This led to some quite easy decisions regarding development environment, programming language and choice of server.

Because the focus was on Windows platforms, such as Windows XP and Server 2003, and all external sources was windows based servers the choice of languages was narrowed down to only two, ASP and ASP.NET.

After some evaluation on these languages, ASP.NET proved to be the best candidate since it had good support for all external servers the project had to deal with. TietoEnator also had access to the .NET development tool, Visual Studio.NET which even more made ASP.NET the main choice.

Choosing .NET also gave the extra choices of picking a programming language for .NET and these were C# (pronounced c-sharp), J# and VB.NET. Out of these, we had earlier

experience with C# so that became the language of our choice.

As ASP.NET was chosen as programming language [1] the matter of picking a web server was easy. As of now, only IIS [2]fully supports .NET and as TietoEnator runs Windows 2003 Server with IIS built in, there wasn't much of a choice to make.

5.3 The result

As written earlier, the goal of the project was to create an embryo, proof of concept of a knowledge-base system. With that, we and TietoEnator agrees, we have succeeded. Of course, more features could have been implemented but lack of time and the request for a robust system forced us to leave out some parts with lower priority [3].

Then, what could have been made differently?

First off, we could have made the application as a stand alone application instead of a web based one. This although, would have been a bit difficult to accomplish in a good way since there are more then one operating system running at TietoEnator and different distributions would have been needed for all of these. It wouldn't present more options then there is now since the main purpose is presenting information to the users. The security part wouldn't have been much different either since all traffic would have been dealt with in pretty much the same manner.

The greatest advantage in making it as a web application is that is does not care about the users' operating system and most operating system today has web browsers and most users are already well understood in how these works and use them in their everyday work.

Aside from implementing the system as a web or stand alone application there were a few points where things in the system that could have been done differently.

5.3.1 Using the SMS more and differently

There are two different ways of accessing the data the SMS stores about users and applications. First, the SMS is based on a relational database implemented on a MSSQL server. Accessing the database directly and fetching data using T-SQL [4] commands was of course possible. Another way of accessing data from the SMS was using its internal query language called WQL[5] and thereby accessing the SMS-server which in turn translated these questions to T-SQL, questioned the database and returned the answer.

Choosing the method using WQL would probably have made the system less sensitive to changes is the SMS database (could happen in updates to the SMS) but our choice fell on asking the database directly since we felt that asking the SMS would only create extra workload for the server.

[1] C# using .NET to be more exact
[2] Internet Information Server
[3] See section *Future work*
[4] Transact-Structured Query Language
[5] WMI Query Language, known as Extended WQL

Another thing that the SMS could have been used for was accessing hardware and user data. The SMS not only carries information about the applications for a client but also collects data about the client hardware and the users. This information, however, might be inaccurate since it fetches information from the AD-database regarding users and this information was not always as complete as wanted.

5.3.2 Using WMI

As WMI does not only holds information about hardware, but information on pretty much everything on the client it could probably have been used for getting, for example, information about software installed on the clients.

Since the support department only handles support on software distributed by TietoEnator and not software the users installed by themselves, using WMI to gather software information would have generated a lot of extra work filtering the information from non-supported software.

5.4 Does the system work

The functionality of the system is decided by the underlying database. Therefore no help is available for the users directly after installation since the support department will have to create general problems and solutions before the users can take use of them.

The users can of course send in error-reports right away and this is necessary so that the support gets some hints on what to provide help for. The more error-reports that the users generates and the support department takes care of and creates generalized problems and solutions for, the more powerful the system and the greater the knowledge-base will get.

Time will make the system better and better until the point where problems and solutions gets outdated in the same pace as new ones arrives.

There is also the question of how willing the users are to use an automated support system instead of talking to the support directly. More work is needed from the users to find information by themselves then calling the support but if the information is easy to find and explains the solutions in an easy to understand fashion the users will hopefully use the system anyway.

5.5 Good and bad

Since the system has only been tested on a fictional network and with only a few users there will be a need of a large scale test-phase on the main network before the system can be taken in use.

But even though it's not yet tested in a large scale, some things can be said already:

The system will put a lot of demand on the support department. The quality and

strength in the system is mainly based on the supports ability to create good generalized problems that are relevant based on the reported problems. If the solutions provided to these problems are not easy to understand or doesn't arrive in a reasonable time, the confidence for the system will go away and the users will stop using it.

If some larger updates are made to the external parts used, such as: SMS and LDAP, there is a possibility that the system wont longer work as planned. This however is a common problem and small changes in the different modules should be able to solve these problems easily.

As the system is based on a lot of stand alone modules it is easy to add new ones as well. For example support for other operating systems is possible by adding a new module for that. Extended information about users and clients is also possible to implement by adding more columns to the table handling this in the database and adding extra stored procedures to the necessary classes.

The system as it is now does not function as a real CBR system but is rather a mix between a KB and a simple CBR system. It would be possible to make it a real CBR system but would need much more time developing. This would make the system more easy to use for the support though and also hopefully save them even more time.

The KB system is mainly built with TietoEnator in mind and cannot be ported to another company without making minor changes to some of the modules. It is however not cost effective for a smaller company to use a system like this since there is a need for a lot of user input before the system can be of any real use.

Chapter 6

Acknowledgements

All of the persons mentioned here have in some way helped us while we were writing our report and developing our project at TietoEnator.

Bengt Erik Nordlund

Our supervisor at TE, always there to help with and comment our work. His deep understanding in the TietoEnator system sure helped us out a lot.

Lars Eriksson

Also a great help regarding software and ideas during the development. A supervisor at TE as well.

Pär Lindholm

Thanks for all the .Net help!

Operating unit at TietoEnator

Thanks for giving us the opportunity to do our Master Thesis for you and thanks a lot for all the Espresso!

Michael J Minock

Thanks for helping us with the writing of this report.

References

[1] E Plaza A Aamodt. IOS Press, not published.

[2] Agnar Aamodth. Case-based reasoning - an introduction. Technical report, University of Trondheim, Trondheim, not published.

[3] Franz-Stefan Hinner Heinz Johner, Larry Brown, Wolfgang Reis, and Johan Westman. Understanding LDAP. http://www.pdc.kth.se/doc/SP/redbooks/pdfbks/sg244986.pdf.gz (visited 2006-03-06).

[4] Globig Kamp S, Lange C. Case-based resoning technology: from foundations to application. Technical report, not published, Berlin, 1998.

[5] Frederico Caldeira Knabben. FCKeditor - The Text Editor For Internet. http://www.fckeditor.net(visited 2006-02-26).

[6] Ian Watson & Farhe Marir. *Case-Based Reasoning: A Review*. Cambridge University Press, Cambridge, 1994.

[7] MSDN. Connecting Through Windows Firewall. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/connecting_through_windows_firewall.asp(visited 2006-03-06).

[8] MSDN. DCOM Architecture. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp(visited 2006-03-06).

[9] MSDN. System Management Server 2003 SP1 Product Overview. http://www.microsoft.com/smserver/evaluation/overview/default.mspx (visited 2006-03-06).

[10] MSDN. WMI: Introduction to Windows Management Instrumentation. http://www.microsoft.com/whdc/system/pnppwr/wmi/WMI-intro.mspx (visited 2006-03-06).

[11] Henrik Månsson. Case-based reasoning. http://www.ida.liu.se/ HKGBB0/studentpapper-02/henrik-mansson.pdf (visited 2006-01-24), 2002.

[12] Jonathan B. Postel. SIMPLE MAIL TRANSFER PROTOCOL. http://www.glocalnet.se/download/kundservice/abuse/smtp.pdf (visited 2006-03-06).

[13] Ian Watson. An introduction to Case-Base Reasoning. Technical report, University of Salford, not published, 1995.

[14] Ian Watson. Knowledge management and case-based reasoning: a perfect match? not published.

[15] Wikipedia. STRIPS. http://en.wikipedia.org/wiki/STRIPS.

Appendix A

Stored procedures

This section describes the stored procedures that operate in the database of the KB. All procedures that does more than one SQL-call does these inside transactions. At the times when an error occurs in any of the SQL-calls inside a transaction a rollback of all SQL-calls inside that transaction is made. This is made to avoid for example that an id-number in one table is based on a false id-number from an other table. During the time of a transaction, all tables that the calls inside the transaction affects are locked for the transaction during its execution. This means that no other user of the application can change any record in the tables during the transaction. All procedures that fetch information from the database returns this in a table. The other procedures returns a string that describes the result of the procedure. Following stored procedures is in the database:

Name: BindProblemReport
Argument: RPID (numeric), PID (numeric)
Creates a binding between a report and a problem by creating a record in t_reportedproblems_problems with the values in RPID and PID.

Name: BindSolutionProblem
Argument: SID (numeric), PID (numeric)
Creates a binding between a problem and a solution by creating a record in t_problem_solution with the values in PID and SID.

Name: CheckProblemBinding
Argument: PID (numeric)
Checks if a problem is bound to a solution by returning the amount of records is the table t_problem_solution that contains PID.

Name: CheckReportBinding
Argument: RPID (numeric)
Checks if a report is bound to a problem by returning the amount of records in the table t_reportedproblems_problems that contain RPID.

Name: DeleteBindProblem
Argument: RPID (numeric), PID (numeric)

Removes a binding between a report and a problem by deleting the record in t_reportedproblems_problems that contains both RPID and PID.

Name: DeleteBindSolution
Argument: PID (numeric), SID (numeric)
Removes a binding between a problem and a solution by deleting the record in t_problem_solution that contains both PID and SID.

Name: DeleteReport
Argument: RPID (numeric)
Removes a report from the table t_reportedproblems by deleting the record that contains id RPID. Since RPID is a foreign key in t_reportedproblems_problems all records in that table with the same RPID will also automatically be deleted.

Name: DeleteSolution
Argument: SID (numeric)
Removes a solution from the table t_solutions by deleting all records that contain id SID. Since SID is a foreign key in t_problem_solution all records in the table with the same SID will also automatically be deleted.

Name: DeleteSolutionRev
Argument: SID (numeric), REVID (numeric)
Removes a revision of a solution from the table t_solutions by deleting the record that contains both SID and REVID.

Name: InsertBindProblem
Argument: p_date (datetime), p_desc (varchar(5000)), GID (numeric), RPID (numeric)
Adds a new problem in t_problems with date p_date and description p_desc. The problem-id that is assigned to the problem is then in the same transaction used as in BindProblemReport to bind the problem to the report RPID and for binding the problem to the problem-group GID.

Name: InsertProblem
Argument: p_date (datetime), p_desc (varchar(5000)), GID (numeric)
Does exactly the same as InsertBindProblem except for not binding the problem to any report.

Name: InsertReport
Argument: rp_date (datetime), rp_desc (varchar(2000)),
rp_username (varchar(50)), rp_fullname (varchar(50)),
rp_telephone (varchar(50)), rp_location (varchar(50)),
rp_ip (varchar(50)), rp_model (varchar(50)),
rp_laptop (bit), rp_applications (varchar(3000)),
rp_path (varchar(1000)),rp_email (varchar(200))
Adds a report in t_problemreports with date rp_date and description rp_desc and many more features that is naturally explained by the name of the arguments.

Name: InsertSolution

Argument: s_user (varchar(50)), s_date (datetime), s_desc (varchar(5000)), s_shortdesc (varchar(1000)), PID (numeric)
Adds a new solution/solution-revision in t_solutions with the features; user s_user, date s_date, description s_desc, revision-description s_shortdesc and binds this solution to the problem with id PID.

Name: InsertSolutionRev
Argument: SID (numeric), s_user (varchar(50)), s_date (datetime), s_desc (varchar(5000)), s_shortdesc (varchar(1000))
Adds in the same way as InsertSolution a new revision of the solution with id SID. The revision-id is automatically fetched.

Name: PublishProblem
Argument: PID (numeric)
Publishes the problem with id PID by setting p_published to 1, meaning true, in the record containing PID.

Name: SelectAllGroupAncestors
Argument: input (numeric)
Gets all the ancestors to a group. An ancestor is a group that in the tree structure is straight above the chosen group. To be able to get these ancestor a recursive method building a table with all ancestors is used. That table is then returned.

Name: SelectAllGroupLeafs
Gets all the groups in t_groups that is at the bottom of the tree structure, in other words all the groups that are leafs.

Name: SelectAllProblems
Gets all the problems in the table t_problems.

Name: SelectAllReports
Gets all the reports in the table t_problemreports.

Name: SelectAllBoundProblems
Gets all the problems in t_problems that is bound to a solution, meaning that they have a solution. Those problems that has a solution has their PID both in t_problems and t_problem_solution.

Name: SelectChildsOfGroup
Argument: GID (numeric)
Gets all child in first generation to the problem-group with id GID.

Name: SelectEmailForProblem
Argument: PID (numeric)
By joining the tables t_reportedproblems and t_reportedproblems_problems all email addresses to the users that created the reports bound to the problem PID is fetched.

Name: SelectGroupForProblem
Argument: PID (numeric)

Gets the id of the group that the problem with id PID is bound to.

Name: SelectGroupIsLeaf
Argument: GID (numeric)
Checks if the group with id GID is leaf, this by fetching its field g_leaf.

Name: SelectLatestProblmeID
Gets the id-number of the latest created problem in the table t_problems.

Name: SelectLatestReports
Gets the 15 latest created reports created in t_reportedproblems.

Name: SelectProblem
Argument: PID (numeric)
Gets the problem with id PID from the table t_problem.

Name: SelectProblemsInGroup
Argument: GID (numeric)
Gets all the problems from t_problems that is bound to the problem-group with id GID and that is published.

Name: SelectReportedProblem
Argument: RPID (numeric)
Gets the problem in t_reportedproblems that has id RPID

Name: SelectRootProblemGroup
Gets all the problem-groups in t_problem-groups that does not have any parent, in other word the root-groups.

Name: SelectSolution
Argument: SID
Gets the solution in table t_solutions that has id SID.

Name: SelectSolutionRev
Argument: SID (numeric), REVID (numeric)
Gets a specific revision of the solution SID, the one with revision-id REVID.

Name: SelectSolutions
Gets all solutions from t_solutions and sorts these regarding to their id.

Name: SelectSolutionToProblem
Argument: PID (numeric)
Gets the latest revision of the solution that the problem with id PID is bound to.

Name: StatsReportsPerMonth
Creates from the contents in t_reportedproblems a table that gives a view of how many report that has been created in every month the latest year. This table is returned by the procedure.

Name: StatsReportsPerMonthYear
Argument: year (numeric)
Does the exact same thing as the procedure above except for creating the table for the chosen year: year.

Name: UpdateBoundProblemSolution
Argument: SID (numeric), PID (numeric)
Changes the binding for the problem with id PID. The new solution is being bound to has id SID.

Name: UpdateProblem
Argument: PID (numeric), p_date (datetime), p_desc (varchar(5000)
Updates the problem with id PID. The problem gets a new description, p_desc and a new date, p_date.

Appendix B

Modules

The Knowledge-base system is using five different modules that each deals with the communication to a specific server.

B.1 ldapDB

This module contains only one class named ldap. The class runs the communication between the knowledge-base and a ldap-server. Its main purpose is in other words to render possibilities of information retrieval about a user in the Active Directory. The class has the following methods.

private string getData(string usrName, string data)
The method creates a connection to the ldap database and retrieves the information described in the string data concerning the user usrName. The information is returned as a string.

public string getFullName(string usrName)
This method uses the method getData to retrieve the full name of the user usrName. The name is returned as a string.

public string getTelephone(string usrName)
This method also uses getData to retrieve information about the user usrName. The information is this time the users telephone number. The number is returned as a string.

public string getLocation(string usrName)
Another method like the two above. The information returned as a string is this time the geografical location of the user usrName:s office.

public string getEmail(string usrName)
The last method in the class and the last to use getData. This method uses it to retrieve the user usrName:s email-address. The address is returned as string.

B.2 smsDB

This module runs the communication between the KB and the database of the System Management Server (SMS). It makes retrieval of information about all client computers advertised programs possible. The status of this program can also be fetched. The module has two classes: smsDB and smsQueries. The first one takes care of the connection creation, connection closing and communication against the database. The other one uses the first to execute its SQL-queries against the database.

B.2.1 smsDB

The class has the following methods.

private bool connect()
The method tries to create a connection to the sms-database and it returns a boolean, true or false depending of the connection result.

private void disconnect()
The method closes an eventual connection to the sms-database. The method does not return anything.

public DataSet select(string SQLString)
The string SQLString contains a SQL-select statement. The method executes this statement against the database. The table retrieved from the database is inserted into a DataSet that is returned.

public bool isConnected
A get-method that returns the status of the connection as a boolean.

public string errMessage
A get-method that returns a string eventually containing an error-message of the class.

B.2.2 smsQueries

The class has the following methods.

public string errMessage
Get-method that returns a string containing an error-message retrieved from smsDB.

public DataSet selectAllOurPrograms()
Fetches the names of all software programs that is in the SMS. The table containing these is returned in a DataSet.

public DataSet selectSomething(string client)
The method fetches name and status of all software programs that is advertised to the

client computer with the name that the string client holds. The table with the information is returned in a DataSet.

public DataSet selectAllPrograms(string client)
The names of all programs advertised to the client computer client is fetched in this method. The table with these names is returned in a DataSet.

B.3 wmiDB

This module runs the communication between the KB and a client computer. The communication is done with the help of the Wmi Query Language (WQL) that is one of many variations of SQL. The module has only one class that is called wmiDB that has the following functions.

public bool isOnline
Get-method that with a boolean returns the status of the connection between the KB and the client computer.

public string operatingSystem
Get-method that returns a string that describes the operative system of the client computer.

public string errMessage
Get-method that returns a string that hold an eventual error-message that is the class has.

public bool isLocalConnection
Get-method that returns a boolean that tells whether the connection is local or not.

private ManagementScope connect(string cl)
This method creates a ManagementScope, that eases the communication, to the client c1 This Scope is returned by the method.

public string getOS()
Method that returns the name of the client computers operative system.

public string getNIC()
This method returns the name of the client computers network adapter in a string.

public string getSoundCard()
Method that in a string returns the name of the sound-card in the client computer.

public string getCDROM()
The name of the client computers CD-rom is returned in this method. The name is returned in a string.

public string getHHD()
This method returns a string containing the name of the hard-drive in the client com-

puter.

public string getMonitor()
Method that in a string returns the name of the monitor attached to the client computer.

public string getMonitorType()
This method returns a string that contains the type of the monitor attached to the client computer.

public string getKeyboard()
Method that in a string returns the name of the keyboard of the client computer.

public string getUser()
This method returns a string. The string holds the name of the user that is logged in on the client computer.

public string getMouse()
The name of the mouse on the client computer is returned in this method. It is returned in a string.

public string getCPU()
The method returns a string with that name of the processor in the client computer.

public string getModel()
Method that in a string returns both the model and version of the client computer.

public string getPcName()
Method that in a string returns the name of the client computer.

public string getChassiType()
This method finds the chassi-type (laptop, desktop, tower etc) of the client computer. The type is returned as a string.

B.4 sendMail

This module handles the communication by mail. The protocol used is the Simple Mail Transfer Protocol (SMTP). The module has only got one class, sendMail, that has the following methods:

public string errMessage
Get-method that returns a string with an eventual error-message of the class.

public bool send(string mTo, string mSubject, string mBody)
This method builds up and sends a mail via a mail-server to the address mTo. The subject of the mail is in the argument string mSubject and the main text of the mail is in mBody. The method returns a Boolean that indicates if the sending was successful.

B.5 SupportDB

This module takes care of the communication with the database where the KB stores all reports, problems and solutions. The module is built up by two classes: supportDB that is responsible of the basic communication and procedures that has a specific method for every stored procedure in the database. These methods are executed through the communication-methods in the class supportDB.

B.5.1 supportDB

The class has the following methods:

public bool isConnected
Get-method that returns a boolean that indicates if there is a connection to the database.

public string errMessage
This get-method returns a string that can contain an eventual error-message from the class.

private bool connect()
This method creates a connection to the database. The connection is to be used by other methods for reading and writing from/to the database. The method returns a boolean indicating if the creation of the connection was a success.

private void disconnect()
This method closes the connection to the database.

private bool dbQuery(string spName, string[] parameters)
The method runs a stored procedure in a transaction. The name of the procedure comes in the argument string spName and any eventual parameters to the procedure is in strings stored in the string-array parameters. The method returns a boolean: true for a successful transaction and false at a failure.

private bool dbQuery(string[] spName, string[][] parameters)
This method is an overload of the one above. The difference between them is that this one runs several stored procedures in the same transaction. The name of the procedures comes in the string-array spName and their arguments in the two-dimensional array parameters.

public DataSet select(string spName)
The method runs a stored procedure inside a transaction. The procedure name is in spName. The table retrieved from the procedure is returned in a DataSet.

public DataSet select(string spName, string[] parameters)
Overloaded method of the one above. This version takes one more argument, parameters that is a string-array with argument to the procedure.

public bool update(string spName, string[] parameters)
Method that through dbQuery runs a stored procedure that updates a table in the

database. The name of the procedure is in the string spName and the arguments to it is in parameters. The method returns the boolean retrieved from dbQuery.

public bool update(string[] spName, string[][] parameters)
Overload of the method above. The difference is that this one runs several updating stored procedures. The names of those is in the array spName. The arguments to these procedures comes in the two-dimensional string-array parameters.

public bool insert(string spName, string[] parameters)
public bool insert(string[] spName, string[][] parameters)
Two methods that are exactly as the two above. The difference is that these two runs inserting stored procedures.

public bool delete(string spName, string[] parameters)
public bool delete(string[] spName, string[][6] parameters)
And here is two more methods of the same kind. These two runs stored procedures that deletes data in the database.

B.5.2 Procedures

In this section is "stored procedures" only written as "procedures". The class has the following functions:

public bool checkReportBinding(int RPID)
Method that by running the procedure CheckReportBinding checks if a report with id-number RPID is bound to a problem. The answer is returned as a boolean; true = bound, false = unbound.

public DataSet selectAllReports()
This method runs the procedure SelectAllReports and then receives a table of all reports in the database. This table is returned in a DataSet.

public DataSet selectLatestReports()
The method runs the procedure SelectLatestReports and receives a table with the 15 latest inserted reports. The table is returned in a DataSet.

public DataSet selectReportedproblem(int RPID)
In this method is the procedure SelectReportedProblem runs. It retrieves a specific report from the database. The argument RPID tells what report to retrieve. The report is returned in a DataSet.

public string insertReport(string desc, string usrname, string fullname, string tel, string location, string ip, string model, bool laptop, string applications, string path, string email)
In this method is the procedure InsertReport used to create a new report in the database. The arguments all but one comes as a string.
desc - A short description of the problem.

usrname - The username of the user that created the report.
fullname - The whole name of the user that created the report.
tel - The office telephone number to the user that created the report.
location - The location of the office were the report were sent from.
ip - The ip-number to the client computer that sent the report.
model - The model of the client computer t that sent the report.
laptop(boolean) - true for laptop client computer, false for not laptop
applications - The software programs that the SMS has distributed to the client computer.
path - The path the user that created the report went in the KB
email - The mail address to the user that created the report.
The method returns a string with the string, indicating the result of the insert, received from the database.

public string deleteReport(int RPID)
The method deletes the report with id RPID from the database. The string indicating the result is returned.

public bool checkProblemBinding(int PID)
Check if the problem with id RPID is bound to any solution, meaning: has a solution. Returns true if it has a solution and false if not.

public DataSet selectProblem(int PID)
The method retrieves the problem with id PID. It is returned in a table in a DataSet.

public DataSet selectBoundProblems(int SID)
Retrieves the problems that is bound to the solution with id-number SID. The table with these are returned in a DataSet.

public DataSet selectAllProblems()
This method retrieves all the problems that is in the database. This are returned in a table inside a DataSet.

public string bindProblemReport(int RPID, int PID)
Binds a report with id RPID to the problem with id PID and returns a string with the result that came from the database.

public string insertBindProblem(string p_desc, int RPID, int GID)
Inserts a new problem into the database and binds this to a report and a problem-group. The string p_desc is a description of the problem, RPID is the id of the report and GID is the id of the problem-group. The method returns the string that it gets from the database (indicating the result).

public string insertProblem(string p_desc, int GID)
This method does the same as the one above except for that it does not bind the problem to any report.

public string updateProblem(int PID, string p_desc)
The method updates a problem with a the new problem description p_desc. The argu-

ment PID denotes the id of the problem that shall be updated. The resulting string from the database is returned.

public DataSet selectSolutionToProblem(int PID)
The method retrieves the solution in the database with id PID. The solution is returned in a table in a DataSet.

public int selectGroupForProblem(int PID)
Retrieves the group that the problem with id PID is bound to. The id of the group is returned as an integer.

public string publishProblem(int PID)
The method publishes the problem that has id PID. The resulting string retrieved from the database is returned.

public DataSet selectEmailForProblem(int PID) Gets the email-addresses belonging to the reports bound to the problem with id PID. The address is returned in a DataSet.

public DataSet selectSolutions()
Gets all existing solutions in the Database. A table with these is returned in a DataSet.

public DataSet selectSolution(int SID)
This method retrieves all revisions of the solution with id SID. These are returned in a table in a DataSet.

public DataSet selectSolutionRev(int SID, int REVID)
The method gets the revision with id REVID of the solution with id SID. The revision is returned in a table in a DataSet.

public string insertSolutionRev(int SID, string s_user,string s_desc, string s_shortdesc)
This method creates a new revision to a solution in the database. SID is the id-number of the solution, s_user the user that created the revision, s_desc the solution in formatted text and s_shortdesc is a shorter description of the revision. The method returns the resulting string from the database.

public string insertSolution(string s_user, string s_desc, string s_shortdesc, int PID)
Inserts a new solution in the database. The arguments is like in the method above except for PID that denotes the id of the problem that this solution solves. The string from the database that indicates the result of the insertion is returned.

public string bindSolutionProblem(int SID, int PID)
This method binds the problem with id PID to the solution with id SID. The result-string from the database is returned.

public string updateBoundProblemSolution(int SID, int PID)
The method changes a binding for the problem with id PID. The solution that the prob-

lem is being bind to has id SID. The result-string from the database is returned.

public DataSet selectRootProblemGroups()
This method retrieves the problem-groups that is in top of the problem-group hierarchy. These groups are returned in a table in a DataSet.

public DataSet selectChildsOfGroup(int GID)
Retrieves the problem-groups that has the group with id GID as parent. These groups are returned in a table in a DataSet.

public bool selectGroupIsLeaf(int GID)
The method check if the problem-group with id PID is a leaf. Returns a boolean: true for leaf and false for not leaf.

public DataSet selectProblemsInGroup(int GID)
In this method is the problems bound to the problem-group with id GID retrieved. These are returned in a table in a DataSet.

public DataSet selectAllGroupLeafs()
The method gets all the problem-groups that are leafs. A table of these is returned in a DataSet.

public DataSet selectAllGroupAncestors(int GID)
This method gets all problem-groups that is above the problem-group GID. A table with these is returned in a DataSet.

public DataSet StatReportsPerMonth()
Retrieves statistics from the database. A table describing incoming reports per month is returned in a DataSet.

public DataSet StatReportsPerMonth(int year)
This method does the same as the method above except for that the statistics is for the year year and not just the last year.