NTNU

Norwegian University of
Science and Technology

Expert system for hydropower stations developed in Volve Knowledge Tools

Erlend Timberlid

Master of Science in Energy and Environment
Submission date: June 2008
Supervisor: Lars Einar Norum, ELKRAFT

Norwegian University of Science and Technology
Department of Electrical Power Engineering

Problem Description

Despite the fact that hydropower stations are equipped with the latest technology in both regulation and remote control, it is not enough to replace the traditional machine expert. The machine expert was a person stationed in the power plant. He had the responsibility for the running and maintenance of the station. This person's experience and human senses made him crucial for the surveillance of the station. Since the machine expert has now been replaced by electronics and newer technology, there are still areas of his expertise that are to be made. Today's facilities are highly sophisticated and contain the latest technology, but there are still situations that the surveillance system may fail to intercept. These situations could for instance be the beginning of an fault in an component.
To help the surveillance system detect these situations, the idea is to implement an expert system. In this way the control system can match the machine expert's skills.
Øystein Fjellheim has already done extensive work in finding a suitable development kit. He explored several software packages looking for one to use for this purpose. The result was the program Volve Knowledge Tools, and the purpose of this report is to further test this program for use in expert systems. This testing considers more advanced and intelligent forms of reading raw data, such as mean calculation, derivation and integration. It also includes testing of how the program learns and uses existing knowledge to solve problems.

Assignment given: 15. January 2008
Supervisor: Lars Einar Norum, ELKRAFT

1 PREFACE

This project is a part of the fourth semester of the 2-year Master program in Electric Power at *NTNU*. The project is weighted as one semester in this program and is a cooperative effort between *NTNU, Voith Siemens* and *Volve AS*. *Voith Siemens* has presented the approach to the problem and also acts as external supervisor. *Volve AS* provides the software one wish to investigate.

This report consists of five main parts. The first part describes the system that is going to be modulated. In the second part it is explained what an expert system is. The third part contains an explanation of how *Volve Knowledge Tools* works and how an expert system can be created in this program. In the fourth part it is discussed why one should use condition monitoring. The last part of the report contains simulations and results. Here, continuing work and advantages/disadvantages will be presented as well.

I hereby send my greatest thanks to Frode Sørmo, Agnar Aamodt, Tore Brede and Jone Rasmussen at *Volve AS* for invaluable help and guidance. I also thank my supervisor professor Lars Norum at *NTNU*, and also Øyvind Holm and Oddbjørn Hansen at *Voith Siemens* for good advice and feedback throughout the process. Finally I want to thank Ole Bjørn Westad at *Maintech AS* and Jan Anders Timberlid at *Sogn og Fjordane University College* for help and support during the whole process.

Trondheim, 9 June 2008

Erlend Timberlid

2 SUMMARY

Despite the fact that hydropower stations are equipped with the latest technology in both regulation and remote control, it is not enough to replace the traditional machine expert. The machine expert was a person stationed in the power plant. He had the responsibility for the running and maintenance of the station. This person's experience and human senses made him crucial for the surveillance of the station. Since the machine expert has now been replaced by electronics and newer technology, there are still areas of his expertise that are to be made. Today's facilities are highly sophisticated and contain the latest technology, but there are still situations that the surveillance system may fail to intercept.
To help the surveillance system detect these situations, the idea is to implement an expert system. In this way the control system can match the machine expert's skills.

Øystein Fjellheim has already done extensive work in finding a suitable development kit. He explored several software packages looking for one to use for this purpose [1]. The result was the program *Volve Knowledge Tools*, and the purpose of this report is to further test this program for use in expert systems. This testing considers more advanced and intelligent forms of reading raw data, such as mean calculation, derivation and integration. It also includes testing of how the program learns and uses existing knowledge to solve problems.

Since hydropower plants can be constructed in so many ways, it is almost impossible to make an expert system that can include all varieties. One solution is to narrow down the system to a *reference system*. This system consists of defined components that are common in today's facilities. Despite this narrowing there are still many processes in a hydropower station that need to be monitored. *Voith Siemens* has promoted that thermal surveillance is desirable. The reason is that, when performing thermal surveillance, the rest of the system would be indirectly monitored as well. A good example of this would be to monitor the bearings. To detect the condition of a bearing, one has to compare several parameters, of which temperature is the most essential. Two conditions of the bearing are simulated, *defective bearing* and *water in the bearing oil*.

When constructing the program, three models were made: an *ontology model*, a *causal model* and a *case model*. All models were made pursuant to the *reference system*. The models are general, but still represent a good approach to the real world. All three models are considered as a good starting point for further work.

By developing a simple expert system in *Volve Knowledge Tools* and using this program for simulations, I gathered a lot of information about how the program operates and functions. After considering advantages versus disadvantages, and the suitability of the program for developing expert systems, I concluded that: *Volve Knowledge Tools* fulfils the demands that the development an expert system entails. I therefore recommend by *Volve Knowledge Tools* as the development kit for evolving an expert system in continuing work.

3 TABLE OF CONTENTS

4 LIST OF FIGURES

5 LIST OF TABLES

6 ABBREVIATIONS

NTNU	-	Norwegian University of Science and Technology
CREEK	-	Case based Reasoning through Extensive Explicit Knowledge
CSV	-	Comma separated values
XML	-	Extensible Markup Language
CBR	-	Case Based Reasoning
MBR	-	Model Based Reasoning
MVA	-	Mega Volt Amper
PLC	-	Programmable Logic Control
SCADA	-	Supervisory Control And Data Acquisition
IGSS	-	Interactive Graphical SCADA System
HIST	-	Sør-Trøndelag University College
IDI	-	Department of Computer and Information Science
GB	-	Giga Byte
HMI	-	Human Machine Interface

1. INTRODUCTION

1.1. Background

Despite the fact that hydropower stations are equipped with the latest technology in both regulation and remote control, it is not enough to replace the traditional machine expert. The machine expert was a person stationed in the power plant. He had the responsibility of the running and maintenance of the station. This person's experience and human senses made him crucial for the surveillance of the station. Since the machine expert has now been replaced by electronics and newer technology, there are still areas of his expertise that are to be made. Today's facilities are highly sophisticated and contain the latest technology, but there are still situations that the surveillance system may fail to intercept.
To help the surveillance system detect these situations, the idea is to implement an expert system. In this way the control system can match the machine expert's skill.

Øystein Fjellheim has already done extensive work in finding the right development kit for the expert system. He explored several software packages looking for one to use for this purpose [1]. The key to finding the right software was to look into the oil business sector. Although the oil drilling process seems very different from creating power from water, the surveillance of these two processes has many things in common. The software that satisfied all of Fjellheims demands is called *Volve Knowledge Tools*. This software is designed for developing expert systems. He also used the program for comparing different sets of parameters to detect a defective generator cooler. In this way the program proved itself useful.

This report is a continuation of "The digital machine expert – Expert system for monitoring hydropower plants [2]. Therefore the approach to the problem is about the same: to investigate whether *Volve Knowledge Tools* can be used for developing an expert system and thus improve the surveillance system of today. This report represents the second part of this investigation.

My purpose is to continue the development of an expert system and thereby further improve the surveillance system of today's power plants. This would allow me to test Volve Knowledge Tools even further and to see if it is suitable for developing expert systems.

Since the situation today is that the highest possible profit is sought from everything, aggregates are driven accordingly. One consequence is more frequent starts and stops of aggregates. This starting and stopping causes more stress to the machines and in shortens the lifetime of the equipment. Since this trend is likely to continue, the need for an improved surveillance system is greater than ever.

1.2. Limitation

Since hydropower plants today can be constructed in so many ways, it is almost impossible to make an expert system that can include all types. In any case this would be a very time-consuming process. Since the time schedule for the present task is limited to twenty weeks, developing such a program is not possible. However, if the system is narrowed, it could be possible. The narrowed system is called the reference system [2]. It consists of defined components that are common in today's facilities.
Even if the task is narrowed into a reference system, there are still many processes in a hydropower station that need to be monitored. As mentioned above, the time available for the present task is relatively short, so further limitation will be to concentrate on one process in the system. The process to be investigated is the detection of a defective bearing. This bearing will be the combined axial and radial bearing located at the top of the machine. The modulation will be a rough approach to the real system, but still easy to expand in continuing work.
Since many of these components do not directly affect the final approach to the problem, they will only be presented in general terms.

1.3. Structure

This report consists of five main parts. The first part describes the system to be investigated. In the second part it is explained what an expert system is. The third part contains an explanation of how *Volve Knowledge Tools* works and how an expert system can be created in it. In the fourth part it is discussed why one should use condition monitoring. The final part contains simulations and results. Continuing work and advantages/disadvantages will also be presented here.

2. THE REFERENCE SYSTEM

To shape the reference system, it was necessary to decide what it should consist of. When choosing the components, commonly-used devices were chosen. In this way the reference system would be a good foundation for developing a specific expert system. One criterion for shaping a reference system was that it should be similar to a facility in the Trondheim region.

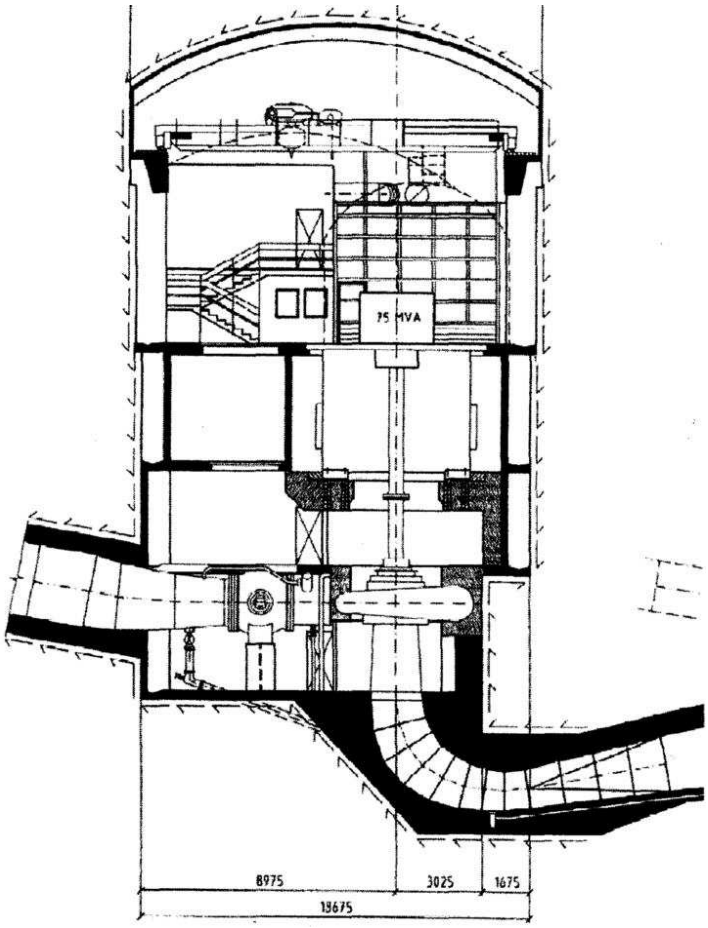


Figure 1, Meråker hydropower plant [2]

The facility chosen as a basis for the reference system is Meråker power plant, Figure 1. It consists of two aggregates one producing 75 MVA and the other 33 MVA. The reference system should, for practical reasons, be a medium-sized power plant. This results in a reference system which consists of [2]:

- Vertical Francis turbine
- Ball valve as main valve
- Submerged station in the bedrock with a draft tube
- Bilge system with two pumps and one ejector
- Three-phase synchronous generator
- Cooling water from the clearance of the turbine
- Cooling system with four separated coolers in the stator, spiral-coolers in the bearings and a separate cooler for the transformer
- Pad type bearing with oil-pressure discharger in the axial bearings
- Bus system for communication
- *IGSS32* as control system

2.1. General

The reference system is a submerged power station embedded in the bedrock. A submerged station means that the centre of the turbine is lower then the level of the tailwater, see Figure 2. The reason for submerging a power station is to exploit the head more effectively, which creates more speed to the rotor. High speed turbines are smaller and less complex than slower ones. A result of this is a cheaper power station.

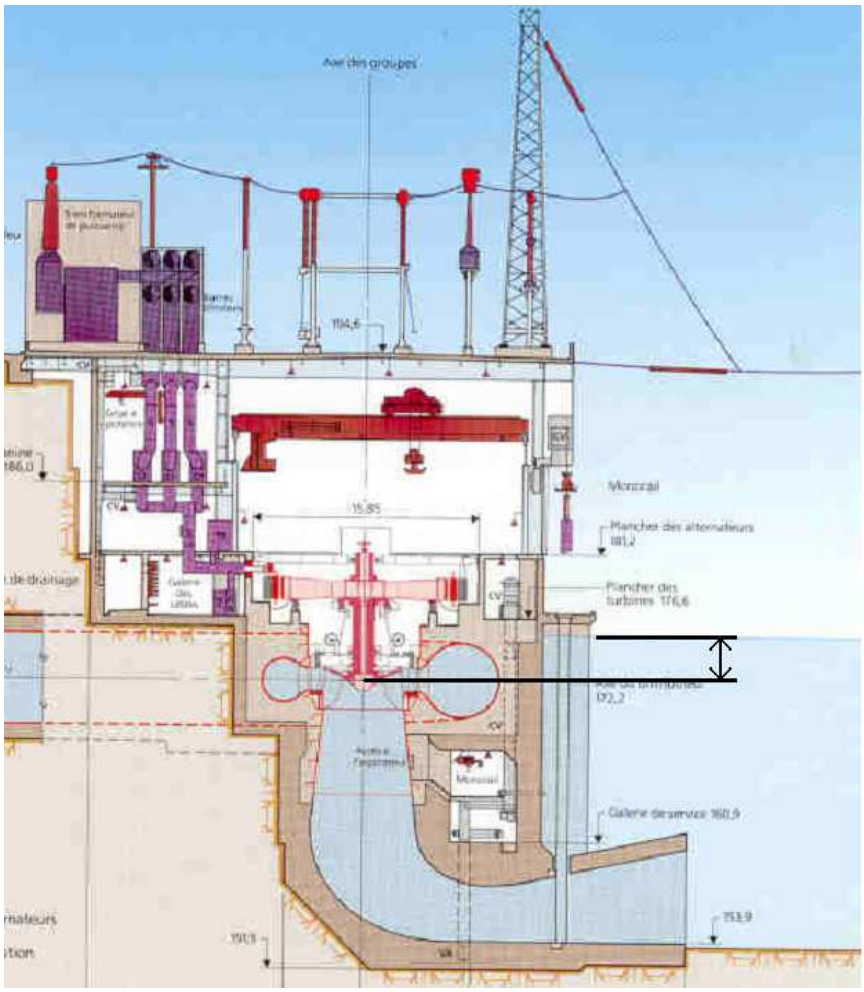


Figure 2, Submerged station [3]

As shown in Figure 2, the reference system consists of a Francis turbine with a draft tube. The purpose of the draft tube is to drain water from the runner and decrease its speed to suit the tailwater [4]. The solution for decreasing the water speed is to gradually expand the draft tubes outlet. At the outlet of the draft tube there is a hatch. This hatch is remotely operated, as are all hatches in the facility. Occasionally rocks loosen in the draft tube. This is unfortunate because rocks can clog the whole tube and cause flooding of the station. To detect any form of landslide in the draft tube or close to it, there are float balls that measure the level of the water. If these are triggered, the sequence of stopping the aggregate is initiated. In that case the hatch of the draft tube will be closed.

2.2. Bearings

There are three types of bearing: roller bearing, ball-bearing and oil lubricated sliding bearing. Lubricated sliding bearings are the type most commonly used in generators. The main task of the bearings in generators is to keep the rotor in place. This is done by using a combination of radial bearings and held bearing. The bearings have another important task; they absorb forces applied by the magnetic and mechanical unstableness of the machine [5].

2.2.1. Placing of bearings

In this chapter, only bearings placed on a vertical machine will be presented.
To see to that the rotor is in the centre at any time, radial bearings are used. There can be several radial bearings, but the minimum requirement is one. The other type of bearings used, are axial bearings. The main task of these bearings is to absorb the weight of the generator and additional hydraulic weight applied by the turbine. A common use of bearings is to combine the axial bearing with one of the radial bearings. The placement of the bearings is individual on each machine, depending on rotational speed, performance and type of turbine.

There are two common ways of placing bearings. One is the W41, which is used for high head Pelton and Francis turbines. This is the most common type in Norway.

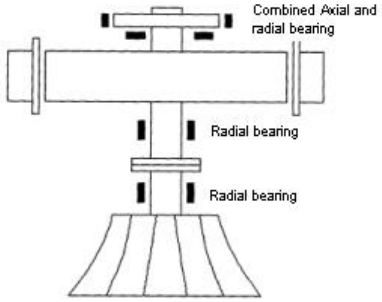


Figure 3, Placement of bearings, model W41 [5]

The other platform is W42. This form is used for low head turbines and where the radius of the stator is rather large.

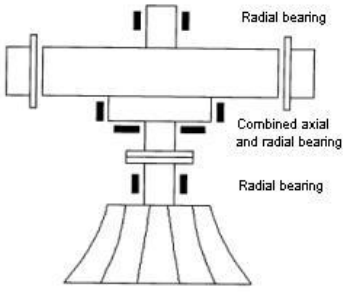


Figure 4, Placement of bearing, model W42 [5]

There are of course other ways of placing the bearings, but those types are not mentioned in this report.

2.2.2. Axial bearing

The axial bearing consists of several parts. Underneath the rotor there is a seal face. The seal face is placed on top of the axial bearings. Between the seal face and the bearing segments there is an oil film. The oil film sees to that the rotor is floating on the top of the bearing segments. In this way there is no physical connection between the segments and the rotor. There are three types of axial bearings. The difference lies in how the segments are shaped:

- Stationary segment
- Tilting-pad segment
- Spring segment

Below, only the stationary-segment type is explained. In Figure 5 a stationary-segment bearing is shown.

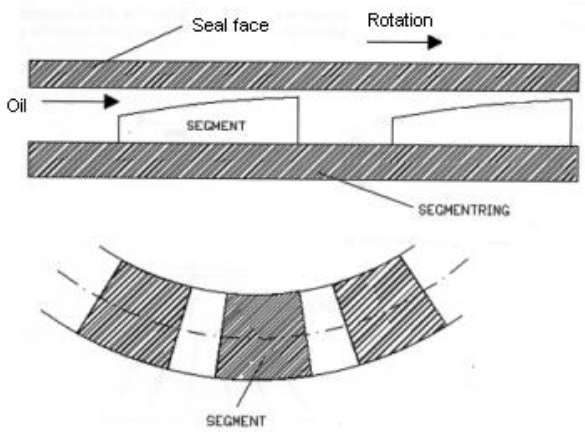


Figure 5, Pad type bearing [5]

As shown in Figure 5, the segments are sloping. The reason for this is that the lateral surface creates an oil film. They also provide replacement of warm oil when the rotor rotates. Thus the rotor functions as a pump and makes sure that the temperature is kept at an acceptable level.
At start-up the oil has a high viscosity and the desired oil film is not established. As the rotor gains speed the viscosity drops, and at a certain rotational speed the oil film is established (Point C). The graph below shows how the friction factor reacts to the increasing speed, where K is the optimal operating point of the machine.

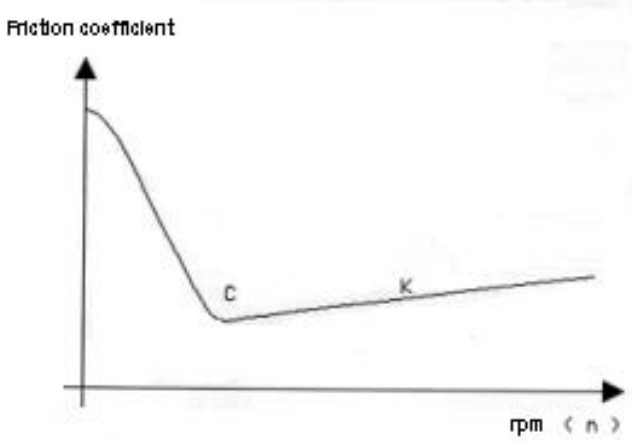


Figure 6, Friction in bearing at start-up [5]

At start-up there is a direct connection between the seal face and the bearing segments. This is not desirable and can be avoided by using oil-pressure discharging. When starting a machine, the oil-pressure discharge raises the rotor so that there is no contact. This is done by pumping additional oil into the bearings. Using oil-pressure discharge prevents unnecessary strain on both the seal face and the bearing segments. In the reference system this type of bearing i.e. pad type bearing with oil-pressure discharging will be used. How the oil-pressure discharge functions is illustrated in Figure 7

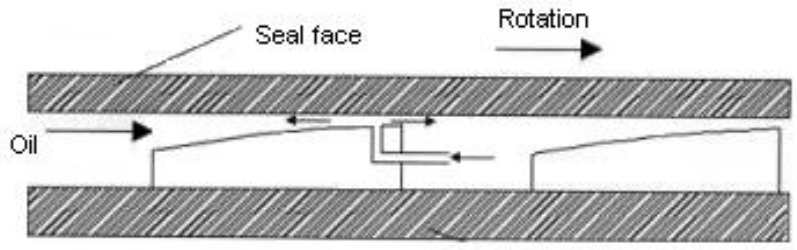


Figure 7, Pad type bearing with oil-pressure discharge [5]

To prevent oil vapour escaping into in the rest of the system (or dirt getting in), labyrinth glands are used.

2.2.3. Radial bearing

The main task of the radial bearings is to absorb the forces in the radial direction.
There are two types of bearings in this category, turbine bearing and lower radial bearing, but their objective are the same; keep the rotor centred. Turbine bearings consist of a drum that encloses the whole shaft and is place right above the turbine. The drum is spilt in two, one part containing the oil. When the shaft rotates, the oil is forced into a pipe. This pipe leads to the upper part of the drum. Here, the oil is released and flows back to the lower part along the shaft. See Figure 8.

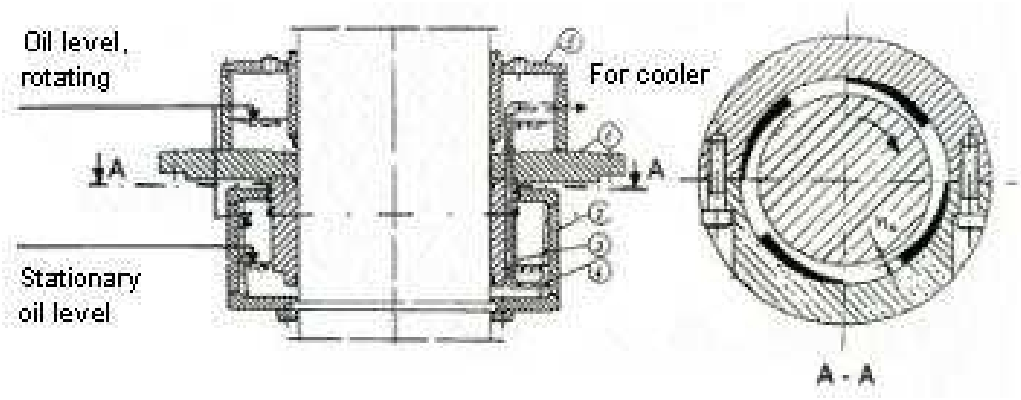


Figure 8, Turbine bearing [4]

The radial bearings operate in the same principle as the axial. The segments see to that an oil film is established and the bearing get lubricated. The only difference is that the bearing is mounted vertically instead of horizontally.

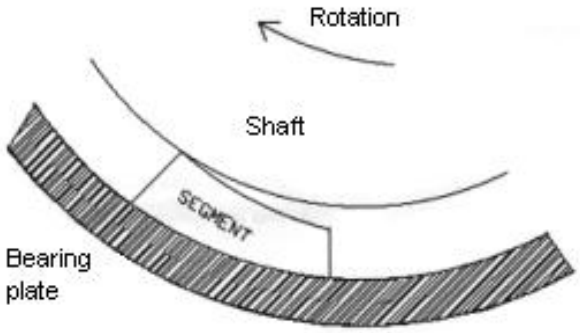


Figure 9, Horizontal pad type bearin [5]

2.2.4. Combined axial and radial bearing

At the top of bearing model W41, the combined axial and radial bearing is placed. As the name indicates, the bearing consists of an axial and a radial bearing. Both of these bearing are placed in an oil sump. The figure below shows how the bearing is built.

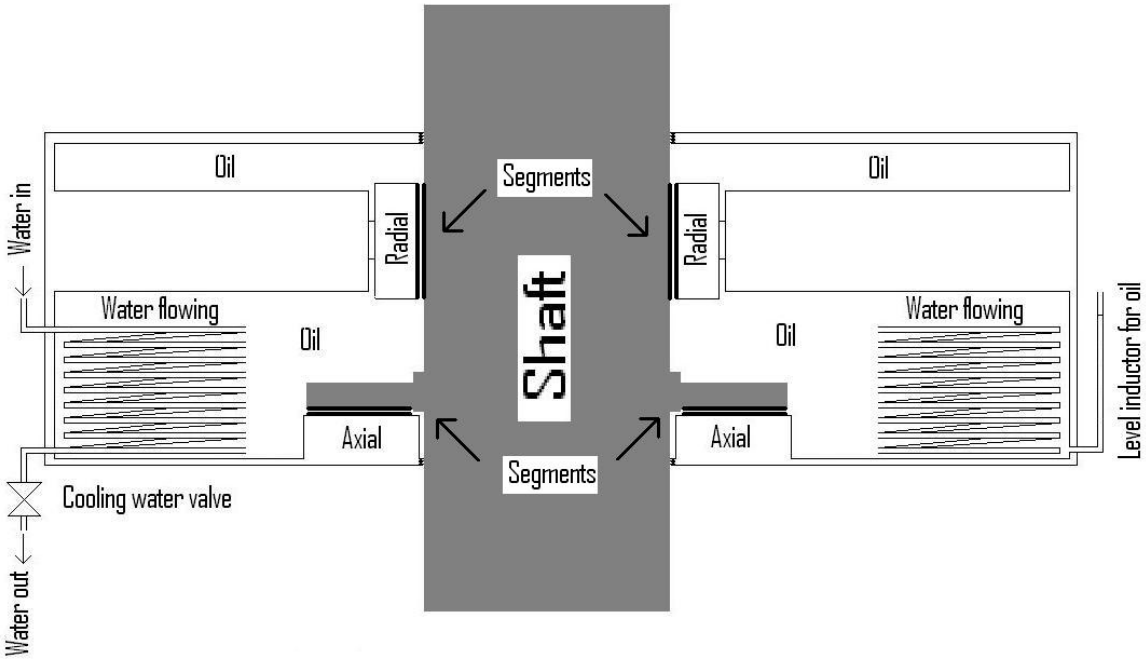


Figure 10, Combined axial and radial bearing

As described in the figure, both bearings are soaked in oil. In this way the bearings always have access to plenty of oil and lubrication is kept at an optimal level.
When the shaft is not rotating, the axial bearing and half the radial bearing are underneath the oil surface. As soon the machine starts to rotate, the oil starts to whirl around in the sump. There are grooves in the radial segments that see to the oil is whirled up so that the whole bearing is lubricated [6].

When the machine rotates, the segment and the oil become very hot. In order to lower the oil temperature, a heat exchanger is used. This exchanger is a spiral pipe that cold water flows through. The water enters the spiral at one point and runs through the spiral and out again only aided by gravity. Since Figure 10 is drawn in 2D, it is easy to interpret the bearing as having two spiral heat exchangers. This is not the case. The spiral goes all the way round the shaft so that the whole oil sump is cooled down. This is an efficient method for keeping the temperature at a reasonable level. The amount of water that runs through the heat exchanger is depends on the cooling water valve. This valve is governed by the control system and will open or close as the bearing temperature goes up or down.

2.2.5. Monitoring

Four parameters are crucial to the bearings: temperature, oil level, vibrations and electrical currents in the shaft. To measure the temperature in the bearing, Pt-100 elements are used. These are placed in the oil and are moulded into the bearing segments. Since the Pt-100 elements are placed in the centre of the segment, the temperature in the segments is the best indication of the temperature in the bearing.
When the temperature reaches a certain value, a pre-warning is sent to the control system to notify an operator. If the temperature reaches an upper level, the aggregate starts the emergency stop sequence.

On the right-hand side of Figure 10 an oil level indicator is shown. This sensor indicators how much oil is in the bearing. When the machine is running the oil level is totally different from when it is stopped. The reason for this is that the oil is splashing around in the bearing. In the simulations only the oil level when the machine is running will be considered.

To measure vibrations, an inductive distance measurer is used. Since vibration is crucial for the aggregate, the measuring of distance is performed in µm and done continuously.

Because of the asymmetry in the stator punching, a voltage is induced in the shaft. This voltage attempts to lead a current through the bearings. This is not desirable because the penetration causes electro-erosion. The erosion will eventually break down the oil film and a bearing breakdown is inevitable. A solution to this problem is to isolate the bearing from ground.

To detect currents in the shaft, a ring-transformer is used. The transformer is mounted around the shaft and sends out a signal when currents are detected. This signal goes directly to the electrical protection switches.

2.2.6. Defective bearing

To detect if a bearing is defective or an error is developing, several parameters must be considered. The most essential parameter is the temperature of the bearing. A high temperature is not desirable and can be an indication that something is wrong. If high temperature and vibration occur, and the rest of the systems is operating normally, it can be a symptom of a defective bearing [4].
A defective bearing can be a result of several things, but it is mainly when the bearing is exposed to heavy stress that its lifetime decreases. Heavy stress on a bearing can be due to several things, including generator on overload and shaft currents, to mention only a few. In this report only the generator on overload will be considered. The reason for excluding shaft currents, which expose the bearing to extremely high strain, is that shaft currents are so critical to the bearing. When a shaft current is detected, the machine goes to quick closing and the bearings are inspected visually by maintenance staff i.e. the monitoring of bearings for shaft currents is already covered in today's surveillance.

2.3. IGSS32

In modern power plants all processes are computer controlled. This means that an operator uses a computer to set the desired parameters and controls the different processes using its software. The software should be simple and easy to use.
Specialized programs have been developed for this purpose. They are called *SCADA* systems. *SCADA* stands for “Supervisory Control And Data Acquisition” [1].
A *SCADA* system mainly consists of several *Programmable Logic Controls (PLCs)*, input/output units, communication protocols and the human interface which is the program the operator uses.

In the reference system, *IGSS32* (*Interactive Graphical SCADA System*) is used. This system has been chosen by *Voith Siemens* and is used in their facilities. The *IGSS* platform was developed by the Danish company *7 Technologies*. The *IGSS* system are very comprehensive and complex, but still easy to customize, flexible and scalable. The program *IGSS* program has a graphical interface. This makes it fast, simple and easy to use. *IGSS* is also a real-time operating system. This means that the temperatures presented on the display are the actual temperatures of the system at that point. A further advantage of *IGSS* is that it can log data. This is practical since, when a situation occurs, it is possible to see if there is a certain chain of events that leads to that type of error. From this information expert systems can be developed. So when developing an expert system, it is sensible to base it on data provided by *IGSS*.

2.4. Stop sequences

When a fault situation occurs in a hydropower station, the aggregate(s) go(es) to a given stop sequence. This sequence depends on how severe the error is. There are three different types of stop sequences [7]; *Quick Closing*, *Stop* and *Emergency Stop*.

2.4.1. Quick Closing

This is the most common way of stopping an aggregate, since it is the gentlest way of bringing the rotating machine to a stop. When the *Quick Closing* command is given, both the guide vane operating mechanism and the valve system are closed before the circuit-breaker and the excitation switch are disconnected. This prevents racing of the machine. At rotational speeds below 60 rpm the mechanical brakes are applied.

2.4.2. Stop

When an electrical fault occurs, the main priority is to stop producing electrical power and get the generator off the grid. This is done by disconnecting both the circuit-breaker and the excitation switch at the same time. As a consequence, the machine loses its electrical counter torque and the rotational speed increases rapidly. Because of this racing, the retardation period is far longer for the *Stop* sequence than for *Quick Closing*. Mechanical brakes are applied at speeds below 60 rpm for this sequence as well.

2.4.3. Emergency Stop

When a critical situation occurs, for example if the temperature in the bearing reaches its critical value, the aggregate goes to *Emergency Stop*. The emergency stop sequence has the same progression as *Quick Closing*, the only major difference is that the brakes are applied at a much higher rotational speed. The brakes can be activated at rotational speeds up to 90 percent of the maximum rotational speed. This exposes the components to a great load of strain. This method of stopping is of course undesirable

3. WHAT IS AN EXPERT SYSTEM?

The most common form of expert system is a computer program. An expert system is often referred to as a knowledge-based system. This means that the program can draw its own conclusions based on knowledge. The knowledge is based on specific information concerning the class of problems. This technology was first developed by researchers in artificial intelligence in the 1960s and was first applied commercially in the 1980s.

Artificial intelligence means the ability of a computer to act like a human being. When a problem occurs, the program should use programmed knowledge to work out a desirable solution, then present the solution to an operator and store the problem situation in a database. The information is stored so that it can be reused if the problem should occur again. This makes the system an intelligent one; it uses its experience to handle problems.
This way of handling problems is desirable in modern society, as most systems nowadays are controlled from a central far away from the actual process. A consequence of this is one use computer system for controlling the process. These systems can only detect that a fault has occurred. An expert system can provide information on what is wrong, why it happened and what can be done to fix the problem. The expert system can also detect a potential fault situation. In this way the process can be stopped and actual problems can be taken care of.

By implementing an expert system to assist the existing system, both the surveillance and operation of a given process are improved.
Typical uses of expert systems include [8]:

- Surveillance
- Interpretation of parameters
- State analysis of systems

An expert system consists of two modules, the *knowledge database* and the *inference engine* [9]. These two modules consist of knowledge about the system and sets of rules on how to find the knowledge and how to present a solution to a user. For development purposes it is practical to split the system into these modules. In case one wants to add something in one part, it is unnecessary to change anything in the other.

3.1. Development

For developing an expert system there are mainly two sorts of tools [9]. One is to develop the whole system using a program language such as C, C++ or Java. This is a time-consuming process and requires a thorough understanding of programming. As a consequence, several companies have developed a new type of programming.
This "new" way of programming is a higher-level language than those mentioned above. The programming is based on evolving expert system, which means that it consists of finished shells for several processes. These shells represent the skeleton of the expert system. This means that after creating the system, one still has to implement program code to make the system do what is required. Still this sort of programming is desirable. It is easy to understand and it is also easy to create the basic structure of the expert system. To create an expert system in this manner, one simply needs to select a shell and implement program codes to the database.

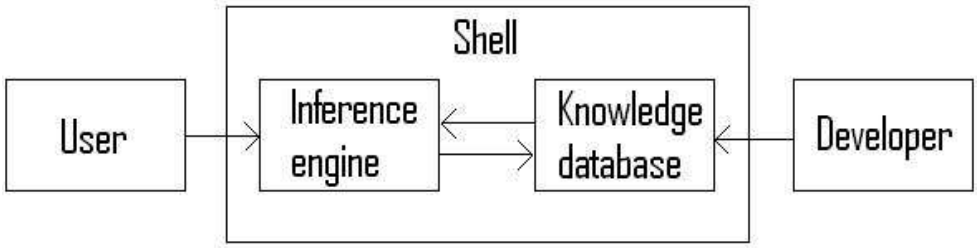


Figure 11, Expert system shell [8]

3.2. Knowledge database

To solve a problem, a program needs knowledge of how to do this. This knowledge is stored in the knowledge database. Here we find large sets of rules on how to handle a given situation. This information can be predefined or collected over time.

When the expert system shell is created, the knowledge database is empty. Supplying the database with relevant information is a time-consuming process and requires expertise within the problem domain [1]. To implement this knowledge, the expert needs to cooperate with a computer specialist. This specialist creates a digitalization of the knowledge the expert possesses concerning the given subject. When programming the expert system, the computer specialist needs to consider the operators that will be used in the system. This means that the program must be user-friendly and easy to understand.
For the system to be able to update the database with new knowledge later on, it has to be self-taught. This is further explained in chapter *3.4.4 Case-based reasoning*.

3.3. Inference engine

When a problem occurs, the inference engine will immediately start reasoning on the basis of the knowledge database and come up with a suitable solution. This reasoning is a search in the database where the inference engine will find a solution (or something close to a solution) that fits the relevant problem best. This searching function is a critical part of the system and is often predefined in the expert system shell. The predefined program code is a set of rules that tell the engine how the search should be performed. These rules must be able to handle the continuous flow of new information as the expert system learns from experience.

The inference engine has another task as well: it presents the solution to an operator. In other words, the inference engine controls the information flow of the system.

3.4. Types of expert systems

Based on how the inference engine searches the database, expert systems are divided into several types of subsystems. In this report, two types of expert systems are presented. These are rule-based and case-based searching [8]. Later in the report one will see that these two types can be combined.

3.4.1. Rule-based system

The rule-based system is based on the predefined rules and facts that exist in the knowledge database. These rules are defined by an expert in that application field and are often represented as a large amount of “if”-statements [10]. This means that if a certain criterion is fulfilled, then an operation is executed. Based on this knowledge, the system can solve the given problem. Some of the “if”-statements are often linked together. An example of this is:

If A then B
If B then C

From these sets of rules, the following can be derived:

If A then C

The rules are often evaluated in terms of how reliable they are. In order to arrange the rules by their reliability, a factor system is implemented. This factor varies between 0 and 1, where 1 is the most reliable rule.

In the rule-based system there are two main methods of reasoning: backward chaining and forward chaining.

3.4.2. Forward chaining

Forward chaining starts with available data and uses the inference rules to conclude more data until a desired goal is reached. This means that the inference engine will search the knowledge database for criteria which are known to be true. It then executes the "then"-statement, saves the data and starts again from the beginning. The inference engine has now updated the data it was previously given and is reusing this data to get new information. This method is also called *data driven*.

3.4.3. Backward chaining

Compared with forward chaining, which is data driven, backward chaining is *goal driven*. To use this goal driven method the inference engine needs a list of goals before it can start searching the knowledge database.
When the inference engine starts the search, it compares the "then"-statements to see which match the goal. When it finds a match, it compares the "if"-statement to the desired solution. If the "if" clause is not known to be true based on the inference rules, it then adds this to the list of goals. And the process starts again from the beginning.

3.4.4. Case-based reasoning

The CBR process is very similar to the human way of solving a problem. This means that when a problem occurs, the program searches its database (case-base) for earlier, similar situations (cases) and uses this information to solve the problem. Often the stored cases are not completely equal to the given case. The result is that the program has to modify these cases so a solution can be presented.
After presenting a possible solution, the program stores the result in the database for later use. The problem solving can be divided into four sequences [11]:

- Retrieve: The inference engine analyses the given problem and searches the database for similar cases.
- Reuse: The matching or near-matching case is presented to the system. This case is compared with the given case. The difference is uncharted. Modification to fit the solution is carried out.
- Revise: The suggested solution is tested. If this is not the right solution, additional modification is performed. This is done until a solution is found.
- Retain: The solution is saved in the database.

These four steps are illustrated in Figure 12:

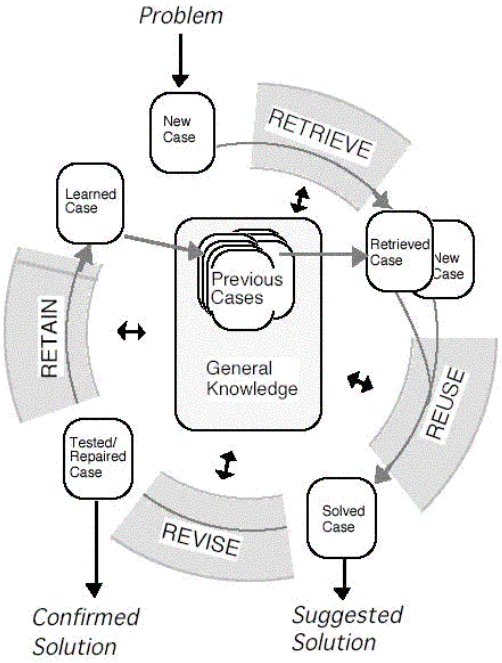


Figure 12, Case-based reasoning [12]

If the case is not solved, the reason for this is saved in the database so that another approach will be used next time the problem occurs.

A CBR system differs from other expert systems in two specific ways [13]:

- It uses knowledge from earlier cases to solve a new case
- Each time a case is treated, the system gains more information which it can use on later occasions.

3.4.5. What is a case?

A case is a predefined situation that may occur as a problem in a system. These situations are defined by a set of demands that have to be fulfilled. An example of a situation may be that the temperature in a generator bearing is increasing:

Case 1:
- Generator is operating normally
- Bearing oil level is normal
- Vibrations are detected
- Cooling water valve to the bearing is opened to max
- Temperature in bearing is increasing

Here we can see the different demands that must be fulfilled in order for Case 1 to be a 100% match to a given problem. This is one part of a case. The other is the part that represents the solutions to the problem. This case has the solution *Defective bearing*.

4. VOLVE KNOWLEDGE TOOLS

Volve Knowledge Tools is mainly intended for problem solving, prediction and decision support for well drilling in the oil industry [12]. The program therefore has a lot of advantages that can be used in an expert system in a hydropower plant. This program has therefore been chosen for continuing work in this project. The main purpose of using this software is to develop a simple version of an expert system and at the same time test if *Volve Knowledge Tools* is suited for use in the surveillance of a hydropower plant. *Volve Knowledge Tools* consists of two modules, *Volve Predictor* and *Volve Knowledge Editor*.

All development and simulations have been performed in this version of *CREEK*. In this report the two modules, *Volve Predictor* and *Volve Knowledge Editor,* will only be referred to as *"Predictor"* and *"Knowledge Editor"* respectively.

4.1. History

Volve Knowledge Tools was developed by a number of persons [14].
In the 1990s, Agnar Aamodts wrote a PhD thesis on "Case based reasoning" where *CREEK* was the raw model of a program. *CREEK* is a shortening for "Case based Reasoning through Extensive Explicit Knowledge". A Lisp version of *CREEK* consisted mainly of the basic architecture and the basic theory of today's program. In the beginning of the 21st century Aamodt and a colleague established the company *Trollhetta*. They developed *CREEK* further and reimplemented it with *Java*. This provided *CREEK* with a better user interface and some of the reasoning processes that are found in the program today. The new version of the program was called *TrollCREEK*.

In 2004, another company called *Volve* was established. This company consisted of four students and three professors from the *NTNU*. The primary goal of this company was to develop a CBR-system that could be used in the oil drilling business. In 2006, *Volve* was sponsored by *Statoil* and with this money *Volve* bought and took over *TrollCREEK*. *TrollCREEK* got an even better user interface and was upgraded in several areas. The new software was called *Volve Knowledge Tools.*

4.2. The CREEK model

The *CREEK* system model consists of three main levels of knowledge [13]. The top-level, g*eneric concepts*, represents the basic structure of each *CREEK* model. This knowledge level is represented in *Volve Knowledge Tools* by the predefined concepts, such as *thing*, *entity* and *symbol* (See Figure 17). This level is common to the whole system and is domain independent. The next knowledge level is the *general domain concepts*. Here we find the knowledge of the process one wishes to describe. This level is represented by the *causal model*. When a case is triggered and the inference engine starts reasoning, the reasoning process mostly takes place in this knowledge concept. The solution to the case is found in this area as well. The final level of knowledge is the *cases.* An illustration of these levels is shown in Figure 13.

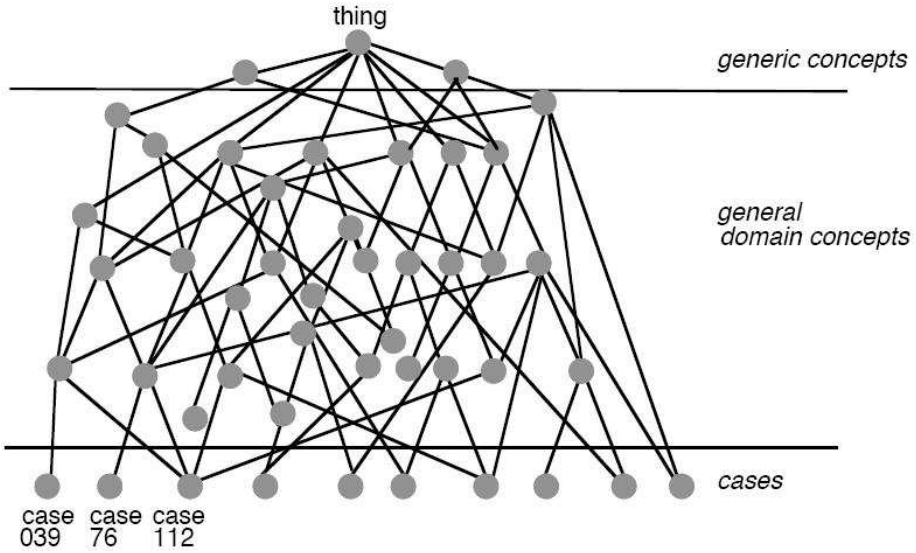


Figure 13, Knowledge model in *CREEK* [13]

In the *CREEK* system there is a strong bond between the cases and the general domain knowledge. This means that the cases in *CRREK* contain a lot of information about the domain application. From Figure 13 it can be seen that the cases are connected directly to the domain concepts. This means that the architecture of *CREEK* contains both model (MBR) and case-based reasoning (CBR). It is primarily a CBR-system with an MBR support component. *TrollCREEK* and *Volve Knowledge Tools* are constructed in this way.

4.3. The reasoning process in CREEK

Reasoning in *CREEK* is based on the same principles as described in section *3.4.4 Case Based Reasoning*, i.e. retrieve, reuse, revise and retain. These are automatic functions implemented in *CREEK*, where each of the four stages is divided into three. These three steps are [15]:

1. Activating relevant parts of the knowledge model
2. Generating and explaining the knowledge that has been activated
3. Focusing towards and selecting a solution to the given problem

4.4. Learning in CREEK

CREEK is a learning program. When a program is self-learning, it means that the program itself has the ability to have new information implemented into its database (case-base). In *CREEK* the program can automatically add new cases into the case-base, making it independent. The cases are added independently of whether the case has been solved or not. If a case is solved by using an old case as reference, the result can be either that a new case is created with the specified solution or that the solution from the old case is modified. In each case the program dedicates more information to its database. Figure 14 illustrates learning in *CREEK* [13].

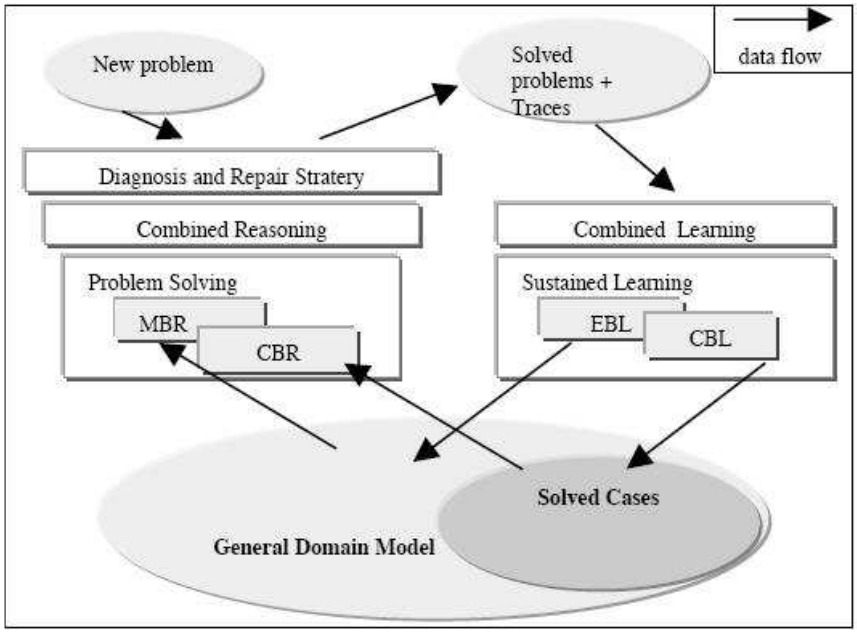


Figure 14, Learning in *CREEK* [13]

4.5. Volve Knowledge Editor

To create a program in *Volve Knowledge Tools* one needs to go through the three steps of creating an *ontology model*, a *causal model* and a c*ase model*. In these three models information about the process is added. When all three steps are accomplished, the program has extensive information about the process. All three models can be made in *Knowledge Editor*, see Figure 15.

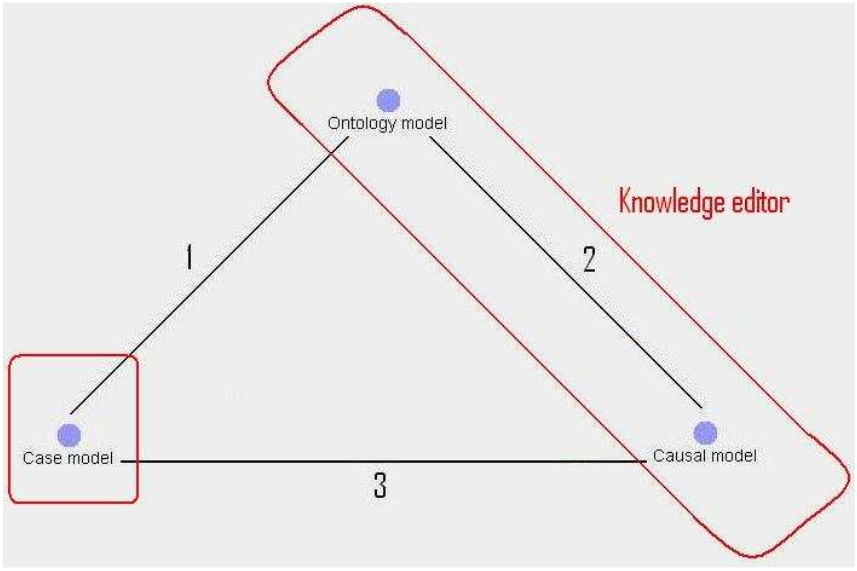


Figure 15, Creating a program in *Knowledge Editor*

In Figure 15 the models are marked with red concerning *Knowledge Editor*. The reason for the dividing of the red blocks is that the *case model* can also be constructed in *Predictor*. This will be further explained in chapter *4.8 Adding cases*.

When the models are created, the program knows which process it will modulate. It also knows how the components work together, how the system reacts to errors and which errors may occur. [12].

4.5.1. Ontology model

The first step in making an expert system is to create the components of the process one wishes to investigate. This is also known as ontology. To add information about a component, a node is added to the *Knowledge Editor*. This node is a representation of a concept and is given a name that suits the concept. For the program to be able to recognise the connection between two concepts, a link must be established between them. This link is called a *relation*. See Figure 16.

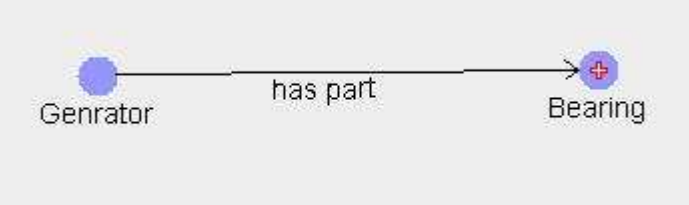


Figure 16, Generator-bearing relation

By adding these two nodes and the relations between them, the program knows that two concepts exist and that one of the concepts is part of the other. In Figure 16 the relation is *has part*. This means that the program is told that the bearing is part of the generator. By creating many of these nodes and relations between them one can finally make a complete system. But there are some precautions that need to be considered. One cannot just add this information directly to the program. The program would not understand what a *generator* and *bearing* are, only that they are somehow connected.
To enable the program to understand that these two concepts are actual components, one needs to add some information. This information is considered as the uppermost subclass level of *Volve Knowledge Tools* and is the basic structure in all *CREEK* programs [16]. See Figure 17.

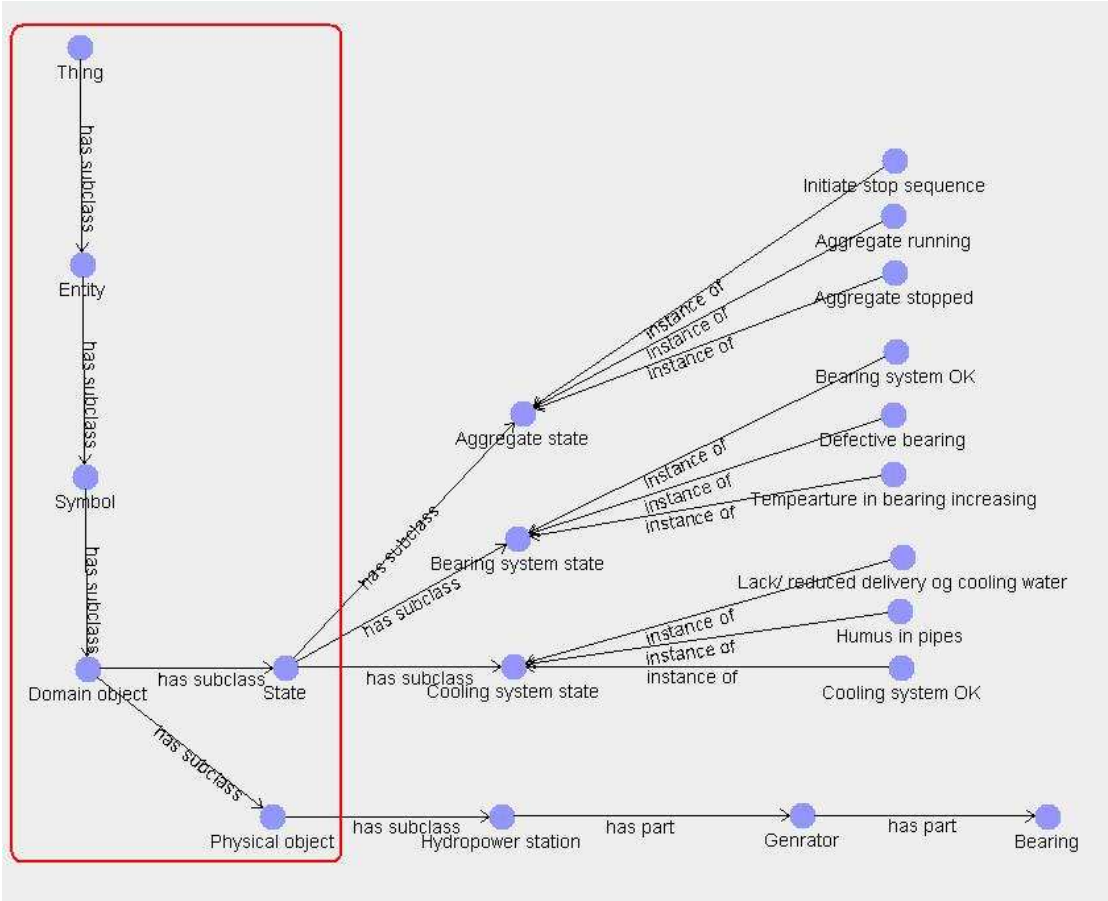


Figure 17, Ontology model

Figure 17 shows how *Volve Knowledge Tools* can be told that a bearing is part of a generator and both are components of a power station. The nodes that are marked with red are the basic structure of *CREEK*. First one must declare that there is a *thing*. A *thing* in *Knowledge Editor* is any thing in the world worth naming or characterising. This thing has a subclass called *entity*. *An entity represents an object from the real world. It can be either a physical object or an abstract* [17].

A subclass of *entity* is *domain object*. By adding this information, the program creates a domain that has an object. This object is a physical object,in this case a power station. As can also be seen from Figure 17 the domain object has a subclass called *state*. In this way the program is told that the physical object one is creating can operate in different states. The definition of *state* in *Knowledge Editor* is *a collection of things at a snapshot of time. A state may be simple, such as a value of a parameter, or complex, such as a total situation* [17]. This chain of information tells the program that a new system model has been created concerning a modelling of a power station. It also tells the program that this power station has several states and components.
From this model it is easy to expand with several new components. For instance the power station also has a cooling system, and the generator has sensors that can detect vibrations. The complexity of the model is dependent on how much of the process one wishes to model.

4.5.2. Causal model

The second step is to create a causal model. The purpose of this model is to tell the program about causes and effects [12]. For instance: If there is humus in pipes, this can cause a lack or reduced delivery of cooling water. This will cause the temperature in the bearings to increase. The increase in the bearing temperature may then cause the aggregate to go to stop.
From Figure 17 it can be seen that the system has three states, the *bearing* state, the *cooling system* state and the a*ggregate* state. As can be seen, these states have describing "sub-states" as well. These "sub-states" are linked together in the causal model. See Figure 18.

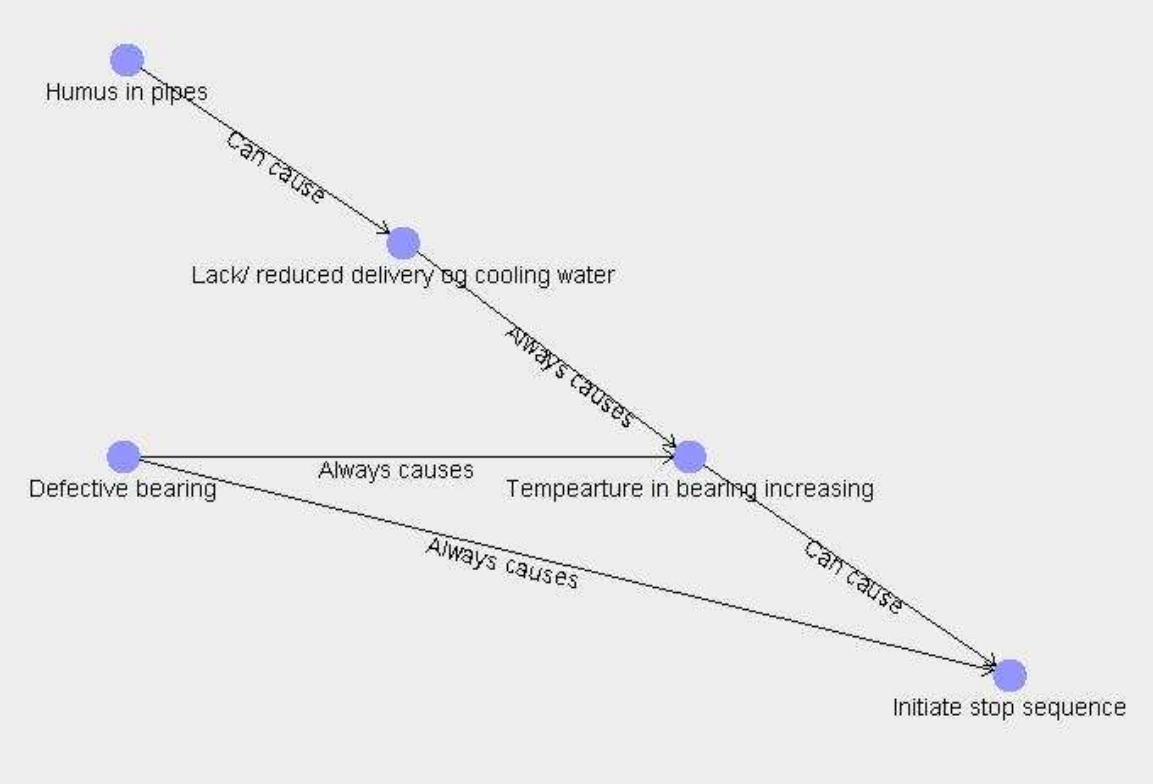


Figure 18, Causal model

In order to reach the *aggregate stopped*-state *Volve Knowledge Tools* can only send a command to the control system to tell the operating program to initiate the stop sequence. The information added in both the *ontology* and *causal* models is knowledge that separates an expert system from another surveillance system.

4.5.3. Map view

Since both the *ontology model* and the *causal model* are built into *Knowledge Editor*, it is important to have a well arranged model. This is done by using several *map views*. A *map view* is like a new page in the *Knowledge Editor*. A good way to use the *map views* is to have one *map view* per model. In this way it is easy to keep control of all the models one is creating. An example of a *map view* is given in Figure 19.

Figure 19, Map view

4.5.4. Relations

Figure 18 shows two relations between the nodes, *can cause* and *always causes*. These two relations are based on how possible it is to go from one node to another. The possibility is given by strength and has a number between 0 and 1, where 1 is a 100% possibility. This means that if the relation between nodes A and B has a strength of 1(such as *always causes*), node A will always lead to node B. For example if a filter is clogged, it is most likely that the pipes have a reduced through-put. In this case the *always* relation is the proper one to use.

When creating a new model in *Knowledge Editor* one often has to add new relations to the knowledge model. By creating these new relations one also creates inverse relations. *Always causes* has an inverse relation, *always caused by*. This inverse relation is a standard relation and may cause some confusion, for instance when looking at Figure 18 and considering why the temperature in the bearings is increasing. To explain why the temperature increases, the program provides the information that it is always caused by the lack of cooling water. This is not completely correct as the temperature can also increase because of a defective bearing. A reasonable name of the inverse relation would be *caused by*. This would make it easier for an human operator to understand why the program has chosen a certain solution.
The reason for inverse relations is that the program should easier understand the relations between two nodes and it is also because the net structure should not be limited to operate only one way. There are also predefined relations in *Knowledge Editor*. These relations are divided into four main classes [16]:

1. Structural relation: Abstract relation class. Consists of relations used for structural purposes in hierarchical systems.
2. Implication relation: Consists of relations used where an action will cause a reaction in the next node with some probability.
3. Associative relation: Consists of relations that associate one node with another.
4. Temporal relation: Consists of relations that are related to a situation or to a state through time-dependent relations.

Some examples of these relations [16]:

Relation type	Relation	Strength
Structural	Has subclass Has part Has instance Has value Has member	1.0
Implication	Always causes Can cause	1.0 0.4
Associative	Occurs in Associated with Described by	0.5
Temporal	Pump-pressure Torque variation Water flow	Individual for each model

Table 1, Relations [16]

Nodes in causal mode are often states that are triggered by an incident. An incident can be, for example a temperature that reaches a certain value. This triggers the state *temperature high*. One can therefore say that nodes in the causal model are states of the system.

In a very large and advanced system, all the relations between the concepts can make the system look messy. See Figure 20:

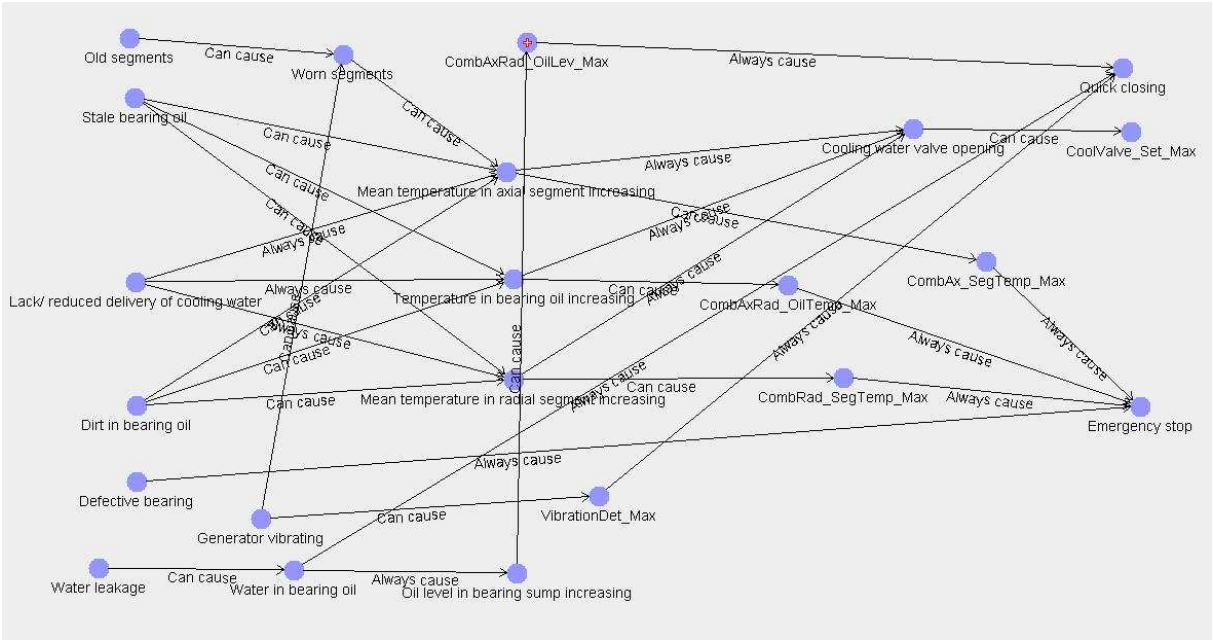


Figure 20, Messy causal model

In order to make the knowledge model easy to understand and easy to read, a function for hiding relations is added. By using this function the model becomes simpler and easier to read. Se Figure 21.

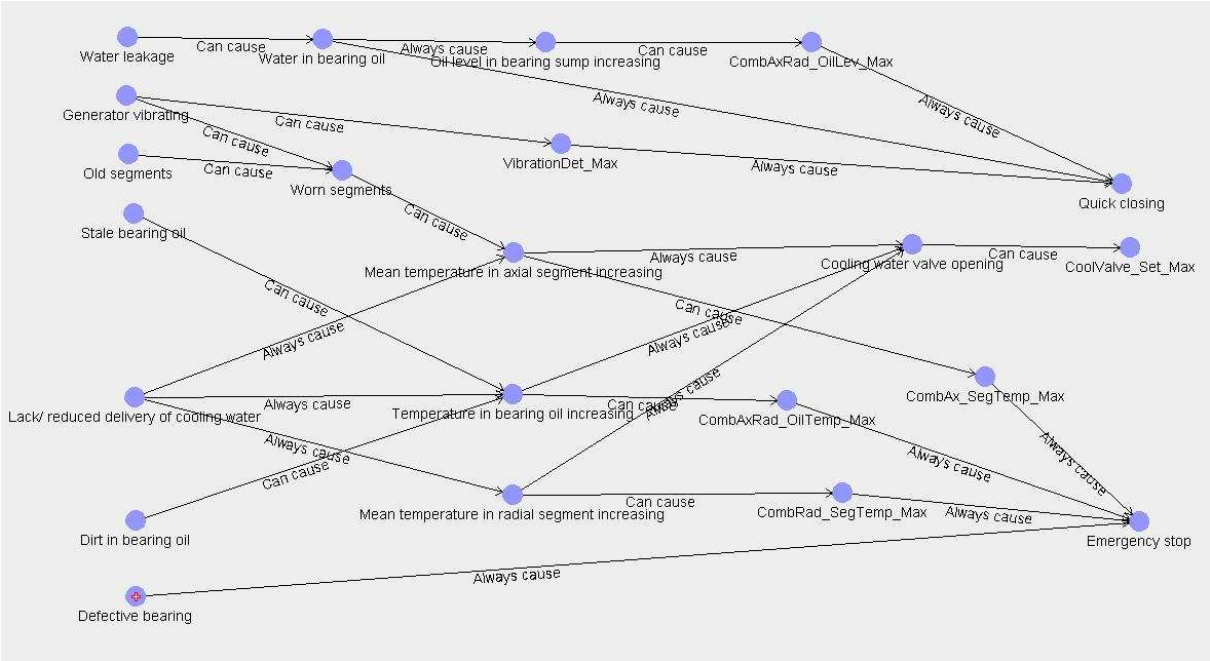


Figure 21, Tidy causal model

One disadvantage of this function is harder to see all the relations to a concept just by looking at the model. To display this information one needs to click on the given concept. The concept is then highlighted and a red dot appears in the node. This is shown in Figure 21, concerning the node *defective bearing*. By mouse-clicking the concepts, a *frame view* label appears in the right-hand part of the program window. The frame contains all the information about the given concept. See Figure 22.

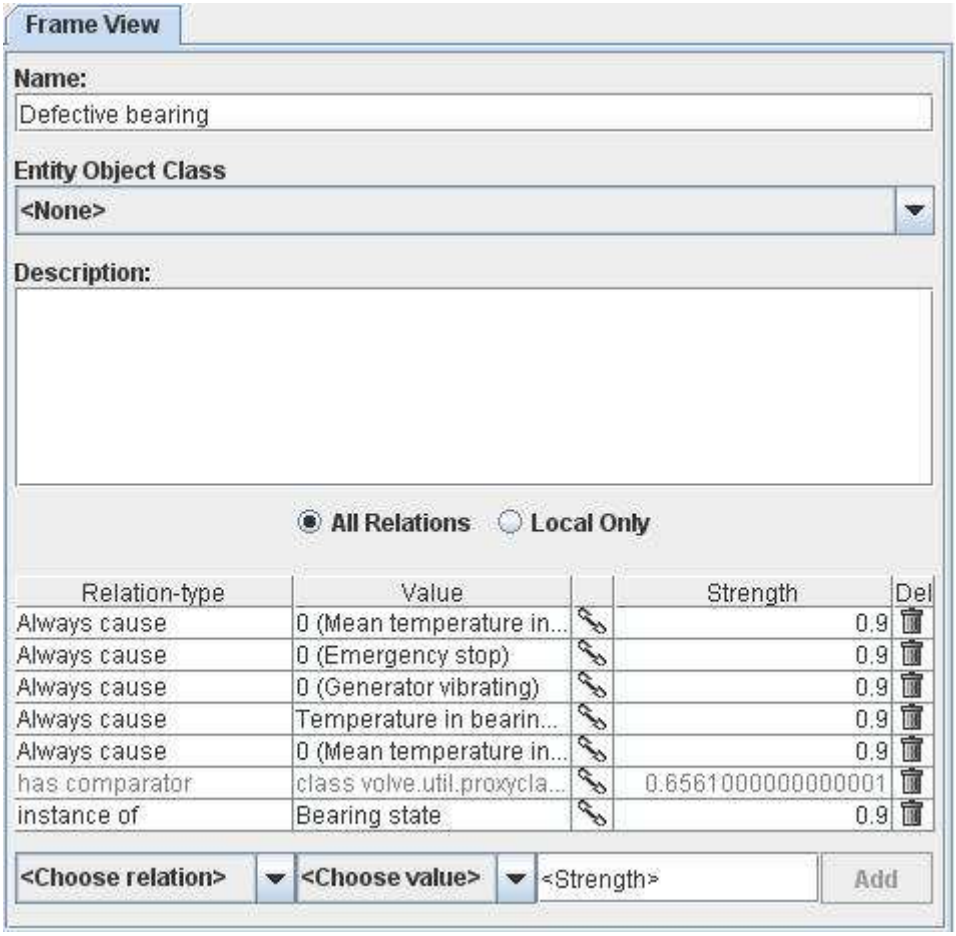


Figure 22, Frame view

As we can see from Figure 21, *defective bearing* has only one relation, e*mergency stop.* But looking at the *frame view* in Figure 22, we see that *defective bearing* has in fact several more. For instance *defective bearing* always leads to:

- Temperature in axial segment increasing
- Emergency stop
- Generator vibrating
- Temperature in bearing oil increasing
- Temperature in axial segment increasing

As a final point the *defective bearing* is also an instance of the *bearing state*. In other words the program is told that *defective bearing* is also a condition of the bearing.
Sewing the different concepts together like this is an important feature of *Knowledge Editor*.

4.6. Volve Predictor

In the *Predictor* module real-time data is read and interpreted. Since *Predictor* and *Knowledge Editor* are two different modules, it is very important that the data in *Predictor* are connected with the right knowledge model in *Knowledge Editor.* This connection is secured by loading a script. The script contains the path of the knowledge model-files and consists of *Java* commands. These *Java* commands also specify how the raw data will be read by creating different sorts of *agents* [18]. An agent is a function of importing the parameter values into the program. These agents are categorised according to how they read the parameters. In *Volve Knowledge Tools* there are large sets of agents. A few examples are listed below:

1. Instant-value agent
2. Derivation agent
3. Integrations agent
4. Mean value agent

When creating an agent, it is very important that its output is assigned to an existing concept in the case-base. If not, there is no link between *Predictor* and *Knowledge Editor,* and the agent will not feed any values into the case-base.
Agents will always interpret the parameters they have been given and do this as long as the program is running. When the agents have finished this interpretation they pass the values on to the case-base. In the case-base the values are compared with the demands of each of the cases. It is here that the case matching begins. Frequently the process the agents have to consider is time crucial. This is no problem for the agents since their performance depends almost exclusively on computer performance. In order to get the agents to function as optimal as possible, they are custom-made for each simulation.

To run an analysis in *Predictor*, there are two things that need to be considered: loading the raw data from the given process and loading the script that tells *Predictor* how to read these data. After doing so, the program is operational. When the raw data are imported into *Predictor*, a list of all the parameters occurs. Now it is up to the human operator to select which parameter he wishes to analyse. If the operator presses *run*, a *CBR Matching*-box appears with all the cases in. The matching of the cases is given in percents according to which one matches the most. All the agents will also appear when *Run* is pressed.
Since *Volve Knowledge Tools* can operate in real-time, the raw data that is loaded into *Predictor* can come straight from the sensors. In this way an operator does not need to press *Run*.

4.7. Adding cases

By adding cases to the knowledge model, the user creates a link between *Predictor* and *Knowledge Editor*. This is because the raw data from the processes are gathered in *Predictor* and it is the cases that interpret these data.
When adding cases to *Knowledge Editor*, several things must be considered, for instance

- which cases/situations one wishes to detect
- which parameters these cases include
- how one wishes to detect these cases.

When this is cleared, one can start to create the cases. The cases can either be constructed in *Knowledge Editor* or in *Predictor*. Cases in *Volve Knowledge Tools* have several descriptions, such as *Case occurrence description*, *Measurements*, *Case Description* and *Normal section weight*. The *Measurements* description is the most important one, and sees to that the case-base is fed with parameters. As Figure 23 shows, the *Agent output* is a parameter of the *Measurement* node.
The purpose of *Normal section weight* is to enable the different cases to be compared with each other so that the program can tell which case matches the most.
Case occurrence description is included to tell the case-base that the cases appear at a certain time. The reason for the *Case Description* is for the user to get information about each of the cases. This description is not vital for the case matching process.

In Figure 23 a simple case-model is shown. This is how a case would look in *Knowledge Editor*.

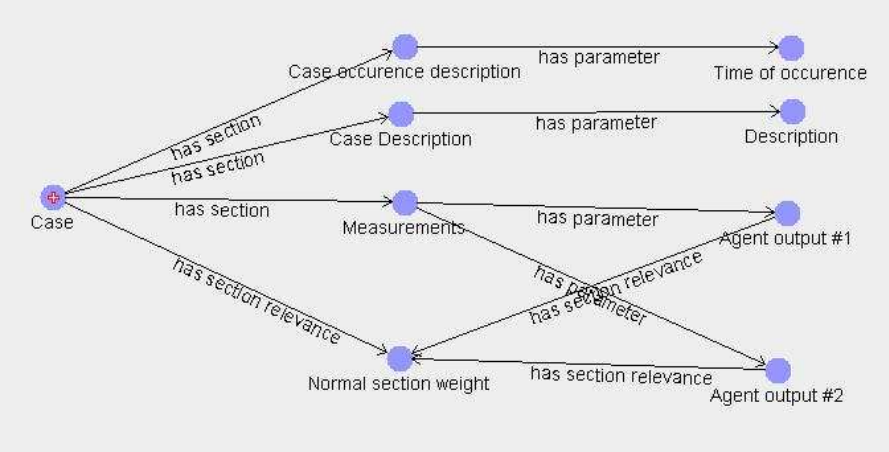


Figure 23, Case model

In order to create a case in *Knowledge Editor*, the case must be constructed using the programming language XML. Once created, the case is stored in the case-base.

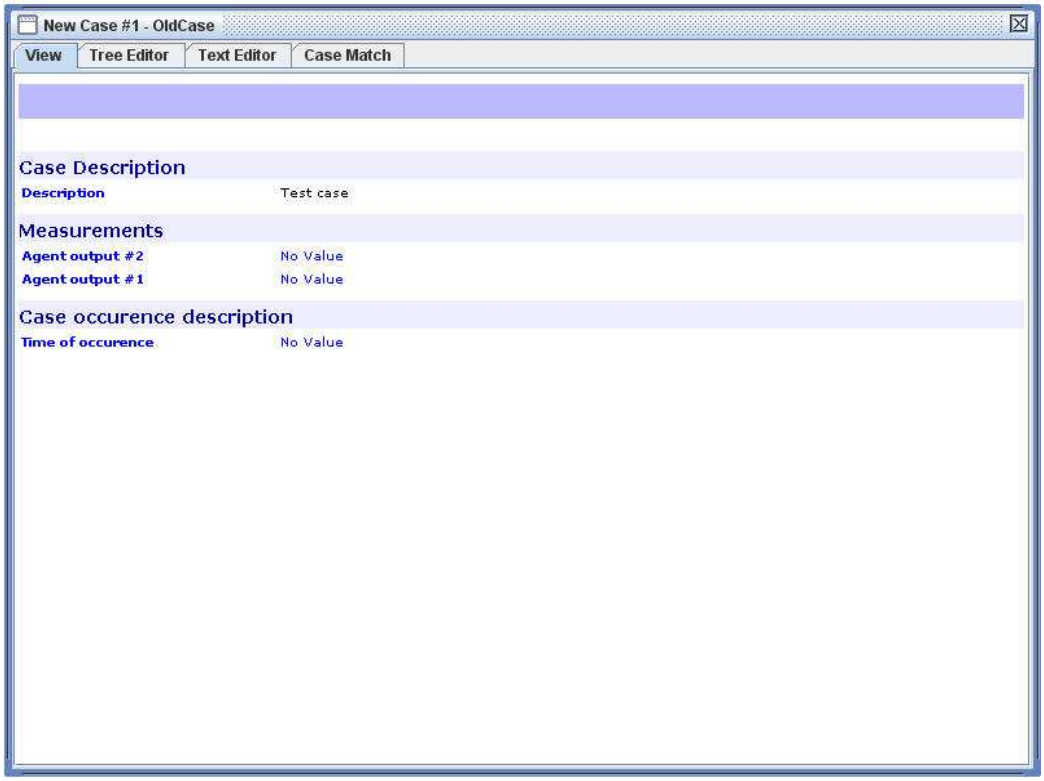


Figure 24, Case in case-base

In Figure 24, a case is shown as it is presented in the case-base. The figure shows that the case has several tabs. In *View*, the cases is described. Here the different measurements are listed. These measurements are the output from the agents. When the cases occurred is also described here. In the tab *Text Editor* the XML-codes are added. The *Case Match* tab contains information about how much the cases matches pursuant to each other.

The other way of creating a case is in *Predictor*. This function is called *Capture case*. When a human operator is considering the graphs in *Predictor* and suddenly sees something that is abnormal, he can just press this button and mark the incident for later investigation and use. In this way the human operator can easily, in small steps, detect new events and expand the case-base. This makes *Volve Knowledge Tools* a learning program and it also includes *Predictor* as a part of the three steps of making the program see Figure 25.

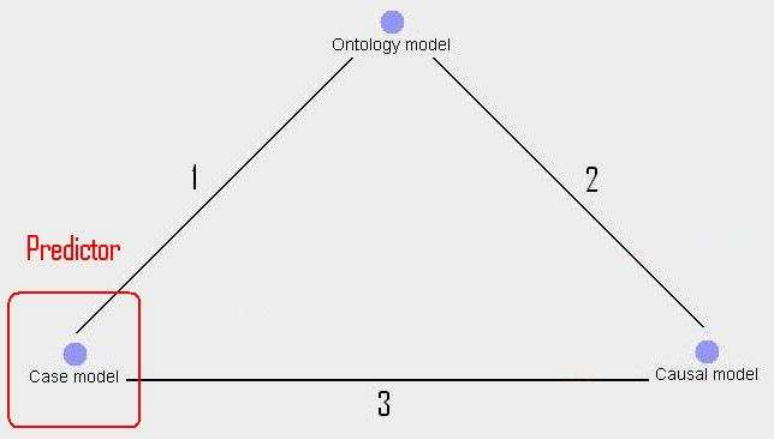


Figure 25, Creating a program in *Predictor*

The cases can have the statuses *Solved*, *Unsolved* or *Processed*. If a case has the status *Processed*, it means that the program has suggested a solution but is not certain that this solution works. The solution must be confirmed by the human operator. After such confirmations, the program knows that this solution was the right one and the status of the case becomes *Solved*. The processed case method is no longer in use in *Volve Knowledge Tools*, but a new version of it will be introduced in the future. This will be further discussed in chapter *4.9 Matching of cases*.

4.8. Matching of cases

In chapter *3.4.5 What is a case?*, five demands are listed. If all demands are fulfilled, Case 1 is a fact and the matching is 100%. Often cases don't match 100%, which means that not all the demands are totally fulfilled. The reason for this may be, for instance, that one of the parameters has not reached the value for that specific case. The result can be that Case 1 is only a 95% match. Case matching is a continuous process. This process compares the solved cases in the case-base with the real-time data that the agent provides. This means that case matching is always running as long as *Predictor* has parameter values, even if everything is running normally.
When *Predictor* is running with a set of raw data, the program searches the case-base for cases that match the data series. In this process the program always presents the case that has the highest matching percent. When searching the case-base for matching cases, *Volve Knowledge Tools* uses the simple method of comparing each of the fulfilled demands with the demands that are in different cases. In Case 1 if vibrations are detected, one demand is fulfilled and so on. This method is called syntactical match.
In previous versions of the program there was another additional case matching method. This method was more sophisticated and is called the relation match method. This means that the program knows that if one demand is fulfilled others will automatically be fulfilled. Looking at Case 1 in chapter *3.4.5 What is a case?*, this method is not easy to use. The method is based on states that always cause another state, independently of external conditions. An example of this is if the bearing does not receive cooling water. The result of this is always that the oil temperature increases. If the purpose was to detect if the temperature increases this method could be used, but this would not had been the case if the purpose was to detect how fast it was increasing. For instance, if the bearing oil was new, the temperature coefficient would had been higher. A result of this, the oil requires much more energy to increase the temperature and the oil temperature would not increase so fast.
Since many things, like how fast the temperature increase, are depending on several criteria, this method cannot be used. In fact almost all things do have states that do not only depending on one parameter. Since very few processes can apply this reasoning method, it has been removed and, as a result the causal model has been made redundant. The co-operation between the models is illustrated in Figure 26. Here the causal model is marked with red to indicate that this model is not a relevant part of the system today. In other words, connections "2" and "3" do not exist.

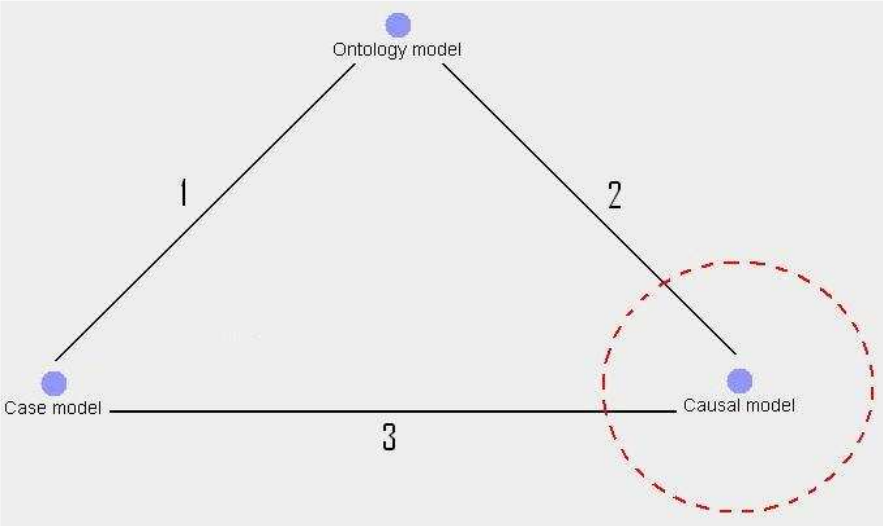


Figure 26, Relation between models

Volve AS is planning to reintroduce the causal model into the system. The consequence of this is that a new and far more advanced case matching method has to be introduced. This method is in the development phase at *Volve AS*, and some of the technology will be introduced already within as little as six months. This technology represents the "2"-connection in Figure 26 and is called *Fuzzy logic*. The "3"-connection is more advanced and needs more time for development. This connection represents adding more information to the causal model based on the cases.
To reintroduce these connections into the model, students can participate by way of a PhD-thesis or similar activity.

4.9. Communication with IGSS32

When making an expert system, one needs to consider communication with the system already existing. The interface, *IGSS Automation Interface* makes this possible. The interface is built on the *Microsoft Automation* platform. This means that every program that supports this standard can communicate with the *IGSS* interface. The interface gives access to:

- Properties of the given objects
- Parameters, alarm limits, i/o-mode
- Alarm lists
- Alarm control and alarm confirmations
- Properties of the system

Since *Volve Knowledge Tools* is constructed on a *Java*-platform, the communication between the two programs can be complicated. To get the programs to communicate with each other, an interface between them has to be created. This is the case with many programs today, two programs with different platform are connected, and it is not unknown to create such an connection.

4.10. Relations between Volve Knowledge Tools, the human operator and the HMI program

Once the expert system is developed, it is ready to be implemented in the hydropower plant. The implementation is a large operation and requires a thorough understanding of how the expert system should cooperate with the already existing systems. Since the approach to the problem has been given by *Voith Siemens*, the HMI program is *IGSS32*. In Figure 27 the relations between the different systems are illustrated.

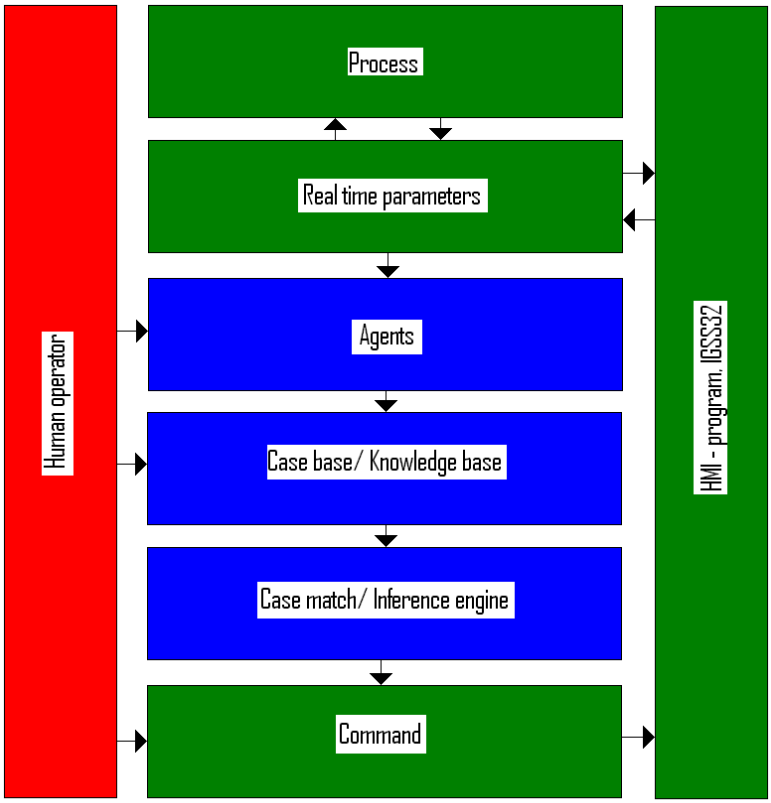


Figure 27, Relation between *Volve KnowledgeTools* and *IGSS32*

In Figure 27 the connection and relations between *Volve Knowledge Tools* and *IGSS32* are visualized. The green sections represent today's surveillance system with *IGSS32* as the HMI program. *Volve Knowledge Tools* is represented as blue sections, and the red section is a human operator. The arrows show how data is sent between the different sections. As shown in Figure 27, the operator can overrun the expert system by giving *IGSS32* a direct command. The operator can also create agents and add new information to the case-base by using the capture-case function.
When a command is given, *IGSS32* changes the real-time parameters, which leads to a modification of the process. The process provides real-time parameters to the agents. These real-time data are interpreted by the agents and sent to the case-base. In the case-base the case matching is running and reaches a conclusion. This conclusion is a command that is sent to *IGSS32*, and the HMI program takes action. The action can be, for example, to open the cooling water valve more, since the temperature in the bearing oil is increasing. Since *IGSS32* and *Volve Knowledge Tools* are built on different platforms, developing the interface between them could be a challenge.

5. WHY USE CONDITION MONITORING

The main purpose of using condition monitoring is to see how the different machines are operating. By getting this sort of information one can easily plan the next maintenance, prevent breakdowns and optimize the running of the machine.

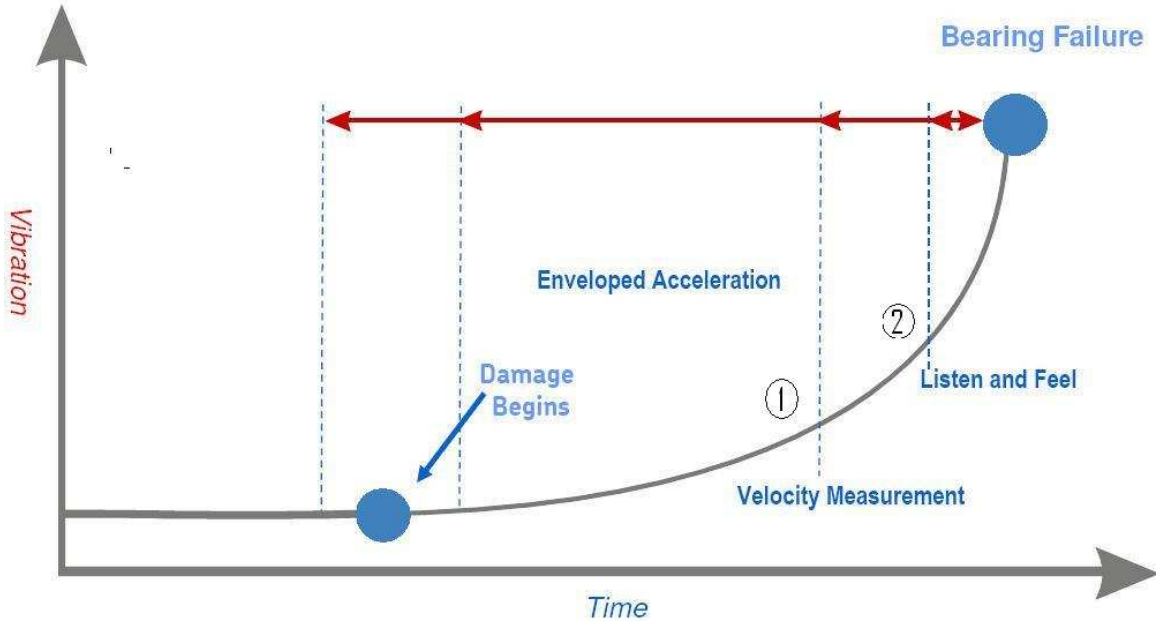


Figure 28, Lifetime of an error

In Figure 28 the progress of a bearing failure is shown. The red arrow illustrates the lifetime of an error from the moment the error occurs until the component is defective. In order to detect the error one can either use sensors and surveillance systems or gamble that the fault will be discovered during the next routine check.
By using condition monitoring of bearings, one can detect an error early (before point 1 in Figure 28) and start planning the maintenance. In this way the maintenance can be performed at a time when electricity prices are low and thus minimize the cost of stopping the aggregate. When the component reaches point 2, the error can be heard and felt. This is the stage where the damage would be detected during a routine check. When studying the remaining lifetime of the error, the time before the bearing is defective is extremely short and the bearing is probably more or less defective already. At this stage it is too late to start planning the maintenance. Often the aggregate has to go to stop immediately even if prices are sky high. This means that condition monitoring is beneficial and very helpful.

5.1. Condition monitoring with and without conditional maintenance

If a company decides to start using condition monitoring of a process, it is very important that the data gathered are used to improve the process. Using condition monitoring alone achieves nothing. Below is a list of situations that will occur if condition monitoring is not followed up by conditional maintenance [19]:

1. Increased running costs
2. Increased need for education of the work force
3. Increased workload
4. No improvement in reliability
5. No reduction of errors in the facility
6. No reduction of stocks
7. No advantages

Based on these facts there is no point in using condition monitoring without conditional maintenance. By doing so, these advantages appear:

1. Optimization of running costs
2. Increased need for education of the work force
3. Optimized workload
4. Potential improvement in reliability
5. Potential for reducing the number of errors
6. Potential for reducing the stocks
7. Significant improvement in the running of the machine

6. SIMULATIONS AND RESULTS

In this report three simulations will be performed, one for each case.
The simulation will be performed in *Predictor* and are based on data series created in *Microsoft Excel*. This will be further discussed in chapter *6.5 Simulations*.

6.1. Time span

The surveillance of parameters can be divided into three sections:

- Momentary: Momentary surveillance of the temperature, dTemp/dt, is often practical, dTemp/dt calculates in real-time how much the temperature in the bearings increase or decrease. These results are compared with how much it is allowed to vary, and a response to given. This is a very important surveillance method because it can detect errors before they really happen, for example if the temperature in a bearing suddenly increases very fast. This is clearly abnormal and is a sign that something is wrong. So instead of the bearing reaching its critical temperature and the machine going to emergency stop, the program detects these abnormal conditions and goes to quick closing or other counteractions. In this way all the components in a power station can be taken better care of and unnecessary stress to the components can be avoided.

- Daily/weekly: For daily/ weekly surveillance, the most practical method is to absolute temperature measurement. This means that the temperature is measured with a certain resolution and the absolute value is compared with a predefined value. In hydropower plants there is a pre-warning system that has a value below the critical one. When the temperature reaches this value the program sends a message to alert the operator. This is the method used in power plants today.

- Long-time: Sometimes the different processes are monitored over longer periods, i.e. months or years. The purpose of this surveillance is to improve the maintenance of the hydropower plant. This sort of surveillance provides a good indication as to whether parts of the station need maintenance, for example if the mean temperature of a bearing has increased over a period of two years. This could be the result if the coolers have gradually been filled with humus and their cooling ability has decreased.

In the following simulations, only the first two sections will be taken into account.

6.2. Measurements

Hydropower stations often employ more than one sensor to measure one parameter. The reason for this is to detect a defective sensor. A common way of doing this is to have three sensors, with the measurements from at least two of them being dominant. This means that the mean value of those two sensors will be the signal that the control system receives. This large amount of information from the process complicates the weighting of each of the signals, placing great demands on the software *IGSS32*. A possible limitation is to have the *PLCs* in the stations take care of this weighting. Such a limitation is very much possible with today's *PLCs.*

6.3. Measurement sampling frequency

In many processes monitored today the sampling frequency of the measurements is a central term. The sampling frequency of the measurements means how often the sensors are triggered to measure a given parameter value. One might believe that the measuring of sensor data is done continuously, this is not the case, but sensors can measure several times per second. In many cases, a sampling frequency of more then one sampling per second, are not needed, since many parameters do not change that much within this short amount of time. An example of this is temperature in water. Since water has a high specific heat capacity it takes a lot of energy to change its temperature. As a result, the temperature does not change vary rapidly. Surveillance of water temperature in a hydropower station is only relevant in the case of the cooling water temperature. These measurements have a resolution of one measurement per half minute or higher.
In some cases the resolution of the measurements is crucial. In these processes the highest possible resolution is desirable. This of course is dependant on several factors, for example how fast a sensor can measure the parameter and continue to pursue the information. Another restriction on how high the resolution can be is data processing and storage. How high the resolution of the measurements should be, is proportional to the amount of data that is going to be processed and stored.
When creating a surveillance system, all these factors must be taken into consideration. This is also crucial for the agents in *Predictor*.

6.4. Cases

In this report, three cases are looked into. The cases primarily concern a defective bearing and water in the bearing oil. For more information on the cases, see appendix *10.3 Cases and agents*.

6.4.1. Case #1

The symptoms of a defective bearing are mainly that the temperature in the bearing segments increases and vibrations are detected. These two parameters are compared with the position of the cooling water valve, the produced effect and the oil level. If vibrations are detected, the temperature in the segments is increasing, the cooling water valve's position is 100%, the aggregates are not running on overload and the oil level is normal, it can be said with great certainty that a bearing is defective.
In order to detect whether the cooling water valve is fully open or not, a limit-switch would be sufficient. Since the measuring of the cooling water valve is a percent indicator, an agent is created to see if the percent is 100 or not. The same type of agent is used for detecting vibrations and whether the generator is operating on overload or not.
Since the oil is splashing around in the bearing, the oil level will vary within certain limits. To measure the oil level a mean value agent is used. The agent integrates 30 of the previous measurements and divides this number by 30. If the level is above a certain limit the oil level is high, if below another limit it is low, and in between the oil level is normal.
To detect if the temperature in a bearing is increasing, a pre-warning system is established. The program compares the instant temperature with a given pre-warning value. When this value is reached, the program investigates if the temperature is still increasing. If it is, the program sends a message to the control system that the temperature in the bearing is increasing after the pre-warning. When all these states occur, the error is detected and the machine goes to *quick closing*.
Another scenario is if the temperature increases and none of the other states occur. In this case the temperature will be investigated individually and if the temperature increases above a certain limit after the pre-warning has been issued, the machine goes to *Emergency stop*.

CASE #1	
1.	The aggregate is running normally at 55 kW
2.	Vibrations are detected.(pre-warning)
3.	Temperature in bearing segment starts to increase
4.	Cooling water valve position opens further
5.	Temperature in bearing segments stabilizes for a moment, then increases further
6.	The temperature is increases along with the cooling water valve until the valve has reached 100% open
7.	The temperature in the axial bearing segment increases further and reaches the pre-warning value: pre-warning is sent to control system.
8.	The temperature increases further. The error is detected and the aggregate goes to quick closing. Message to operator: *"Defective bearing"*

Table 2, Case #1

6.4.2. Case #2

Sometimes when a bearing is defective the temperature increases very fast. Today's surveillance does not include this form of monitoring, but only observes the maximum level of the absolute temperature. When the temperature increases rapidly in a hydropower station today, nothing is triggered until the temperature reaches its pre-warning level. By the time the temperature reaches its critical value, the aggregate goes to emergency stop. As mentioned in chapter *2.4.3 Emergency stop*, the emergency stop exposes the components in the station to a lot of stress and is not desirable. An easy way to prevent this is to monitor the derived function of the temperature.
If the temperature is increasing rapidly, the program should detect this at an early stage and send a command to IGSS32 that the aggregate should go to *quick closing*. This should prevent unnecessary stress to the different components.
This will be compared with whether the generator is running on overload.

To detect if the temperature increases rapidly within a short amount of time, a definition of 'short time' must be provided. In these simulations a limit of two-and-a-half minutes is set. This corresponds to 30 measurements. The agent used in this case is a derivation agent. This agent derives the 30 previous measurements and sums the result. If the result is above a certain value, and the oil level is normal, and the generator is not operating on overload, the machine goes to *quick closing.*

CASE #2	
1.	The aggregate is running normally at 55 kW
2.	Temperature in bearing suddenly increases rapidly, i.e. the temperature is increasing by one or two degrees per measurement
3.	Oil level and produced power are at normal values
4.	Aggregate goes to quick closing. Message to operator: *"Temperature in bearing increasing rapidly, possible defective bearing, go to quick closing*

Table 3, Case #2

6.4.3. Case #3

To detect if there is water in the bearing oil, the oil level in the bearing needs to be considered. This level will be more or less constant when the bearing is operating under normal conditions. As a comparison with this parameter the running time of the reserve bilge pump and the temperature in the bearings are monitored. These parameters are compared with the produced power.
Since there is no parameter called *On-time of bilge bump*, one can create a counter that starts every time the pump is switched on. The reason for monitoring the on-time of the bilge pumps is because it is an indirect indicator of whether the water level in the sump is increasing. If it is, a water leakage has occurred. This leakage could be responsible for the increasing water in the oil. The reason for monitoring just the reserve pump and not both is that the reserve pump will only operate if the water level reaches a certain value. If this value is reached, it is a sign that there is something wrong with the system. Most likely a water leakage has occurred.
The temperature in the bearing functions as a final comparison and is also an indicator that there is water in the oil. If there is water in the oil, the viscosity of the oil changes and the oil has reduced load carrying capacity. This results in more friction between the rotor and the bearing surface and the temperature rises. All these parameters are compared with the produced power. If the generator is not running on overload, a fault has occurred.

CASE #3	
1.	The aggregate is running normally at 55 kW
2.	Reserve bilge pump starts
3.	Oil level in bearing increases
4.	Temperature in bearing segments increasing.
5.	Aggregate goes to quick closing. Message to operator: *"Oil level unnaturally high, possible water in oil"*

Table 4, Case #3

6.5. Simulations

The purpose of these simulations is to see if *Volve Knowledge Tools* can handle this way of detecting fault conditions.
Due to the short time available for performing the investigation, the parameters are limited to a general point of view. A real model would be far more advanced. For example the detection of a leak involves more than just considering how long the reserve bilge pump is running. But by using the bilge pump as a basic approach, it is easy to further expand the system. In this way one get to test *Volve Knowledge Tools* can be tested more thoroughly than Fjellheim's work. Another means of testing *Volve Knowledge Tools* is to use derivation for detecting abnormal behaviour at an early stage.

The parameters involved in the simulation are listed in Table 5.

Signal name	Parameter name	Resolution
Vibration	**Vibration**	**5 sec**
Temperature in axial segments	**CombAx_SegTemp**	**5 sec**
Temperature in radial segments	**CombRad_SegTemp**	**5 sec**
Temperature in oil	**CombAxRad_OilTemp**	**5 sec**
Oil level	**CombAxRad_OilLev**	**5 sec**
Cooling water valve position	**CoolingWaterValve_Setpoint**	**5 sec**
Produced power	**Produced_Power**	**5 sec**
Reserve Bilge pump on/ off	**RedBilge_Pump**	**5 sec**

Table 5, Simulation parameters

Because of the simplicity of the system and the few parameters, the raw data series is constructed in *Microsoft Excel*. The values of the parameters are realistic as compared with the real world. This means that if the temperature in the bearing segment increases, the temperature of the bearing oil will also increase after a small delay. At the same time the cooling water valve is opens further to lower the temperature. These data are transformed into *Excel CSV*-format so that *Predictor* can read them. Since Fjellheim has already performed simulation in *Predictor* using log-files from *IGSS32*, this is no important issue, but one that has been considered.

The total simulation time is limited to one hour. This is of course too short compared with the real world. But since the purpose of the simulations is to see if *Volve Knowledge Tools* can handle the new measuring method, it is acceptable. The new measuring methods are derivation, integration and mean values. In this way the software can be further tested.

Figure 29 shows a screenshot of *Predictor* in operation. The different cases are further described in appendix *10.3 Cases and agents*. Here all the agents are mentioned as well.

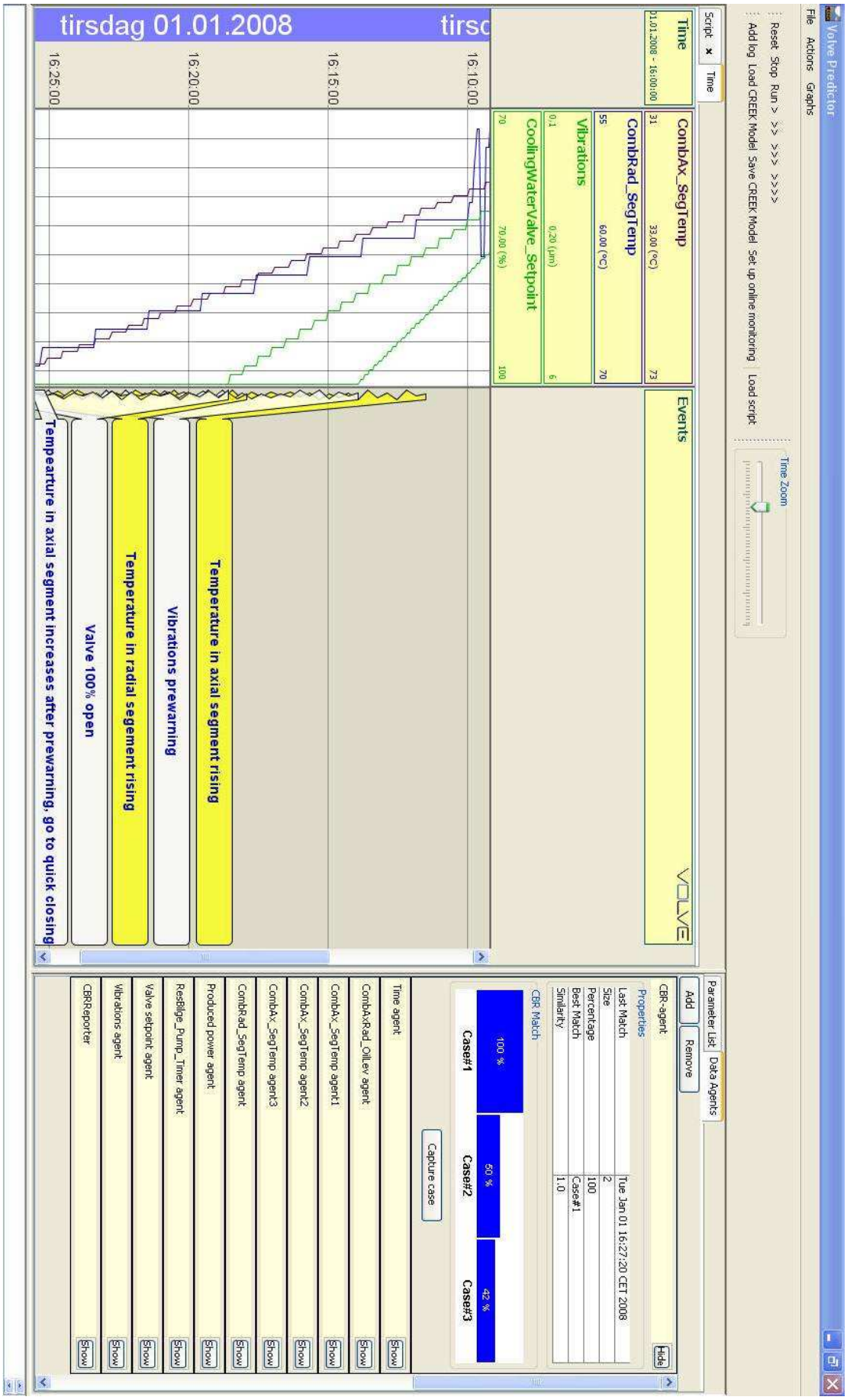
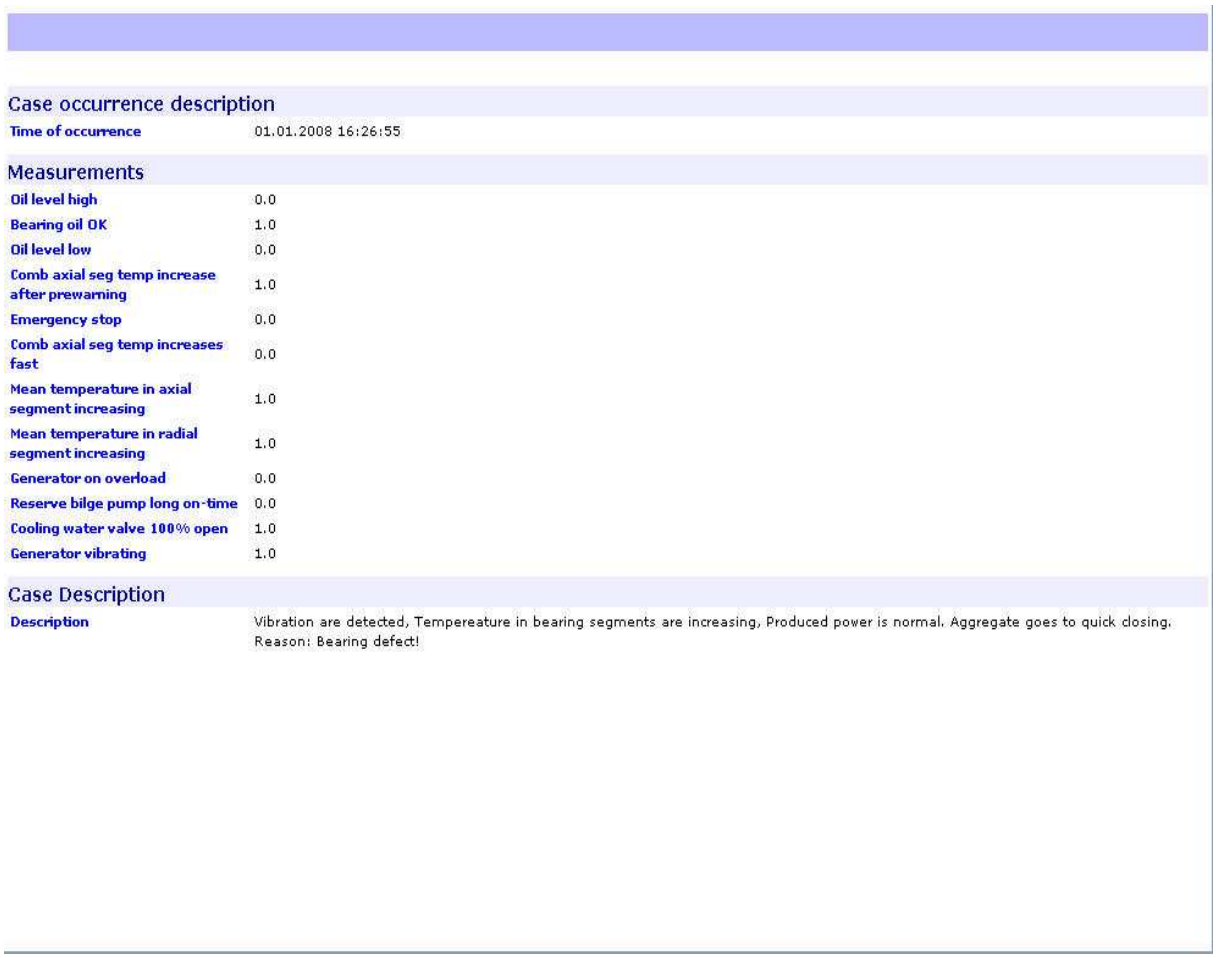


Figure 29, Screenshot of *Predictor*, case #1

In Figure 29 shows how the parameters changes over time. The red line in the centre of the window represents the present time and moves along the vertical time-axis. In the figure the time resolution is shown for every fifth minute, this can be whether zoomed in or out by using the time zoom scroll bar.
When the parameters reach a certain value, the agents display a message to the human operator. The function is called *Event*, and makes *Predictor* more user friendly. As shown in Figure 29 these messages can come in two different colours, yellow and white. This is can be an indication of how severe a message can be.
On the right-hand side the parameters are listed. Below these there is a *CBR Matching*-box. Inside this box there are four indicators. These indicators illustrate how much each of the cases, *Case #1*, *Case #2* and *Case #3*, match the data that are being read. The matching is measured in percent. By double-clicking one of these bars, a window appears which contains information about the given case. In this window the demands of each case are listed and there is also a case description and information when the case matched 100%. This is the same window that is shown in Figure 24, only more complete. See Figure 30.



Case occurrence description

Time of occurrence	01.01.2008 16:26:55

Measurements

Oil level high	0.0
Bearing oil OK	1.0
Oil level low	0.0
Comb axial seg temp increase after prewarning	1.0
Emergency stop	0.0
Comb axial seg temp increases fast	0.0
Mean temperature in axial segment increasing	1.0
Mean temperature in radial segment increasing	1.0
Generator on overload	0.0
Reserve bilge pump long on-time	0.0
Cooling water valve 100% open	1.0
Generator vibrating	1.0

Case Description

Description	Vibration are detected, Tempereature in bearing segments are increasing, Produced power is normal. Aggregate goes to quick closing. Reason: Bearing defect!

Figure 30, Case description, case#1

In Figure 30 there is presented a case description. It contains why case#1 is matched 100%, and which commends have been send to the HMI program. The reason is *defective bearing*. For more information about the simulations, see appendix *11.4 Screenshots from simulations*.

6.6. System model in Volve Knowledge Tools

As mentioned in chapter *4.7 Predictor* one has to create a knowledge model that matches the simulations performed in *Predictor*. This model consists first of a hardware part, in which the system to be modulated is created. The different states of the system are also defined in the system model. A screenshot of the system is shown in Figure 31. For more information concerning the system model see appendix *10.1.1 Ontology model*.

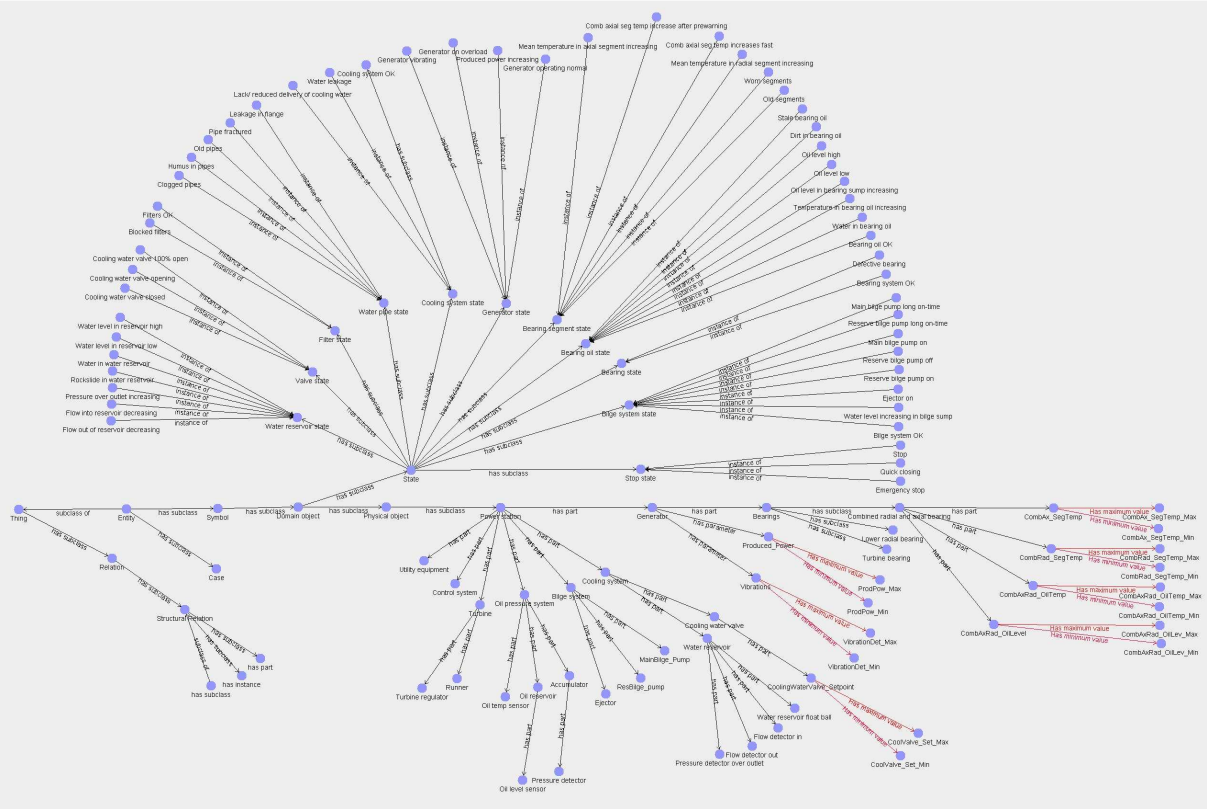


Figure 31, Hydropower station ontology

The model in Figure 31 is very good starting point for continued work. One reasons for this is that it already includes many other systems in addition to the bearing system. The model is well suited for further work, whether to improve what has already been done or to consider other parts of the power station. In Figure 31, some of the relations are marked with red. The red relations are the parameters that are going to be used in this simulation. The reason for this marking is to separate the parameters from the other signals as a matter of form.

6.7. Knowledge model in Volve Knowledge Tools

In the knowledge model all the different states are linked together, and it is here that the system gets its knowledge of how the processes in the system work. When an abnormal state occurs in the system, the inference engine searches this model for an explanation of what is wrong.
In Figure 32, one of the knowledge models is illustrated. For more information concerning the knowledge model, see appendix *10.1.2* and *10.1.3 Causal model*

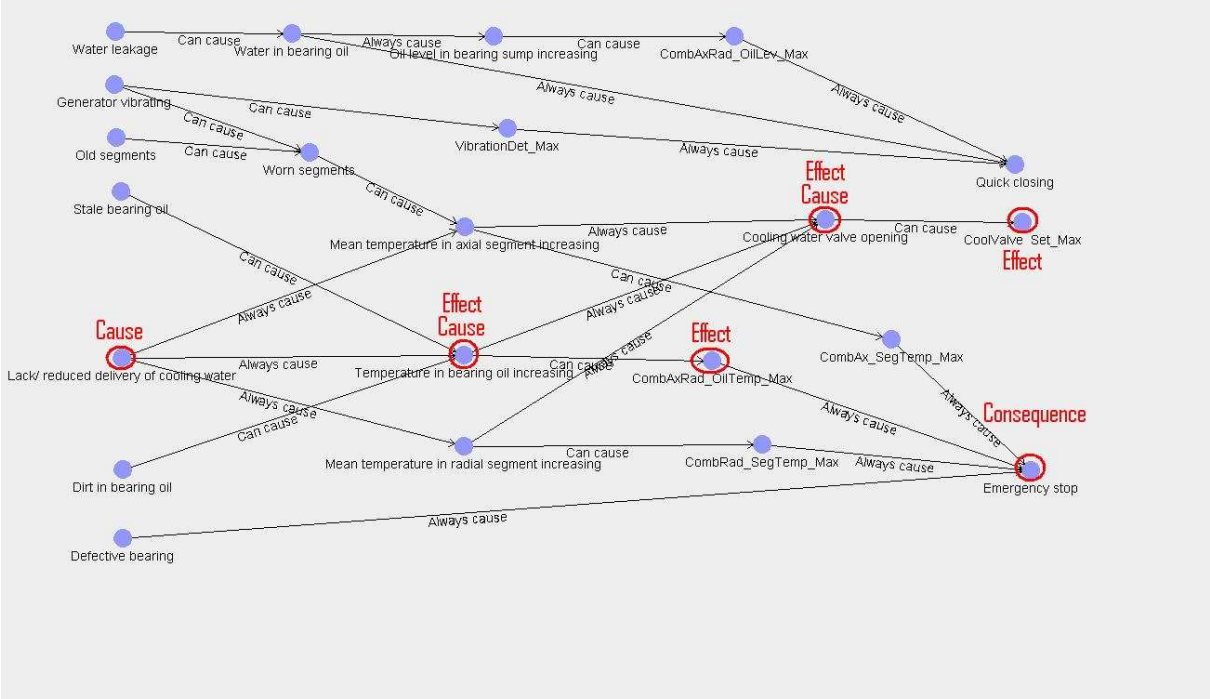


Figure 32, Causal bearing system

In Figure 32 the causes and effects are illustrated. One can see how the system reacts to one state and changes into another. In the figure the system detects a lack of cooling water. This causes the temperature in the bearing oil to rise, and a new state is reached. In this way the *Temperature in bearing oil increasing*-node is the effect of missing cooling water. Furthermore, the increasing temperature in the oil causes the cooling water valve to open further. This adjustment will continue until the valve goes to full opening. If the temperature in the bearing still increases and reaches its maximum value, the generator starts the emergency stop procedure.

As mentioned in chapter *4.6.1 Ontology model* the model is a good starting point for continuing work. This especially concerns the causal knowledge model. In the latest versions of *Volve Knowledge Tools*, the causal model has been excluded from the monitoring process. The reason why is explained in chapter *4.8 Matching of cases*.

6.8. Case model in Volve Knowledge Tools

The third and final step is to create the cases. In this process it is important that *Predictor* and *Knowledge Editor* correspond to each other.

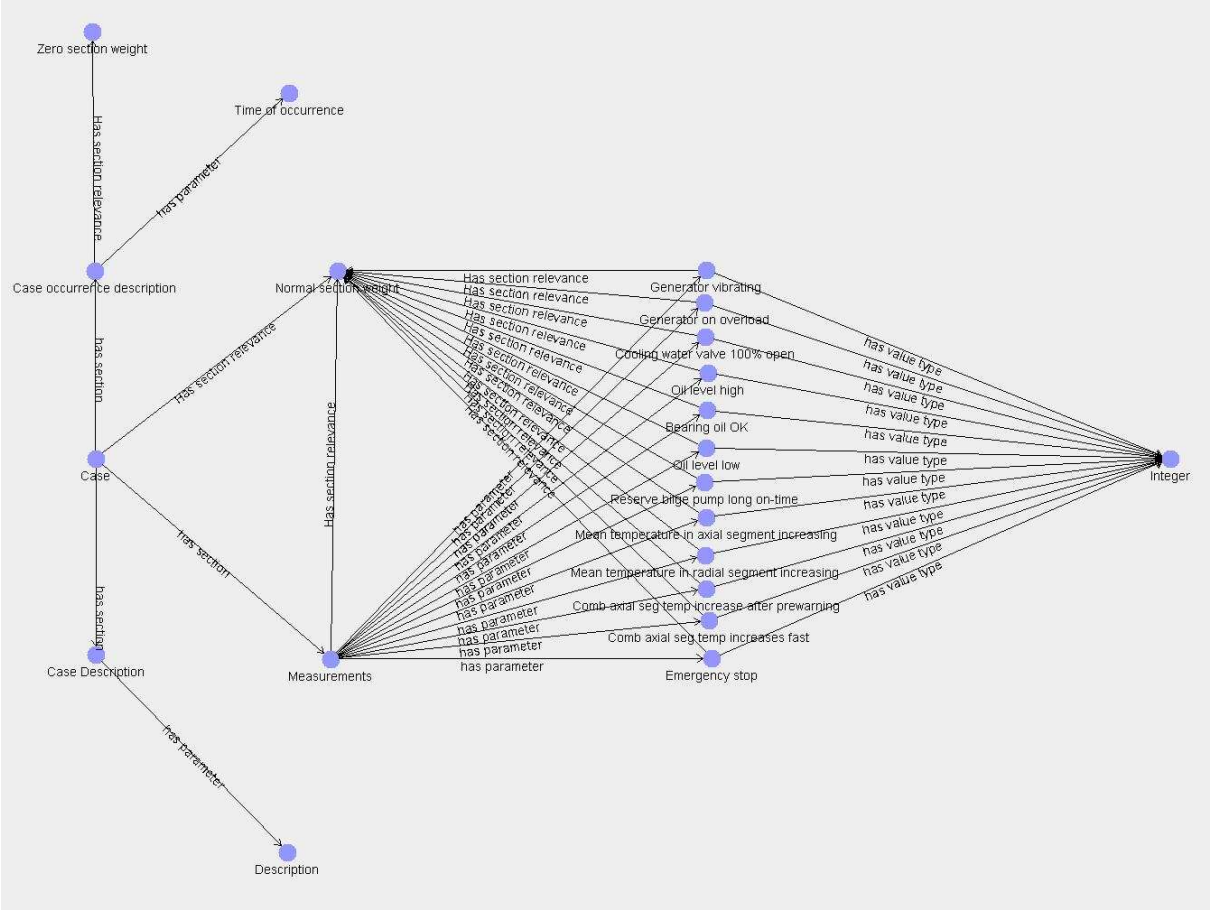


Figure 33, Cases

Figure 33 illustrates how the parameters are imported into the case-base by agents. It also shows that the parameter values have the value type *Integer*.
For further information on the model see chapter *4.8 Adding cases*.

7. DISCUSSION

The intention of this report is to investigate if *Volve Knowledge Tools* is capable of performing the functions required of an expert system. This investigation is about exploring the software and mapping the features it possesses. To explore the software a program has been developed and tested through simulations.

7.1. Developing the program

To create a program in *Volve Knowledge Tools* it is necessary to go through three steps: creating an *ontology model*, a *causal model* and a *case model*. To these three models information about the process is added. When all three steps are accomplished, the program has extensive information about the process. The models consist of information about which process the program will modulate. It also knows how the components work together, how the system reacts to errors and what errors that may occur.

Since *Volve Knowledge Tools* uses a graphical interface, the program is clear, easy to use and easy to create for a new operator. The graphical interface sees to it that the system is easy to read even if the system is large and sophisticated. A tutorial on creating a simple CBR-system for condition monitoring of a car, gives a quick introduction to the software. The introduction gives the user knowledge of how the three models work together, and hence how the program works. After completing the tutorial, the user is ready to create a system on his own.

In the final step of developing the program, cases are created. The cases represent events that may occur in the system. A feature of *Volve Knowledge Tools* is that one can add information about the cases. The information indicates why the case occurred and which actions should be considered. This is useful when an event occurs and the operator is perplexed. This function makes the program user-friendly.

Volve Knowledge Tools is a program that can be installed on an average computer. The only specification required is at least 2 GB of memory. The reason for this requirement is that the *Predictor* can load huge amounts of external raw data. A general recommendation for the computer utility is that the computers memory capacity should be four times the size of the external raw data. Since 2 GB is the average standard nowadays, this would not be a problem.

Since the program is still under construction, it has a huge potential for becoming an even better learning program. When the program gets its new learning ability, it can gain its own knowledge and thus be very suitable for use in expert systems. The developing of this feature is already in progress and will be included in the program in approximately one year. Still, *Volve Knowledge Tools* can already be considered a learning program, thanks to the *Capture case*-feature in *Predictor*.

Even though the program is evolving in a positive direction, the developing phase sometimes creates bugs. This can of course result in some program errors. Since *Volve Knowledge Tools* is used by oil drilling companies, security standards are extremely high. In order to maintain this security standard, extensive troubleshooting procedures are initiated. Therefore, computer

errors that might occur will be insignificant to the monitoring. In the simulations performed in connection with this report no errors occurred.

Volve Knowledge Tools is created on the *Java* platform. Since *IGSS32* was built on another platform, *Microsoft Automation* platform, the communication between the two programs may cause some problems. However, many of the communication programs today are built on different platforms, so communication between *Volve Knowledge Tools* and *IGSS32* should be possible.

7.2. The simulations

Because of the simplicity of the system and the few parameters, the raw data series is constructed in *Microsoft Excel*. The values of the parameters are realistic as compared with the real world. For instance if the temperature in the bearing segment increases, the temperature of the bearing oil will increase after a small delay. At the same time the cooling water valve is opens further to lower the temperature. These data are transformed into *Excel CSV*-format so that *Predictor* can read them. Since Fjellheim has already performed simulation in *Predictor* using log-files from *IGSS32*, this is no important issue, but one that has been considered.

Even though the data is created as a log in these simulations, *Volve Knowledge Tools* can handle real-time data, directly from sensors in a process. This is an obvious criterion for a development kit for expert systems. Since the amount of sensor data processed is only limited by the memory capacity of the computer, *Volve Knowledge Tools* can handle large amounts of data. Since the agents in *Volve Knowledge Tools* are custom-made for each measurement, the program is very flexible and can interpret the raw data in ways to suits any customer.

When simulating in *Predictor*, the layout of the program is very tidy. The simple layout makes the program easy to operate and easy to use. *Predictor* also has the function *Capture Case*. This function makes it easy to implement new cases to the case-base, thus expanding the knowledge of the program. When the program is detects cases, it uses previous experience with the real-time data. This is a form of artificial intelligence which is also a criterion for making an expert system.

Because of the *Case description* function in the cases, it is easy for an inexperienced operator to know what to do when a fault/case occurs. The description tells the operator why the case occurred and what actions should be taken into consideration.

One disadvantage of *Volve Knowledge Tools* is that the program can not function as a surveillance system alone. The program can not give direct orders to the process, for instance tell an actuator to open a valve. If the program detects an error, it can only send an error message to the HMI program telling it what to do. The HMI program then carries out the order. However, in the present investigation of using *Volve Knowledge Tools* to create an expert system, the program will work as a supplement to today's surveillance system. This so called disadvantage becomes insignificant.

8. CONCLUSION

The background of this report is the desire to improve the surveillance system of hydropower stations. This improvement involves implementing an expert system. To make an expert system, development software is needed, and in this case *Volve Knowledge Tools* was used. The intention of this report is to investigate how suitable *Volve Knowledge Tools* is for developing expert systems.
By developing a simple expert system in *Volve Knowledge Tools* and using this program for simulations, I gathered a lot of information on how the program operates and on its functions. After considering the advantages versus disadvantages and the suitability of the program for developing expert systems, I concluded that *Volve Knowledge Tools* has the capabilities required for developing an expert system. I therefore recommend *Volve Knowledge Tools* as a development kit for evolving an expert system in further work.

9. RECOMMENDATIONS FOR FURTHER WORK

By describing how the different parts of a hydropower station functions, one can detect the different errors and faults that may occur as well. Some of these situations are not detectable by surveillance systems today. Below is a list of ideas that would help to improve today's surveillance systems [2].

1. Frequency monitoring of the machines: By performing an analysis of the sound pattern of the machine, deviations from normal operation will be detected quickly and easily. Partial discharges and cavitations may also be discovered. A disadvantage of frequency analysis is that the system would need to be expanded. When expanding a system there are always some difficulties in integrating the new component into the old system. The expansion requires the implementation of a microphone.

2. Surveying the start-up and stopping times of the aggregate: By performing this simple surveillance one can get an indication that something is wrong if the start-up-time or the stop-time is suddenly much shorter or longer than usual. A microphone would be useful here as well because it can detect slouch scuffling. Investigating this time interval compared with other parameters can provide useful information concerning the system.

3. Identify ways to get the *case model* to add new information to the *causal model*.

4. Surveillance of the amount of clearance water from the turbine. If the amount of clearance water increases and vibrations are detected, the labyrinth gland may be worn.

5. Surveillance of the outdoor climate, temperature in the water and the magnetic ability of valves and sluices. By monitoring this, heat cables or pressure air can be activated if equipment such as valves and sluices threaten to freeze. In this way the station can at any time go to stop in all weather conditions.

6. What happens to the expert system at power failure?
   - How should the expert system be reset?
   - What happens if the communication with *IGSS32* is down?

7. Further improvement and expansion of the bearing model

8. Comparison of parameters to detect fault situations in other areas of the hydropower station.

9. Establish a cooperation with an IDI student with the purpose of making an interface between *Volve Knowledge Editor* and *IGSS32*.

10. Establish a cooperation with the automation or machine engineers at HIST with the purpose to make a hydropower station emulator.

10. REFERENCES

[1] Fjellheim, Ø., Ekspertsystem i Vannkraftverk, Master thesis, Norwegian University of Science and Technology, Department of Electrical Engineering, 2007.

[2] Timberlid, E., The digital machine expert – Expert system for monitoring hydropower plants.(Preliminary study for master thesis) Norwegian University of Science and technology, Department of Electrical engineering, 2007

[3] Lecture notes, Subject: Dimensjonering, drift og vedlikehold av strømningsmaskiner, TEP15., Norwegian University of Science and technology

[4] Energiforsyningens fellesorganisasjon, Tilstandskontroll av Vannkraftanlegg, Håndbok Francisturbin, Oslo, 1995

[5] Mårdalen, Helge, J., Vedlikehold av vannkraftgeneratorer, Master thesis, Norwegian University of Science and Technology, Department of Electro- and Computer technique, 1994

[6] Westad, Bjørn, O. / Maintech, Visit at Maintech, 15.05.08

[7] Fjellheim, Ø., Den digitale maskinmester.(Preliminary study for Master thesis, Norwegian University of Science and Technology, Department of Electrical Engineering, 2006).

[8] Jackson, P., Introduction to Expert Systems, Essex: Addison Longmann Limited, Third Edition, 1999

[9] Harmon, P., Sawyer, B., Creating Expert Systems. For business and Industry, USA: John Wiley & Sons, Inc., 1990

[10] Cawsey, A., Artificial intelligence, Edinburgh: Heriot Watt University. [online] http://www.macs.hw.ac.uk/~alison/ai3notes/all.html [Accessed 09.10.07]

[11] Aamodt, A., Plaza, E., Cased-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches, AICom - Artificial Intelligence Communications, IOS Pres, Vol 7: 1, 1994

[12] Aamodt, A., Brede, T., Bø, K., Sørmo, F., TrollCreek, A Knowledge Modelling Editor and Testing Environment for Knowledge-Intensive Case-Based Reasoning, Tutorial version 0.9 TrollCreek, 2004 [online] http://www.idi.ntnu.no/emner/it3704/lectures/papers/TrollCreek-tutorial.pdf [Accessed 03.02.08]

[13] Aamodt, A., Knowledge-Intensive Case-Based Reasoning in CREEK [online] Norwegian University of Science and Technology, Department of Computer and Information Science, [online] http://www.idi.ntnu.no/~agnar/publications/eccbr04-aamodt.pdf,
[Accessed 15.05.2008]

[14] Aamodt, A., Volve AS, Interview at Volve AS, 20.05.08

[15] Aamodt, A., Knowledge-Intensive Case-Based Reasoning and Intelligent Tutoring, Norwegian University of Science and Technology, Department of Computer and Information Science, [online] http://www.idi.ntnu.no/~agnar/publications/sais-05.pdf
[Accessed 07.05.08]

[16] Aamodt, A., Skalle, P., Sørmo, F., How to modell in TrollCreek in the Petroleum Engeneering domain, [online], Instruction of modelling in TrollCreek, TrollCreek Documentation. http://creek.idi.ntnu.no/docs/TrollCreek-modelling.doc
[Accessed 03.02.08]

[17] Source code of Volve Knowledge Tools software, created by Volve AS 2008

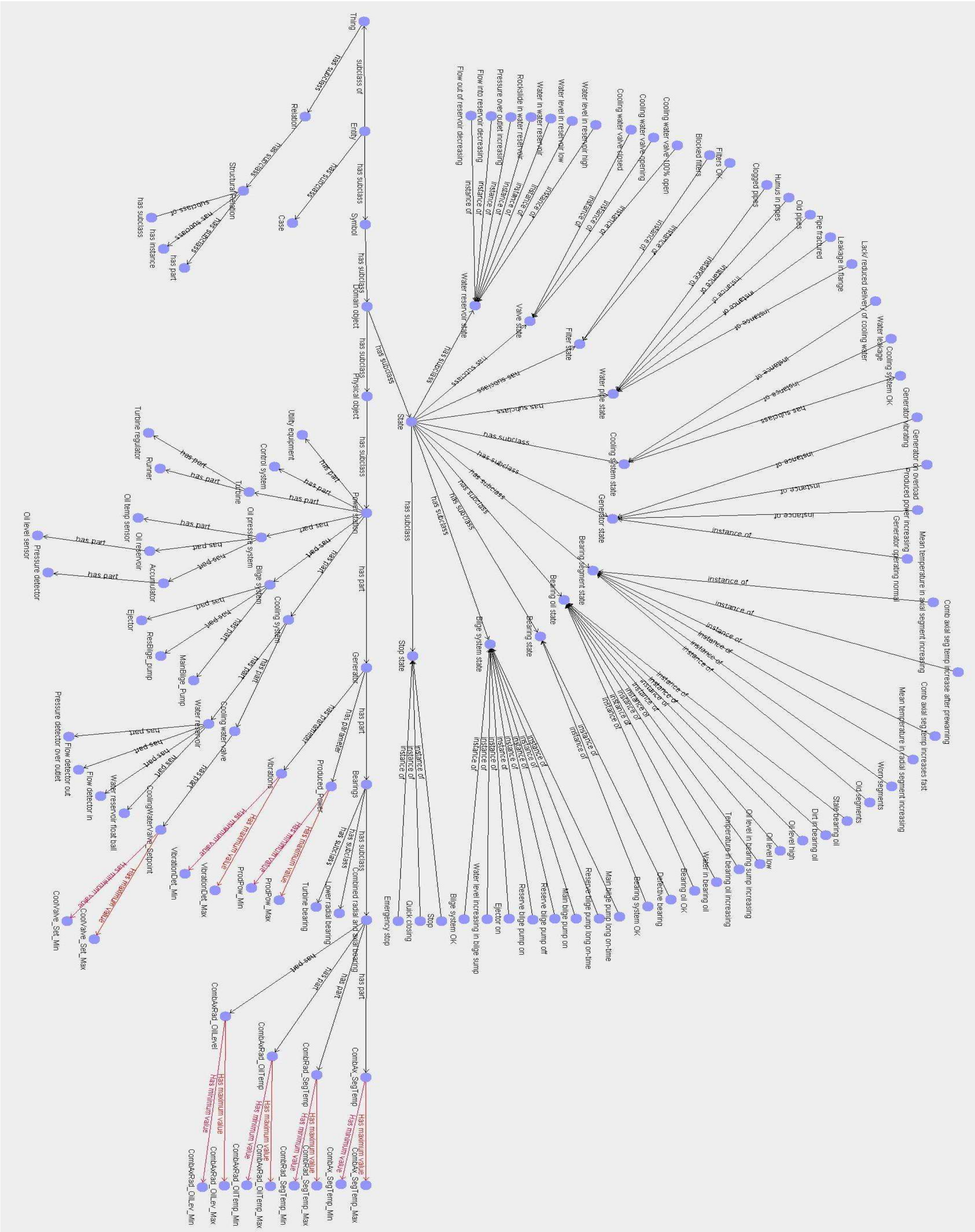
[18] Brede, T., Volve AS, Visit at Volve AS, 19.05.08

[19] Power point presentation from maintenance seminar 28. Mars in Maintech http://www.maintech.no/maintech/entrypage.aspx?t=2008&containerid=10121&parentid=10041&entrypage=true&guid=1&lnodeid=8&pageid=5001
[Accessed 15.05.08]

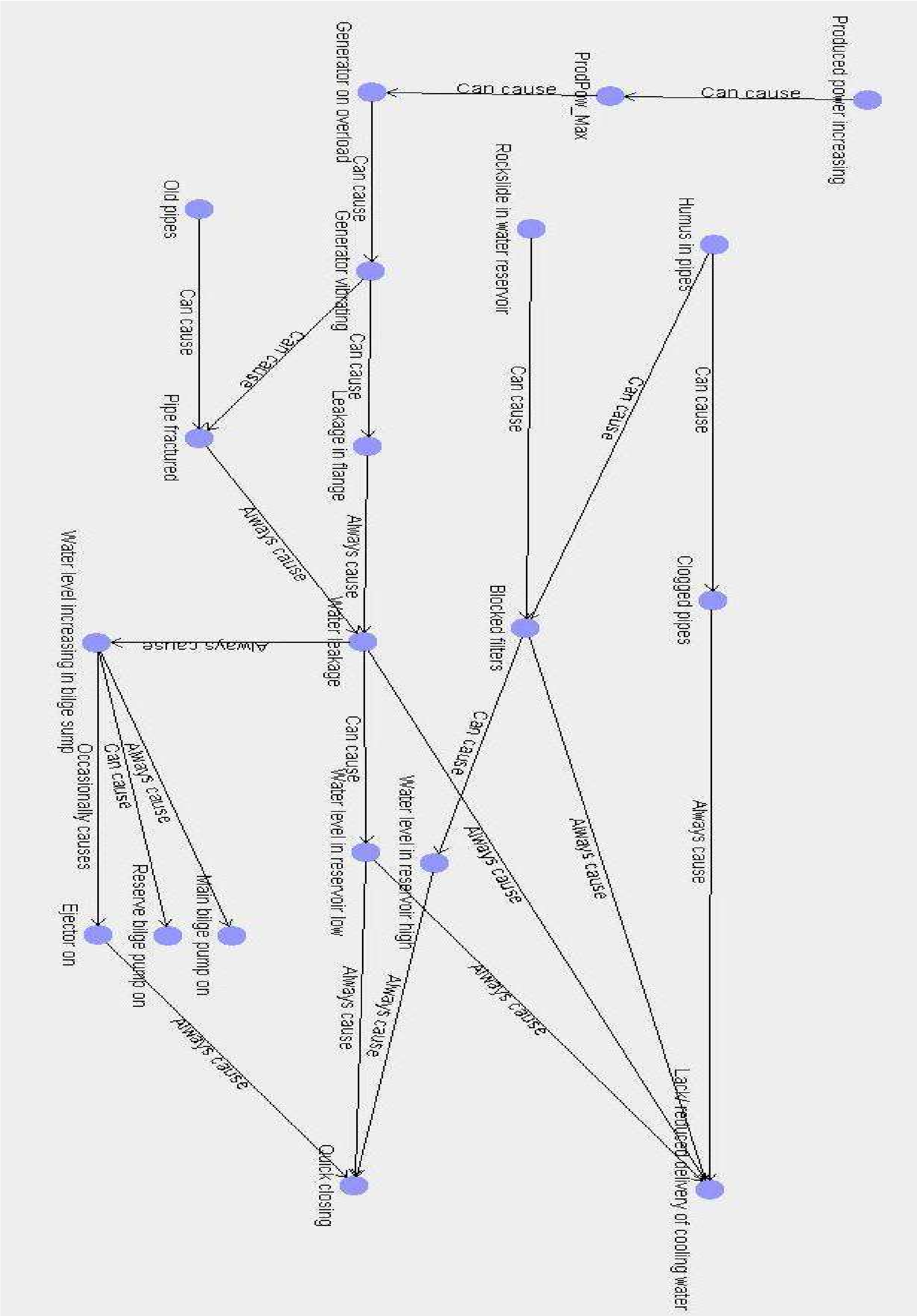
11. APPENDIX

11.1. Condition monitoring of bearings in Volve Knowledge Tools, the whole program

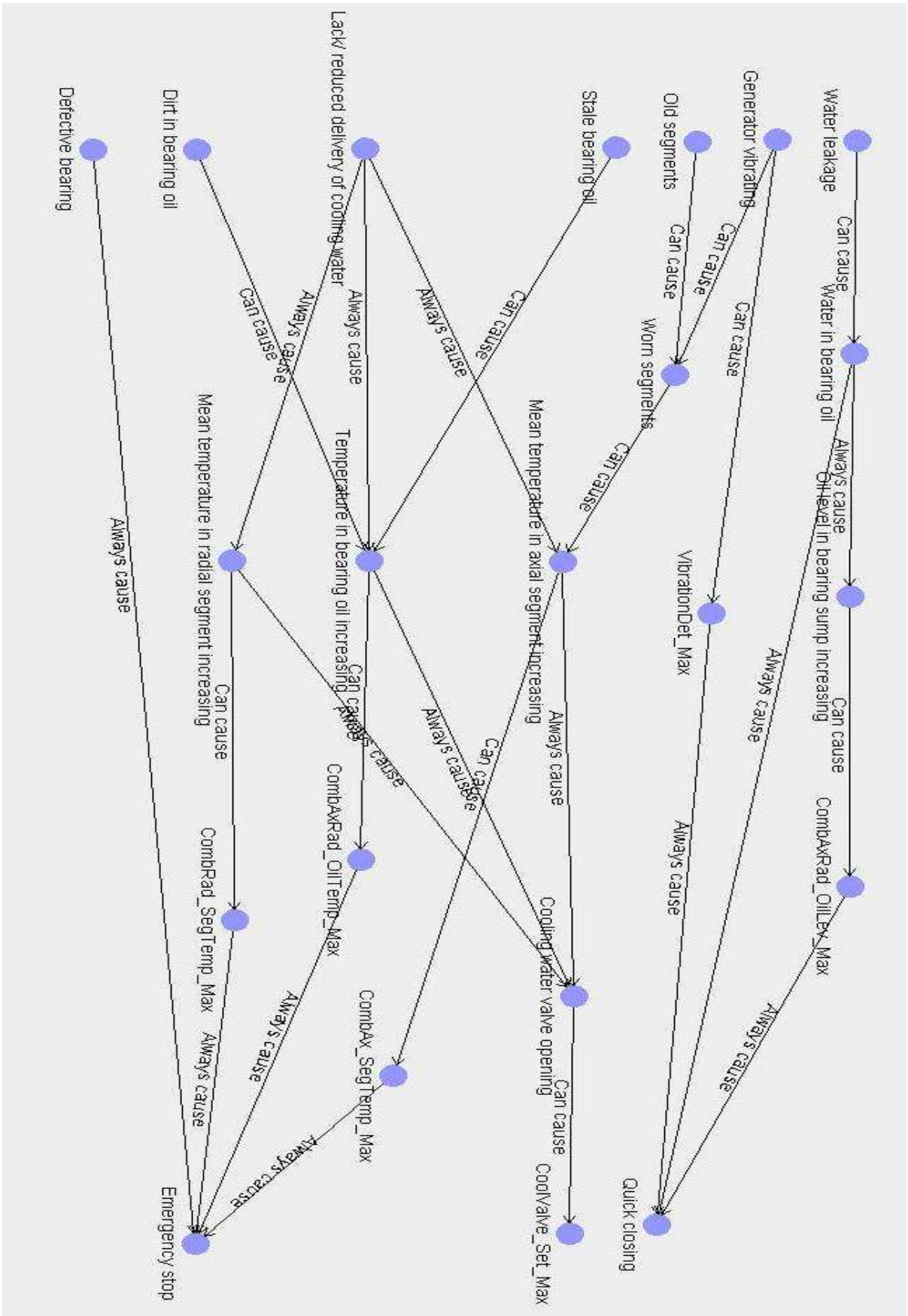
11.1.1. Ontology model



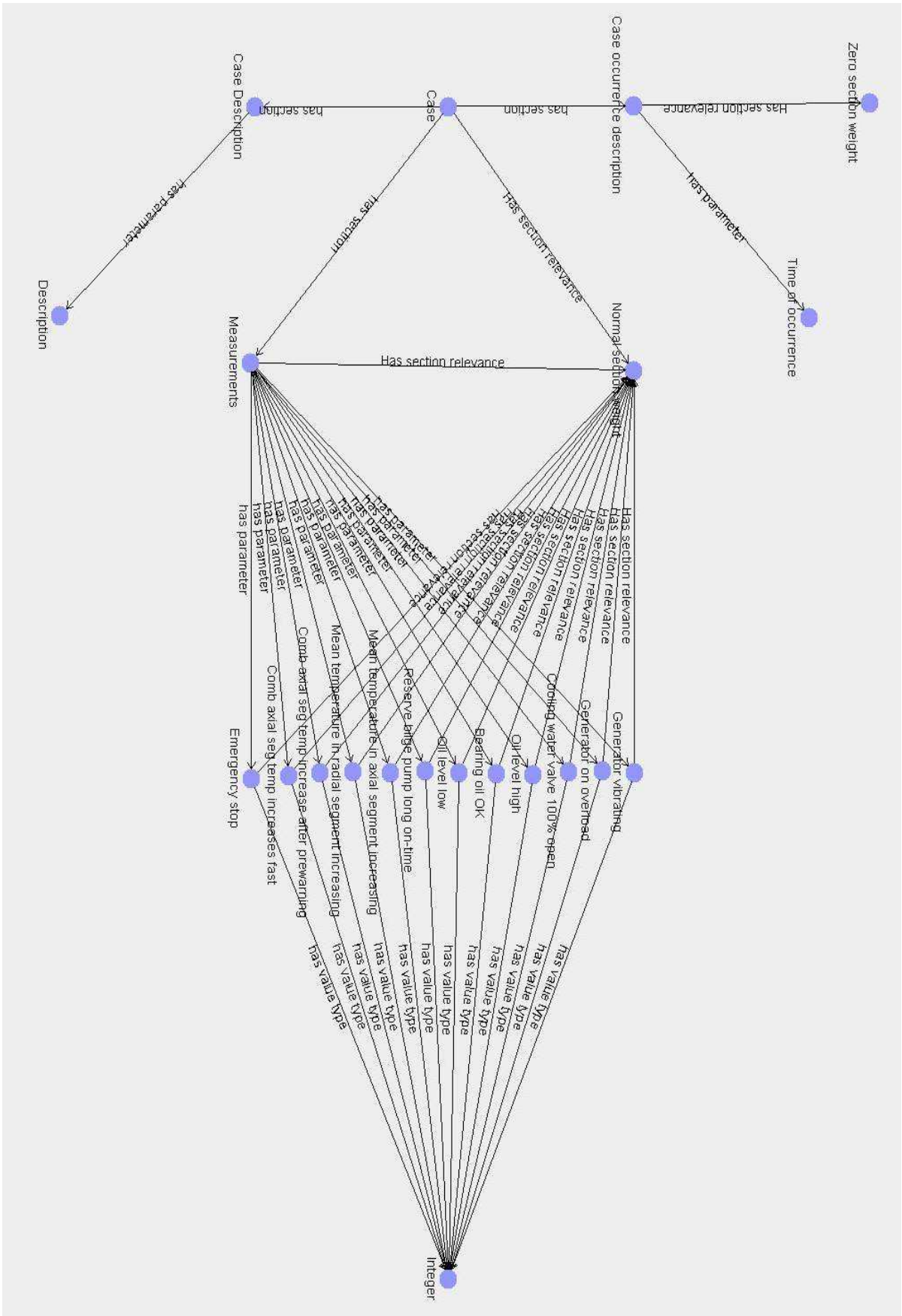
11.1.2. Causal model, cooling system



11.1.3. Causal model, bearing system



11.1.4. Case model



11.2. Predictor script

```
var outputDirectory = "D:\\Skule\\4. Semester Eletric Power Engineering\\Master
thesis\\Work\\Volve Knowledge Tools\\Sensordata";
var inputDirectory = "D:\\Skule\\4. Semester Eletric Power Engineering\\Master
thesis\\Work\\Volve Knowledge Tools\\Sensordata2";

var d = new java.util.Date();
var sdf = new java.text.SimpleDateFormat("dd-MM-yyyy_HH-mm-ss");
var nameAdd = sdf.format(d);

var knowledgeModel = "D:\\Skule\\4. Semester Eletric Power Engineering\\Master
thesis\\Work\\Volve Knowledge Tools/HydroPowerPlant_with_cases.km";
var knowledgeModelSaveName = "D:\\Skule\\4. Semester Eletric Power Engineering\\Master
thesis\\Work\\Volve Knowledge Tools/HydroPowerPlant_with_cases.km";

var sim = api.getTracker();

//captureCases();

runInPredictor();

function runInPredictor() {
        var cbrAg = api.addAgent("CBRAgentImpl", "CBR-agent", api.getTracker(), new
        java.lang.Boolean("true"));
        cbrAg.setComparisonController(new
        Packages.volve.creek.cbr.DefaultComparisonController());
        api.loadDrillLog("D:\\Skule\\4. Semester Eletric Power Engineering\\Master
        thesis\\Work\\Volve Knowledge Tools\\Sensordata/Case#1.csv");
        api.loadKnowledgeModel(knowledgeModelSaveName);
        setupAgents(cbrAg);
        var cbrReporter = api.addAgent("CaseMatchingResultLogger", "CBRReporter",
        cbrAg, 0.1, 3, api.getTracker());
}

function captureCases() {

        var captureTimes = java.lang.reflect.Array.newInstance(java.lang.String, 1);

        // Case 3
                captureTimes[0] = "01.01.2008 16:57:02";
                var cbrAg = api.addAgent("CaptureAgent", "Capture agent", "Time", "Case#",
                captureTimes, sim);
                api.loadDrillLog("D:\\Skule\\4. Semester Eletric Power Engineering\\Master
                thesis\\Work\\Volve Knowledge Tools\\Sensordata/Case#3.csv");
                api.loadKnowledgeModel(knowledgeModelSaveName);
                setupAgents(cbrAg);
                runSimulation();
                sim.getKnowledgeModel().saveAs(knowledgeModelSaveName);
        api.clear();
```

```
	// Case 1
	captureTimes[0] = "01.01.2008 16:26:57"; // The two added seconds are a hack...
	var cbrAg = api.addAgent("CaptureAgent", "Capture agent", "Time", "Case#",
	captureTimes, sim);

	api.loadDrillLog("D:\\Skule\\4. Semester Eletric Power Engineering\\Master
	thesis\\Work\\Volve Knowledge Tools\\Sensordata/Case#1.csv");
	api.loadKnowledgeModel(knowledgeModel);
	setupAgents(cbrAg);
	runSimulation();
	sim.getKnowledgeModel().saveAs(knowledgeModelSaveName);
	api.clear();

	// Case 2
	captureTimes[0] = "01.01.2008 16:33:37";
	var cbrAg = api.addAgent("CaptureAgent", "Capture agent", "Time", "Case#",
	captureTimes, sim);

	api.loadDrillLog("D:\\Skule\\4. Semester Eletric Power Engineering\\Master
	thesis\\Work\\Volve Knowledge Tools\\Sensordata/Case#2.csv");
	api.loadKnowledgeModel(knowledgeModelSaveName);
	setupAgents(cbrAg);
	runSimulation();
	sim.getKnowledgeModel().saveAs(knowledgeModelSaveName);
	api.clear();
}

function runSimulation() {
	api.runSimulator(0);
	api.waitForEnd();
	api.println("Simulation finished");

	//api.exportWITSMLDrillLog(outputDirectory +logName +nameAdd +".xml");
	//sim.exportEvents(outputDirectory +logName +"_events_" +nameAdd +".csv");
	//sim.getKnowledgeModel().saveAs(outputDirectory +nameAdd +"_test.km");
}

function setupAgents(cbrAg) {
	cbrAg.setPropertyFile("D:\\Skule\\4. Semester Eletric Power Engineering\\Master
	thesis\\Work\\Volve Knowledge Tools/KraftCBR.properties");

	var timeAgent = api.addAgent("TimeAgent", "Time", "Case", sim);
	cbrAg.addCaseParameterMapping(timeAgent, "Data value", "Time Of Occurrence",
	"Case occurrence description", true);

	// Set up agents reading data from the data log
	var combAxRadOilLevAgent = api.addAgent("CombAxRadOilLevAgent", "Oil level
	high", "Oil level low", sim);
	cbrAg.addCaseParameterMapping(combAxRadOilLevAgent, "Oil level high", "Oil
	level high", "Measurements", true);
```

```
        cbrAg.addCaseParameterMapping(combAxRadOilLevAgent, "Bearing oil OK",
        "Bearing oil OK", "Measurements", true);
        cbrAg.addCaseParameterMapping(combAxRadOilLevAgent, "Oil level low", "Oil
        level low", "Measurements", true);

        var combAxSegTempAgent = api.addAgent("CombAxSegTempAgent1", "Axial segm
        prewarning", "Temp increase abnormal fast, defective bearing", "Emergency stop!",
        sim);
        cbrAg.addCaseParameterMapping(combAxSegTempAgent, "Comb axial seg temp
        increase after prewarning", "Comb axial seg temp increase after prewarning",
        "Measurements", true);
        cbrAg.addCaseParameterMapping(combAxSegTempAgent, "Emergency stop",
        "Emergency stop", "Measurements", true);

        var combAxSegTempAgent2 = api.addAgent("CombAxSegTempAgent2",
        "Temperature increase rapidly", sim);
        cbrAg.addCaseParameterMapping(combAxSegTempAgent2, "Comb axial seg temp
        increases fast", "Comb axial seg temp increases fast", "Measurements", true);

        var combAxSegTempAgent3 = api.addAgent("CombAxSegTempAgent3",
        "Prewarning: \"Temperature in axial segment rising\"", sim);
        cbrAg.addCaseParameterMapping(combAxSegTempAgent3, "Mean temperature in
        axial segment increasing", "Mean temperature in axial segment increasing",
        "Measurements", true);

        var combRadSegTempAgent = api.addAgent("CombRadSegTempAgent",
        "Prewarning: \"Temperature in radial segement rising\"", sim);
        cbrAg.addCaseParameterMapping(combRadSegTempAgent, "Mean temperature in
        radial segment increasing", "Mean temperature in radial segment increasing",
        "Measurements", true);

        var producedPowerAgent = api.addAgent("ProducedPowerAgent", "Generator on
        overload", sim);
        cbrAg.addCaseParameterMapping(producedPowerAgent, "Generator on overload",
        "Generator on overload", "Measurements", true);

        var resBilgePumpTimerAgent = api.addAgent("ResBilgePumpTimerAgent", sim);
        cbrAg.addCaseParameterMapping(resBilgePumpTimerAgent, "Reserve bilge pump
        long on-time", "Reserve bilge pump long on-time", "Measurements", true);

        var valveSetpointAgent = api.addAgent("ValveSetpointAgent", "Valve 100% open",
        sim);
        cbrAg.addCaseParameterMapping(valveSetpointAgent, "Cooling water valve 100%
        open", "Cooling water valve 100% open", "Measurements", true);

        var vibrationAgent = api.addAgent("VibrationAgent", "Vibrations prewarning", sim);
        cbrAg.addCaseParameterMapping(vibrationAgent, "Generator vibrating", "Generator
        vibrating", "Measurements", true);
}
```

11.3. Cases and agents

11.3.1. Case #1

```
%*********************************************************************
%*                                                                   *
%*                            CASE #1                                *
%*                                                                   *
%*                                                                   *
%*    SIMULATION TIME         :     01.01.2008  16:00:00 -           *
%*                                  01.01.2008  16:27:20             *
%*                                                                   *
%*    FAULT OCCUR             :     01.01.2008  16:04:00             *
%*    FAULT DETECTED          :     01.01.2008  16:26:55             *
%*                                                                   *
%*                                                                   *
%*                                                                   *
%*  Sequence      Happening                      Time        Value    *
%*     1.         Vibration, prewarning       16:13:55      6,0 um    *
%*     2.         Valve setpoint, message     16:18:35      100 %     *
%*     3.         CombAx_SegTemp, prewarning  16:25:10       70 Deg   *
%*                                                                   *
%*                                                                   *
%*********************************************************************


%***********************      AGENTS      ******************************
%*                                                                   *
%*                Vibration               agent                      *
%*                Valve setpoint          agent                      *
%*                Produced Power          agent                      *
%*                CombAxRad_OilLev        agent                      *
%*    Comparator  CombAx_SegTemp          agent1                     *
%*                                                                   *
%*********************************************************************


%***********************      PROGRAM     ******************************


      dim double  Vibration agent                                = 0;
      dim bool    Generator vibrating                            = 0;

      dim int     Valve setpoint agent                           = 0;
      dim bool    Cooling water valve 100% open                  = 0;

      dim int     Produced_Power agent                           = 0;
      dim bool    Generator on overload                          = 0;

      dim int     CombAxRad_OilLev agent                         = 0;
      dim bool    Oil level high                                 = 0;
      dim bool    Bearing oil OK                                 = 1;
      dim bool    Oil level low                                  = 0;

      dim int     CombAx_SegTemp agent1                          = 1;
      dim bool    Comb axial seg temp increase after prewarning  = 0;

      dim bool    Emergency stop                                 = 0;
      dim bool    Quick closing                                  = 0;
```

```
main()
{

%Vibration agent

      Vibration agent =  Instant value;

      If Vibration agent > 5 then
            Message: "Vibrations prewarning";
            Generator vibrating = 1;
      Else
            Generator vibrating = 0;
      End if

%Valve setpoint agent

      Valve setpoint agent = Instant value

      If Valve setpoint agent = 100%
            Message: "Valve 100% open";
            Cooling water valve 100% open = 1;
      Else
            Cooling water valve 100% open = 0;
      End if

%Produced_Power agent

      Prodced_Power agent = Instant value,

      If Produced_Power > 55
            Message: "Generator on overload";
            Generator on overload = 1;
      Else
            Generator on overload = 0;
      End if

%CombAxRad_OilLev agent

      CombAxRad_OilLev agent = Integrate 30 measurments,
                        devide by 30,

      If CombAxRad_OilLev > 255
            Message: "Oil level high";
            Oil level high    = 1;
            Bearing oil OK    = 0;
            Oil level low     = 0;
      Elseif CombAxRad_OilLev < 245
            Message: "Oil level low"
            Oil level high    = 0;
            Bearing oil OK    = 0;
            Oil level low     = 1;
      Else
            Oil level high    = 0;
            Bearing oil OK    = 1;
            Oil level low     = 0;
      End if

%CombAx_SegTemp agent1

      CombAx_SegTemp agent1 = 70 - Instant value
```

```
      If CombAx_SegTemp agent1 =  0
            Message: "Axial segment temp prewarning"
      ElseIf CombAx_SegTemp agent1 = -3
            Message: "Temp increase abnormal fast, defective bearing"
            Comb axial seg temp increase after prewarning = 1;
            Emergency stop = 0;
      ElseIf CombAx_SegTemp agent1 < -9
            Message: "Emergency stop!";
            Emergency stop = 1;
      Else
      End if

%Case #1

      If (Generator vibrating = 1;
          Cooling water valve 100% open = 1;
          Generator on overload = 0;
          Bearing oil OK = 1;
          Comb axial seg temp increase after prewarning = 1)
            Message: "Defective bearing";
            Quick closing = 1;
      End if
}
```

11.3.2. Case #2

```
%*********************************************************************
%*                                                                   *
%*                            CASE #2                                *
%*                                                                   *
%*                                                                   *
%*    SIMULATION TIME         :     01.01.2008  16:30:05 -           *
%*                                  01.01.2008  16:34:00             *
%*                                                                   *
%*    FAULT OCCUR             :     01.01.2008  16:31:10             *
%*    FAULT DETECTED          :     01.01.2008  16:33:35             *
%*                                                                   *
%*                                                                   *
%*                                                                   *
%*  Sequence      Happening                      Time        Value    *
%*     1.         CombAx_SegTemp, rapid incr  16:13:55      1 deg/10 s*
%*     2.         Vibrations, incr            16:18:35      0,1 um/5 s*
%*     3.         Valve setpoint, opens       16:25:10      1 %   /5 s*
%*                                                                   *
%*                                                                   *
%*********************************************************************


%************************      AGENTS      ******************************
%*                                                                   *
%*                Produced_Power          agent                      *
%*                CombAxRad_OilLev        agent                      *
%*    Derivation  CombAx_SegTemp          agent2                     *
%*                                                                   *
%*********************************************************************


%************************      PROGRAM     ******************************


      dim int     Produced_Power agent                              = 0;
      dim bool    Generator on overload                             = 0;

      dim int     CombAxRad_OilLev agent                            = 0;
      dim bool    Oil level high                                    = 0;
      dim bool    Bearing oil OK                                    = 1;
      dim bool    Oil level low                                     = 0;

      dim int     CombAx_SegTemp agent2                             = 0;
      dim bool    Comb axial seg temp increases fast                = 0;

      dim bool    Emergency stop                                    = 0;
      dim bool    Quick closing                                     = 0;
```

```
main()
{

%Produced_Power agent

      Prodced_Power agent = Instant value,

      If Produced_Power > 55
            Message: "Generator on overload";
            Generator on overload = 1;
      Else
            Generator on overload = 0;
      End if

%CombAxRad_OilLev agent

      CombAxRad_OilLev agent = Integrate 30 measurments,
                          devide by 30,

      If CombAxRad_OilLev > 255
            Message: "Oil level high";
            Oil level high    = 1;
            Bearing oil OK    = 0;
            Oil level low     = 0;
      Elseif CombAxRad_OilLev < 245
            Message: "Oil level low"
            Oil level high    = 0;
            Bearing oil OK    = 0;
            Oil level low     = 1;
      Else
            Oil level high    = 0;
            Bearing oil OK    = 1;
            Oil level low     = 0;
      End if

%CombAx_SegTemp agent2

      CombAx_SegTemp agent2 = Derivation 30 measurments,
                         Sum up measurments

      If CombAx_SegTemp agent2 >= 15
            Message: "Temperature increase rapidly";
            Comb axial seg temp increases fast = 1;
      Else
            Comb axial seg temp increases fast = 0;
      End if

%Case #2

      If (Generator on overload = 0;
          Bearing oil OK = 1;
          Comb axial seg temp increases fast = 1)
            Message: "Temperature increase rapidly, defective bearing";
            Quick closing = 1;
      End if
}
```

11.3.3. Case #3

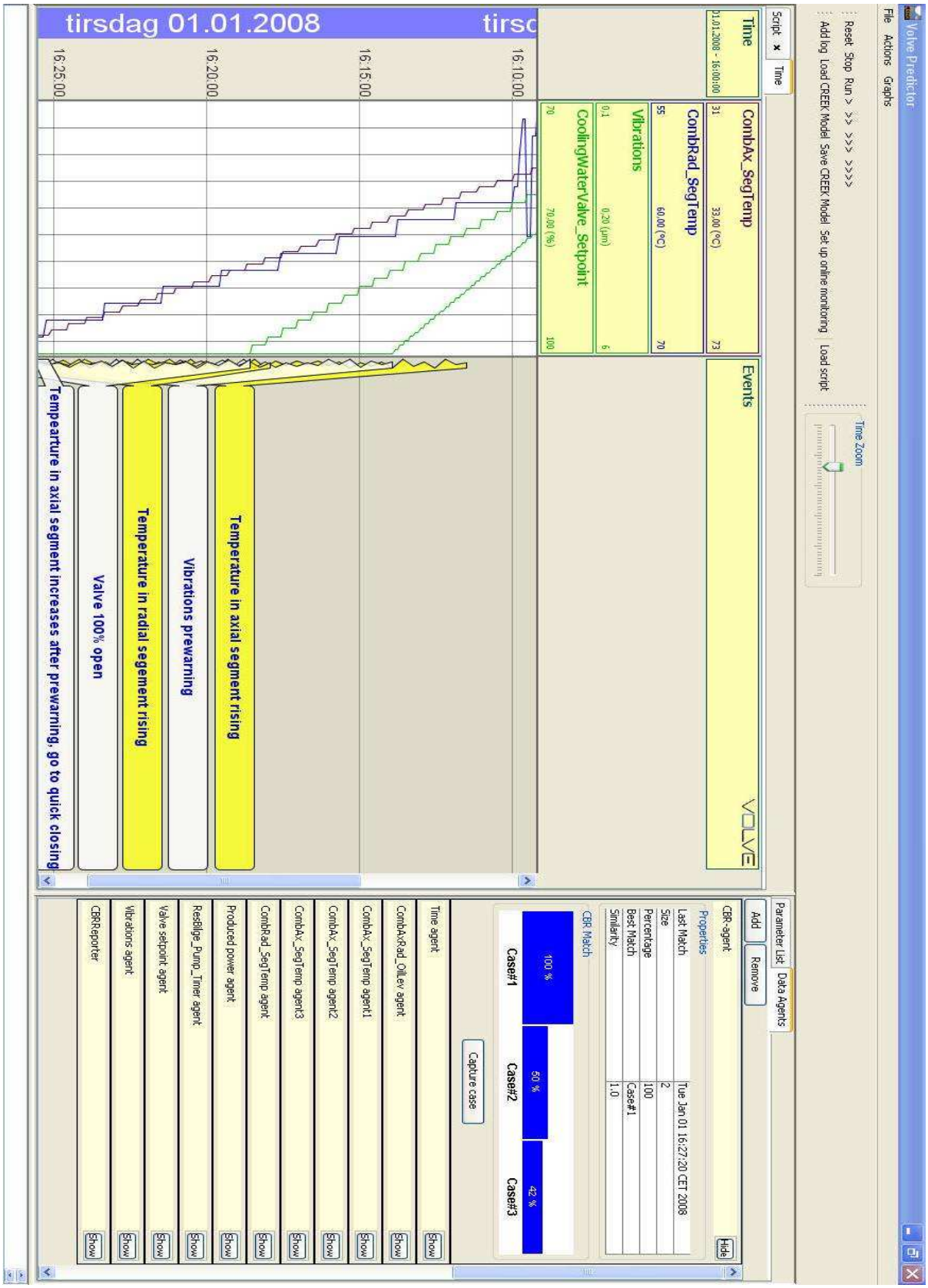
```
%*********************************************************************
%*                                                                   *
%*                            CASE #3                                *
%*                                                                   *
%*                                                                   *
%*    SIMULATION TIME         :     01.01.2008  16:36:45 -           *
%*                                  01.01.2008  16:57:10             *
%*                                                                   *
%*    FAULT OCCUR             :     01.01.2008  16:40:45             *
%*    FAULT DETECTED          :     01.01.2008  16:57:00             *
%*                                                                   *
%*                                                                   *
%*  Sequence      Happening                       Time        Value     *
%*     1.         ReserveBilge pump, start    16:52:00        1       *
%*     2.         Mean_CombAxRad_OilLev, inc  16:55:30      256   mm  *
%*     3.         Mean_CombAx_SegTemp, inc    16:56:00       40   Deg *
%*     4.         Mean_CombRad_SegTemp, inc   16:56:40       65   Deg *
%*     5.         ResBilge_PumpOnLong = 1     16:57:00        5   Min *
%*                                                                   *
%*********************************************************************

%************************      AGENTS      ******************************
%*                                                                   *
%*                ResBilge_Pump           agent                      *
%*                Produced_Power          agent                      *
%*                CombAxRad_OilLev        agent                      *
%*    Mean value  CombAx_SegTemp          agent3                     *
%*    Mean value  CombRad_SegTemp         agent                      *
%*                                                                   *
%*********************************************************************


%************************      PROGRAM     ******************************


      dim int     Produced_Power agent                              = 0;
      dim bool    Generator on overload                             = 0;

      dim int     CombAxRad_OilLev agent                            = 0;
      dim bool    Oil level high                                    = 0;
      dim bool    Bearing oil OK                                    = 1;
      dim bool    Oil level low                                     = 0;

      dim int     ResBilge_Pump agent                               = 0;
      dim bool    ResBilge_Pump_Timer                               = 0;
      dim bool    Reserve bilge pump long on-time                   = 0;

      dim int     CombAx_SegTemp agent3                             = 0;
      dim bool    Mean temperature in axial segment increasing      = 0;

      dim int     CombRad_SegTemp agent                             = 0;
      dim bool    Mean temperature in radial segment increasing     = 0;

      dim bool    Emergency stop                                    = 0;
      dim bool    Quick closing                                     = 0;
```

```
main()
{

%Produced_Power agent

      Prodced_Power agent = Instant value,

      If Produced_Power > 55
            Message: "Generator on overload";
            Generator on overload = 1;
      Else
            Generator on overload = 0;
      End if

%CombAxRad_OilLev agent

      CombAxRad_OilLev agent = Integrate 30 measurments,
                        devide by 30,

      If CombAxRad_OilLev > 255
            Message: "Oil level high";
            Oil level high    = 1;
            Bearing oil OK    = 0;
            Oil level low     = 0;
      Elseif CombAxRad_OilLev < 245
            Message: "Oil level low"
            Oil level high    = 0;
            Bearing oil OK    = 0;
            Oil level low     = 1;
      Else
            Oil level high    = 0;
            Bearing oil OK    = 1;
            Oil level low     = 0;
      End if


%ResBilge_Pump agent

      If ResBilge_Pump agent = 1
            ResBilge_Pump_Timer = +1;
      End if
      If ResBilge_Pump_Timer > 59
            Reserve bilge pump long on-time = 1;
            ResBilge_Pump_Timer = 0;
      End if

%CombAx_SegTemp agent3

      CombAx_SegTemp agent3 = Integrate 30 measurments,
                              devide by 30

      If CombAx_SegTemp agent3 > 44
            Prewarning: "Temperature in axial segment rising"
            Mean temperature in axial segment increasing = 1;
      Else
            Mean temperature in axial segment increasing = 0;
      End if

%CombRad_SegTemp agent

      CombRad_SegTemp agent = Integrate 30 measurments,
```

```
                        devide by 30

      If Mean_CombRad_SegTemp > 64
            Prewarning: "Temperature in radail segement rising"
            Mean temperature in radial segment increasing = 1
      Else
            Mean temperature in radial segment increasing = 0
      End if

%Case #3

      If (Reserve bilge pump long on-time = 1;
          Generator on overload = 0;
          Oil level high   = 1;
          Mean temperature in axial segment increasing = 1;
          Mean temperature in radial segment increasing = 1)
            Message: "Oil level high, possible water in oil";
      End if
}
```

11.4. Screenshots from simulations

11.4.1. Predictor Case #1



11.4.2. Case description, Case #1

Case occurrence description

Time of occurrence	01.01.2008 16:26:55

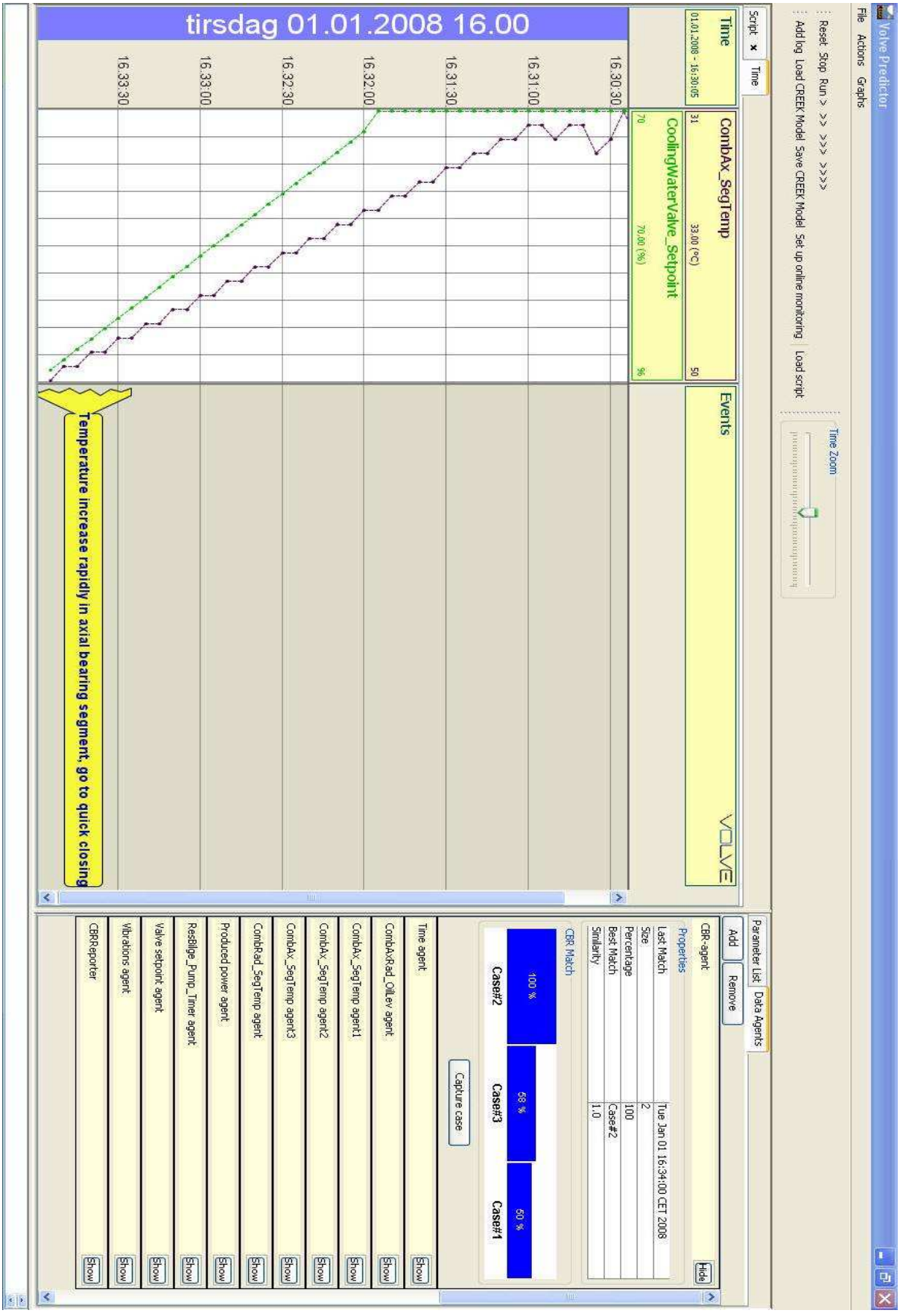
Measurements

Oil level high	0.0
Bearing oil OK	1.0
Oil level low	0.0
Comb axial seg temp increase after prewarning	1.0
Emergency stop	0.0
Comb axial seg temp increases fast	0.0
Mean temperature in axial segment increasing	1.0
Mean temperature in radial segment increasing	1.0
Generator on overload	0.0
Reserve bilge pump long on-time	0.0
Cooling water valve 100% open	1.0
Generator vibrating	1.0

Case Description

Description	Vibration are detected, Tempereature in bearing segments are increasing, Produced power is normal. Aggregate goes to quick closing. Reason: Bearing defect!

11.4.3. Predictor Case #2



11.4.4. Case description Case #2

Case occurrence description

Time of occurrence	01.01.2008 16:33:35

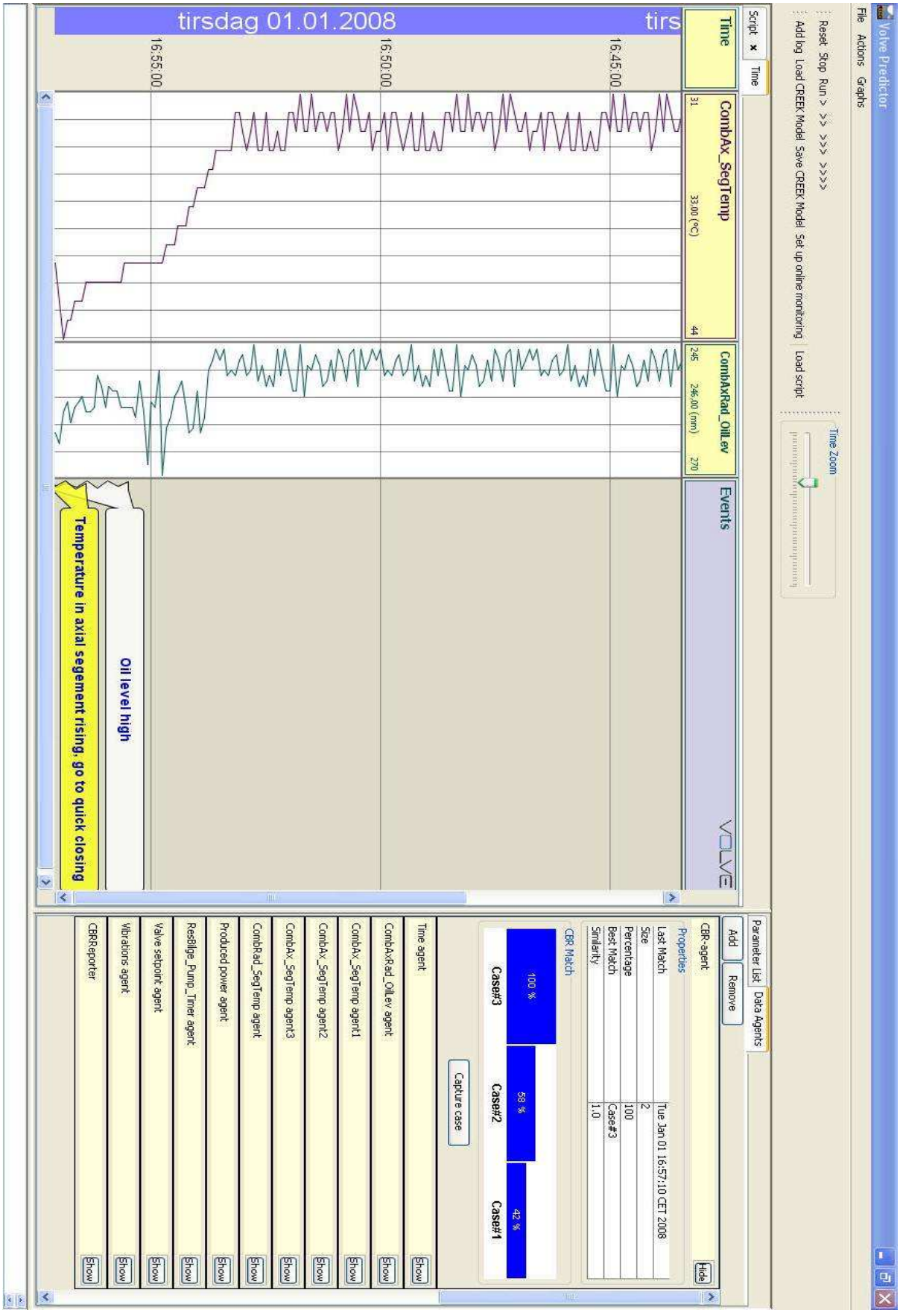
Measurements

Oil level high	0.0
Bearing oil OK	1.0
Oil level low	0.0
Comb axial seg temp increase after prewarning	0.0
Emergency stop	0.0
Comb axial seg temp increases fast	1.0
Mean temperature in axial segment increasing	0.0
Mean temperature in radial segment increasing	0.0
Generator on overload	0.0
Reserve bilge pump long on-time	0.0
Cooling water valve 100% open	0.0
Generator vibrating	0.0

Case Description

Description	Temperature in bearing segment increase rapidly,aggregate runs normally. aggregate goes to quick closing. Reason: Possible defect bearing

11.4.5. Predictor Case #3



11.4.6. Case description Case #3

Case occurrence description

Time of occurrence	01.01.2008 16:57:00

Measurements

Oil level high	1.0
Bearing oil OK	0.0
Oil level low	0.0
Comb axial seg temp increase after prewarning	0.0
Emergency stop	0.0
Comb axial seg temp increases fast	0.0
Mean temperature in axial segment increasing	0.0
Mean temperature in radial segment increasing	1.0
Generator on overload	0.0
Reserve bilge pump long on-time	1.0
Cooling water valve 100% open	0.0
Generator vibrating	0.0

Case Description

Description	Oil level abnormal high, temperature in bearing is increasing, reserve bilge pump on-time is unnaturally long. Aggregate goes to quick closing. Reason: Water in bearing oil