A community based expert system

van de Laar, J.G.A.

Award date:
2009

A community based expert system

Graduation paper by:	Johnny van de Laar 0583727
Faculty :	Wiskunde & Informatica Computer Science and Engineering
Supervisor:	Marc Voorhoeve
Date:	2008/12/14

A community based expert system

Graduation paper

Technische Universiteit Eindhoven - fac. Wiskunde en Informatica - dep. Computer Science and Engineering

Written by: Johnny van de Laar
Student number: 0583727
Supervisor: Marc Voorhoeve

A community based expert system

Graduation paper written by Johnny van de Laar – [j.vandelaar@byelex.nl]

During his graduation project from 1st June 2007 – 31st January 2008

Document version 1.0 - last modified: 02/01/2009

Special thanks to:

Herman Vissia and my colleagues at Byelex Multimedia B.V.

Marc Voorhoeve and all my teachers of the Technische Universiteit Eindhoven

faculty Mathematics and Computer Science

Sergey Gafurov of the Belarusian State University

Byelex
The Minsk Silicon Valley Company

TU/e technische universiteit eindhoven

Belarusian State University

PREFACE

"A community based expert system project" is the subject of my graduation project for my Masters study in Computer Science and Engineering at the Technische Universiteit Eindhoven. The project was conducted in the Architecture of Information Systems group within the Mathematics and Computer Science faculty and supervised by dr. Voorhoeve.

The practical project has taken place at Byelex Multimedia B.V. in Oud Gastel, from 1st of June 2007 till 31st of January 2008. I have been an employee of Byelex during the project and my colleagues have supported me by giving advice and helping me with setting up the graphical layout of the product. The director of Byelex Multimedia B.V., Herman Vissia, has been my supervisor within Byelex. During the project a number of research studies have been made and a product has been developed. This document will cover the studies and the products that have been made.

Byelex has developed expert systems for many years in cooperation with the Belarusian State University. During the project professors of the Belarusian State University have provided source code, support and documentation of the expert system that they developed in cooperation with Byelex.

TABLE OF CONTENTS

INTRODUCTION

Computers and the internet have an increasing importance for many areas of expertise. Computers become more powerful and more easy to use. With easier ways to use computers like graphical user interfaces, multimedia and the internet, people from all kinds of fields are able to use computers. Computers increase their level of productivity of people when they have the correct tools, they make their work and life easier and decrease the amount of errors that they make.

But computers still rely heavily on the knowledge of the user itself. Someone can make a medical encyclopaedia with descriptions of all diseases. But when you are ill the medical encyclopaedia will not tell you what sickness you have. You have to know your way around the encyclopaedia and have to know yourself where to look, you have to be an expert to make the system work for you.

Logo of the @dvisor program

This project is about automated advice systems; these systems are called **expert systems**. Expert systems can assist a user, without much experience of the field of the expert system by asking questions and based on the answers of the users the expert system calculates an advice. During this project I have modified an existing expert system built by Byelex Multimedia B.V. This expert system, called @dvisor, originally consisted of two parts. A tool for knowledge engineers such that they can build up a knowledge base (the knowledge inside an expert system) and the actual expert system that uses the knowledge base for giving advice.

Building a new knowledge base consists of two phases. A first phase where the knowledge engineer builds up the knowledge base. This knowledge engineer is an experienced knowledge base builder and does not necessarily have to be an expert in the field of the expert system. Usually the knowledge engineer will gather information about the field by interviewing experts.

The second phase consists of verification of the knowledge base by experts in the field for which the expert system is made. The knowledge base creation tool, called the Knowledge Designer, is developed for the first phase. The Knowledge Designer is made for knowledge engineers and therefore usually too difficult for people unfamiliar with knowledge base designing. Therefore the tool cannot be used in the second phase by these experts.

Before this project, experts were not able to verify the knowledge base by just using the @dvisor system. Verification of a knowledge base was solved by making a tree containing the knowledge inside the knowledge base. This tree was made by hand by the knowledge engineer, which was a time consuming practice. The tree contained the questions, answers to these questions and the advices coupled to these answers.

The solution for this problem, developed during this project, is a new system. This new system provides a, completely new, web-based interface to the knowledge base. This web-based system can be used by multiple people at the same time. By this, it becomes possible for multiple people to verify a knowledge base at the same time. Especially in the field of medical expert systems there is a need for verification by several experts before the system can be used in hospitals, etc. The title of this project, “a community based expert system”, is based on this group of experts (a community) that together are able to verify a knowledge base.

In a typical situation the system operates as follows: an expert logs in to the web-based system and gets a personal copy of the knowledge base. Within his personal copy the expert is able to modify the knowledge in the knowledge base, making additions and/or correcting mistakes. Along with these modifications, the expert can enter feedback messages. When all experts have finished reviewing their personal copy of the knowledge base, the knowledge engineer gathers all knowledge bases. Based on all verified knowledge bases and the feedback messages, the knowledge engineer is able to make a new and modified version of the knowledge base. By iterating, a consensus is reached, resulting in a correct knowledge base that has been verified by a community of experts.

Because the experts can fix mistakes themselves, it becomes easier to verify the knowledge base. This also introduces the possibility for a knowledge base engineer to set up the general structure of the knowledge base by using the Knowledge Designer tool. Followed by the experts who finish the knowledge base with the actual knowledge, by using the web-based tool.

When the system is placed on the internet it would even be possible to make it accessible for a large community, making it possible for people from all over the world to build up the knowledge base. This would make the system similar to web 2.0 community systems like Wikipedia.

Wikipedia is an online encyclopedia

This document starts with a description of expert systems in general and the @dvisor system in more detail. The second chapter describes the technical designs of the @dvisor system and the during this project developed web-based system. The third chapter describes the testing period of the project and the fourth chapter describes the final result of the project. This final result includes screenshots and a description of how the system operates. The final chapter contains the conclusion and future improvements for the system.

1. EXPERT SYSTEMS

This chapter describes what expert systems are, a short history about expert systems, how expert systems work and in more detail the technology behind the @dvisor expert system.

DEFINITIONS

Expert systems are used to transform the knowledge of an expert into an advice for a user of the expert system. This advice is based on the answers that the user gives to a questionnaire.

For a better understanding of this system the following terms will first be described in more detail:

1. Knowledge
2. Expert
3. Advice

KNOWLEDGE

There exist a multitude of definitions of the term "knowledge" but there does not exist a single definition which is agreed correct. The Oxford Dictionary of English (2nd edition revised) definition describes knowledge as follows:

Knowledge
→ noun [mass noun]

1. *facts, information, and skills acquired through experience or education; the theoretical or practical understanding of a subject: a thirst for knowledge | her considerable knowledge of antiques.*
    - *the sum of what is known: the transmission of knowledge.*
    - *information held on a computer system.*
    - *(Philosophy) true, justified belief; certain understanding, as opposed to opinion.*
2. *awareness or familiarity gained by experience of a fact or situation: the program had been developed without his knowledge | he denied all knowledge of the incidents.*

The Greek word for knowledge is "episteme". The science of "Epistemology", theory of knowledge, is a branch of philosophy that researches knowledge. This includes the definition of knowledge, acquisition of knowledge.

The science of Epistemology goes back to the time of Plato and his teacher Socrates. In the Theaetetus [1], Plato describes, but does not agrees with, a conversation between Socrates, Theodorus and Theaetetus. In this discussion Socrates and Theaetetus gave a definition for knowledge. This definition has been source of discussion ever since and are shown incomplete. But it does form one of the most important sources of discussions and research in the science of Epistemology.

Statue at the Library of Celsus in Turkey. The statue symbolizes "episteme" which is the Greek word for wisdom.

The definition describes that in order for something to be considered knowledge if it fulfils at least three criteria's. Knowledge has to be true, justified and believed.

Problem with this definition, also described by Plato, is that the justification of the knowledge must itself be knowledge and therefore results in a circular definition. But for the sake of this paper this definition contains several important features. Within knowledge bases statements are captured and the expert system then tries to find justification for these statements by asking the user questions. When the system found enough justification then the statement is believed to be true. Therefore the definition of knowledge in the Theaetetus is a good basis for expert systems.

EXPERT

The Oxford Dictionary of English (2nd edition revised) definition describes the term expert as follows:

Expert
→ noun
1. *A person who is very knowledgeable about or skilful in a particular area: an expert in health care | a financial expert.*

So basically an expert is a person who has an extensive amount of knowledge in a particular area. For building up and verification of an expert system it is also important that the expert not only has an extensive amount of knowledge. It also has to be proven that the expert has this extensive amount of knowledge. Depending on the field this can mean different things (diploma's, etcetera).

ADVICE

The Oxford Dictionary of English (2nd edition revised) definition describes the term advice as follows:

Advice
1. *Words given or offered as an opinion or recommendation about future action or behaviour.*
2. *Information given; news.*

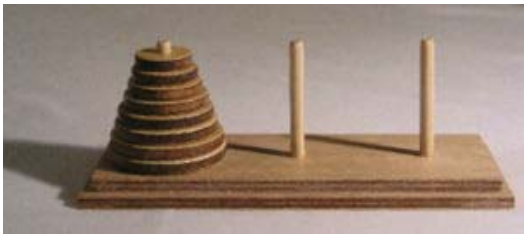
The advice provided by an expert system falls within the first definition. Usually the advice will be a recommendation about future action or behaviour. The purpose of this project is to get experts to verify if the advice given by the expert system is actually a good advice.

HISTORY OF EXPERT SYSTEMS

Expert systems belong within the field of artificial intelligence. In the field of artificial intelligence systems expert systems are one of the first and most used systems. Especially during the 60-s and 70-s, the expert systems topic has been popular in computer science research. But since the mid 80-s, expert systems became less popular. Reasons for this are discussed later in this chapter.

During the early days of computers, the most common use of computers was for calculation purposes. But even during these early days, a small group of scientists tried to use computers for artificial intelligence. The first artificial intelligence programs tried to solve mathematical theorems. These early artificial intelligence systems had a very general purpose and therefore could not use domain specific knowledge.

This made the programs less efficient for solving problems influenced by a large amount of variables. But the systems where advanced enough to solve simple puzzles that depend on a small number of fixed variables. One of these problems were the towers of Hanoi (see the image on the right). One of these first systems was GPS (General Problem Solver) by A. Newell, H.A. Simon and J.C. Shaw. GPS was developed in 1957. While solving a theorem, GPS would try every possible path. Although this made the tool highly inefficient, GPS is seen as the starting point of research in expert systems and more generally artificial intelligence. [2]



Puzzle game called towers of Hanoi

The invention of heuristic search algorithms made it possible to develop more specialized reasoning algorithms. Before the invention of heuristic search algorithms, most algorithms in artificial intelligence just calculated every possible outcome and returned the best result found. Calculating every possible outcome could result in expensive calculations even for simple problems. Therefore, in the early days of artificial intelligence, only small problems could be solved by these algorithms. When a system is built for a specific purpose, during the search for a solution the algorithm can use domain specific information to select better paths and to ignore paths that will never produce a feasible outcome.

Heuristics can do this by attaching a cost to an outcome and by trying to find the lowest cost. The algorithm can detect that a certain path will never lead to a cheaper outcome and therefore the algorithm can skip these paths. With this mechanism the algorithm is able to reduce the amount of paths that have to be checked.

Artificial intelligence systems are quite common for medical diagnosis. During the 60-s and 70-s most of the expert systems where built for the medical field. One of the best known expert systems built in the 70-s was the system Mycin (finished in 1974) [2]. Mycin was an expert system in the field of medicine and was developed at the University of Stanford by E.H. Shortliffe. Mycin could help by detecting and treating various infection diseases.

Mycin possessed a fairly simple inference engine (the system that does the actual reasoning) containing around 600 rules. When the user fills in a questionnaire, the system calculates which bacteria is causing the infection and what medicines should be used. It was one of the first systems that used uncertainty in its calculations. Studies showed that the Mycin system was right in 69% of the cases, which surpasses real experts in the same field. But still the system was never used in practice, this is caused by ethical and legal issues caused by accountability when the system gives a wrong advice.

Still, Mycin was very important for the development of future expert systems, as it was one of the first systems that used uncertainty in its calculations. In real life, an answer to a problem seems plausible if there is enough confidence in the answer, but this confidence always contains a level of uncertainty.

Initially, all elements in the knowledge base have 100% uncertainty. By answering questions, the uncertainty of some facts decreases and by deduction this propagates through the graph. The answers thus suggest a number of possibilities, that can be reduced further by asking and answering more questions. By iteration, a very limited set (preferably of size one) of possible answers is arrived at.

Mycin was the starting point of the popularity in expert systems but it also showcases one of the biggest problems with them. Although they perform on average not worse than real experts, there still is a high level of mistrust in expert systems.

These initial expert systems were always built for one specific purpose, this was due to the combination of both the inference engine and the knowledge base into the same system. A system for a new purpose required rebuilding the entire system. Emycin (Essential Mycin), an improved version of Mycin, solved this problem. This system separated the knowledge base from the inference engine. The inference engine was generalized to work with a separate, and replaceable, knowledge base. The knowledge base could be built by using a separate development platform.

There exist multiple knowledge presentation methods for use inside knowledge bases. These presentation methods are used for displaying the knowledge, storing the knowledge and processing the knowledge by the inference engine. Examples of these presentation methods are logics, production rules, semantic webs and frames. The main advantage of the separation of knowledge base and expert system is that the knowledge in a system can be replaced. Also bugs in the knowledge base can easily be fixed and the knowledge in a system can be extended/improved after the system has been deployed. Emycin was developed in 1979, also on the Stanford University, by Bill VanMelle. [2]

During the 80-s and 90-s many new expert systems have been build using technology as described above. Current research on expert systems is mostly based on flaws of the current implementations. Construction of a big knowledge base is one of the biggest problems when creating a new knowledge base. In many fields an extensive amount of knowledge is required in order to give good advice. Therefore, new research focuses on machine learning and knowledge sharing with other systems (semantic web technologies).

TECHNOLOGY BEHIND EXPERT SYSTEMS

An expert system contains two parts: a knowledge base and an inference engine. The knowledge base can be compared to a database that stores knowledge instead of data. In knowledge bases a knowledge presentation format is used to store knowledge. The most common knowledge presentation formats are:

1. Boolean logics
2. Production rules
3. Frames

Every presentation format has its own advantages and disadvantages, which will be described later on in this chapter.

KNOWLEDGE BASE

The knowledge base usually is developed in a separate tool. This tool then exports the knowledge base such that it can be imported in the expert system. The inference engine is usually a fixed component of an expert system and thus cannot be replaced. The knowledge base is a loosely coupled component and can easily be replaced. This makes it possible to use the same expert system for completely different purposes and it is possible to modify/improve the knowledge inside a system.

INFERENCE ENGINE

The inference engine is the component in the expert system which does the actual reasoning. For reasoning in expert systems there globally exist two methods:

1. Forward chaining (also called data-driven or bottom-up inference)
2. Backward chaining (also called goal-driven or top-down inference)

Forward chaining works on the data available. If the knowledge base contains the following rule base:

1. **if** eats grass and gives milk **then** cow
2. **if** eats corn and lays eggs **then** chicken
3. **if** chicken **then** feathers
4. **if** cow **then** fur

The goal is to find out what kind of skin the animal has. The available data is that the animal eats corn and lays eggs. Based on the available data first the second rule is selected because it matches the available data. Because the available data matches the antecedent (eats corn and lays eggs) the consequent (chicken) gets added to the available data. Then the rule base is searched again resulting in the third row being selected. Based on the available data the consequent is added to the available data. Because the consequent is the information that we were seeking (feathers) the inference engine stops and returns the answer (feathers) to the question.

Backward chaining works in the other direction. Based on the goals, evidence is being searched to prove that goal. We use the same goal as before, that we want to find out the skin of an animal. Based on this goal rules 3 and 4 are selected because their consequent (feathers and fur) are of the type that we are looking for (skin). The antecedents (chicken and cow) become the new goals. Based on these new goals rules 1 and 2 are selected because their consequents (cow and chicken) where in the goals. The antecedents of rules 1 and 2 become the new goals (eats grass, gives milk and eats corn, lays eggs). Because the goals eats corn and lays eggs are known to be true, rule 2 becomes true. The algorithm then backtracks the tree and feathers in rule 3 becomes true and will be the final result.

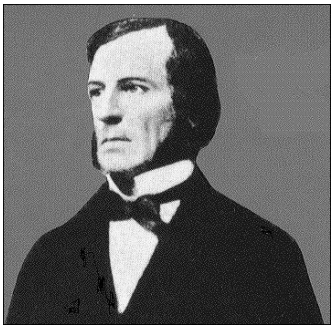
Forward chaining and backward chaining can be used to solve the same problems. Every problem that can be solved by forward chaining can also be solved with backward chaining and vice versa, sometimes one method is more efficient than the other method.

Basically, if the system already knows what it is looking for and if the goal is to prove that the assumptions where right, backward chaining is the best method. For example a doctor is pretty sure that a patient has pneumonia then with backward chaining it is easier to find proofs that the patient actually has pneumonia. Another benefit of backward chaining is that it can backtrack the proof and get a detailed justification of the initial assumption.

Forward chaining systems work better in situations where it is unclear what the result might be. So if initially there are zero clues available and during that examination more information becomes available then forward chaining will be a more efficient solution. This is partially due to the possibility of reaching sub goals, from these sub goals the questions that can be asked will be adapted.

Another possibility is to use a combination of both approaches. For example it is possible to use a forward chaining system to find an advice, after which the advice is used in the backward chain for providing more detailed information about how the advice was reached. This information can be useful to describe the reasoning how a certain advice was found. This can also be an important step in verification whether the given advice is feasible.

BOOLEAN LOGIC



George Boole, first person who formalized an algebraic system for logic in the 19th century

In the beginning of the chapter the different types for storing the knowledge where named. Now these three methods will be explained in more detail. The first method is Boolean Logic.

Boolean Logic is the formalism on which most of the concepts of Computer Science are based. George Boole, the person after which the formalism is named, was the first person to formalise an algebraic system for logics in the mid 19 century.

In 1938 Claude Shannon created a model for Boolean Logic with relays in electric circuits. This model was the first step in creating an electronic computer.

Boolean logic is a formalism that consists of a number of symbols:

- $\Leftrightarrow$ (bi-implication)
- $\Rightarrow$ (implication)
- $\vee$ (or)
- $\wedge$ (and)
- $\neg$ (negation)

It is not within the scope of this paper to explain the entire formalism of Boolean logics. Instead some examples will be given how knowledge can be stored by this formalism.

This rule base, described in the examples mentioned before:

1. **if** eats grass and gives milk **then** cow
2. **if** eats corn and lays eggs **then** chicken
3. **if** chicken **then** feathers
4. **if** cow **then** fur

can be rewritten to:

1. *eats grass* $\wedge$ *gives milk* $\Rightarrow$ *cow*
2. *eats corn* $\wedge$ *lays eggs* $\Rightarrow$ *chicken*
3. *chicken* $\Rightarrow$ *feathers*
4. *cow* $\Rightarrow$ *fur*

Notice that the implication is used and not the bi-implication. Having fur does not automatically mean that the animal is a cow.

The problem of using Boolean logic in its pure form is that some elements cannot be expressed. For example concepts like counting, time, unsure information and exceptions cannot be expressed with Boolean Logics. This means that in expert systems Boolean logics usually are extended with extra features.

PRODUCTION RULES

Most knowledge presentation formalisms are based on Boolean logics. One of these formalisms are production rules. The rule base given before where already in a production rule formalism:

1. **if** eats grass and gives milk **then** cow
2. **if** eats corn and lays eggs **then** chicken
3. **if** chicken **then** feathers
4. **if** cow **then** fur

Production rules are a very old formalism. Already in 500 B.C. the Indian writer Panini used production rules to define the grammar of the Sanskrit language in a series of books called Astadhyayi (which roughly means “eight books”) [3].

The first expert system, Mycin, was based on production rules and still a lot of expert systems are based on the production rules formalism. There exist a number of variations on production rules. The rules in this paper are based on the formalism described in [2]. The rules mentioned before are based on a less strict, but more commonly used, formalism.

The formalism described in this paragraph contains more strict rules about how formulas can be constructed, but also contains more possibilities to construct more advanced formulas. The rules in the following formalism are based on the Disjunctive Normal Form (DNF) formalism. DNF formulas are built up first from a collection of disjunctions and the disjunctions are built up from a collection of conjunctions.

An example of a production rule in this more advanced formalism is:

If same(eats, grass) **and** same(produces, milk) **then** assign(animal, cow)

With the “assign” statement it is possible to assign a constant “cow” to a variable “animal”. The “same” statement compares the variable “eats” with the constant “grass”. With this the production rule formalism is much more expressive than the Boolean logics formalism. A variable in a consequent can appear again in a antecedent and by this entire graphs can be built up where changes can propagate through the entire graph. Because of the strict DNF formulation it is easier for the computer to reason with the formalism.

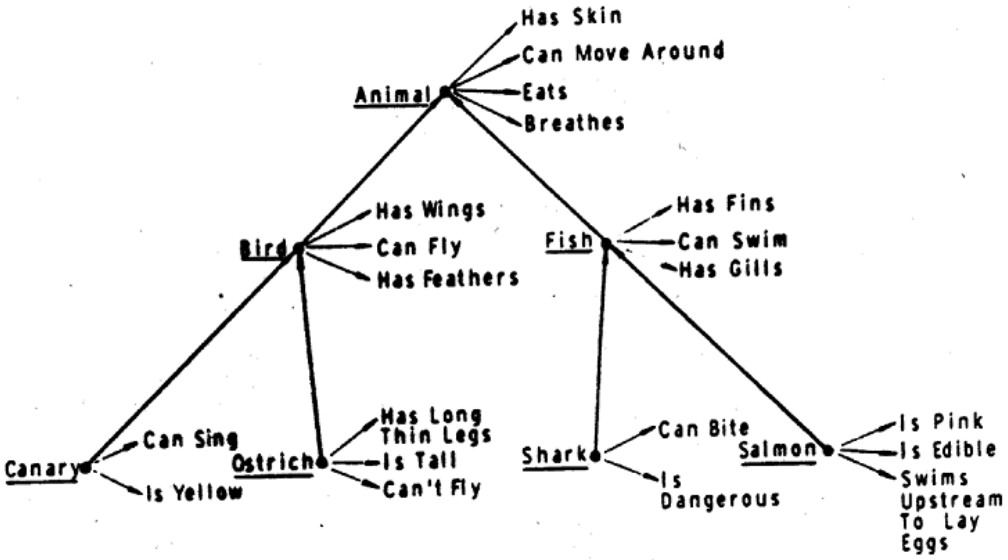
An extension to this production rule formalism is an object oriented production rule system. An object contains a number of attributes. With this it becomes possible to have multiple variables that describe the same object. An example of assigning a variable to an object is as follows:

assign(Pete, hair, brown)

The statement assigns the constant “brown” to the variable “hair” which is a variable of object “Pete”.

FRAMES

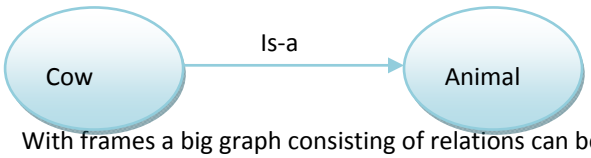
Production rules are the most used formalism within expert systems. But they have one major problem. An enormous amount of rules is needed to describe a big knowledge field. Therefore a third knowledge presentation formalism has been developed that is based on production rules. This formalism is called frames and they are based on semantic webs. Semantic webs are graphs that where invented by M. R. Quillian to describe the meaning of English words.[4]



An example of a semantic web is in the graph below, taken from the original paper of Quillian

In the 70-s M. Minsky used the semantic web technology from Quillian to define the frames formalism for guiding image recognition. This formalism will be described in more detail in this paragraph.

An example of a semantic web is the following graph:



With frames a big graph consisting of relations can be made from objects. The frames formalism is based on the Object Oriented paradigm. A frame is an object with certain attributes, multiple frames can be related to each other as a child/parent relation. In the graph above the “Cow” and the “Animal” are frames and “Is-a” is a relation between these frames.

A single frame can have attributes describing the object. An object can have multiple instances, for example a “Cow” can have multiple instances (the individual cows). A child object inherits the attributes of its parent object, unless overridden by a same named attribute in the child object.

The advantage of frames over production rules is that they reduce the amount of rules needed to define a knowledge base. By assigning an attribute to a parent object, all child objects inherit this attribute. By this, there is no need to assign this attribute for each object. Basically, frames can be compared to the production rules formalism with objects, extended with inheritance.

REASONING WITH UNCERTAINTY

Lotfi A. Zadeh, Iranian-American mathematician who proposed the theory of Fuzzy Logics in 1965.

The topic discussed in this paragraph is reasoning with uncertainty. In many areas of expertise, both the knowledge and the collected information have a level of uncertainty. This means that apart from the knowledge of the expert, an expert system should also take this uncertainty into consideration.

Reasoning with uncertainty is based on an extension of the Boolean Logic formalism. This new formalism is called Fuzzy Logic and is defined by Lofti A. Zadeh in 1965. Fuzzy Logic is based on the uncertainty that exists whether a statement really is true or not. This uncertainty is reflected in the Fuzzy Logics formalism by using values between 1 and 0 instead, instead of either the value 1 or 0.

Fuzzy Logic is based on the formalism of Boolean Logic. This means that the formalism still works with and, or, implies and equals statements. For example the statement “A and B” in Fuzzy Logics is equal to minimum(A,B) and “A or B” is equal to maximum(A,B).

For reasoning with uncertainty the production rule formalism will be used, extended with Fuzzy Logics. If there is a rule base like the following:

1. **if** A and B **then** C

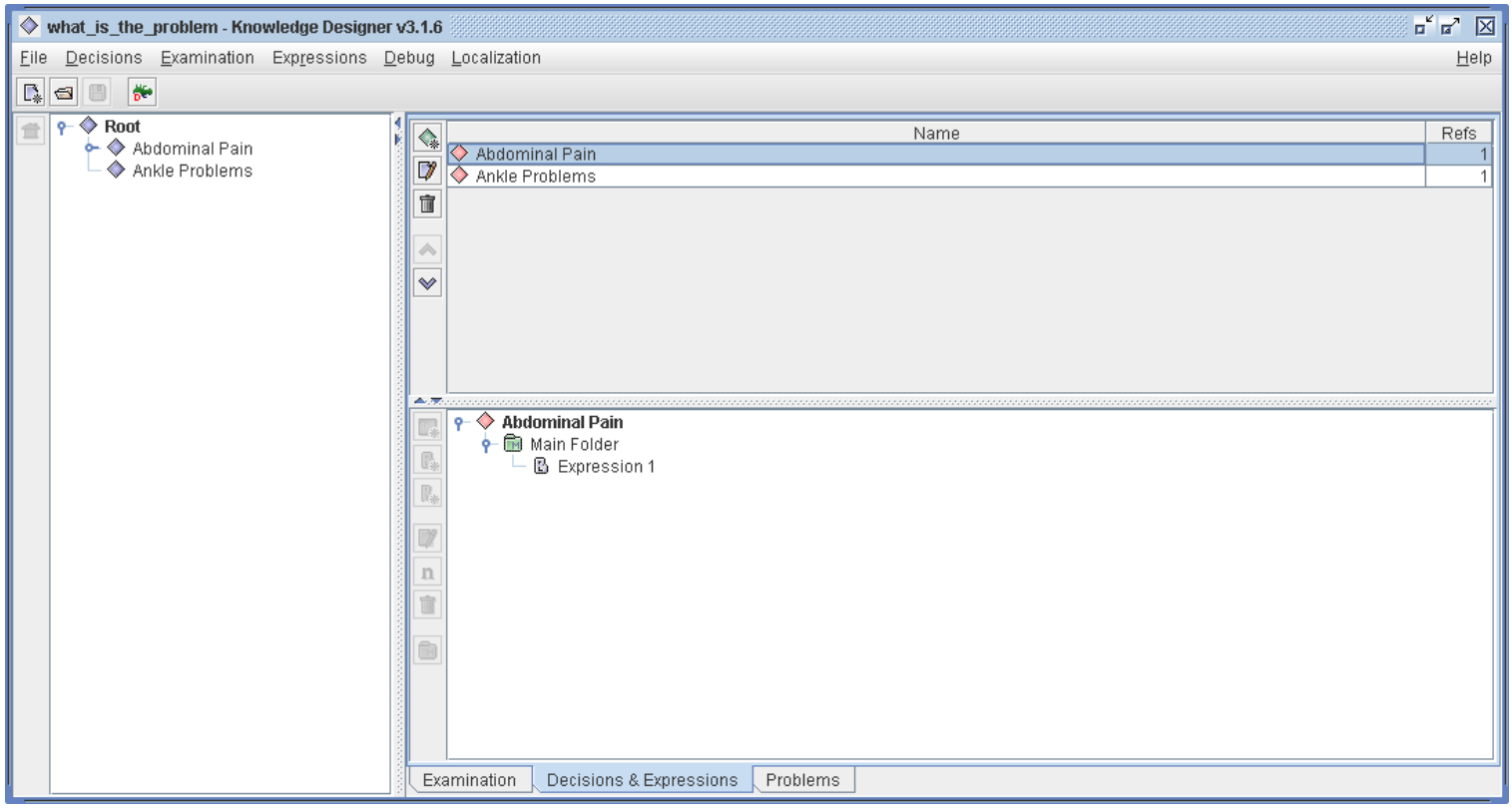
If A and B are true with a level of uncertainty then this level of uncertainty will propagate to C. This propagation is calculated with a propagation function. This could be an averaging function that takes the average of all confidences in the antecedent and the average will be the confidence in the consequent. Propagation functions are usually based on probability formulas. For reasons of space, we will forego a description. More information about different mechanisms can be found in [2].

When there is no information whether a certain fact is true or false, then its uncertainty will be very high, but because of the selected propagation function it can still be possible that the consequent has a low level of uncertainty. This shows one of the advantages of reasoning with uncertainty: it is possible to draw conclusions even when the amount of information available is low.

THE @DVISOR EXPERT SYSTEM

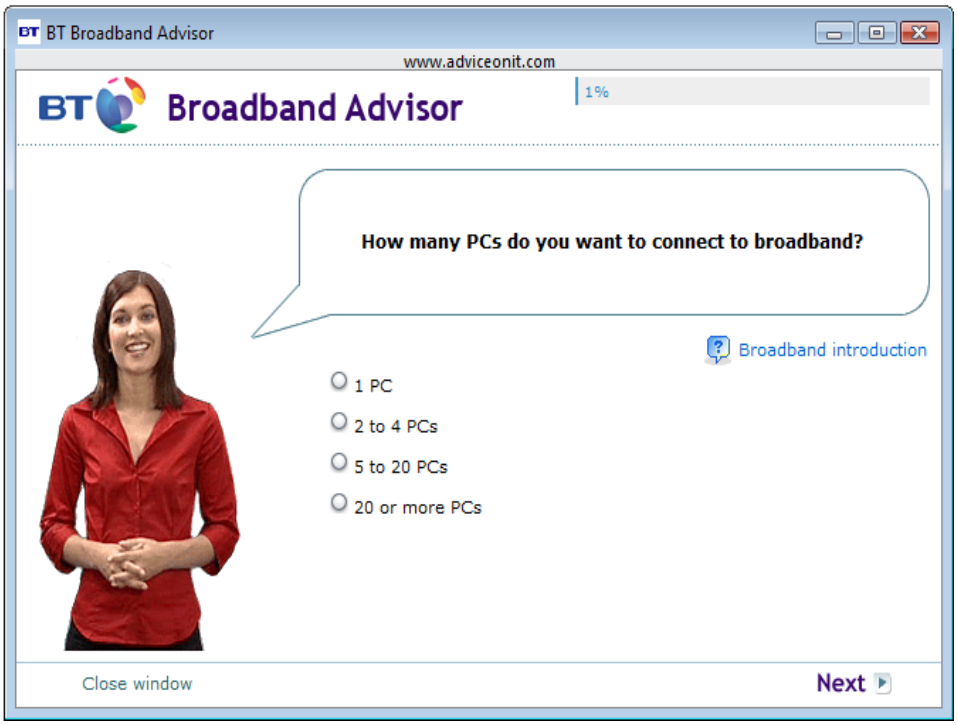
The original @dvisor system is an expert system built by Byelex. It consists of two separate parts:

- The Knowledge Designer, the Knowledge Designer is used to build and maintain the knowledge base



Screenshot from the @dvisor Knowledge Designer containing a medical knowledge base.

- Java Servlet., the Java Servlet is the expert system that asks the user questions and when the system gained enough confidence in will show an advice.

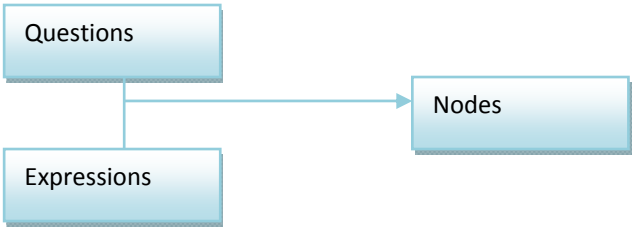


Screenshot from an @dvisor expert system made for Brittish Telecom that gives an advice about which broadband connection is best for a customer's specific needs.

This paragraph describes the technology behind the @dvisor system. The more advanced features will not be described or will only get a brief description.

STRUCTURE OF THE KNOWLEDGE

The knowledge in the system is stored in 3 different objects: questions, expressions and nodes. Questions are used in the questionnaires, as the name implies these are the questions asked to the user. The nodes are the advices/decisions that the system calculates based on the answers the user gives to the questions that were asked. Expressions are formulas that use the questions and their answers for calculating an advice. These 3 types of objects are connected together according to the graph below:



NODES

Nodes form a graph beginning in a root node. Nodes can contain child nodes, child nodes can be reused as child nodes from other parents. The program does not allow cycles of Nodes as this could create a deadlock in the Inference Engine. So if node A has children then it is impossible for any of the children or grandchildren of node A to have node A as its child.

There are 3 types of nodes:

1. Boolean nodes
2. Fuzzy nodes
3. Otherwise

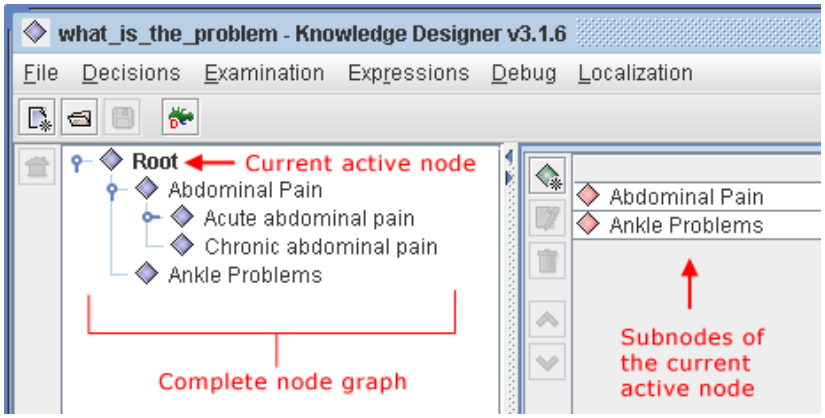
The node type is important for expressions. If an expression leads to a node of the Boolean node type then the node either has 100% confidence or 0% confidence. If an expression leads to a node of the Fuzzy node type then it gets a confidence value between 0% and 100%.

An expression of the otherwise type cannot contain expressions and will always have 100% confidence. If another node from another type, Boolean or Fuzzy, has 100% confidence then this Boolean or Fuzzy node will have preference over the otherwise node. For example the following statement:

If A **then** B **otherwise** C

If the nodes A and B would be Fuzzy nodes, node B can only be reached via node A and if A would have 80% confidence, then B will have less than 80% confidence. Node C is of type otherwise and therefore this node has 100% confidence, resulting in a situation where node C becomes the active node

For Boolean nodes it is usually needed to add otherwise nodes to be able to reach a final advice. This is due to Boolean nodes either having 0% or 100% confidence.



In the screenshot above the “Root” is the root node, nested under the root node is the “Abdominal Pain” node, this is an intermediate node. Nested under that node is the “Chronic abdominal pain” node which forms a final decision.

QUESTIONS

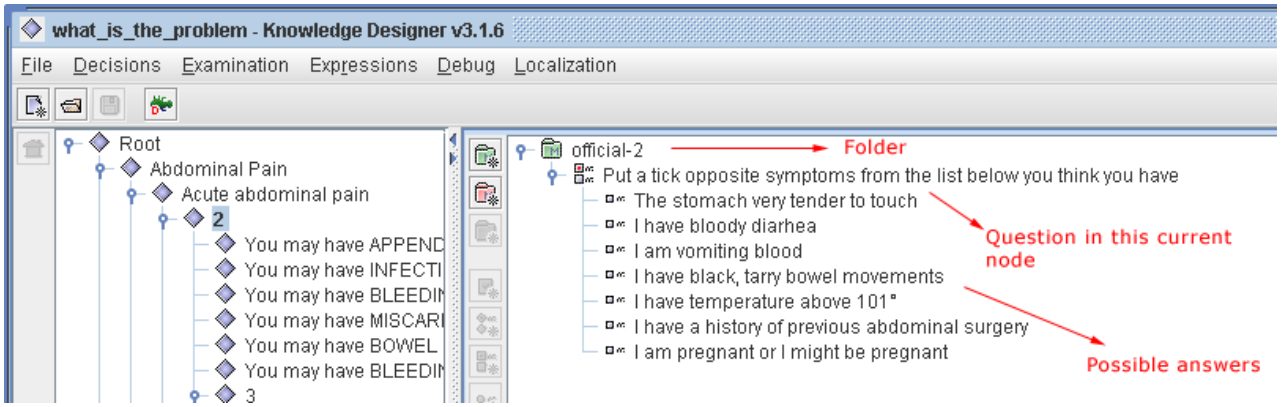
Nodes can contain folders and folders contain questions. When a user initiates a questionnaire then the root node will be the active node. The user will receive the questions that are within the folders that are inside the active node. The system calculates the confidence for every child node of the currently active node. When a user finished answering all questions in the currently active node, the system will select the child node in which it has highest confidence. This node becomes the new active node.

When there are no folders or questions in this node, then this node becomes the final decision/advice and the questionnaire is finished. Otherwise this node becomes an intermediate node and the user receives a new list of questions, based on the currently active node.

The same question can occur in multiple folders, but it is not allowed to reuse the same question within the same node. When a question is reused in different nodes, and first one node becomes active and then the other, then the question will only be asked the first time and the answer will be reused.

The types of questions that can be asked are:

- yes/no questions;
- singular questions (a question with a list of answers, single choice);
- checklist questions (a question with a list of answers, multiple choices).

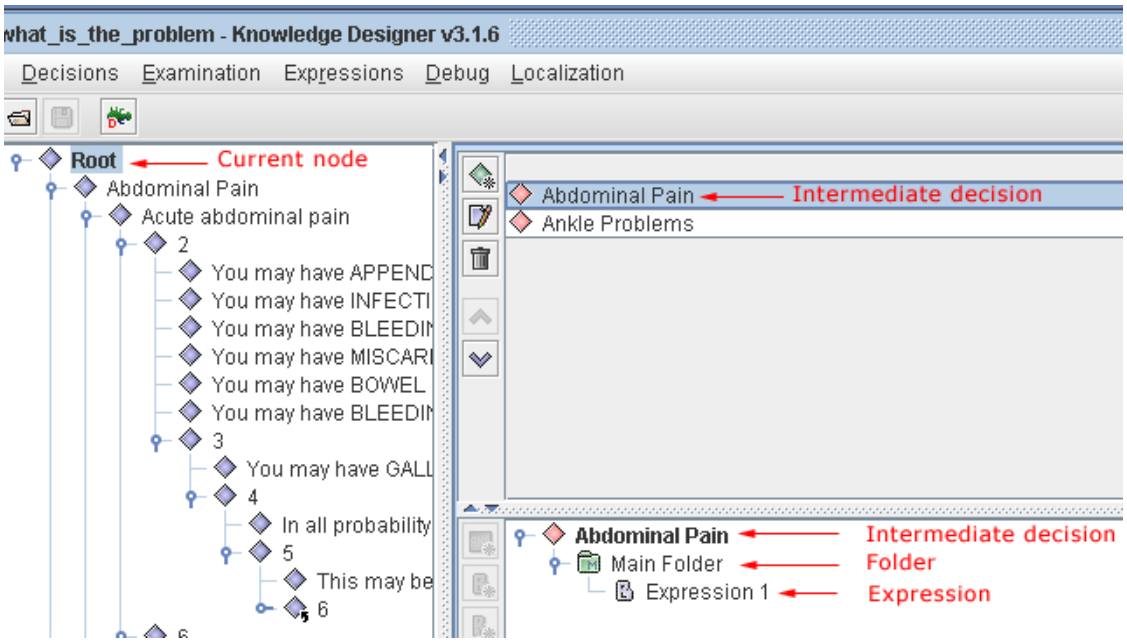


The screenshot above shows a pane with questions nested under folders. The current active node is "2" and the folders/questions that are shown belong to the "2" node.

EXPRESSIONS

Expressions are stored in folders (which are part of nodes) and the expressions are built up with possible answers to questions that are stored in the same folders. An expression is connected to a child node of the node in which the expression is stored.

An expression evaluates to true if enough confidence in the expression is found. The confidence in a expression is calculated based on the type of the node to which the expression leads. The confidence in a node is calculated based on all expressions that lead to this node. If enough confidence is reached in a node, the node will become the new active node.

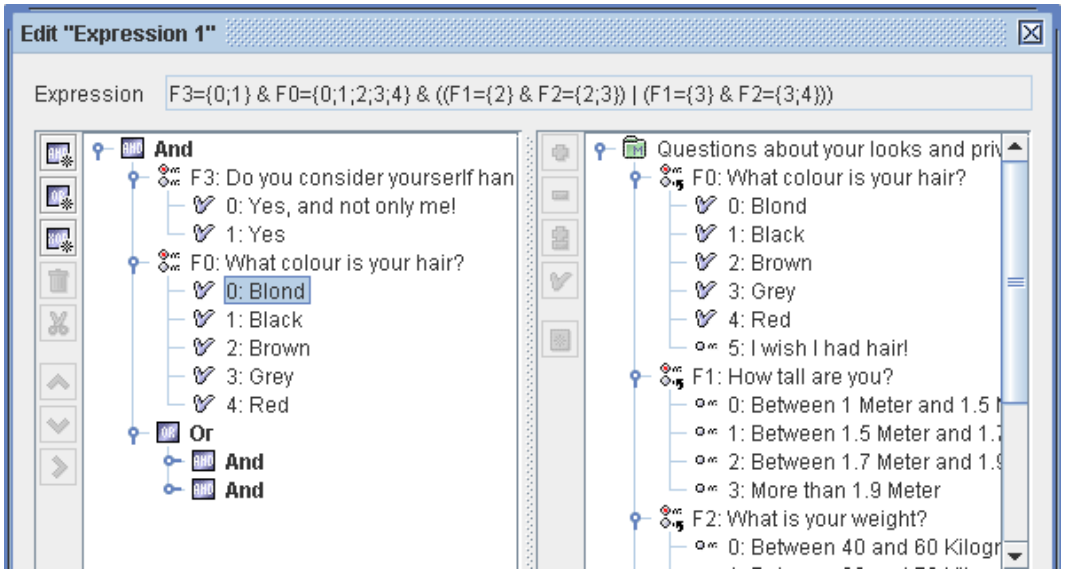


The screenshot above contains an expression, “Expression 1”, which, if there would be enough confidence in this expression, would lead to the decision, “Abdominal pain”. The expression is based on questions from folder “Main Folder”. This folder is part of the currently active node, which is “Root”.

There exist two types of expressions:

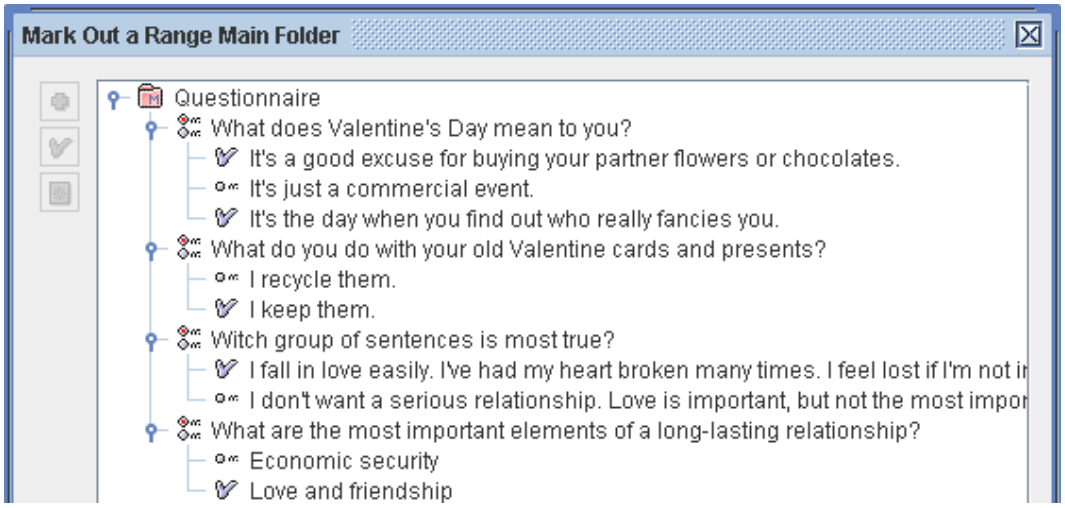
1. Boolean expressions
2. Ranged expressions

In the following window it is possible to create a Boolean expression for a certain decision:

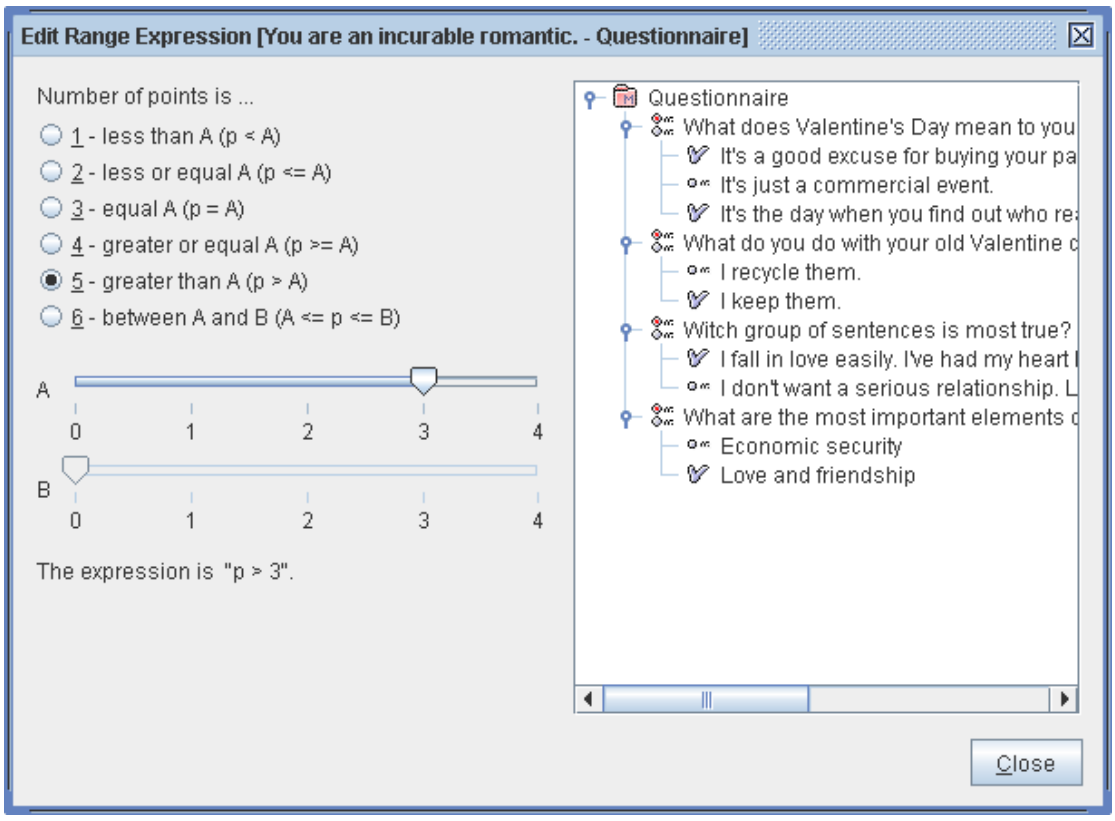


By using Boolean logic, displayed in a tree like structure, it is possible to build up a formula that describes the relation between the question answers and the decisions. There exist and, or and xor connectors to make very advanced formulas.

A ranged expression is an expression where every “correct” answer gives a “point”. The screenshot below shows the marking of how a question should be answered to be “correct”.



An expression defines a range of points and if the score falls within this range then the expression evaluates to true. The screenshot below shows a ranged expression and its range. The defined range below, says that the amount of points should be larger than 3.



INFERENCE ENGINE

The inference engine of @dvisor works with a forward chaining strategy, explained in the first paragraph. Forward chaining works better in situations where the system gains more knowledge by answering each question and therefore this approach is used instead of backward chaining.

When new data arrives into the system by answering a question, all expressions in which this question occurs will be selected by the system. The system will update the variables that occur in these expressions and will update the certainty it has in these changed expressions.

With these modified certainties, the certainty in each node will be updated. Besides the certainty that the system currently has in a node the system also calculates a maximum certainty. The current certainty is based on all questions answered this far. The maximum certainty in a node is based on the maximal certainty value possible by answering all remaining questions.

The system will reduce the amount of possible advices based on these maximum certainties. Child nodes that have a maximum certainty of 0% certainty will be excluded from the calculations. The amount of possible advices will be reduced, until only one advice is left or all questions are answered.

The methods how certainty is calculated and how certainty propagates through the knowledge base depends on the different settings in the knowledge base.

One of these settings are the type of nodes that are used. With Boolean nodes all expressions will either have 0% or 100% certainty. Therefore the certainty of other nodes will also be either 0% or 100%. If node C can only be reached via node B and node B has 0% certainty then node C will have 0% certainty as well. When nodes can be reached via multiple paths then the maximum certainty can still be 100%, if one of the intermediate nodes that lead to this node still has a maximum of 100% certainty.

With Fuzzy nodes, the expressions can have a certainty between 0% and 100%. When a node has certainty of 80%, the child nodes of this node will have a maximum of 80% as well, given that no other nodes with higher certainty can lead to this child node, otherwise this node with higher certainty will decide the maximum certainty of this child node.

2. THE WEB BASED @DVISOR SYSTEM

This chapter first describes the problem that this project is going to solve, this includes a more detailed description of the user requirements. This part of the chapter will be followed by a technical description of the original @dvisor expert system. This description contains structure of the knowledge base in more detail.

The chapter will then continue with a description of the new web based @dvisor system. This description will contain the user requirements, the structure and the behaviour of the new web based @dvisor system

The inference engine is not used in the web based @dvisor system as the web based @dvisor system is only used for managing and verifying knowledge bases, therefore the technical part of the inference engine will not be described in more detail.

PROBLEM DESCRIPTION

The new web based @dvisor system, which was the goal of this project, seeks to provide a system that experts can use to verify a knowledge base. The original @dvisor system does not provide functionality for experts to verify a knowledge base. Verification of knowledge bases with the original @dvisor system was a time consuming process. The process consisted of the following steps:

1. The knowledge engineer creates a knowledge base by using the @dvisor Knowledge Designer
2. The knowledge engineer draws a graph, either on paper or by using a modelling tool, like Microsoft Visio, containing the entire knowledge base.
3. The expert(s) verify the graph and will provide feedback based on the graph.
4. The knowledge engineer gathers the feedback and will try to improve the knowledge base by reiterating this step and the previous steps, until the knowledge base is verified correct.

The time consuming part of this process is the translation of the knowledge base into an understandable format for experts in step 2 and transforming the feedback of the expert into modifications to the knowledge base in step 4. Some knowledge bases are verified by multiple experts simultaneously which makes the entire process even more time consuming.

Another problem with the process, described above, is the amount of translation steps. Due to the expert misinterpreting the graph, from step 2, or the knowledge engineer misinterpreting the feedback, from step 3, information might be misinterpreted or get lost. These translations errors might reduce the quality of the knowledge base and might result in extra iterations of the verification process to repair these errors.

The goal of this project was to research if, and how, a web based system is able to solve the described problems. Based on this research a system was developed to solve these problems. The implemented system replaces step 2 and 3, by the web based @dvisor system.

Step 2, of the process described above, will be replaced by functionality that displays the knowledge base in the web based system. Experts from various fields of expertise will use this system, therefore the method that displays the knowledge has to be understandable for these experts. By having a web based system that displays the knowledge in an understandable form, the translation in this step is done automatically. The web based system will therefore reduce the amount of time needed for this translation step and will remove the amount of errors that might occur during this step.

Step 3 will be replaced by functionality to modify the knowledge in the knowledge base. Experts would not have been able to modify the knowledge in the Knowledge Designer themselves. Modifying the knowledge in the web based system is possible, because the knowledge base is represented in a method that is more understandable for these experts. By letting the expert directly modify the knowledge base, instead of writing down the feedback and implementing this feedback by the knowledge engineer, the chance of translation errors and the time needed to gather feedback from experts will be reduced.

The system is developed according to the following user requirements:

1. A web-based system to access the knowledge base
2. Possibility for multiple experts to access the web based system simultaneously.
    a. An expert can register for an account to login to the system.
    b. Newly registered accounts have to be accepted first by a administrator.
    c. An expert can login to the system.
    d. An administrator can modify account details, remove accounts and send e-mails to all experts registered to the system.
    e. An expert account gets a personal versions of the knowledge bases that (s)he imported. These knowledge bases cannot be accessed by other experts.
3. Possibility to modify the knowledge base (only basic functionality).
    a. Possibility to import/export knowledge bases.
    b. Possibility to switch between knowledge bases.
    c. Possibility to create/modify/delete/reuse nodes.
    d. Possibility to create/modify/delete folders.
    e. Possibility to create/modify/delete/reuse questions.
    f. Possibility to create/modify/delete expressions.
4. Possibility to store remarks/feedback with a modification made to the knowledge base.
5. Functionality for knowledge base engineers to see the modifications that an expert made.
6. Displaying the knowledge base in an understandable way, for people with different fields of interest.

THE @DVISOR SYSTEM

This paragraph describes how the @dvisor knowledge base is stored in a file and how this file is translated into a class structure.

The knowledge base in the @dvisor system is stored in an XML file. The XML format makes it easy to write multiple applications that can use the knowledge base. The applications that use the file are the Knowledge base Designer, that can modify the knowledge base, and the expert system servlet, that uses the file for the advice giving process. One of the special features of the used XML format is that it can contain multiple languages, which is very useful in international systems.

There currently does not exist a universal standard format for storing knowledge bases,therefore all expert systems currently use their own file format. Because @dvisor uses an XML format, other systems are able to use the same format.

An example version of the XML file is shown below. Sections of the file have been omitted for reasons of readability, therefore the code will not work when used in @dvisor.

```
<?XML version="1.0" encoding="UTF-8" standalone="yes"?>
<knowledge base XMLns="http://www.byelex.com/advisor/kbxs/kbxs30">
  <nextID>148</nextID><locale>en_EN</locale>
  <activeLocale>en_EN</activeLocale>
  <qstContainer>
    <rootFolder>
      <name><value locale="en_EN">All Questions</value></name>
      <multiQst kind="qs" qstId="9">
        <name><value locale="en_EN">Choose your main complaint</value></name>
      </multiQst>
    </rootFolder>
  </qstContainer>
  <clfContainer>
    <linkedAdvice>
      <linkedAdviceId>51</linkedAdviceId>
      <name><value locale="en_EN">SELF-CARE</value></name>
      <text><value locale="en_EN">CALL YOUR DOCTOR RIGHT AWAY.</value></text>
    </linkedAdvice>
    <rootClfNodeId>1</rootClfNodeId>
    <clfNode clfId="1">
      <name><value locale="en_EN">Root</value></name>
      <clfDM dmNodeId="2">
        <multiDiagnostics>false</multiDiagnostics>
        <dmClass type="boolean">
          <realId>4</realId><zeroPoint>-1.0</zeroPoint>
        </dmClass>
        <mainFolder folderId="8">
          <name><value locale="en_EN">Main Folder</value></name>
          <qst qstId="9"/>
          <expression dmClassId="4">
            <formula>
              <name>Expression 1</name>
              <rootOperation opKind="and" negated="false">
                <singularOperand itemId="10"/>
              </rootOperation>
            </formula>
          </expression>
        </mainFolder>
        <simpleAutoExmScheme/>
      </clfDM>
    </clfNode>
    <clfNode clfId="4">
      <name><value locale="en_EN">Abdominal Pain</value></name>
      <clfDM dmNodeId="5">
        <multiDiagnostics>false</multiDiagnostics>
        <dmClass type="boolean">
          <realId>12</realId><zeroPoint>-1.0</zeroPoint>
        </dmClass>
        <mainFolder folderId="16">
          <name><value locale="en_EN">Time of pain</value></name>
          <qst qstId="17"/>
          <expression dmClassId="12">
            <formula>
              <name>Expression 1</name>
              <rootOperation opKind="and" negated="false">
                <singularOperand itemId="18"/>
              </rootOperation>
            </formula>
          </expression>
        </mainFolder>
        <simpleAutoExmScheme/>
      </clfDM>
    </clfNode>
    <clfNode clfId="28">
      <name><value locale="en_EN">You may have APPENDICITIS</value></name>
      <linkedAdviceId>51</linkedAdviceId>
      <clfDM dmNodeId="29">
        <multiDiagnostics>false</multiDiagnostics><simpleAutoExmScheme/>
      </clfDM>
    </clfNode>
  </clfContainer>
</knowledge base>
```

The XML is divided in two containers, not taking into account the activeLocale tag and the nextId tag that are used for storing the currently active language and the identifier number that should be used next. The two containers are:

1. qstContainer (the blue part)
2. clfContainer (the red part)

The qstContainer contains all the questions. The wrapping tag “rootFolder” does not carry any special purpose besides being a wrapper for all these questions. This container does not contain the folder hierarchy, it only lists all the available questions. Questions that are used multiple times in different folders will only appear once in this container.

The clfContainer contains the nodes, the folders and the expressions. This section describes the hierarchic structure of the knowledge base. This describes the hierarchy of the nodes, the folders in every node and the questions and expressions in every folder.

The clfcontainer starts with the linkedAdvice tags, linkedAdvice are used to provide more detailed text descriptions for a final advice.

Besides the linkedAdvice the clfContainer contains clfNodes. The clfNode represents a node in the knowledge base, it contains the name of the node, an optional reference to a linkedAdvice and a clfDM tag. The clfDM tag contains dmClass tags and folder tags.

The dmClass tag represents a child node of this node. The tag contains a realId tag, this realId is a reference to the clfId attribute that can be found in clfNode tags.

The folder tag represents a folder inside a node. The folders contain references to the questions that are used in this folder and the expressions in this folder. The questions in a folder are represented by the qst-tag. This qst-tag contains an id attribute, which contains the same identifier as a question in the qstContainer.

An expression tag contains a dmClassId attribute, this attribute is a reference to the node to which this expression leads. The formula tag, inside the expression tag, contains the expression formula. This formula contains references to question tags and in case of yes/no questions and multiple option questions the way how each answer should be answered.

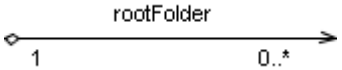
The previous XML section contains a Boolean expression. The following XML snippet contains ranged expressions. The example contains two nodes (a parent node, clfId=”1” and a child node clifId=”17”).

```
1.      <clfNode clfId="1">
2.        <name>
3.          <value locale="en">Root</value>
4.        </name>
5.        <showIntermediateDecision>false</showIntermediateDecision>
6.        <useGlobalConfidenceThreshold>1</useGlobalConfidenceThreshold>
7.        <confidenceThreshold>100</confidenceThreshold>
8.        <clfDM dmNodeId="2">
9.          <useGlobalMultidiagnostics>true</useGlobalMultidiagnostics>
10.         <multiDiagnostics>false</multiDiagnostics>
11.         <dmClass type="fuzzy">
12.           <realId>17</realId>
13.           <zeroPoint>0.0</zeroPoint>
14.         </dmClass>
15.         <rangeMainFolder folderId="3">
16.           <name><value locale="en">Questionnaire</value></name>
17.           <qst qstId="7"/>
18.           <markupItem itemId="8"/>
19.           <rangeExpression dmClassId="17">
20.             <a>0</a>
21.             <b>0</b>
22.             <kind>equal</kind>
23.           </rangeExpression>
24.         </rangeMainFolder>
25.         <simpleAutoExmScheme/>
26.       </clfDM>
27.     </clfNode>
28.     <clfNode clfId="17">
29.       <name>
30.         <value locale="en">Love for you is a business.</value>
31.       </name>
32.       <showIntermediateDecision>false</showIntermediateDecision>
33.       <useGlobalConfidenceThreshold>1</useGlobalConfidenceThreshold>
34.       <confidenceThreshold>100</confidenceThreshold>
35.       <clfDM dmNodeId="18">
36.         <useGlobalMultidiagnostics>true</useGlobalMultidiagnostics>
37.         <multiDiagnostics>false</multiDiagnostics>
38.         <simpleAutoExmScheme/>
39.       </clfDM>
40.     </clfNode>
```

How the questions should be marked to make a certain ranged expression become true is stored per folder, inside the rangeMainFolder tag, with markupItem tags. The rangeExpression tags contain the expressions, where the a, b and kind tag describe the range of this expression.

The XML structure is closely mapped to the class structure of the knowledge base. The class diagram on the next page shows the knowledge base that is loaded into the Knowledge base designer. The class diagram is made conform the UML standard, information about the formalism that is used, is shown below.

Aggregation

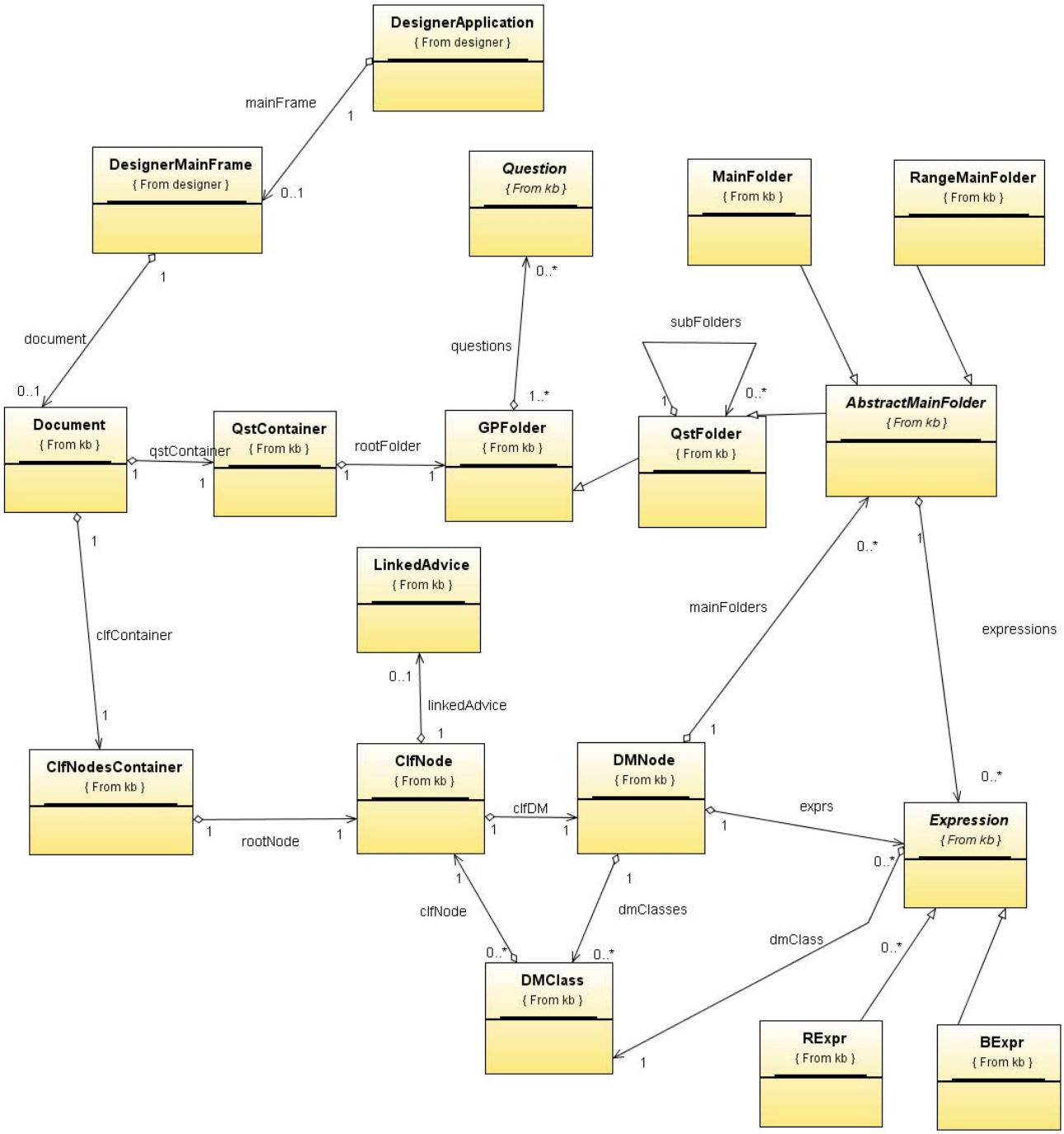


Means that an instance of the left class can contain a reference to one or more instances of the right class. The class on the right lives independent from the class on the left. Meaning that the class on the right can exist without a class on the left referencing to it. The 1 on the left means that an instance of the right class always belongs to one instance of the class on the left. The 0..* on the right means that an instance of the class on the left can contain zero or more instances of the class on the right. The label *rootFolder* is the name of the relation.

Inheritance

Means that the left class is the parent of the right class. A child class inherits the properties of the parent class.

The classes that describe the inner structures of the expressions and the questions and some relations are taken out of this diagram and are placed in separate diagrams. The classes were taken out to create a more clear overview of the knowledge base classes.



The Knowledge base Designer program communicates with the knowledge base via an instance of the DesignerApplication class. The DesignerMainFrame class is the main class that contains the user interface. The DesignerMainFrame class has a one on one relation to the DesignerApplication class, in the Knowledge base Designer only one instance of both classes will exist. The DesignerMainFrame. will communicate with the knowledge base by calling functions on the DesignerApplication.

The Document class provides communication with the file in which the knowledge base is stored. This class will perform the reading and writing of the XML file. The class will read the file, parse the contents and will build up the knowledge base structure. The Knowledge base Designer program can open only one knowledge base at the same time, therefore there is a one to one relation between the Document class and the DesignerApplication class.

The Document class consists of two containers, like the XML document described earlier, the ClfNodesContainer class and the QstContainer class. Between the Document class and each of these containers is a one to one relation. Because the application can only have zero or one instances of the Document class, an application will have only zero or one instances of each of these containers.

The class diagram shows that the ClfNodesContainer class contains a root node, this root node is of type ClfNode. An instance of the ClfNode class is able to contain an instance of the LinkedAdvice class, which contains a more detailed description of an advice. The ClfNode class has a one to one relation with the DMNode class (in the XML file represented as the clfDM tag), an instance of the DMNode class can contain sub nodes, folders and expressions. The ClfNode and DMNode classes together represent the Node functionality.

An instance of the DMNode class can contain zero or more instances of the DMClass. These instances of the DMClass class represent the possibility that a node contains sub nodes. An instance of the DMClass class contains a reference to one instance of the ClfNode class, but an instance of the ClfNode class can belong to multiple DMClass instances. This makes it possible that a node can be reused as a child node of multiple nodes and a node can contain multiple child nodes.

The QstContainer class contains one rootFolder of type GPFolder. This rootFolder contains all questions of the knowledge base. The GPFolder class contains functionality for creating questions into and deleting questions from the knowledge base.

An instance of the DMNode class also contains folders, these folders are subclasses of the AbstractMainFolder class, which is a subclass of the QstFolder. The QstFolder is a subclass of the GPFolder, the folder type that is used in the QstContainer, this QstFolder can contain zero or more subfolders. The AbstractMainFolder class inherits this functionality from its parent class, therefore instances of the AbstractMainFolder class, in an instance of the DMNode class, are able to contain subfolders as well. The GPFolder does not contain this functionality therefore the folders in the QstContainer do not contain subfolders. The purpose of subfolders, is that questions in a folder are shown grouped on the same page. By putting questions in a separate folder, a separate page is created for the questions in this folder.

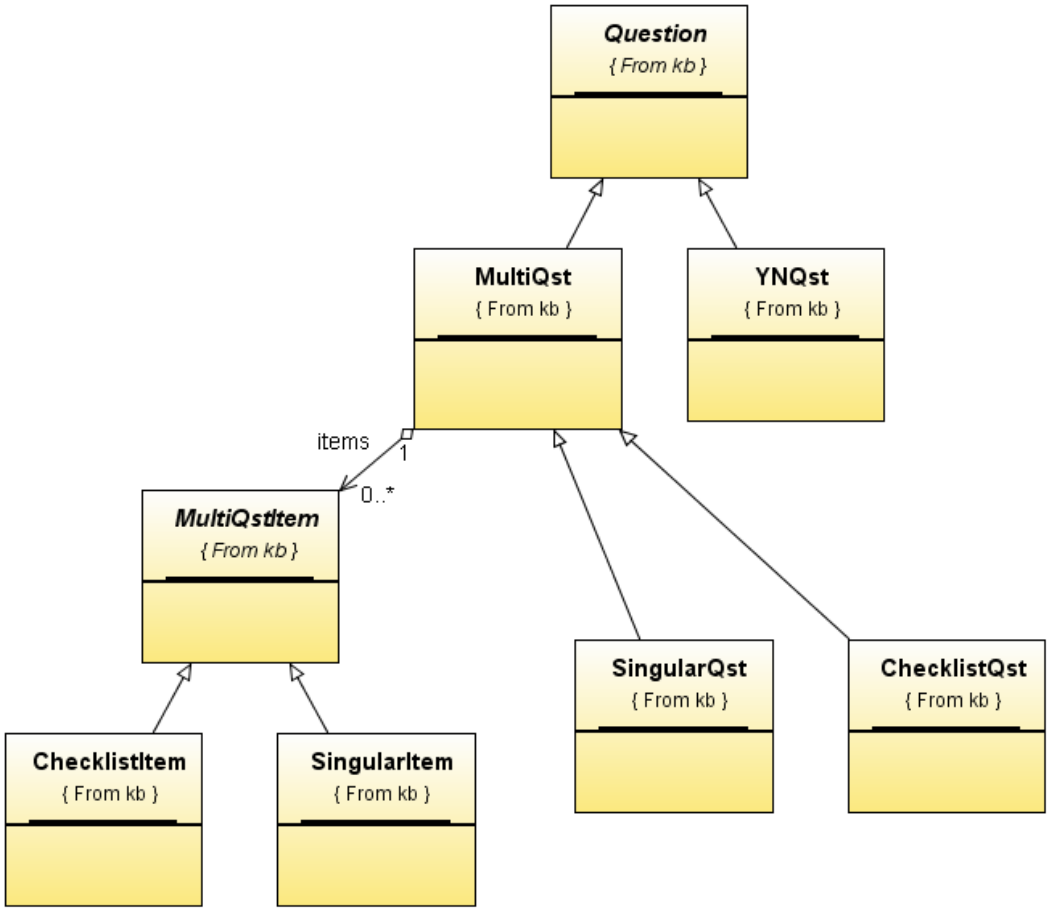
The AbstractMainFolder has the following two subclasses: the MainFolder class and the RangedMainFolder class. A folder class contains zero or more questions. In the XML file, described before, all questions are located in the QstContainer tag and the ClfNodesContainer tag contains references to these questions. The same approach is used in the class diagram, both the instance of the QstContainer class and the instance of the ClfNodesContainer class contain references to the same instances of the Question class.

Below on this page, the structure of the Questions is described in more detail, by a separate class diagram.

The AbstractMainFolder class also has a one to many relation to the Expression class. An instance of the AbstractMainFolder class can contain zero or more instances of the Expression class.

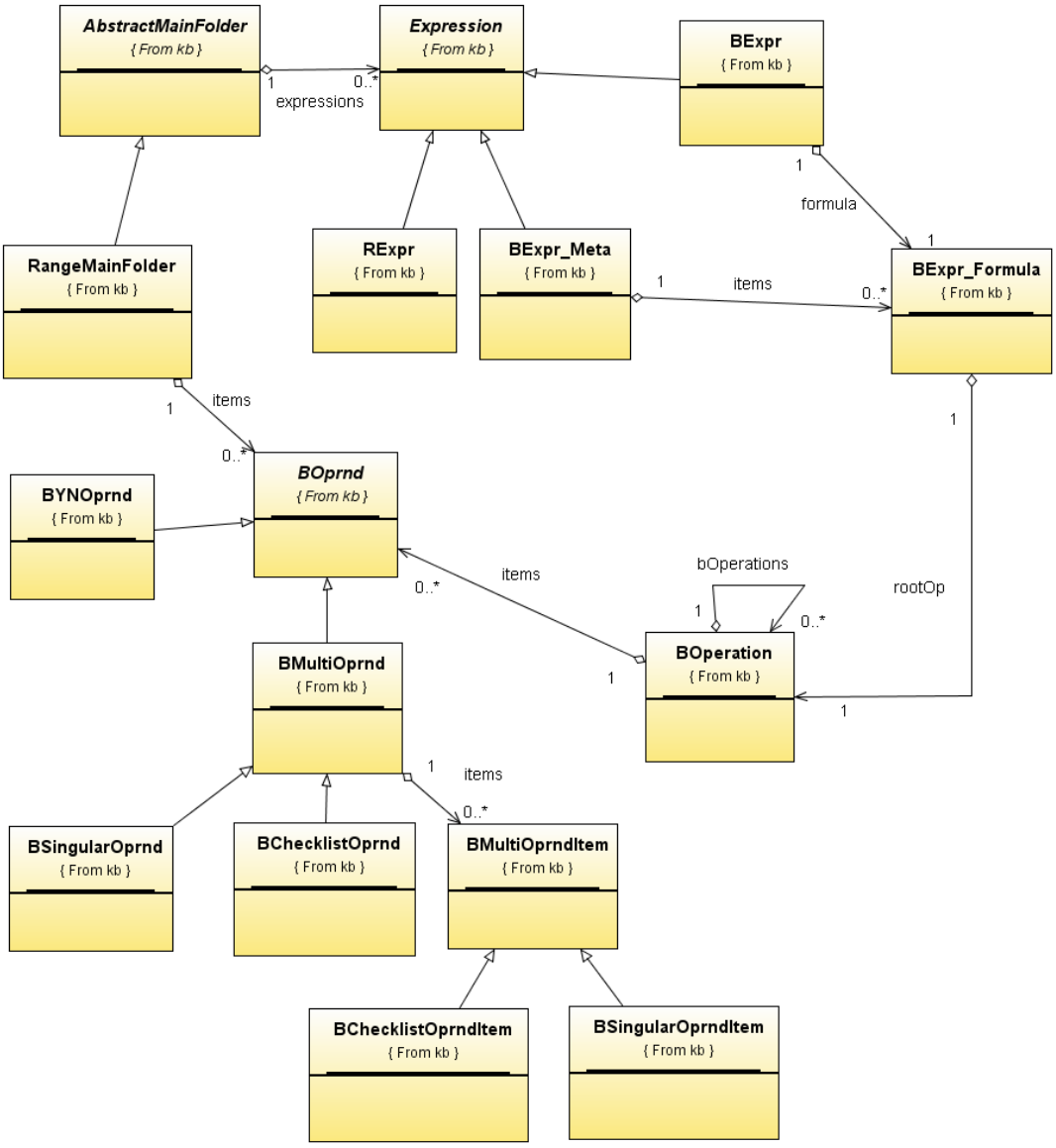
An Expression points to a node, the Expression class therefore contains a one to one relation with the DMClass class. This DMClass has a relation with the ClfNode therefore the Expression class is able to contribute to the confidence of the Node that is represented by the ClfNode. If enough confidence in this node is gathered, this node will become the next active node.

The structure of the Expressions and its subclasses are described in more detail on the next page.



The above structure shows the Question class and its child classes. There exist three different types of questions: YNQst class (yes/no questions), SingularQst class (multiple choice questions with one answer), ChecklistQst class (multiple choice questions with multiple answers).

The SingularQst class and ChecklistQst class have a parent class, the MultiQst class. This MultiQst class has a one to many relation with the MultiQstItem class, one instance of the MultiQst class can have zero or more instances of the MultiQstItem class. The MultiQstItem class is the parent of the ChecklistItem class and the SingularItem class. Instances of the ChecklistItem class represent the answers to a ChecklistQst class and instances of the SingularItem class represent the answers to a SingularQst class.



There exist three different types of expressions, which are subclasses of the Expression class: BExpr, BExpr_Meta and RExpr.

The BExpr represents a Boolean Expression. This Boolean expression consists of a tree of objects, that are instances of a number of different classes. The BExpr class has a one to one relation to the BExpr_Formula class. This BExpr_Formula class contains the Boolean expression tree. The BExpr_Meta class, also a subclass of the Expression class, has a one to many relation to this same BExpr_Formula class, by this the BExpr_Meta class can contain multiple Boolean expression trees.

The BExpr_Formula has a one to one relation with the BOperation. The BOperation class represents a Boolean connector, this BOperation can be of 3 types: AND, OR and XOR. An instance of the BOperation class can contain one or more instances of the BOperation class and one or more instances of the BOprn class. The BOprnd class refers to a possible answer to a question. The combination of BOperation objects and BOprnd objects makes it possible to build formulas. For example the formula: $(A \wedge B) \vee C$. Where the A,B,C would be instances of the BOprnd class and $\wedge$,$\vee$ would be instances of the BOperation class. The $\vee$ would be the top BOperation, the $\wedge$ and C would be children of this top BOperation. The A and B would be children of the $\wedge$ BOperation.

The BOprnd class and its subclasses, have a similar structure as the Question class. The BOprnd class differs from the Question class, as the Question class represents the questions and it is possible answers while the BOprnd class represents the answer to a Question that is needed for this Expression to evaluate to true.

The BOprnd class has two subclasses: BYNOprnd (related to a YNQst question) and BMultiOprnd (related to a MultiQst question). The BMultiOprnd also has two subclasses: BSingularOprnd (related to a SingularQst question) and BChecklistOprnd (related to a ChecklistQst question). An instance of the BMultiOprnd class can contain zero or more instances of the BMultiOprndItem class, similar as an instance of the MultiQst class that has zero or more instances of the MultiQstItem class. An instance of the BMultiOprndItem class refers to an answer of an instance of the MultiQst class, which is represented by an instance of the MultiQstItem class.

The RExpr class differs from the BExpr and BExpr_Meta class that an instance of the RExpr class does not contain a formula of how the questions should be answered, to make this expression evaluate to true. The RExpr class only contains variables that specify the range of this RExpr. The actual formula of how the questions should be answered is located in the RangeMainFolder, in which the instances of the RExpr are located. This formula in a RangeMainFolder class describes how the questions should be answered and a RExpr class describes a range of how many answers should be answered “correct” to make this RExpr evaluate to true. The RangeMainFolder has a zero to many relation with the BOprnd class, which means that that the formula is not built up as a tree but as a list.

NEW WEB BASED @DVISOR SYSTEM

This part of the chapter describes the functionality that the new web based @dvisor contains in more detail and how these functionalities are implemented.

The behaviour of the new @dvisor system will be described by UML case diagrams and the structure of the program will be described by UML class diagrams.

USER INTERFACE

The user interface will be divided in 3 sections:

1. A section for the Node hierarchy.
2. A section for the Folder/Question hierarchy.
3. A section for the Folder/Expression hierarchy.

When a node is selected, the two other sections, question section and expression section, will update their content and show the content of the selected node in these two sections.

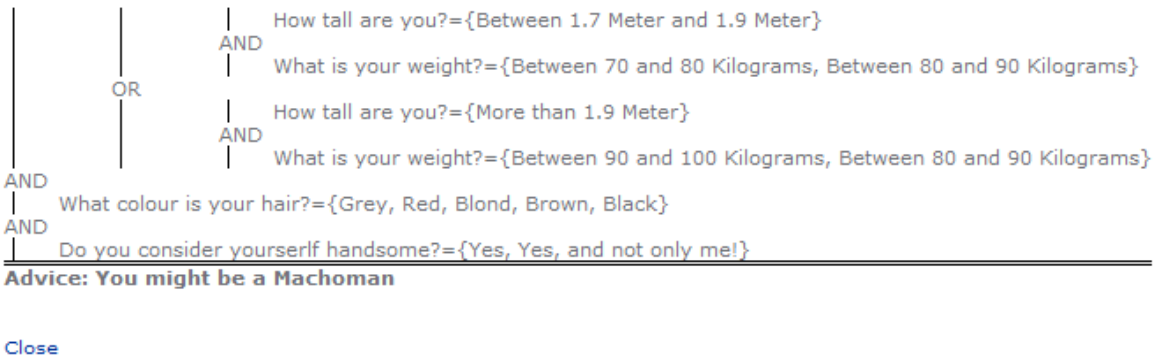
The main difference between this layout and the layout used by the original @dvisor system is that the complete knowledge base is shown in one page. All knowledge is accessible without switching tabs, windows, etcetera.

Nodes	Questions
Root Child 1 Child 2 Child 3	Folder 1 Checklist question 1 Checklist answer 1 Checklist answer 2 Yes/No question 1 Folder 2 Singular question 1 Singular answer 1 Singular answer 2
	Expressions Folder 1 Expression 1 Expression 2 Expression 3 Folder 2 Expression 1 Expression 2 Expression 3

On top of the window, a bar with buttons will be shown, with which experts can modify the knowledge base. In this bar are three buttons (create, modify/delete and copy). When an element in one of the sections is selected, the three buttons get disabled or enabled, depending on whether this button applies to the selected element. The create button will display a popup that provides functionality to create a new element. The modify/delete button will display a popup that provides functionality to modify and delete elements. The copy button will display a popup that provides functionality to copy an already existing element to this place.

Besides these three buttons, the bar also contain buttons to export a knowledge base, import a knowledge base, switch between multiple knowledge bases that are imported and a button for administrators to manage the user accounts that have access to the system.

When someone clicks on a Boolean expression, a production rule will become visible. The following screenshot contains the production rule how it is shown on the screen:



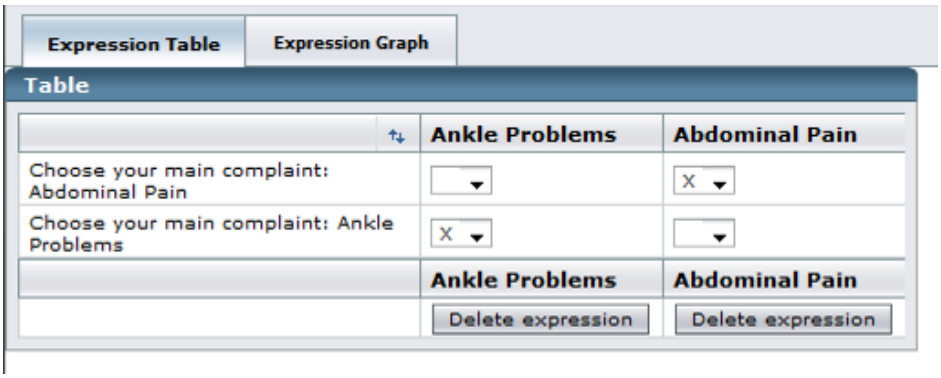
The bottom line contains the node to which this expression will lead (Advice: You might be a Machoman). The other lines contain the formula, the formula is built up by AND and OR statements, questions and the answers to these questions.

The leftmost vertical line contains the top level connectors of the formula (AND). On this top level there are 3 elements:

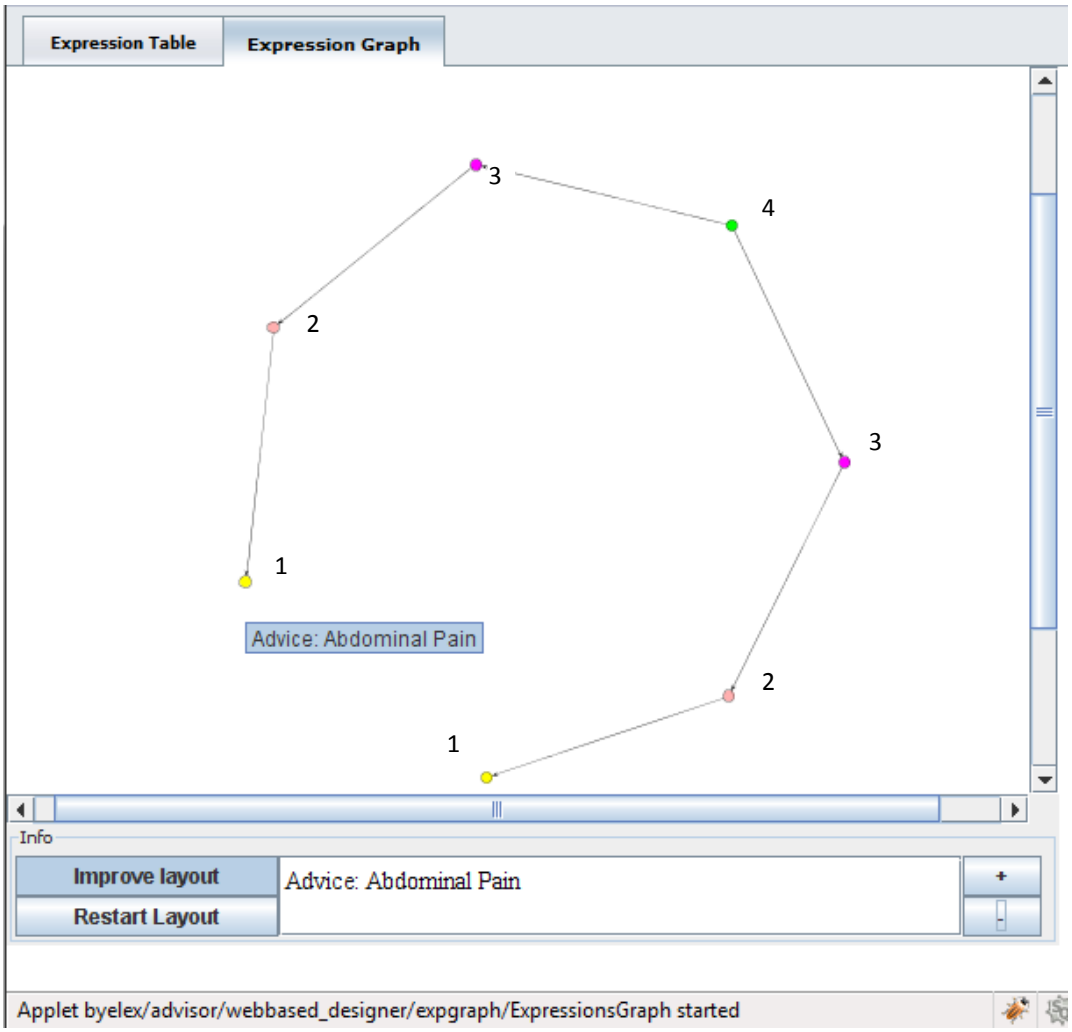
1. “Do you consider yourself handsome” which could contain the values “yes”, “Yes, and not only me!”.
2. “What color is your hair?” which could contain the values “Red”, “Grey”, “Brown”, “Blond”, “Black”.
3. A sub formula.

This sub formula has the OR connector which again contains two sub formula’s, etc.

When a folder in the expressions section is selected then a popup comes up with two tab frames. The first tab contains a decision table and the second tab contains a graph.



The screenshot above shows the decision table. In the columns the node is shown to which the expression leads, on the left the questions are shown that are inside the folder. The cells contain drop down fields with which it is possible to select the answer to this question. For example the “X” marks that the question should be checked.



The screenshot above shows the expression graph. The graph shows all expressions in a particular folder. By hovering over an element in this graph the text area on the bottom shows more information about the selected element. The colour of a circle reflects the type of item it represents (for monochrome prints the colours are also numbered).

The green circle (4) represents the folder that was selected. The folder, in the screenshot above, contains two paths, these paths are the two expressions within this folder. The two yellow circles (1) at the end of these two paths represent the destination nodes of these expressions. The magenta circles (3) represent the questions within the active folder and the pink circles (2) represent the answers to these questions.

The expression graph contains multiple paths from the green circle (4), which represents a folder, to the final yellow circle (1), that represents the target node. The expression diagram contains the same folder as the expression graph.

By using DNF (disjunctive normal form) the expressions in the graph are translated to the multiple expressions that are shown in the expression table. The paragraph about production rules describe DNF in more details. It suffices here to say that because a formula in DNF is built up from a collection of disjunctions and each disjunction is built up from a collection of conjunctions that each collection of conjunctions can be seen as a separate expression and therefore every column shows one collection of conjunctions.

SCENARIOS

The use case diagrams and scenarios in this paragraph describe the different actions that a user of the system is able to perform.

There exist 3 different actors within this system:

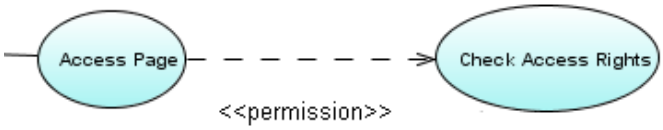
1. Not logged in users
2. A logged in expert
3. A logged in administrator

A user that is not logged is able to use the log in action, if the user has a correct account then the user becomes logged in. The logged in user becomes either a logged in expert or a logged in administrator, based on the user rights of the profile to which the user logs in. Every administrator can perform the same actions as an expert. So the functionality that is described for an expert also applies for administrators.

Association

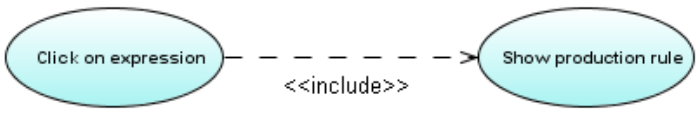
Means that the Actor on the left, labeled with User – not logged in, is able to perform the action on the right, labeled with Create Account.

Permission



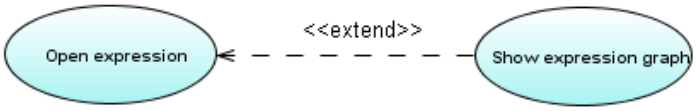
Means that the action on the left can only be performed if the action on the right grants permission.

Include



Means that the action on the left always uses the action on the right.

Extend

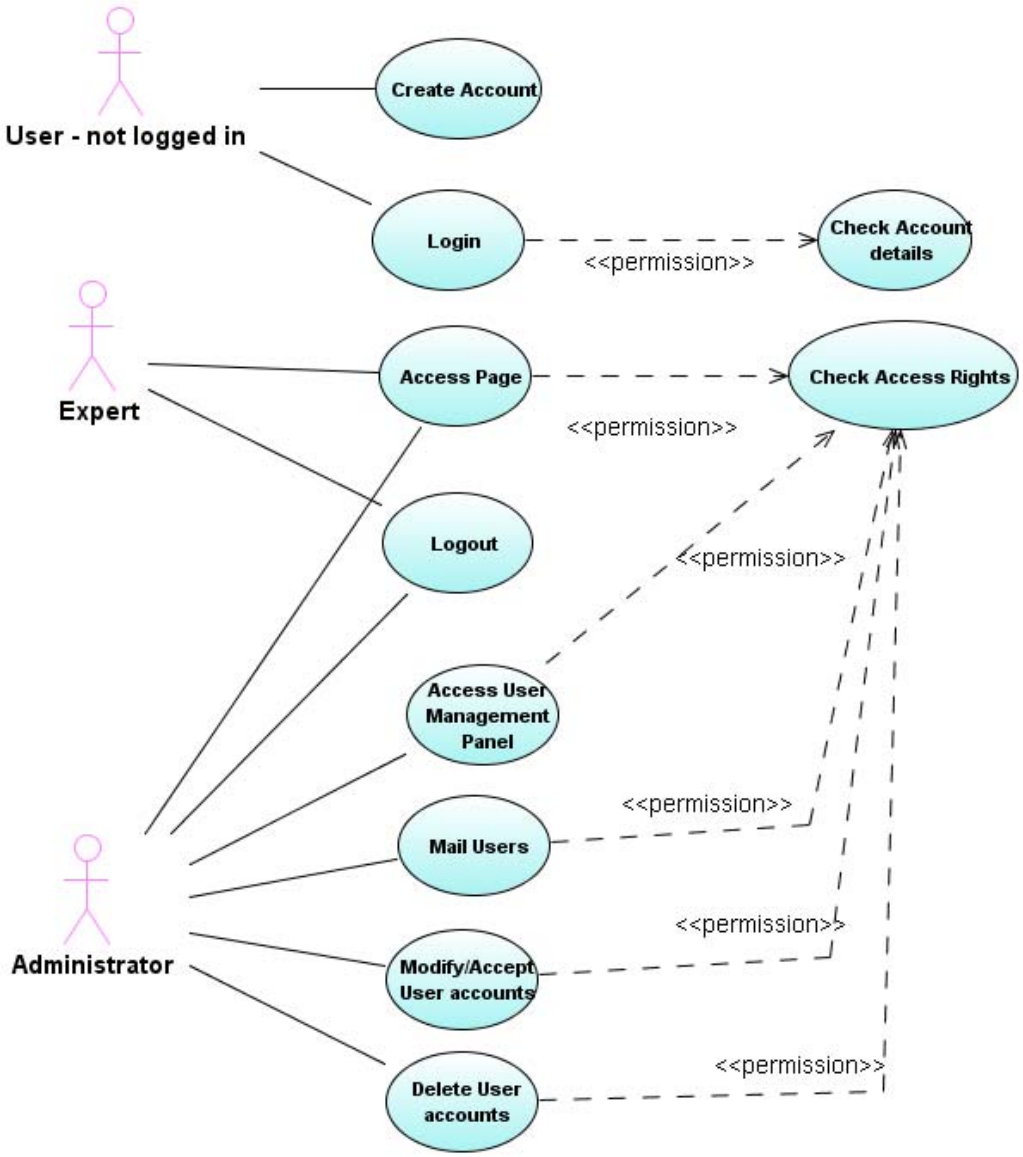


Means that the action on the left can be extended by the action on the right.

In the descriptions underneath the use case diagrams, words surrounded by quotation marks refer to use cases in the diagram.

USER ACCOUNT MANAGEMENT

In the following use case diagram the different actions that a not logged in user, an expert and an administrator can perform related to user accounts and the management of these accounts are shown.



When a user accesses the system, without being logged in, the user is able to perform two different actions: "Create Account" and "Login". The "Create Account" action allows a user that is not logged in to register for a new account. This account has to be accepted by an administrator, for it to become active. The "Login" action allows a user, that is not logged in, to login to an already existing and accepted account. The "Login" action first have to be granted permission by performing the "Check Access Rights" action, if granted permission the user will be logged in. Based on the account details the user either becomes an Expert user or an Administrator user. This state change is not shown in the Use case diagram above.

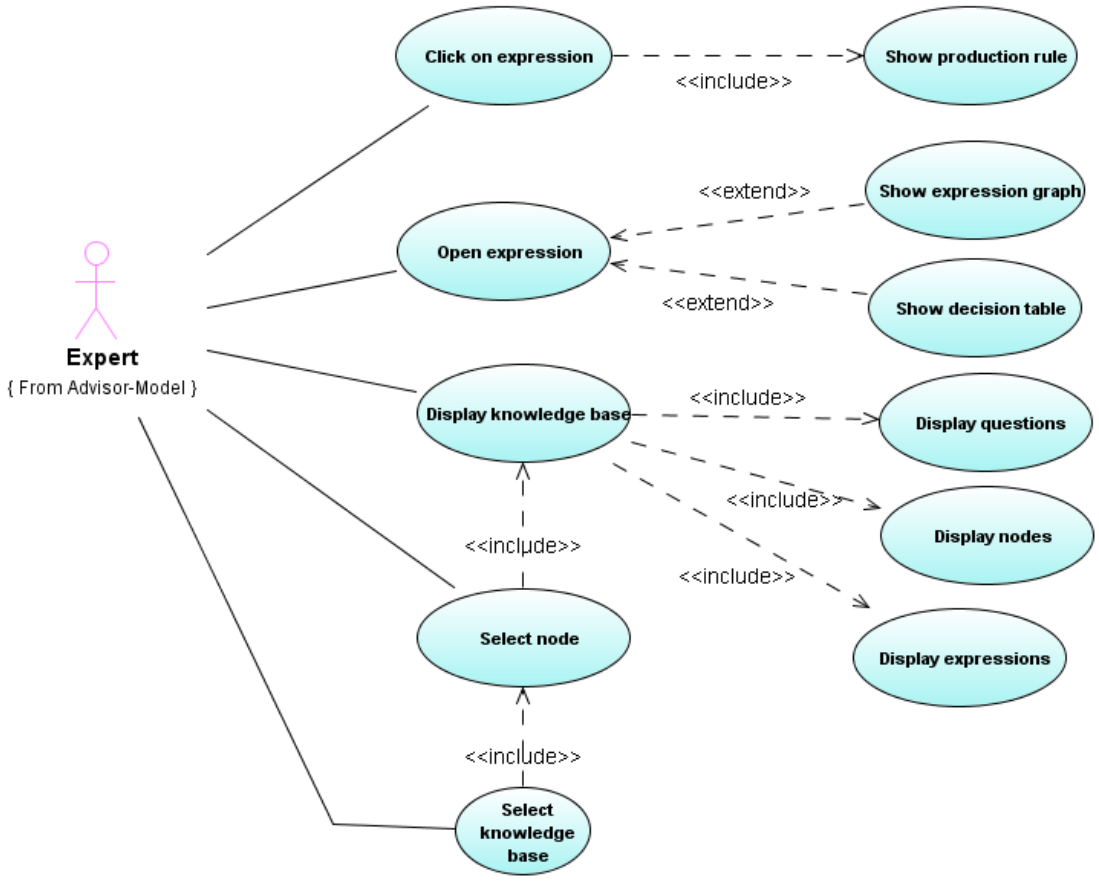
Both experts and administrators are able to perform the “Access Page” action. This action performs the opening of a secured page inside the system. To be able to execute this action, the system first performs the “Check Access Rights” action to check if the current user has sufficient permissions to open the requested page. Experts and administrators are also able to perform the “Logout” action. By performing this action the system will end the session of the logged in user.

An administrator is a user that is able to perform a number of additional actions related to user account management, these actions are:

1. “Access User Management” displays the user management panel.
2. “Mail User” sends a mail to another user
3. “Modify User” modifies the account details of a user, for example an administrator can enable a user account
4. “Delete User” removes a user from the system.

DISPLAYING THE KNOWLEDGE BASE

The following use-case diagram describes how an expert can display the knowledge inside a knowledge base. To be able to perform the actions described in the use case, the system requires an expert to be logged in to the system, as described in the previous paragraph.



When an expert enters the system for the first time, no knowledge bases are selected and therefore no knowledge bases are shown. When the expert performs the “Select knowledge base” action, the system will open the selected knowledge base. When a knowledge base is selected, the system also has to select a node, therefore the “Select node” action is included. This action will select the root node of the knowledge base. Via the include relation the system will then perform the “Display knowledge base” action. This action shows the root node of the knowledgebase to the expert. The “Display knowledge base” action includes 3 other actions “Display nodes”, “Display questions” and “Display expressions”, these 3 actions display the nodes, questions and expressions that occur in the currently active node.

If the expert returns to the system on a later time, the “Display knowledge base” action will be performed automatically to show the previously selected knowledge base and the last active node to the expert.

If the expert performs the “Select node” action, the system will select a new node. The “Select node” action includes the “Display knowledge base” action, which will update the screen with the newly selected node and the child nodes, questions and expressions that occur within the selected node.

When an expert performs the “Click on expression” action, the system will include the “Show production rule” action, which will provide more information about the selected expression.

When an expert selects an expression and performs the “Opens expression” use case, a popup will be shown. This popup will either perform the “Show decision table” action or the “Show expression graph” action, depending on what tab panel is opened.

MODIFY THE KNOWLEDGE BASE

The use case diagram below shows the different actions that an expert of the system can perform to modify the knowledge in the knowledge base.

When an expert starts the system for the first time, the system will not contain any knowledge bases. Therefore, the first and only action that an expert performs will be the “Import knowledge base” action. When the system contains a knowledge base, then the other actions in the system become available.

A knowledge base is created per expert, an expert can only modify its own knowledge base and other experts will not be able to see these changes. The expert also is able to perform the “export the knowledge base” action.

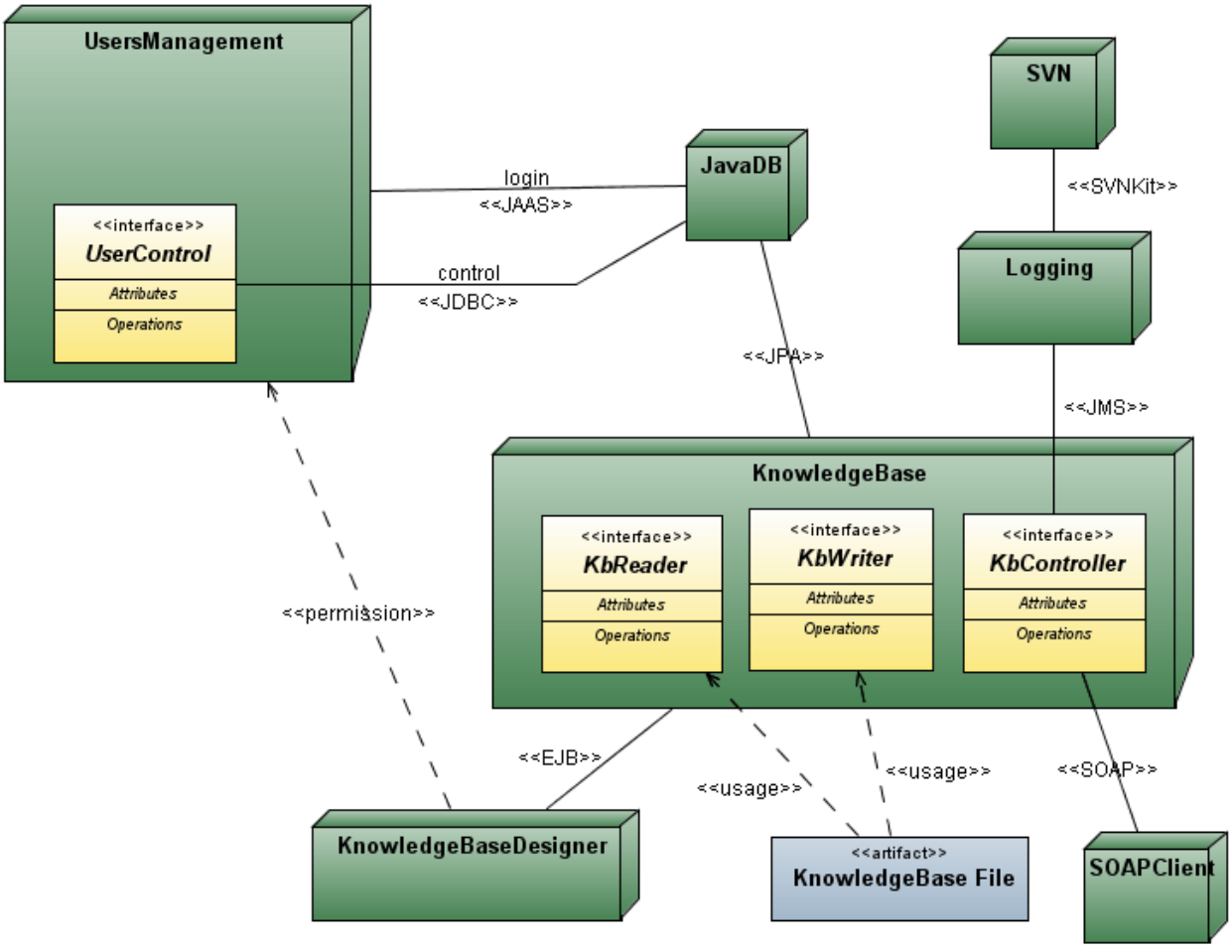
When an expert performs the “delete knowledge base” action, the system will remove the knowledge base and it will also automatically perform the “delete node” action on the root node of the knowledge base. By removing a node, the system will remove all child nodes and folders inside this node. When a folder is deleted, the system will also remove the expressions/questions inside the folder, etc. So when the knowledge base is removed, all elements inside this knowledge base will be removed automatically.

The expert is also able to create/modify/delete a node/folder/linked advice/question/etc. manually.

A node can also be moved to a new location. When the expert performs this action, the system will first check whether the new parent node is a child of the moved node or one of its children. This is done to prevent cycles in the knowledge base which will result in deadlocks in the inference engine. The expert also is able to clone questions to other folders/nodes.

SYSTEM DESIGN

The following deployment diagram describes the different subsystems of the @dvisor web based application.



The diagram, on the previous page, shows multiple nodes (the green boxes). Each node can be placed on a different computer, but it is also possible to place multiple nodes on the same computer. This makes the system more scalable for situations with large amounts of experts that use the application simultaneously. As application server, the open source Glassfish v2 Java application server is used. This application server is based on the Java EE 5 specifications and provides a web server that is able to run Java code to build up web pages.

The Java EE 5 specification is an industry standard for Java application servers. The specification describes particular features that should be included in a server before it can be called Java EE 5. Some of these features are used in this project and are needed to be able to run the application (JPA, JMS, EJB). Besides Glassfish, there are a number of other Java EE 5 certified application servers, for example BEA Weblogic, SAP Netweaver, Apache Geronimo, IBM Websphere, Oracle Containers for Java EE.

Glassfish is used because it is one of the more feature-rich application servers that performs acceptably in most benchmarking tests, while it is an open source system, and therefore cheap to use. Another advantage of the Glassfish application server is that it is maintained by Sun Microsystems, the developers of Java. Because of this, new functionality will be implemented into the Glassfish application server first, as a method to showcase the functionality and to provide a point of reference for other servers. Sun also provides, paid, support for users of the Glassfish server. This support sometimes is required by larger companies.

The Knowledge base node is the central node of the system, which contains all the functionality for reading, writing and modifying the knowledge base. The entire system is based on the MVC pattern (model, view, control). The Knowledge base node contains all the Model classes. These classes build up the knowledge base.

The Knowledge base node communicates with other nodes via the KbController interface. This interface is the main Controller class of the system. The KbController interface is a Java EJB class. Enterprise JavaBeans (EJB are objects that live independently from the application in the application server. Thus they can be accessed by multiple applications without being located on the same physical machine.

By using the EJB technology, other nodes can communicate with the KbController class, as if the class is in the same application and even on the same computer (which may not be the case). This makes it much easier for a developer to build scalable applications, as the developer does not need to take socket connections, client/server interfaces, etcetera into account.

There exist three types of EJB classes, stateful and stateless and message driven beans. Stateful beans are persistent and thus keep their variables in a session. The garbage collector will destroy the stateless beans instance after the page has been loaded. Therefore stateless beans cannot be used to store variables that can be used on several different pages.

Stateless and stateful beans provide synchronous communication, therefore the object that calls the bean has to wait until the bean has finished its operations. Message driven beans provide asynchronous communication, therefore the calling class can proceed with its work immediately after it called the message driven bean. Even when the message driven bean still is busy with its work. Like stateless beans, message driven bean do not keep their variables in a session.

The KbController class is a stateless bean, that provides functionality to store and collect information from the knowledge base. It does not need to remember any data itself because the knowledge base itself will be stored by other mechanisms, that will also provide caching mechanisms.

The Knowledge base node is able to import and export a knowledge base. A knowledge base is stored in an XML file that retains the structure of the XML file of the original @dvisor system. By using the same XML structure, it is possible to use the exported/imported XML files in other applications of the @dvisor system.

The knowledge base is stored in a database. In this edition of the system a simple JavaDB, based on Apache Derby, will be used. The JavaDB database is written in Java and is very light weight (only a 2 MB memory footprint). JavaDB is used during the testing period because of this light weight.

Connection with the database is made via a JDBC connection. JDBC is a technology that provides the communication between Java and a database server. JDBC consists of a database-independent API and a database-dependent JDBC driver. The database independent API makes it possible to replace one database by another database without any modifications to the Java application, given that this other database provides a JDBC driver. This is useful in case the system provides a more feature-rich or scalable database, like Postgresql or Oracle, to replace the basic JavaDB database.

For storing data to the database and collecting data from the database the JPA technology is used. The JPA technology (Java Persistance Api) provides an ORM coupling (Object Relational Mapping), by which the JPA system will automatically generate SQL queries, Caching operations etcetera.

In a class, annotations will describe the mapping of variables to database fields, the cardinality of relations to other classes, etcetera. With these annotations, the JPA system is able to automatically create the database structure and upon saving, the JPA system will be able to create database queries to store the objects into the database. The JPA system makes it much easier to develop and extend the system. The JPA technology will be used inside the web based @dvisor for storing the knowledge base inside the database.

Toplink, the JPA implementation by Oracle, will be used inside the web based @dvisor because this implementation is combined with the Glassfish v2 application server. There also exist different JPA implementations like Hibernate and OpenJPA.

Besides being an EJB class, the KbController also provides a SOAP interface, Simple Object Access Protocol. SOAP interfaces make it possible to write applications that connect to the knowledge base node via the world wide web. SOAP is an XML protocol that is used by applications to communicate with each other. With SOAP it is possible to send an object to another application, even when the applications are written in different languages or are installed on different operating systems.

Because the SOAP protocol uses port 80, the same port that is used by HTTP, it is possible to use the SOAP connection via most proxies/firewalls. EJB technology uses different ports and protocols, i.e. CORBA, which makes it difficult to use EJB via proxies/firewalls. This makes it harder to use this technology for the same purpose over the web.

Via a SOAP interface, external applications can be developed that are able to display and modify knowledge from the knowledge base. In this project, an applet is developed that shows a graph containing a folder and its expressions. The applet belongs to the View part of the web based @dvisor.

The KnowledgeDesigner node contains the other View classes, that provide the functionality to display the knowledge base on the screen and provide controls to modify the knowledge base. The actual modifications are done via an EJB connection to the KbController.

The pages in the KnowledgeDesigner node are made with the JSF technology (Java Server Faces). JSF is a technology that is based on JSP (Java Server Pages) that is built on an XML file where the XML tags are bound to Java functions in backing Java classes. With this, the Java classes can provide the input to build up the JSF pages.

The KnowledgeDesigner node can only be accessed by a user if the user received sufficient permission from the UsersManagement node. The UsersManagement node provides functionality for logging in to the application by experts and maintaining user accounts by administrators. The maintenance functionality consists of accepting accounts, assigning an account to a user group, removing an account and mailing an account.

Logging in proceeds via the JAAS technology (Java Authentication and Authorization Technology). The JAAS technology provides functionality for users to authenticate to the system and when a user accesses a page the JAAS system will grant authorization to the system, or not when the user does not have sufficient access rights.

Because the JAAS system is built inside the application server, security via this mechanism is much more reliable. Authorization to the system is done via user groups. The accounts and groups are defined in a security realm. There exist a number of different security realms. The difference between these realms differ from each other by how the realm scores its user accounts, for example there exist a realm that stores its accounts in a database or in files. Because of scalability, the web based @dvisor uses a database based security realm.

The SVN node (Subversion) is used for storing/logging changes in the knowledge base. SVN is a revision control system used for maintaining files and changes in files. It is possible to track which changes are made and revert a change to an earlier situation. SVN is chosen, above CVS, Bazaar or any other revision control system.

There exist Java libraries for both SVN and for CVS, which makes it easier to combine one of these systems with the web based @dvisor. Bazaar is a quite new revision control system and therefore Java libraries are not yet developed for this system. Advantages of SVN over CVS is the possibility to commit files via transactions. If a commit action would fail, the entire transaction will be marked as failed and the SVN server will rollback the transaction. CVS will not rollback a failed commit and therefore the revision control system will be corrupt until the system is repaired manually. Because of the availability of transactions in SVN, SVN is chosen as revision control system for the web based @dvisor.

SVN servers operate by committing an initial version of a file to the server. Every time the expert commits a new version of the same file, the server will calculate the differences between the previous committed version and the current committed version. Only these differences will be stored, instead of the complete new version. This storage method, called delta compression, decreases the amount of data stored in the repository, i.e. the place where the revision control system stores its data.

Committing a file to the SVN repository can take quite some time, therefore an asynchronous communication method between the KnowledgeBase node and the Logging node should be used. Synchronous communication would be to slow for committing data to the SVN server. The Logging node therefore contains a message driven bean. For communication with a message driven bean the Java application server uses the JMS mechanism (Java Messaging System).

JMS is an asynchronous communication mechanism built into Java application servers. By using JMS, committing a change to SVN happens in the background, without delaying the web based system.

The KnowledgeBase node contains a stateful bean that communicates to the Logging Node. The Logging Node will commit the knowledge base to the SVN server. The stateful bean, LoggingBean, stores the login credentials to the SVN server in its session. The KbController bean will communicate with the LoggingBean and the LoggingBean communicates with the Logging node. The KbController and the Logging Node do not contain sessions and therefore cannot store the login credentials to the SVN server.

The data sent to the SVN node consists of the following data:

- The knowledge base XML file
- Feedback entered by experts
- System generated logs, tracking changes made to the knowledge base

The knowledge base XML file is stored in the SVN repository and the expert feedback and system generated logs are stored as comments to the commit. Every expert has a personal SVN repository for all its knowledge bases.

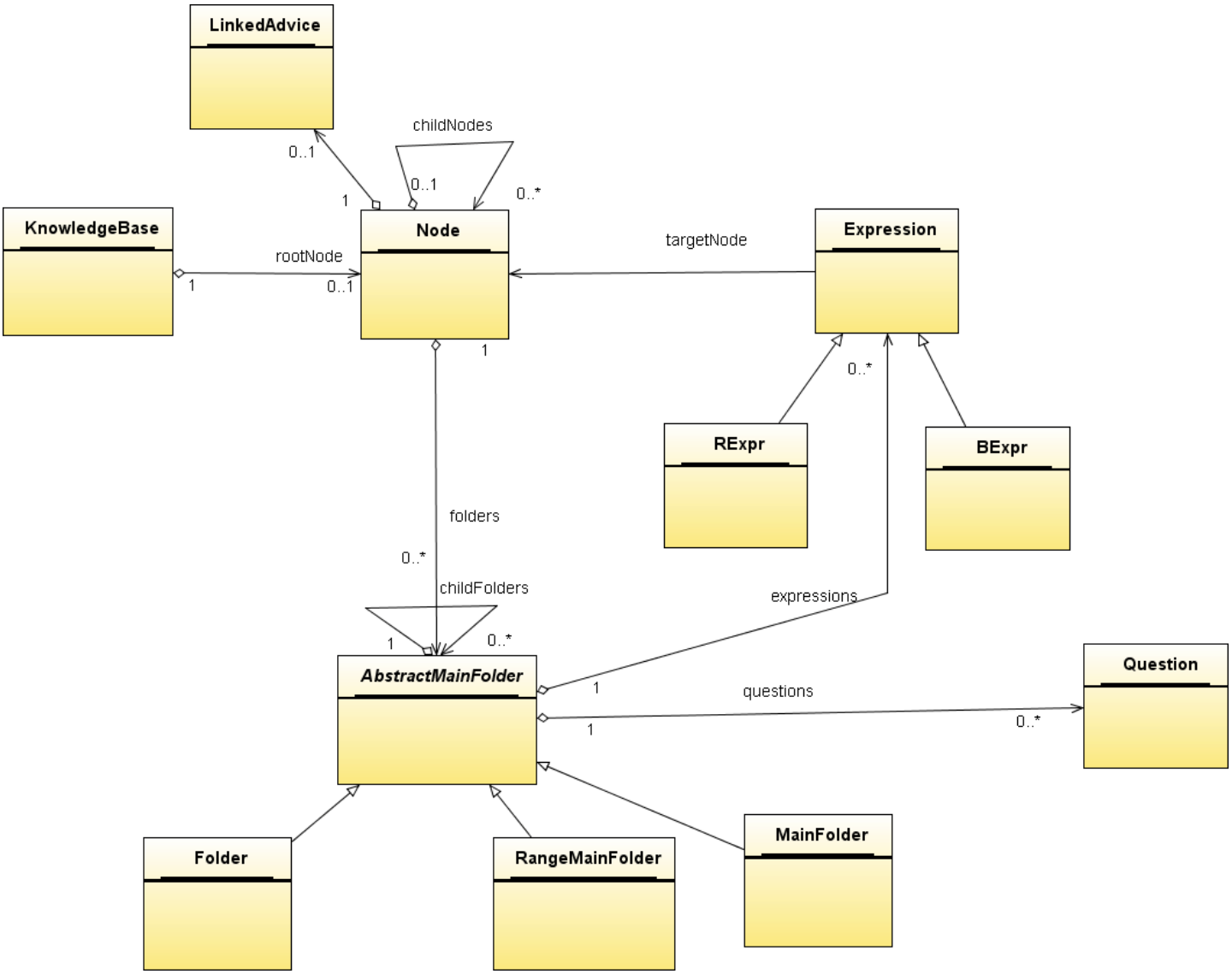
The SVN repository can be accessed with a standalone SVN client. With a standalone SVN client, an expert is able to retrieve the latest version, or earlier versions, of the knowledge base from the SVN repository. A repository is a place inside a SVN server that contains the knowledge base files.

OBJECT DESIGN

The diagram on the next page describes the structure of the Model classes in the Knowledge base node. The subclasses of the Questions and Expressions classes are taken out of this diagram to reduce the complexity, but the subclasses are described in a separate diagram.

The structure is quite similar to the class diagram of the original knowledge base, but the class diagram has some optimizations for presentation and processing by the JPA platform.

When the program imports a new knowledge base, the original knowledge base structure is transformed into this new structure. When the program exports a knowledge base, the new knowledge base structure is transformed into the original structure.

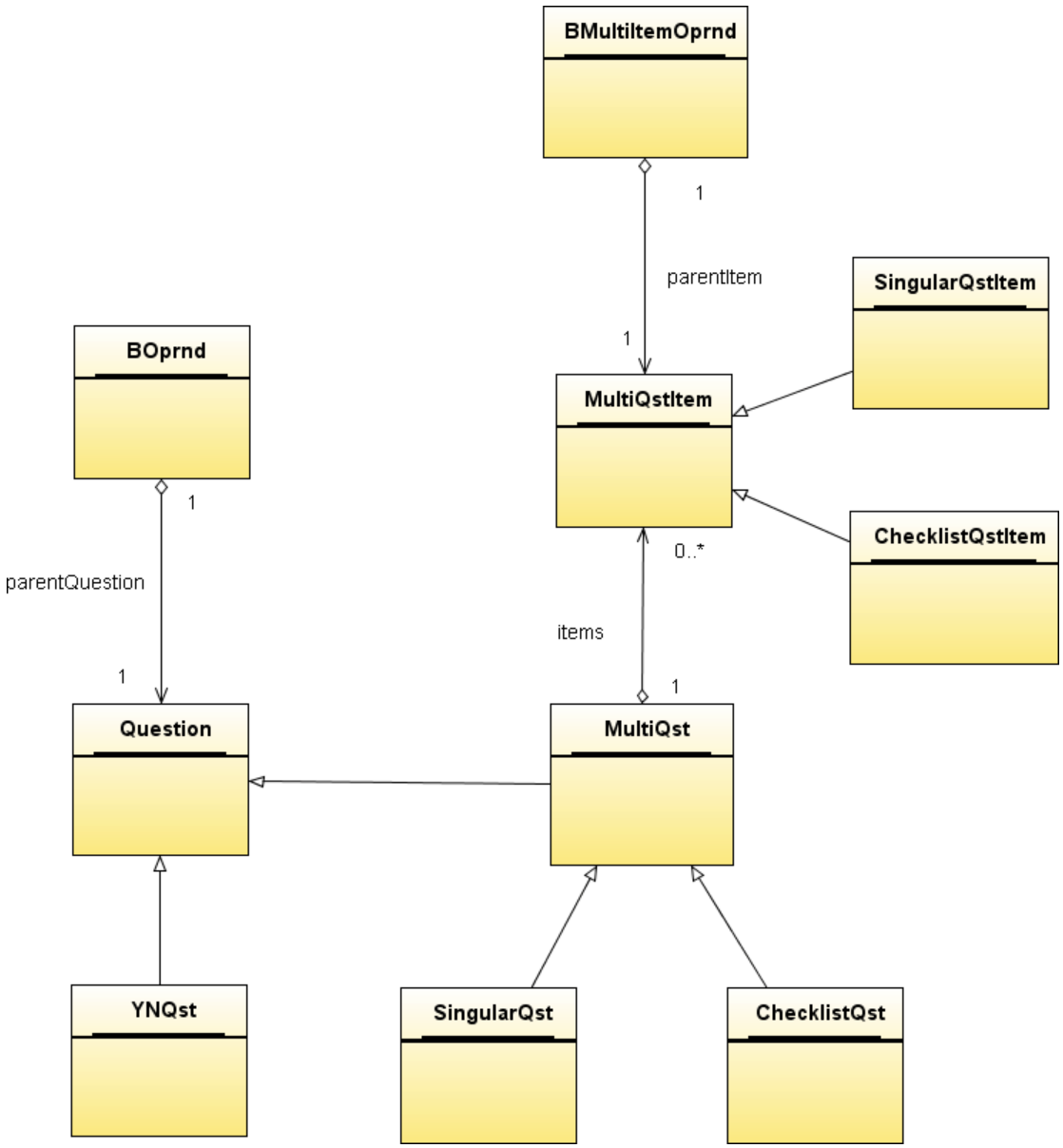


The KnowledgeBase class is similar to the Document class from the original structure, the class contains the entire knowledge base. An instance of the KnowledgeBase class can contain an instance of the Node class, this instance of the Node class forms the root node of a node graph. An instance of the Node class can contain an instance of the LinkedAdvice class, this LinkedAdvice represents a more detailed description of a node, this information is shown when the node is used as an advice.

An instance of the Node class can contain zero or more child nodes, instances of the same Node class. These child nodes represent nodes that can be reached from the node, via expressions. Every node in the node graph has to be connected to the root node and the root node cannot have a parent node. In the original knowledge base structure, the Node functionality was built with 3 different classes (ClfNode, DMNode, ClfClass). In the original knowledge base structure this division in 3 classes was necessary because of the inference engine and because the structure is saved as XML file. This new knowledge base structure is used for modification and presentation of the knowledge base and therefore the structure is simplified on many points.

Besides child nodes and an instance of the LinkedAdvice class, an instance of the Node class can contain zero or more instances of the AbstractMainFolder. The AbstractMainFolder contains expressions and questions. The AbstractMainFolder class is the parent class of three classes: MainFolder, RangeMainFolder and Folder. The MainFolder contains Boolean expressions and and the RangeMainFolder contains ranged expressions. The MainFolder and RangeMainFolder can have child folders, these child folders are from the Folder subclass of the AbstractMainFolder.

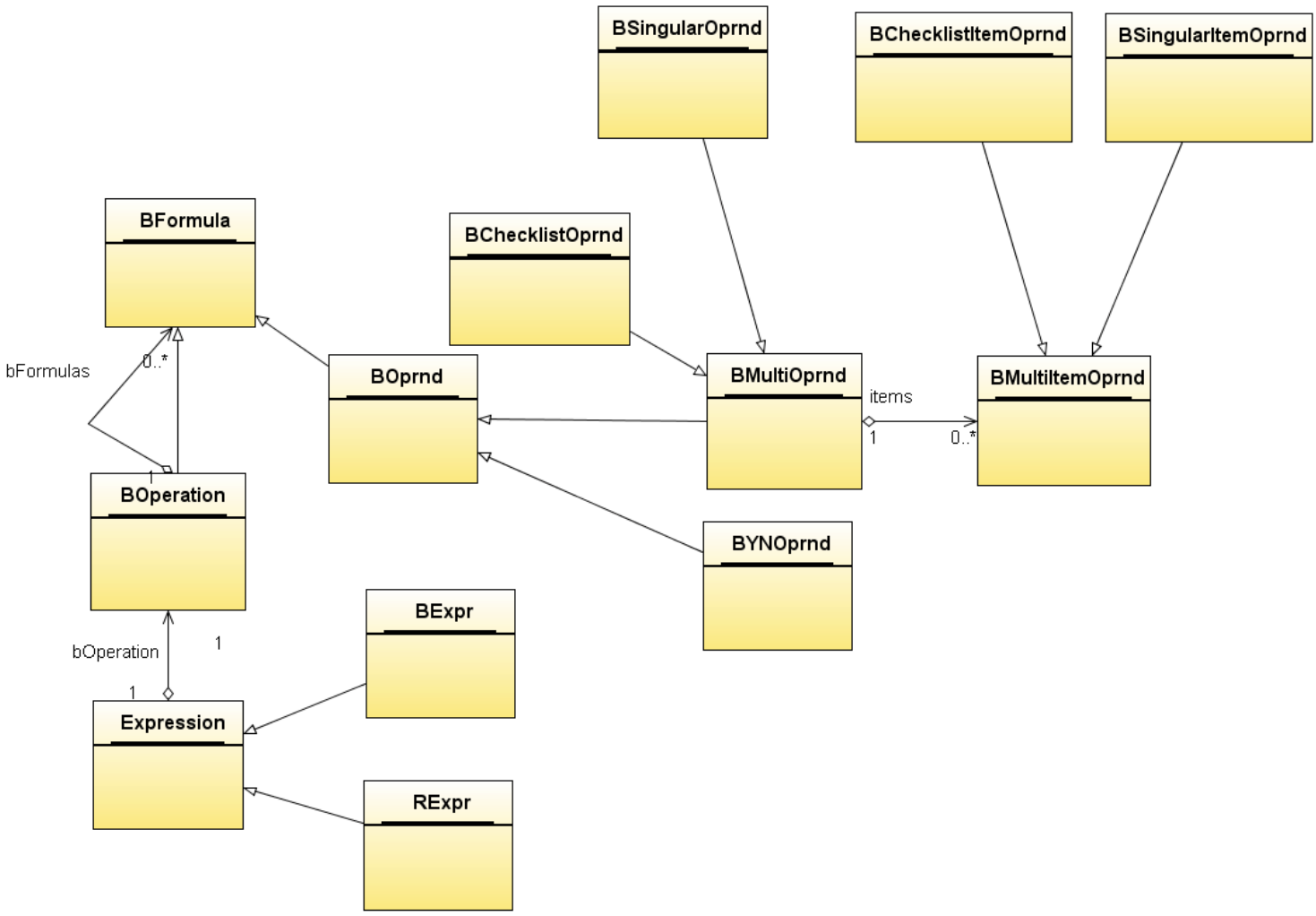
An instance of the AbstractMainFolder contains zero or more instances of the Question class and zero or more instances of the Expression class. The Question class is a parent class of various different questions types. The Question class and its subclasses are described in more detail in the following class diagram:



The Question class has a similar structure as the original class diagram. The main difference is that the Question class has a one to one relation with the BOprnd class and the MultiQstItem class has a one to one relation with the BMultiItemOprnd class. In practice this means that a subclass of the Question class has a relation to a similar subclass of the BOprnd class (e.g. an instance of the YNQst class will have a relation with an instance of the BYNOprnd class) and that a subclass of the MultiQst class has a relation to a similar subclass of the BMultiItemOprnd (e.g. an instance of the SingularQst class will have a relation with an instance of the BSingularOprnd).

The first class diagram, of the new knowledge base structure, shows that the Expression class has a one to one relation with the Node class. The Node class, with which an Expression class shares a relation, is the target node that will become active when this expression becomes true.

The Expression class is a parent class of 2 Expression types BExpr and RExpr, which are the Boolean expressions and Ranged expressions. The BExpr_Meta class from the original knowledge base structure is not built into the new knowledge base structure. This is due to the simplified structure of the new knowledge base structure, this simplified structure can be seen in the class diagram below.



The BOprnd class and its subclasses have a similar structure as the BOprnd class in the original knowledge base structure had. The main difference with the original knowledge base structure, is that the BOprnd class and the BOperation class are subclasses of the BFormula class. These classes in the original knowledge base structure where related, but didn't share the same parent class. The BOprnd class represents an answer to a question and the BOperation class represents a Boolean Operator (AND, OR, XOR).

Another difference in this new knowledge base structure, is that the RangedMainFolder class does not have a direct relation to the BOprnd class. All Expressions are built up by the same structure, an instance of the Expression class has one BOperation. If the Expression class is of the RExpr type, the BOperation will always be an AND Boolean operator, otherwise the BOperation can be an AND, OR or XOR Boolean operator.

A BOperation can contain zero or more subclasses of the BFormula class, this makes it possible to built complex expression trees. If the Expression of the RExpr type, the BOperation that is contained in the RExpr class will only contain subclasses of the BOprnd class and no other BOperation classes. This constraint is not shown in the class diagram.

3. TEST RESULTS

This chapter describes the testing of the web based @dvisor system. All tests listed below where executed successfully, some after a number of bug fixes.

TESTING THE IMPORT AND DISPLAY MECHANISM

In this test the importing and displaying mechanism of the web based @dvisor system will be tested by importing some knowledge bases and display these on the screen. The knowledge displayed on the screen will be compared with the original Knowledge base Designer.

The following knowledge bases from the @dvisor tutorial will be imported and displayed:

1. Machometer
    - *a root node with several advice nodes that are nested under this root node*
    - *the root node contains several folders*
    - *the folders contain several singular and checklist questions*
    - *the folders contain several Boolean expressions*

2. Machometer 2
    - *a root node*
    - *inside the root node are several sub nodes*
    - *every sub node contains several advice nodes*
    - *the root node and sub nodes contain several folders*
    - *the folders contain several singular and checklist questions*
    - *the folders contain several Boolean expressions*

3. Are you romantic? check
    - *a root node*
    - *several advice nodes*
    - *the root node contains several folders*
    - *the folders contain several singular questions*
    - *the folders contain several ranged expressions*

The following real world knowledge base made for medical examination will be used

4. What is the problem
    - *a root node*
    - *multiple levels of sub nodes*
    - *several advice nodes*
    - *the root node and sub nodes all contain one folder*
    - *the folders contain several singular and checklist questions*
    - *the folders contain several Boolean expressions*

5. The following knowledge base contains all the things not tested yet:

6. Linked advice, multiple languages, yes/no questions, sub folders

These knowledge bases will be loaded into the system and all pages will be checked. This checking consists of opening all pages, comparing them to the advisor software and verifying if the expressions are the same. If all elements of the knowledge base are equal then the test is successful.

TESTING THE MODIFICATION AND EXPORT MECHANISM

In this test, the modification and exporting mechanism of the web based @dvisor system will be used for remaking the same knowledge bases from the first stage and exporting them. The exported @dvisor files will be loaded into the original Knowledge base Designer and these files will be compared with the versions that are made with the Knowledge base Designer. If the knowledge bases are equal, then the test is successful.

The same knowledge bases as used in the previous test will be recreated, except the "what is the problem" knowledge base that is too big to recreate:

1. Machometer
2. Machometer 2
3. Are you romantic? check
4. A simple knowledge base that contains all the things not tested yet (Linked advice, multiple languages, yes/no questions, sub folde)

TESTING THE SUBVERSION REPOSITORY

This test will test the logging to SVN mechanism. The final test in the previous section:

1. *A simple knowledge base that contains all the things not tested yet (Linked advice, multiple languages, yes/no questions, sub folders)*

will be used in this test. All the modifications will be verified with the SVN logs and the final version when the knowledge base is completed will be exported with the export tools and will be compared to the last version in the SVN repository. If the knowledge bases are equal then the test is successful.

TESTING THE SOAP INTERFACE

All elements of the knowledge base can be created/modified with the soap interface. A script will be made that will test the interface like this:

1. For every element in the knowledge base (node, folder, question, etc) the script will run the create function.
2. Then the script will run the retrieve function to retrieve the created element.
3. The script will test if the created element is equal to the retrieved element.
4. The script will then run the modify function to change a number of elements of the created element.
5. The script will run the retrieve function to retrieve the created element.
6. The script will test if the created element is equal to the retrieved element.
7. The script will run the delete function of the created element.
8. The script will run the retrieve function to try to retrieve the created element.
9. The script will test if the retrieve function returned NULL.

If all steps succeed without errors then the test is successful.

TESTING THE MULTI LANGUAGE FUNCTIONALITY

A knowledge base can be translated into multiple languages. This test will test a knowledge base in multiple languages and will test the system if it can handle special characters and different character sets.

For this one of the knowledge bases in the first test “machometer” will be tested with the following settings:

1. Knowledge base in English
2. Knowledge base in French (for characters like è, é, ê, etc)
3. Knowledge base in Russian (for Cyrillic script)

The following actions will be done to test this multi language functionality:

1. Knowledge bases in these languages will be imported.
2. Knowledge bases in these languages will be exported.
3. Knowledge bases in these languages will be displayed by the web based system.
4. Knowledge bases in these languages will be exported from SVN.

The test will be successful if all these actions can be done without errors and encoding mistakes.

TESTING THE SYSTEM FOR MULTIPLE SIMULTANEOUS EXPERTS

The system should be able to be used by 100 simultaneous experts and therefore the system will be “stress tested” by using the JMeter tool. The tool will do the following actions:

1. Log in a user
2. Open a node
3. Open a folder
4. Open an expression
5. Open a question
6. Log out

These actions will be done by 100 simultaneous users that will log in 1 second after another. The test will be successful if all actions can be finished without errors but also within reasonable time (max 3 seconds per page).

TESTING THE SECURITY OF THE SYSTEM

The system will be tested on several security issues. These issues are:

1. Cross site scripting (XSS)
   Injection of javascript code by the expert

2. SQL injection
   Injection of SQL queries

3. Session hijacking
   Stealing the session of a logged in expert

4. Accessing pages without being logged in
   Testing if all pages are protected

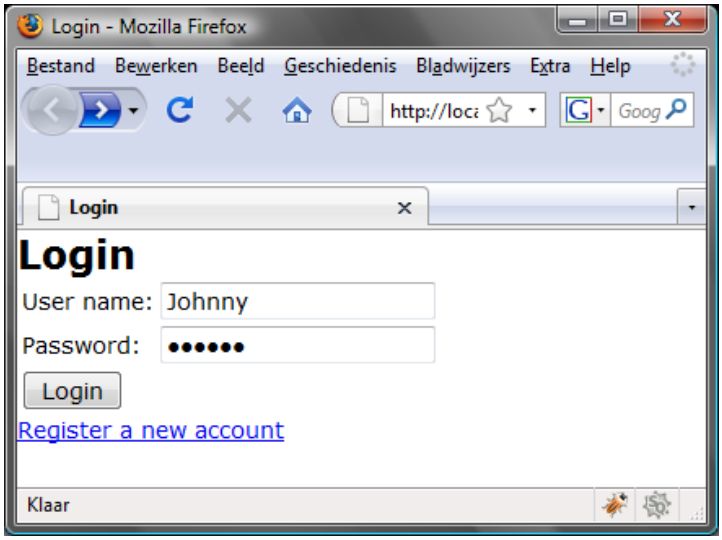
5. Accessing pages that should be not accessible by the logged in account
   Testing if all pages have the proper access rights

If all these tests succeed then the test is successful.

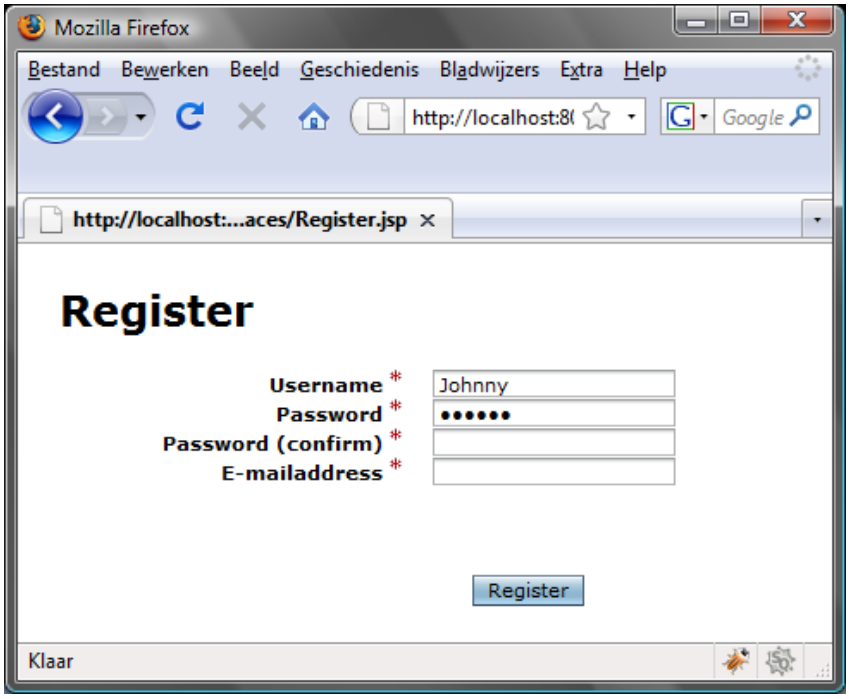
4. END RESULT

This chapter will display screenshots made in the web based @dvisor application to describe the different functionality of the system.

The following screen contains the login page. When an expert enters a correct user name and password combination the application will log in to the application.

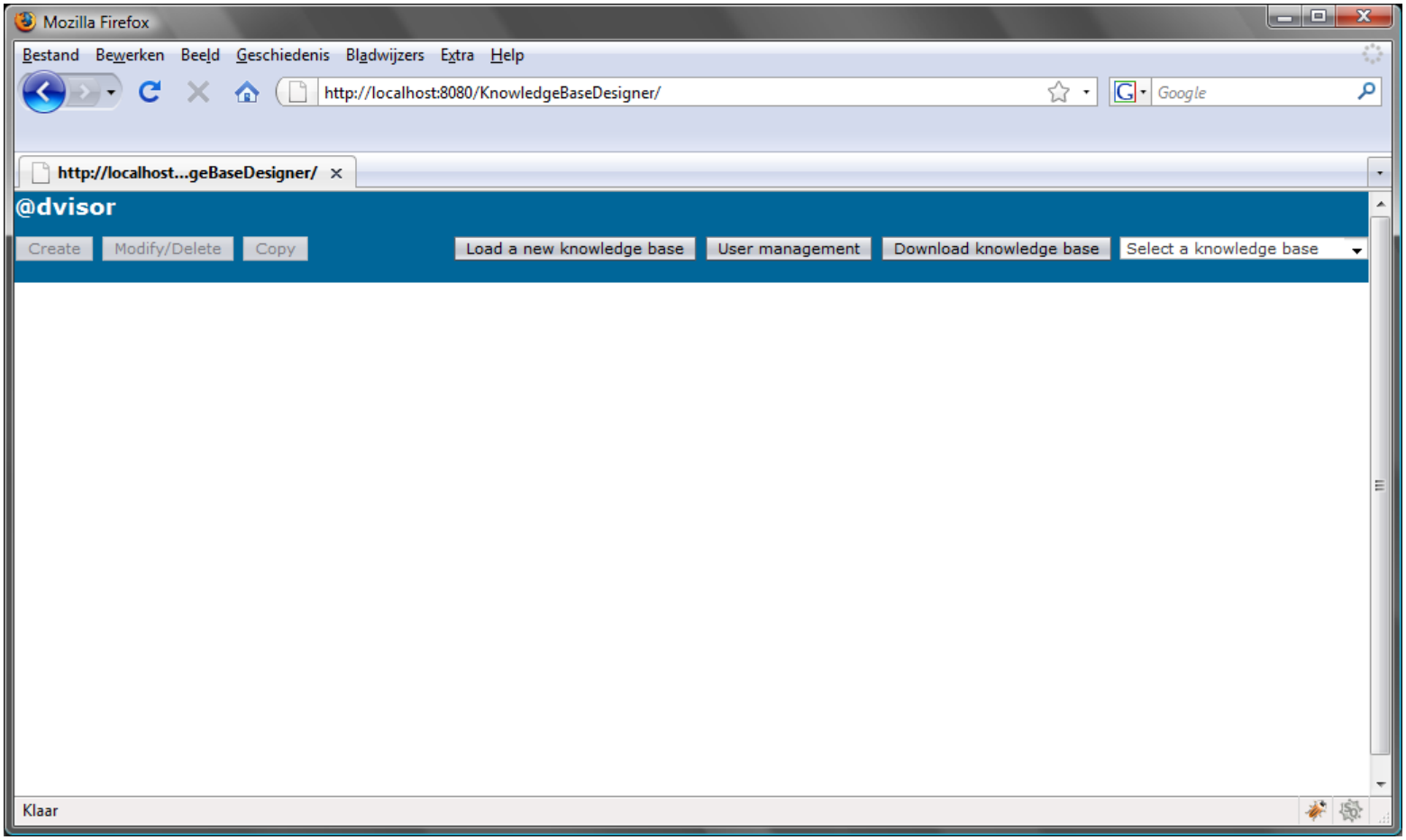


When an expert clicks on the “register a new account” the application will navigate to the following registration page:

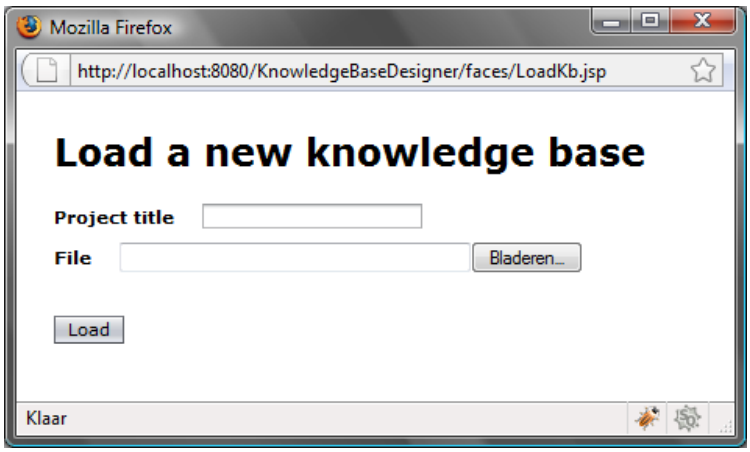


The system requires a unique user name, a password consisting of at least 6 characters and a correct e-mail address.

When an expert is logged in to the application, for the first time, the following screen is shown:

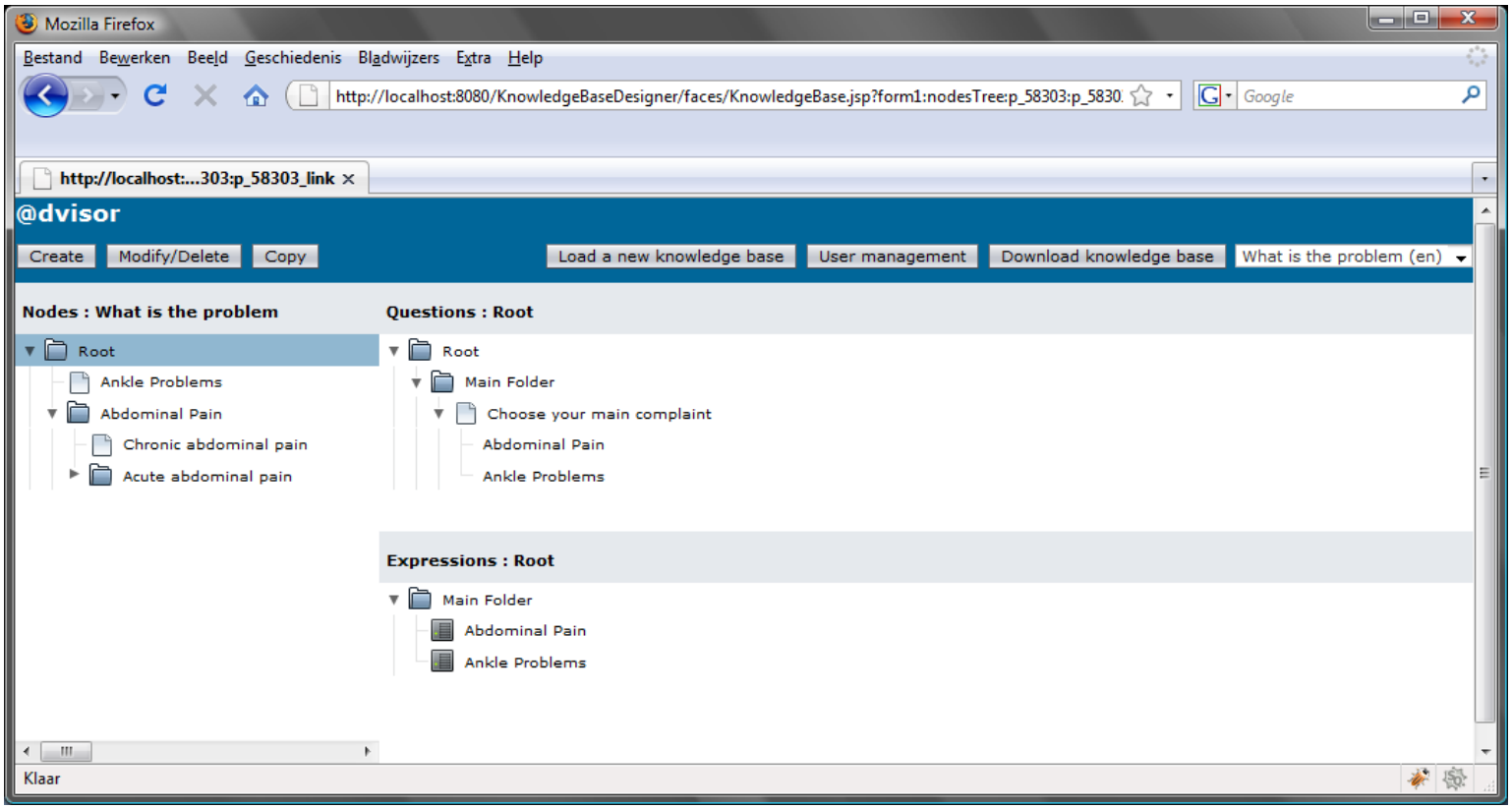


Because it is the first time that the expert logged in the system currently does not contain any knowledge bases and therefore a new knowledge base has to be loaded via the “Load a new knowledge base” button. By clicking on the button the following popup shows up:



When an expert fills in the name of an already existing knowledge base then this knowledge base will be overwritten. When the name of a not existing knowledge base is entered the system will create a new entry.

In the main window the added knowledge base will show up within the drop down list on the right. By selecting the knowledge base name from this drop down list the knowledge base will be loaded and displayed.



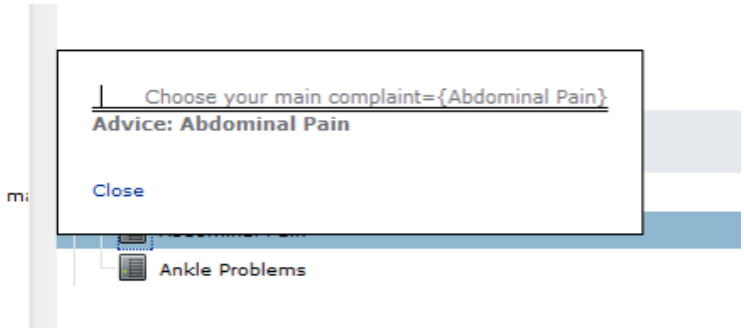
The top part of the window contains the controls with which it is possible to modify the knowledge base. By clicking the Download knowledge base button the system will start downloading the currently active knowledge base in the format that can be read by the other @dvisor programs. The other buttons will be described in more detail in the remaining of this chapter.

The left part of the screen displays the Nodes in this knowledge base starting with the Root node and ending with the advice nodes. The top part in the middle shows the folders in the currently active node (which is the “Root” node at the moment) and within these folders the questions.

The bottom part in the middle shows these same folders and within these folders the expressions. Within the current active node, the “Root”, is one folder, called the “Main Folder”. This folder contains one questions “Choose your main complaint”. The answers to this question are “Abdominal Pain” and “Ankle Problems”.

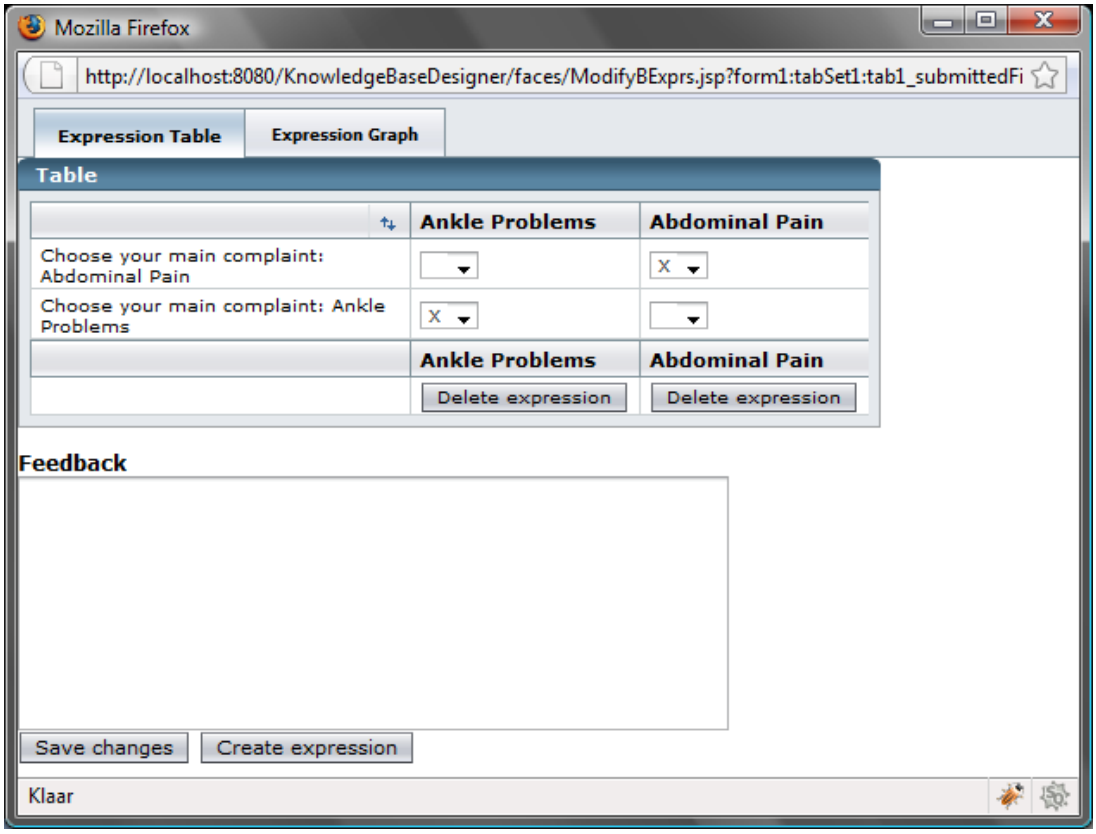
From the “Root” node the expert system can go to the “Ankle Problems” node and to the “Abdominal Pain” node. This is done via the two expressions that are shown in the bottom part in the middle.

By clicking on an expression a production rule will be shown. The following production rule describes that the “Abdominal Pain” node will become active if the “Choose your main complaint” question is answered with the “Abdominal Pain” answer.



By selecting a node, question, folder or expression the “Create”, “Modify/Delete” and “Copy” buttons will get enabled and disabled. By pressing one of the buttons a popup window will be shown that provides functionality to change, delete, create and reuse objects.

When an expression is selected in the bottom right section, followed by clicking the modify/delete button the following popup will be opened:

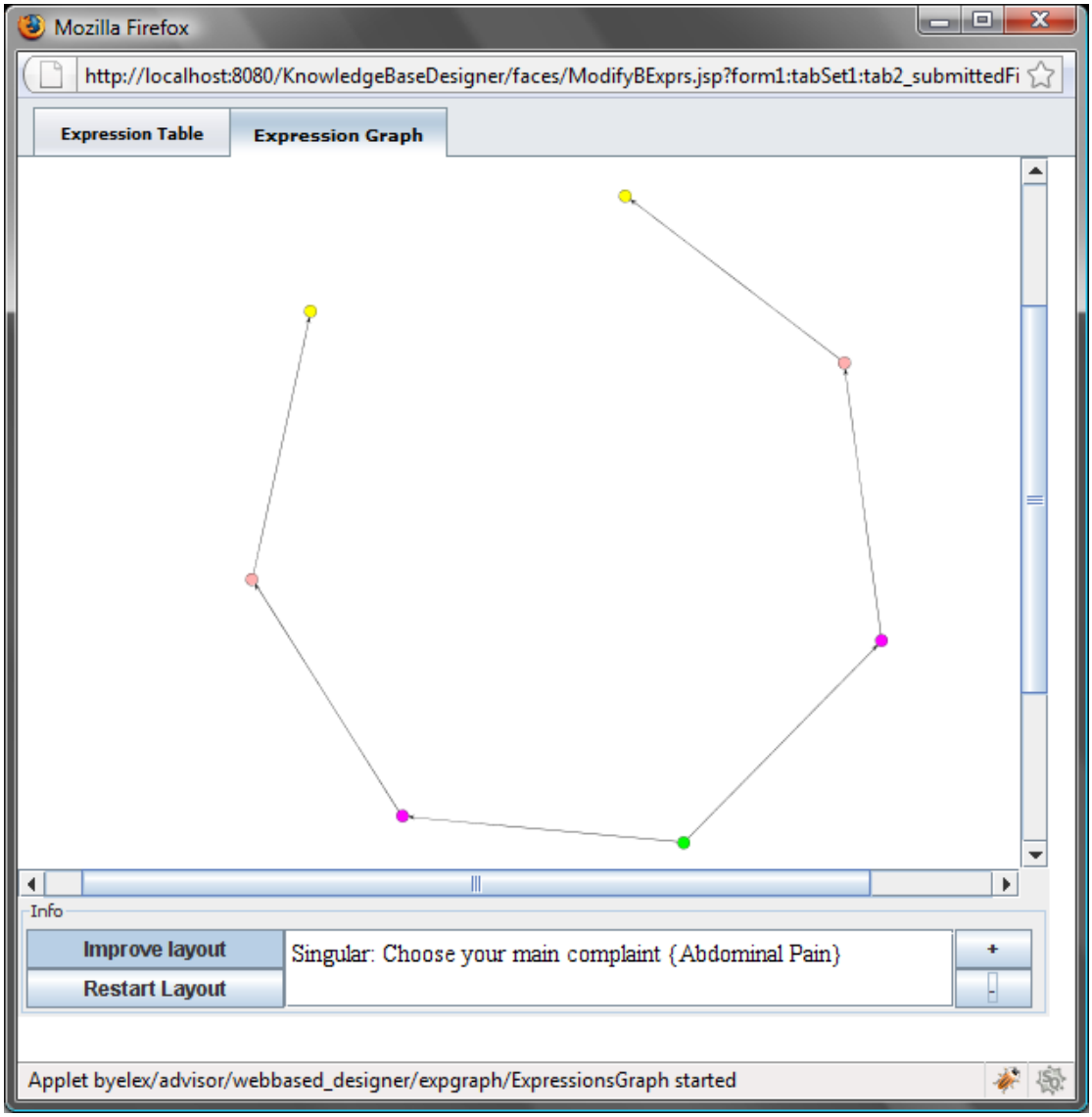


This popup contains two tabs. The first tab, shown above, contains the expression table. With this tab the expression can be modified. Expressions can be created and deleted and inside an expression there is the possibility to change the answers to questions, questions and their answers are shown in the screenshot above at the row headers.

The expressions made in the table will always be in the Disjunctive Normal Form, which is used by the inference engine. But when an expression is modified/deleted/created the system will also combine the expressions that lead to the same target node (the target node is shown in the screenshot above as the column header).

This combined expression will be used to display the production rule, shown on the previous page, and the expression graph that can be found on the second tab, shown below.

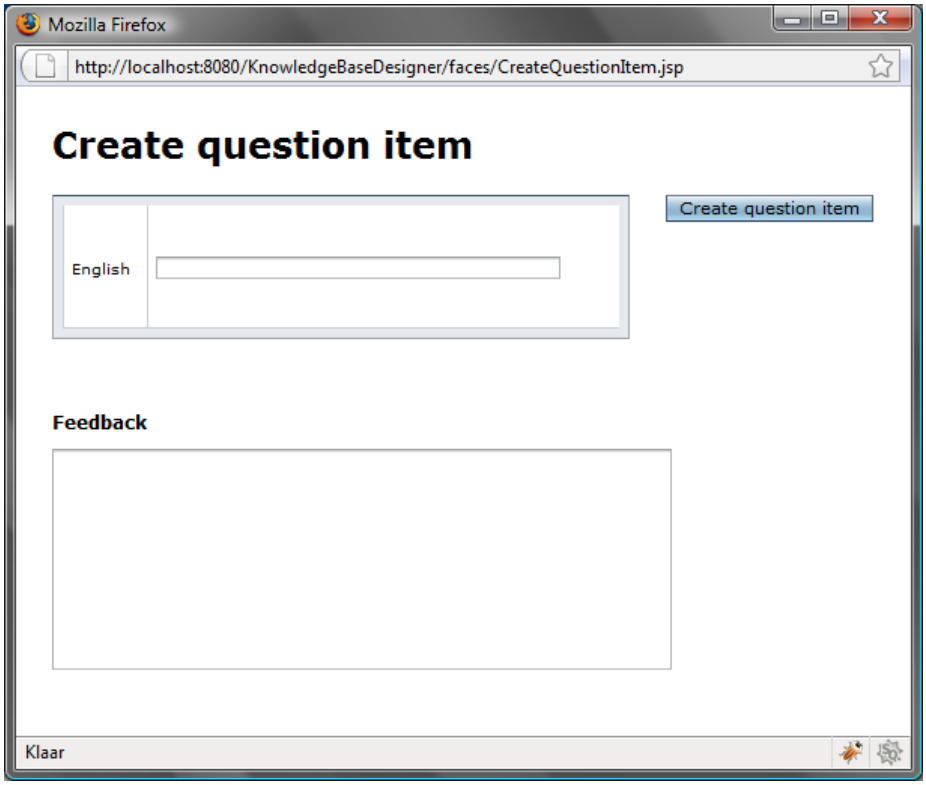
The expert also is able to enter a feedback message to describe the reasons why the expression is modified. This kind of feedback is useful for the knowledge base engineer, who will extract the knowledge bases from all experts that verified the knowledge base from the revision control system. The feedback of all experts combined will be used by the knowledge engineer to make a new knowledge base. The experts will then be able to verify this new knowledge base and after a number of these iterations the result will be a knowledge base that is verified correct by all, or enough, experts.



The screenshot on the previous page shows the expression graph that can be found in the second tab. The graph cannot be modified from this tab. The purpose of this window is to display the structure of the expressions and how you can go from the current node to other nodes. Below the graph are several buttons. The + and – buttons are used to zoom in and out. The Improve layout and Restart layout buttons are used to redraw the graph.

Drawing the graph is done with a "spring" algorithm and this algorithm can have a different result after each run. Sometimes the algorithm will result in a cluttered graph and by restarting the algorithm it might result in a more readable graph.

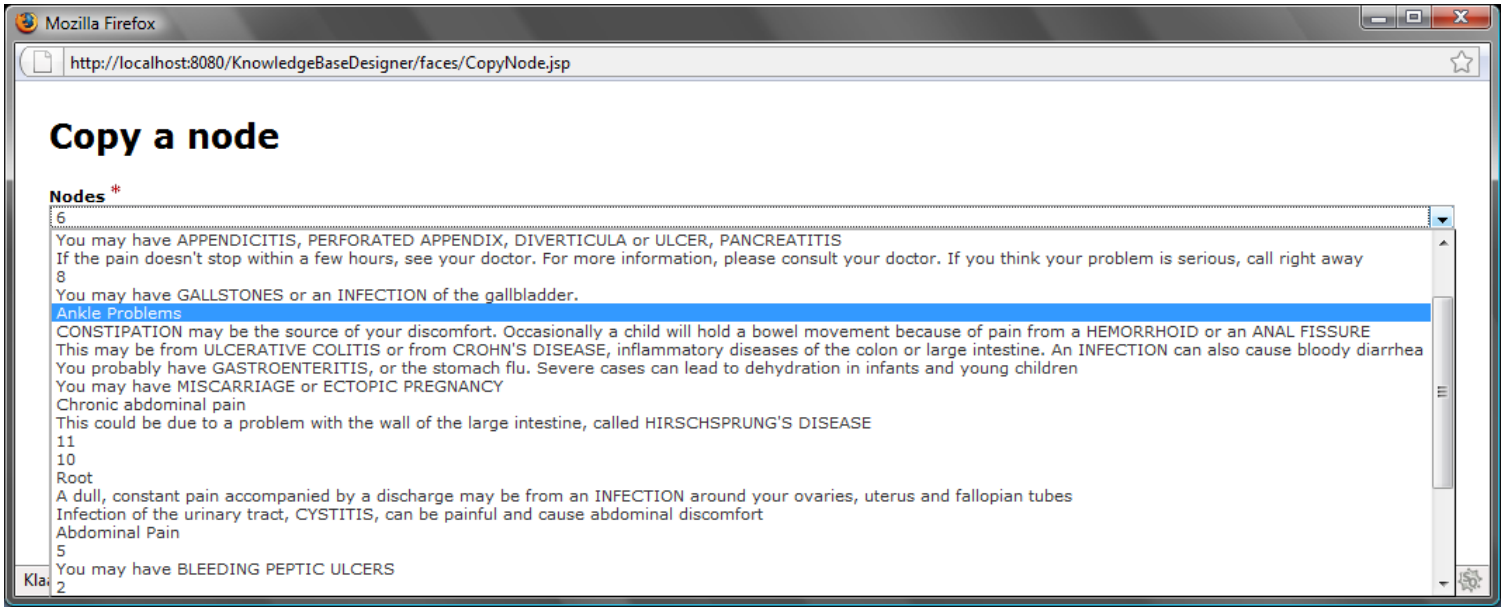
The expression graph is made with a Java Applet and therefore the user must have an updated Java VM installed.



The screenshot above is the window that will show up after an expert pressed the create button while a question is selected. A question item is a possible answer to a multiple choice or a single choice question. As can be seen the expert can enter a text string for every language (only English is available for the current knowledge base).

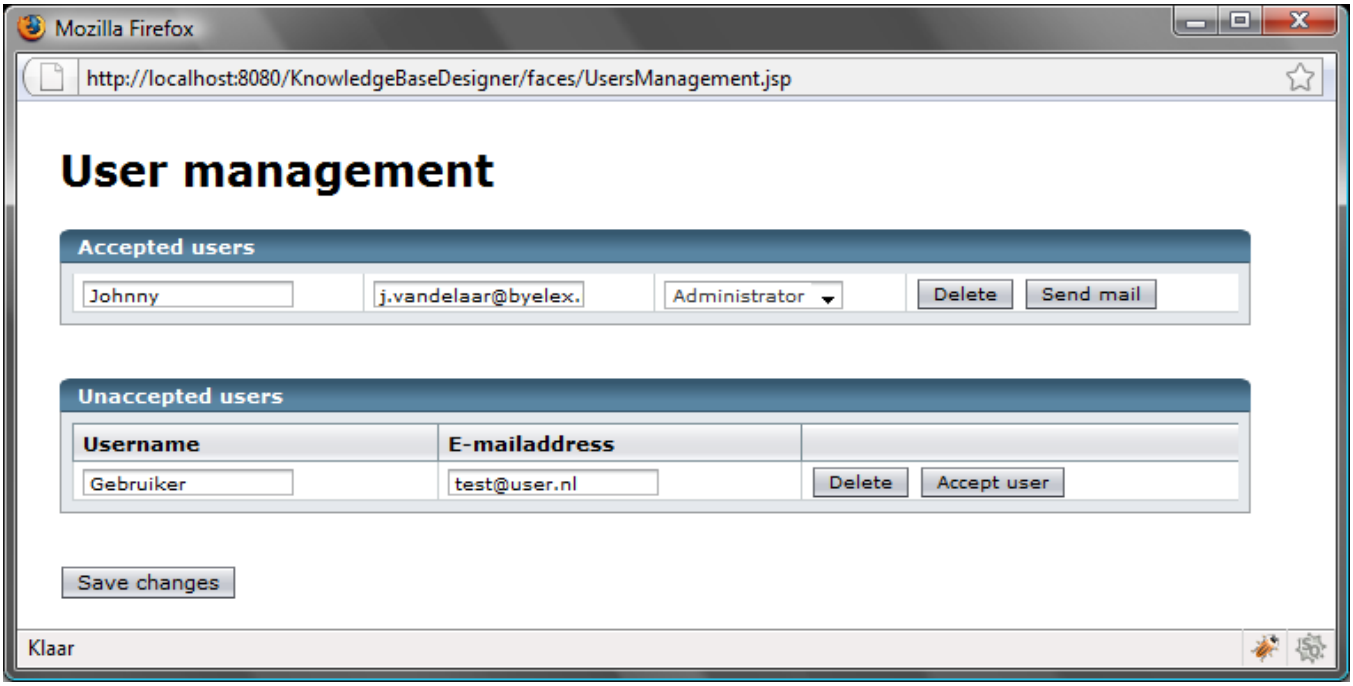
Not all individual creation/modification windows will be discussed as they all look similar and provide the same functionality as the "create question item" window does.

The following screenshot shows the clone node window. By selecting a node, this node will become a child node of the current active node.



An important check that is done when a node is clones is a check for cycles. The node graph cannot contain cycles, as this will result in deadlocks in the inference engine. Therefore the clone node operation will check the resulting graph for cycles and will prevent cycles from happening (it will not accept the operation if it results in a cycle).

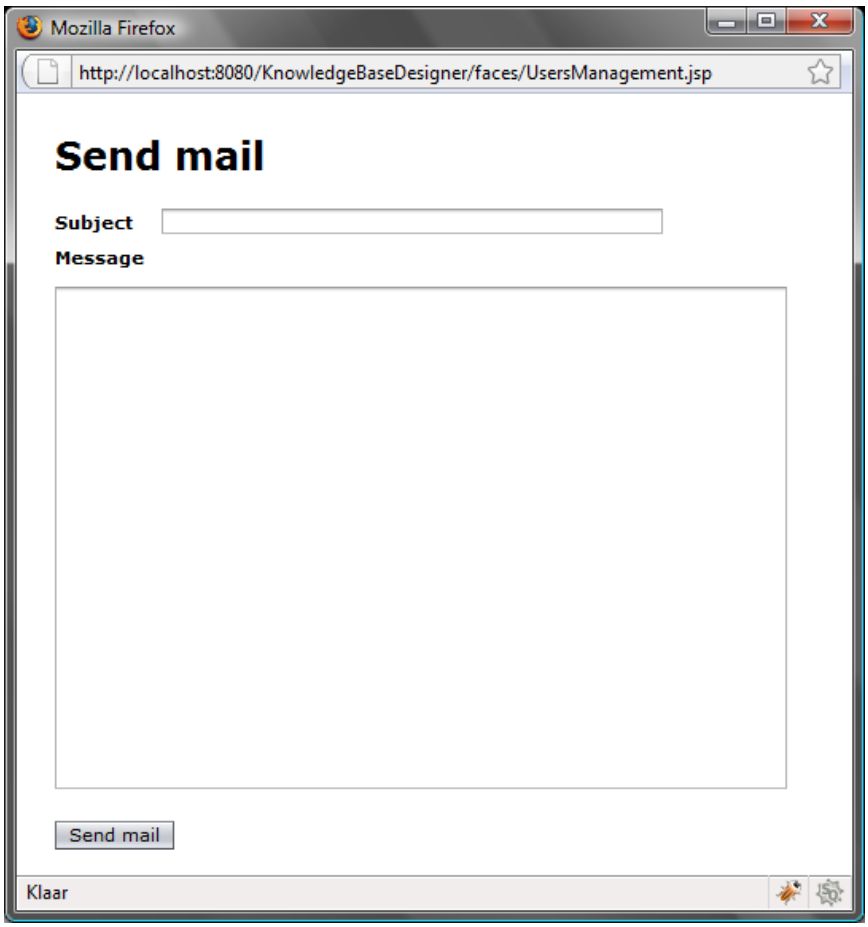
The User management functionality that will be described next will only be available to users with administrator privileges. By clicking on the user management button in the main screen, the following popup will be shown:



The popup on the previous page consists of two parts: accepted users and unaccepted users. The unaccepted users are newly created accounts which have not been approved yet by an administrator. The accepted users are user accounts that are accepted by an administrator and thus can be used by an expert to log in. There is the possibility to change the username and email address of a user and to set a user group for the user (administrator or normal user).

The delete button will remove the account and the “save changes” button will save the changes made to the usernames, e-mail addresses and user groups.

By pressing the send mail button, the following popup will be shown:



With this popup, the administrator can send an e-mail to the users e-mail address.

5. CONCLUSIONS

The goal of this project was to develop a web based system that can be used by experts in different fields to verify knowledge bases that are made by knowledge engineers. A system that contains the functionality that was set by the goals has been built. This system is based on the @dvisor program developed by Byelex Multimedia B.V. This chapter will describe the problems encountered during this project, but will also describe the parts of the project that where accomplished without any problems. Furthermore, this chapter will contain a paragraph describing how this project can be improved with new functionality in the future.

Fitting in the original source from the @dvisor program proved to be easier than expected. The original @dvisor program was written at the State University of Minsk, Belarus, and thus all source code contained Russian comments. But the Yahoo! Babel fish page proved to be very helpful in translating Russian to English and therefore the source code of this project was quite easy to port to the new web based @dvisor.

During my initial research, it was quite easy to find good documentation on the concepts of knowledge bases and expert systems. This is partially due to the fact that expert systems and knowledge bases are quite an old field of research. Already 50 years ago, there was research in the field of expert systems. During these 50 years of research some quite decent books have been written. Upcoming internet technologies, like the semantic web, provide a decent ground to revive the research in expert system technologies. More information on this subject is in the paragraph on the future of the web based @dvisor system.

During the actual development of the project , a number of problems where encountered. The main problem is that some parts of the Java EE environment are quite difficult platform to master. As a developer I have a decent experience in the Java programming language but this experience only consists of writing standalone programs. The Java EE environment consists of a number of frameworks and some of them proved to be easier to understand then others.

The tutorials that can be found on the various Sun websites are sufficient for building simple applications but there hardly are any tutorials for building more advanced applications. For building this project, especially the JPA and the JSF framework provided a number of difficulties.

In the end, I do not regret making the decision to use the Java EE environment but I do regret choosing Toplink over Hibernate and using the Woodstock JSF framework over a number of its competitors. JPA itself is not a very difficult framework to master. Being able to set various configuration options via annotations, instead of via separate configuration files, made setting up the JPA system quite easy. The main problem of the Toplink JPA platform is that it contains a number of strange issues and the framework is used by less people than the Hibernate JPA framework. Because of these strange issues in the JPA framework, I had to change a number of design decisions.

For example, importing a very large knowledge base into the system sometimes resulted in a strange error message: “it’s not possible to change the value of a primary key”. The problem here was that the JPA framework was keeping the variable in its cache and did not persist the variable to the database yet. In the meantime, the application tried to fetch the variable from the database which resulted in an error. This problem was fixed by changing the persisting strategy of the JPA framework.

The biggest problem in this respect, is that it is quite difficult to debug an application where the error message seems very unrelated to the problem and little information about the issue is available on the internet. Being new to the application makes it even harder.

The idea behind the JPA framework is good and without the glitches of the Toplink JPA framework it is not too difficult to create entity beans. So probably, using a framework that is used by more people, like the Hibernate framework, would have solved a number of problems that I encountered.

The Woodstock JSF framework, which comes standard with Glassfish version 2, simply lacks a lot of functionality and adding this functionality oneself is quite hard as the framework is not built for easy extension. If someone wants to use JSF out of the box without the need for very advanced functionality, then JSF is a nice framework to use. With the Netbeans IDE Visual Web extension, it is possible to built a JSF website via a WYSIWYG (what you see is what you get) editor. This makes it very easy to build a JSF application with the Woodstock JSF framework.

When someone wants a feature that is not already in the framework, like a table that does not have a fixed amount of columns or a button that opens a popup, then the JSF framework becomes more difficult to extend and thus more difficult to use.

The EJB and Web services framework provided by the Java EE platform proved to be easier to use. It is not too difficult to build an enterprise java bean and even easier to extend this bean with Web services functionality. Embedding a EJB or a Web service inside another project is fairly easy, as the Netbeans IDE takes care of this itself. Making scalable applications with EJB's and Web services becomes very easy, because the Netbeans IDE and Glassfish cooperate very good.

Because of the difficulties with building the application, a number of features could have been implemented better. For example the algorithm that combines multiple expressions in DNF into one expression is very basic. For displaying these expressions in the graph it would have been better if I had implemented an algorithm that tried to minimize the size of the expression; probably a greedy algorithm would have resulted in a decent result.

FUTURE

This version of the system contains the functionality to verify a knowledge base. In the future, the application can be extended with the other functionality that is currently available in the standalone @dvisor Knowledge Designer program. This functionality will make the application a full featured replacing tool for the knowledge base designer.

But a more important extension of this program would be to make the system compatible with semantic web technologies like RDF and OWL. By combining the reasoning power of the @dvisor system and the amount of knowledge available if the web switches over to semantic web technologies, sometimes marked with the web 3.0 buzzword.

By using a semantic web enabled search engine, like the Yahoo! Search Monkey project, the @dvisor system could import a large amount of structured data. By this, the @dvisor system can take advantage of resources available on the internet containing structured knowledge and thus reducing the amount of work needed to build op knowledge bases.

Another important extension would be the integration of the inference engine into the Web service. By using the abilities of semantic webs, huge knowledge bases can be gathered. By sharing the @dvisor inference engine, a remote application could use the huge knowledge base and the power of the @dvisor technology.

This could lead to traditional expert systems like medical advice systems, product advice systems, automated product helpdesks, etcetera. The @dvisor technology could also be used for smarter search engines. These search engines will not be based on search terms but will query the user with specific questions, which will lead to an answer in the form of search results.

The difference with traditional search engine techniques is that the new search engine “understands” the question but also “understands” the answer it will give. Because of the semantic web technology, the system will contain a large structure of words that are related to other words. A possible implementation of this system could be that the user will first be asked to describe what he is looking for. Based on the search query that is entered, the system will determine, by using the semantic web, what the meaning is of the entered search query. Based on the meaning of the words and their relation to other words the system will be able to ask more questions to narrow down the search results.

The power of the @dvisor system Is that it can show an intermediate result at every moment. This means that the user could provide feedback on the direction in which the system is looking. Based on this feedback the system should be able to “learn”. The semantic web is a big graph containing nodes (the terms) connected by arcs (relations between the terms). By learning, the system will adapt arc weights, such that the inference engine will improve the quality of its advice.

6. REFERENCES

Books

1. Title: Plato's Theory of Knowledge: The Theatetus and The Sophist
   Section: Theaetetus [201c-210b]
   Writer(s): Plato, translated by Francis M. Conford
   Publisher: Dover Philosophical Classics

2. Title: Principes van expert systemen
   Writer(s): P.J.F. Lucas and L.C. van der Gaag
   Publisher: Academic service

3. Title: The Astadhyayi of Panini
   Writer(s): Panini, translated by S.D. Joshi and J.A.F. Roodbergen
   Publisher: Sahitya Akademi

Papers about expert systems

4. Title: Semantic Memory
   Writer(s): M. Quillian
   Published in: Semantic information processing by M. Minsky
   Publisher: MIT Press

5. Title: Web-based expert systems: benefits and challenges
   Writer(s): Y. Duan, J.S. Edwards, M.X. Xu
   Published in: Information and Management, volume 42, issue 6
   Publisher: Elsevier

6. Title: Foundation and Application of Knowledge Base verification
   Writer(s): A.D. Preece, R. Shinghal
   Published in: International Journal of Intelligent Systems, volume 9, issue 8
   Publisher: Wiley Periodicals

Papers specifically about @dvisor

7. Title: Decision-Making by Precedence: Modelling, Technology and Applications
   Writer(s): V. Krasnoproshin, V. Obraztsov, H. Vissia
   Published in: Proceedings of MS'2002, Internation Conference on Modelling and Simulation in Technical and Social Sciences
   Publisher: Universitat de Girona

8. Title: Inductive algorithm for solving diagnosis problem
   Writer(s): J. Bergmans, V. Krasnoproshin, V. Obrazstov, H. Vissia
   Published in: Proceedings of PRIP'97, Pattern recognition and information processing
   Publisher: University of Szczecin

9. Title: A technological solution for modern knowledge management systems
Writer(s): S. Gutnikov, V. Krasnoproshin, V. Obrazstov, H. Vissia
Published in: Proceedings of SSIT'98, International conference on systems and signals in intelligent technologies
Publisher: Belarus state university

10. Title: Knowledge as an object of mathematical formalization
Writer(s): V. Krasnoproshin, V. Obrazstov, H. Vissia
Published in: Proceedings of MS'99, Internacional conference on modelling and simulation
Publisher: Universidade de Santiago de compostela

11. Title: Computer-based support to decision-making in orthopaedics
Writer(s): J. Bergmans, S. Gunikov, V. Krasnoproshin, S. Popok, H. Vissia
Published in: Proceedings of ITHURS'96, International conference on intelligent technologies in Human-Related sciences