Abstract

A high amount of complex and urgent information needs timely attention in an operational environment. This requires specialized systems. These systems should provide immediate access to accurate and pertinent information when troubleshooting or controlling abnormal situations.

This study is a collaboration between NTNU and Statoil Research Center. It aims at designing and developing a prototype to improve the operator's awareness of alarms, by means of a multi-modal virtual environment. This will be achieved by creating an extension to the virtual model SnøhvitSIM, using a spatial auditory display in addition to visual elements. The auditory display will provide (1) spatial information about new alarms and (2) information about the overall state of the facility. The system also offers (3) beacons to aid navigation within the virtual environment.

To reach these goals, a comprehensive literature study was carried out, investigating similar concepts and various techniques for developing such systems. The development was prioritized in the following order, according to the main objectives: (1) design, (2) functionality and (3) architecture. Within the design-process, the main focus has been on the spatial auditory display.

The development of the prototype proved successful. The feedback on the prototype reflects its value as a showcase for future development, containing new and potentially very effective solutions for tomorrow's alarm management systems.

Preface

This study is carried out as a part of Håvard Sjøvoll's MSc studies at the Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, Department of Computer and Information Science. It is the result of a master thesis in the field of Algorithm constructions and Visualization.

The study is conducted as a collaboration between NTNU and Statoil Research Center, and this report has been developed at Statoil's facilities at Rotvoll.

I would in particular like to thank my mentor at Statoil/NTNU, Dr. Bjørn Sæther, for the opportunity to perform this study and his guidance throughout the development.

I also want to thank my mentor at NTNU, Sigurd Saue, for invaluable guidance and feedback regarding the acoustic topics of this study. This also goes for Thomas Hammer and musician Øyvind Brandtsegg. Thanks to Peder Blekken at Systems In Motion for helping out with the implementation of the SnøhvitSIM AMEX prototype. One should neither forget all Statoil employees which have been available for questions and interviews, like Per Ivar Karstad, Vidar Hepsø, Kjell Bjerkeli, Christian Salbu Aasland, Frank Sinnes, Jan Kjeldstad, Svein Fredrik Fredriksen and the operators at Tjeldbergodden.

A special thanks to Solrun Furuholt Pedersen, for her encouragement and help reading through the study.

Trondheim, 23. Juni 2005

Håvard Sjøvoll

While visual instrumentation has received considerable attention and guidelines, sound has remained analogous to a neglected child in light of what could be accomplished.

(Begault 2000)

Contents

List of Figures

List of Tables

Abbreviations

API Application Programming Interface

ADSR Attack, decay, sustain and release

CCR Central Control Room

CCTV Closed Circuit Television.

CTC Cross Talk Cancellation

DCS Distributed Control System

CLOD Continuous Level of Detail

EAX Environmental Audio Extensions

GUI Graphical User Interface

HTML HyperText Markup Language

ITIL IT Infrastructure Library

HUD Head Up Display

MIDI Musical Instrument Digital Interface

OSC Onshore Support Center

OpenAL Open Audio Library

OpenGL Open Graphics Library

PCM Pulse Code Modulation

RGB Red Green Blue

SA Situation Awareness

SCSSV Surface Control Subsurface Safety Valve

VE Virtual environment

VR	Virtual reality
VRML	Virtual Reality Modeling Language
SnøhvitSIM	Snøhvit Simulator

Part I

Introduction

Chapter 1

Introduction

1.1 Domain

The domain of *virtual reality* (VR) explores ways in which technology can be used to immerse people in *virtual environments* (VE). The research within this area has lately focused on other sensory stimulus than vision, such as audition, sense of touch and olfaction[1]. This is done in order to complement the more traditional unimodal VR-system, which in many cases will be inadequate to fulfill the intended purpose of a virtual environment.

LoBrutto states that *"film sound serves the story, creates a mood, helps define the director's aural vision, and can be the key to bringing the visuals to life"* (LoBrutto 1994). Similarly, sound can be used in VEs, not only to immerse the user in a new environment, but also to mediate information from the system. This could be done in order to enhance the visual representation, or to communicate information which lies beyond our normal field of view. A multi modal VE can also improve and intensify the user's situational awareness and sense of locality within the environment compared to a system using only visual aids.

Using sound to achieve such communication is an area of research denominated *auditory displays*. In its original form, the term auditory display derives from an attempt to use sound to present information that previously has been mediated through visual aids. This is becoming more and more relevant, as computer systems, and technology in general, are increasing in complexity and projected amount of information. This also goes for the systems used in operational environments, such as a control rooms. From the earlier mimic boards where the size of the panels restricted the amount of available information, the development has lead to a whole new situation. Today's control rooms are often characterized by complex control systems and high amount of mediated information. Combining this with a stressful situation and little or no room for failure, enforce a need for specialized

[1]Sense of smell

systems. These systems should provide immediate access to accurate and pertinent information when troubleshooting or controlling abnormal situations.

The combination of multi-modal VEs, more specifically virtual models assisted by 3D sound, and operational environments is the focus and domain of this master thesis.

1.2 Problem

Control room operators, regardless of the type of industry, are facing many challenges when troubleshooting and controlling their systems. Providing the right tool for this task is consequently of great importance.

History is rich on examples where a combination of inappropriate system designs and/or human failure have lead to fatal consequences, sometimes even with human casualties. The accident at the Three Mile Island nuclear power plant is an example of such (Bishton 2003). In March 1979, a valve in the condensate system failed to close properly. This alone should normally have been taken care of by the operators. However, a series of unfortunate events resulted in catastrophe. The system used to control the plant was designed to provide audible and visual indications for more than 1500 alarms. Later investigations revealed that some of these did not work properly. For instance, one light indicated that a valve had been closed when it still was open. Another light indicator was covered by a caution tag attached to another valve controller. In other words, the control panels did not indicate to the operators the true state of affairs in the reactor.
This example illustrates what consequences a faulty control room system can lead to. Another tragic event worth mentioning, is the Åsta train-accident. The 4th of January 2000 two trains collided, causing 19 lives. The casual analysis indicated several weaknesses in the control system and the routines. One interesting point was stated by the investigating committee: *"The Commission consider that audible alarms on the rail traffic control center as an very important and necessary aid to avoid dangerous situations as an result of safety critical errors. [...] The Commission recommends that an audible alarm for safety-critical faults should be installed at all rail traffic control centers as soon as possible."* (NOU 2000).
Even though it is not said anything about its design or mode of operation, the committee emphasizes the importance of audible alarms in control rooms. This is an interesting statement which actually has been taken seriously by another type of industry; the oil industry. In this domain, most control rooms and DCSs use audible alarms for safety-critical faults. Many of these systems also use sound to warn about other unexpected events, such as alarms and messages. However, the original design of these systems, in-

cluding the audio design, may not be scalable with the increasing amount of data that are produced. The reason for this is the usage of more advanced technology and new and more detailed types of instrumentation and sensor technology. Figure 1.1 pinpoints this problem, and emphasize the fact that more data not necessarily equals more information.

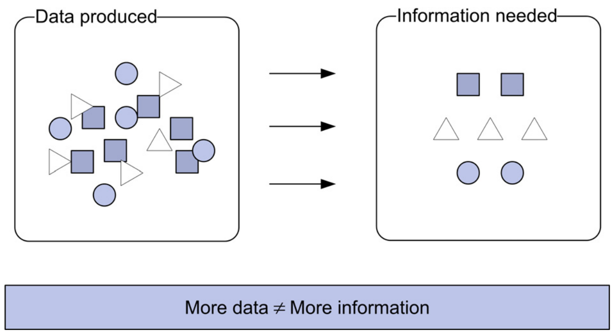


Figure 1.1: The information gap in today's information systems.

More effort is required to design the systems in such a way that they provide the operators with the right amount of pertinent information. If this problem is left unsolved, it is reasonable to believe that the situation sooner or later will come out of hand.

The problem which will be in focus in this master thesis, is how to improve the operator's situation awareness and their overview of the facility's condition. This includes how to better provide information about new alarms. The improvement will be seen in relation with today's DCSs.

1.3 Objectives

Simultaneous with the construction of the Snøhvit facility at Melkøya and in the Barents Sea, Statoil is developing an interactive virtual model called SnøhvitSIM. The model, containing detailed information about the entire land based- and subsea installations, will provide real time updates from the production wells and transport system.
The objectives of this master thesis are to design and develop an extension to SnøhvitSIM to improve the user-interface by means of 3D sound techniques and improved visual elements. This will be done in order to provide a better overview of the facility's condition to the operator.

To summarize, the major objective of this thesis is to develop a multi-modal prototype that fulfills the following hypothesis:
An interactive virtual representation of a facility assisted by 3D

sound will give the operators a better overview of the facility's condition than a traditional control-room. A better overview refers to identifying possible problems regarding type, severity and localization.

1.4 Research methodology

Software engineering is a rather young discipline, seen in a research perspective (Sim 2004). Within this discipline, the scientific methodology are built on very few formal sources. Most approaches are built on other well-established research paradigms. The experimental model of physics is an example of such (Shaw 2003). The intention of such an methodology is to answer a number of questions: What is the problem to be solved? What previous work exist, and what is used as a foundation for the thesis? What is the contribution? Are the results well grounded?
These are all questions which are covered in this master thesis. A hypothesis is defined to lead the author throughout the study. A review of the theoretical basis and similar research, have been done to build the foundation for the further work. The selection of relevant topics is seen in context with the objectives of the master thesis, and it has been emphasized to identify resembling work if available. Using this knowledge, a design has been suggested and implemented. This includes justification of the chosen approach. The implementation has combined the available framework of SnøhvitSIM with the author's own work. Throughout the research and development, informal tests are performed on friends and colleagues for individual feedback. This has been done to refine the prototype to a better design, but is not mentioned specific in the thesis.
A final evaluation has been carried out to examine whether the objectives and goals are achieved. This has been carried out with two different subjects, representing different backgrounds and views.

1.5 Structure of the report

This report has been structured in 5 different parts:

- **Part I, Introduction**
  - *Chapter 1, Introduction* presents the domain, problem and objectives of the master thesis.
- **Part II, Prestudy**
  - *Chapter 2, The task* gives a brief overview of the task.
  - *Chapter 3, Theory and technology* contains theory and descriptions of technology relevant to the objectives.

  - *Chapter 4, Situation today* describes today's situation and its challenges.
  - *Chapter 5, Related work* presents different work and research related to the objectives of this master thesis.

- **Part III, Own contribution**
  - *Chapter 6, Task specification* contains the requirements and main priorities for the system.
  - *Chapter 7, Architecture* briefly describes the suggested architecture.
  - *Chapter 8, Design* presents and discuss the chosen design in accordance with the requirements.
  - *Chapter 9, Implementation* documents the implementation of the prototype.
  - *Chapter 10, Evaluation* gives an analysis of the prototype, including informal feedback from test subjects

- **Part IV, Conclusion**
  - *Chapter 11, Conclusion and further work* concludes the thesis and discuss future work.

The reader should be familiar with common vocabulary and basic expression used in the areas of acoustics and 3D modeling.

Part II

Prestudy

Chapter 2

The task

Prior to the prestudy, a brief overview of the task will be given. This overview supplements the hypothesis defined in section 1.3, and gives the necessary foundation to understand the topics covered in the prestudy.

2.1 Overview

Given the objectives of this master thesis, it is the overall goal to develop an alternative to the conventional control-room design planned at Snøhvit. This refers to the the control room operator's point of view, and the alarm management system in specific. The alarm management system takes care of all incoming alarms and provides support for monitoring and managing the facility. An example of such a system and the operator's desk is depicted in figure 2.1. The intention of the new design suggested in this master thesis is to replace parts of the traditional routines and systems. The new alarm management system will use a virtual model of the facility. Combined with 3D sound, this system will provide a new way of localizing alarms and problems, as well as giving information about the overall status of the system. Hence, the operator's desk will possibly end up looking like figure 2.2, with a virtual view and a surround sound system.

It is worth mentioning that the scope of this master thesis is limited to suggesting a new system for localizing alarms and problems, and not to replace the entire alarm management system.

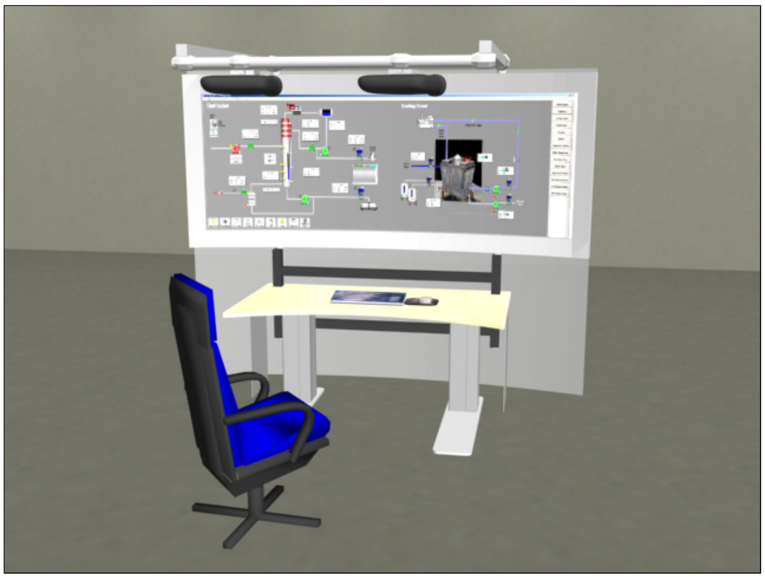


Figure 2.1: Traditional operator's desk with alarm management system.

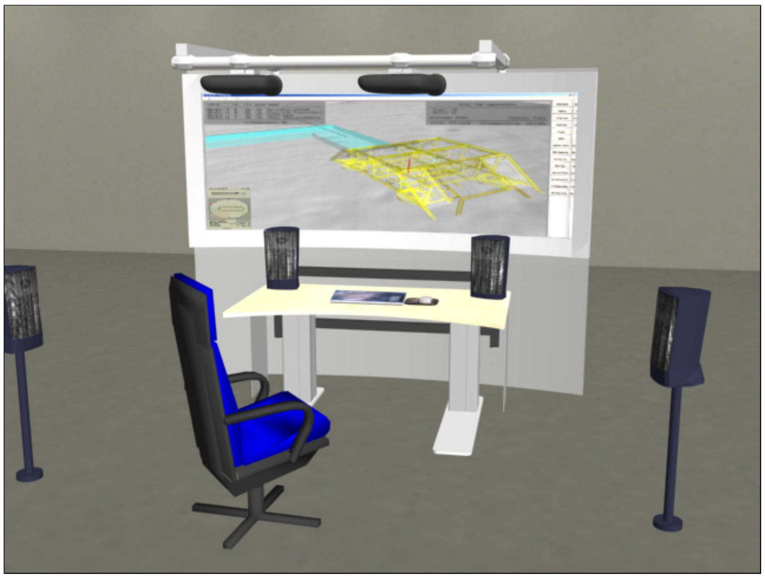


Figure 2.2: Possible new design of operator's alarm management system using virtual models and 3D sound.

Chapter 3

Theory and technology

This chapter studies a selection of different theories and technologies within the fields of acoustics, auditory displays and virtual reality. The selection is based on the guidelines given in the objectives of this master thesis.
It should be noted that parts of this chapter is based on earlier studies by the author (Sjøvoll 2004).

3.1 3D Sound

The scientific basis of 3D sound can be categorized in 3 parts: *physical acoustics*, *auditory neurophysiology* and *psychoacoustics*. While physical acoustics describes the sound waves and how they interact with the environment, psychoacoustics studies the relationship between acoustic waves at the eardrums and the perception of the listener. This relationship is important to understand when trying to immerse the user in a virtual reality environment.
After describing the scientific basis, the most common modeling-principles will be described followed by a description of 3D sound properties which are used when implementing 3D sound applications.

3.1.1 Physical acoustic perspective

Sounds are normally created by objects oscillating in air, for instance a loudspeaker membrane. These oscillations will in a normal environment create sound waves which will propagate in all directions from the source, as depicted in figure 3.1.

The waves will encounter objects in the environment, and according to the laws of physics, they will be reflected and diffracted. Combined with both constructive and destructive interference, a rich acoustic field is created (Kendall 1995).

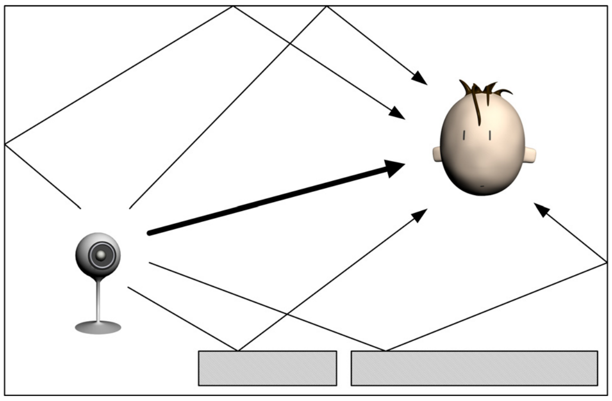


Figure 3.1: Direct and reflected sound waves.

As shown in figure 3.1, sound waves will interact with the listener in different ways. The direct sound is the one following the shortest path from the source to the listener, shown as the thick line in the figure. Other waves will reach the listener after being reflected from a wall or an object, leading to a time delay compared to the direct sound. These indirect sounds provide the listener with information used to analyze the environment and the relative position of the sound event.
Before the sound wave reaches the listeners eardrum, it is also affected by the interaction with the listener's body. Both the shoulders and the pinna work as filters, reinforcing and attenuating different frequencies of the sound wave. This modification depends on characteristic of both the wave and the physical attributes of the listener. These properties are measurable and can be captured and described as a *head-related transfer function*, HRTF (Montan 2002). This is normally done by inserting miniature microphones into the ear canals of a human head or an artificial head. The measurements are done for different *azimuth* and *elevation* angles, giving a detailed description of how the sound reaches the two ears differently.

Before proceeding with more details about the HRTFs, some of the terms which are used must be defined.

When a sound is equidistant from the two ears of a human, the sound arrives at each ear from the same direction. This area, in which the sound sources are equidistant from the two ears, is called the *median plane* (see figure 3.2) (Kendall 1995). Sounds originating from the *horizontal plane* will have very similar HRTFs, but due to slight asymmetries of the human head, they will not be identical. The horizontal plane and the *frontal plane* divides the head at ear level as depicted in figure 3.2.

Whenever a sound is not equidistant from the two ears, it will arrive at the eardrums at different times and from different directions. These directions are described in a coordinate system centered halfway between the ears,

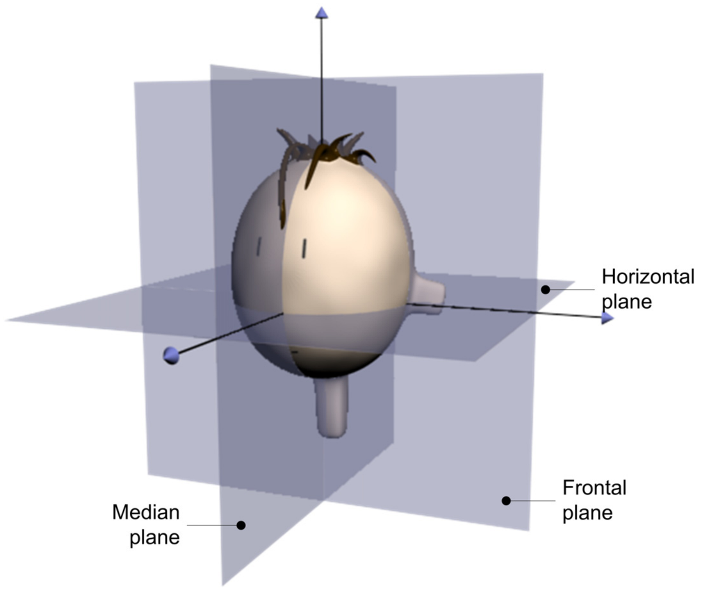


Figure 3.2: Relationship of the median, horizontal and frontal (lateral) planes relative to the listener's head.

pointing forward. The azimuth is defined as the deflection from front center (0 °) in the horizontal plane (see figure 3.3).

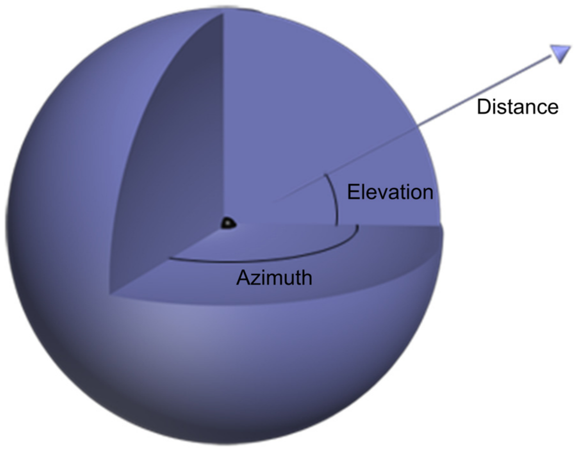


Figure 3.3: Specifying the position of a sound event relative to the head in terms of azimuth, elevation and distance.

Elevation is the horizontal deflection, giving e.g. 90 ° directly over the head. The distance is a scalar describing the actual distance from the center

of the listeners head to the sound source.
It is normal to label the two ears differently, according to which one is closest to the sound source. The closest ear is called the *ipsilateral ear*, while the ear furthest away is called the *contralateral ear*.

As described earlier, HRTFs can be created by doing measurements with miniature microphones in a person's ear canals. These functions can later be used to simulate a sound source situated in space at a given distance, azimuth and elevation. Comparing the signals at both ears shows that the sound arriving at the ipsilateral ear is more intense and arrives earlier that at the contralateral ear. These differences are called the *interaural intensity difference* (IID) and the *interaural time difference* (ITD), respectively (Kendall 1995). Figure 3.4 illustrates these effects.

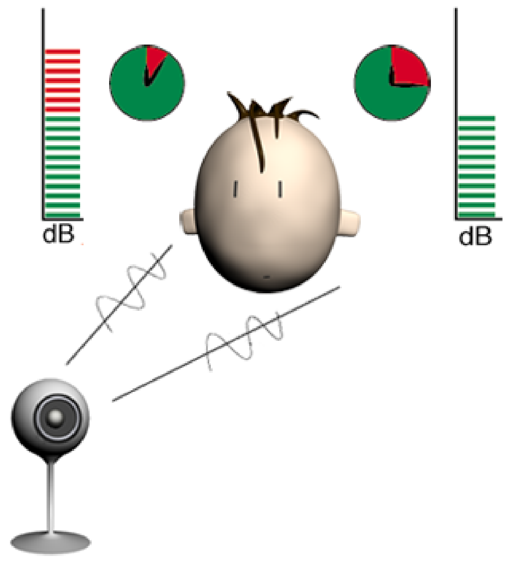


Figure 3.4: Interaural intensity difference and interaural time difference caused by a sound source with unequal distance to the two ears.

However, the HRTFs are not completely without faults. It is not always possible to locate a sound source based on only the HRTFs. This problem often appears when the origin of the sound is in the median plane, either in front of, behind, or above the listener. A normal respond to such a signal is for the listener to move his head to determine the origin of the sound. For this to work in a virtual environment, one must be able to track the listener's head movements.

Figure 3.5 shows a sample HRTF in both time and frequency domain (Albertalli 2003). As can be seen from the diagrams, the right ear is obviously the ipsilateral ear, meaning that the sound source is nearer to the right ear than the left ear. Comparing the two upper diagrams, one can see that the

ITD is approximately 18 samples long.

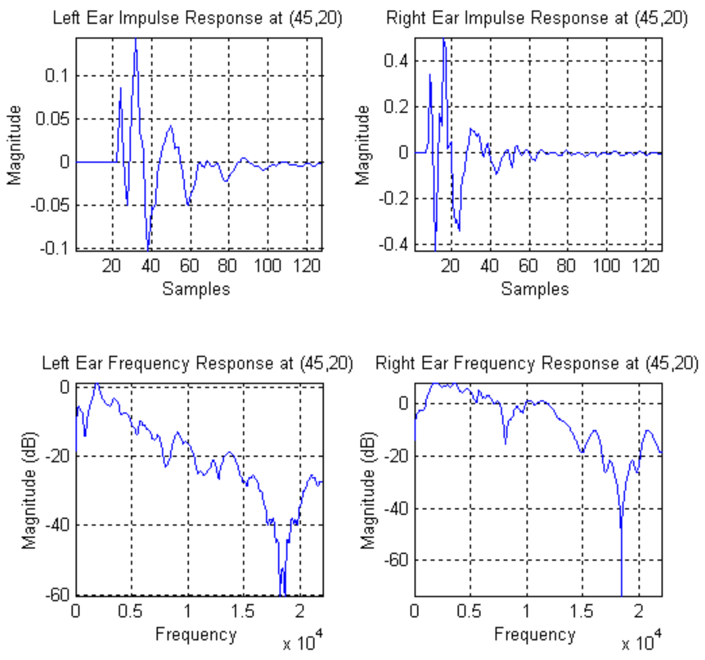


Figure 3.5: Example of HRTF, which is the representation of an impulse response (above) in the frequency domain (below)
Courtesy of Albertalli *et al.*

3.1.2 Psychoacoustic perspective

So far in this chapter, the physical acoustic properties and neurophysiological properties have been discussed. The theory of psychoacoustic, which will be discussed in this chapter, tells how we perceive the sounds around us (Rudi 2000). More specifically, the focus will be on the psychoacoustic mechanisms for spatial interpretation of sound source location. In particular its angle and elevation relative to the listener, distance perception, and the effects of the environmental context will be studied.

Azimuth and elevation perception

The most important cues for localizing a sound source's angular position involve the relative difference of the sound wave at the two ears on the horizontal plane (Begault 2000). This difference is described by the two frequency-dependent cues; interaural intensity difference and interaural time difference. These descriptors have been in focus for several studies related to psychoacoustic theories, and are sometimes referred to as the *duplex theory of sound localization* (Strutt 1907).

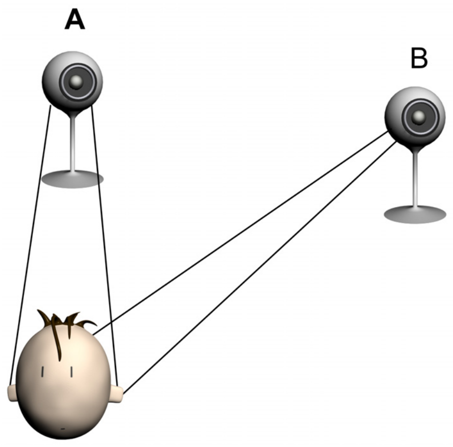


Figure 3.6: Unequal path lengths from different sound sources give different perception of the sound.

As shown in figure 3.6, a sound source "B" placed at approximately 60 ° azimuth will give unequal path lengths to the two ears, and also unequal intensities. Experiments show that the impact the IID and the ITD make on the perceptual judgments vary in different frequency ranges.
Above 1500 Hz the head will act as an obstacle and create an acoustic shadow, causing the localization judgments to be dominated by the intensity difference IID. Below 1500 Hz, longer wavelengths will not be significantly affected by the head but diffract around it and thereby minimize the intensity difference. However, the interaural time difference is still a fact, and will consequently dominate the judgment of localization.

When the ITD outreaches the values predicted by the already mentioned models, another theory become effective: The precedence effect (Wallach, Newman & Rosenzweig 1949). The precedence effect explains the mechanism of the auditory system, allowing localization of sound sources in the presence of reverberation. This means that even though the sound waves arrive from different directions because of reflection, the listener's judgment of the direction is dominated by the sound reaching him along the most direct path. Referring to figure 3.1, the listener will easily localize the sound source correctly and not be confused by the reflected sound waves (thin lines).

Even though the IID and ITD are very useful when trying to localize a sound source along the interaural axis, they cannot separate acoustic events in the relative positions above, below, behind or in front (as depicted in figure 3.7). This ambiguity of location has been called the *cone of confusion*.

As can be deducted from these results, the classic psychoacoustic experiments supporting the duplex theory of localization do not manage to position

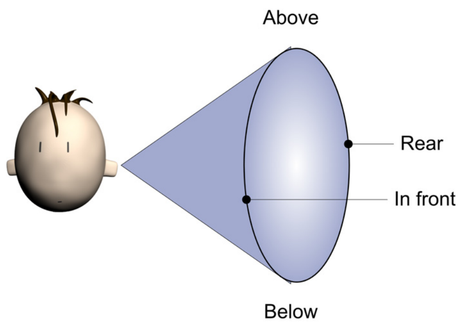


Figure 3.7: Cone of confusion.

all sources precisely enough.

A more modern psychoacoustic field of research is *binaural hearing* by use of HRTFs in localization (Kendall 1995). As mentioned earlier, the shape of the head, the torso and especially the outer ear (pinna) affects the spectral contents of the sound as it propagates from free space to the inner ear. Because of the irregular shape of the upper body and pinna, it is theorized that these filtering effects varies with the location of the sound source, and therefore being spatially dependent. A HRTF will not only contain the spectral filtering effects, but also both ITD and IID cues as described in the duplex theory earlier.
Since each person with given physical characteristics shape the spectrum of a sound based on the sound's location, it implies that the localization will be easier if the sound contain more spectral energy (Morgan 1998). An example of such a spectrally rich sound is noise. A good illustration of this is the mating and warning signals of some species of birds. Whenever they feel endangered, they employ pure tones in order to avoid revealing their own location. These same birds use spectrally rich mating calls so they can be easily located (Morgan 1998).

Normally, the HRTFs used are non-individualized HRTFs. This means that they are measured from a head where both the size and the pinna may be quite different from your own. This may have different outcome, but generally individuals can localize a sound better with their own HRTFs than with those of others. Some individuals however, have superior HRTFs which sometimes can be used to improve other people's localization skills. If the use of non-individualized HRTFs causes a reduction in localization skills, this will first and foremost be shown as a front/back confusion and elevation errors (Gardner 1999).

Based on experiments by Blauert (Blauert 1974), an effect called *localization blur* has been documented. This effect explains the difference between the vertical and horizontal resolution, in which people can locate a sound source. The experiments show that the horizontal dimension has a higher resolution, down to a minimum audible angle of 2° in the front. At the sides and in the back, the angle is increased to 10° and 6°, respectively. In the vertical dimension, the minimum audible angle starts at 9° in the front, and increases to 22° straight above the head.

The effect of localization blur can be minimized by the use of head movements. By nature, humans make small head movements to gain a better foundation when trying to localize an acoustic event. This plays an important role when trying to resolve front/back confusions, especially when the sound sources are in the median plane where other acoustic information provides few interaural differences (Kendall 1995).

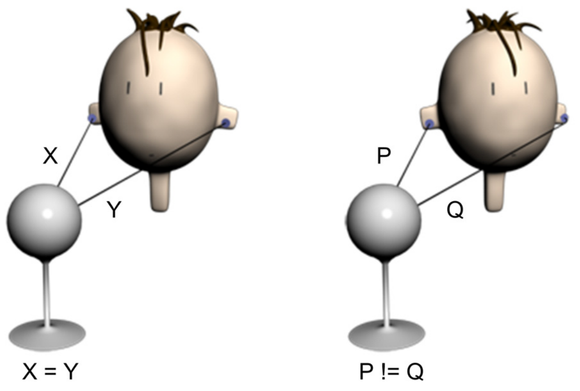


Figure 3.8: A dynamic head turn to the right disambiguates whether a sound source is in front or in back of the listener.

As depicted in figure 3.8, turning the head will disambiguate the possible front/back confusion. This natural response must be considered when choosing system for reproducing 3D sound. While a normal speaker setup will utilize this head movement, the use of headphones will demand headtracking to be able to synthesize the result.

Distance cues and reverberation

There are four principal cues to a sound event's distance: *echoes*, *reverberation*, *overall attenuation* and *high-frequency attenuation* (Morgan 1998). According to the inverse square law a sound event's intensity decreases with $\frac{1}{r^2}$, given the distance r from the center of the sound. For example, at the distance of two meters from the sound source, the intensity is $\frac{1}{2^2} = \frac{1}{4}$ of the sound intensity at one meter (see figure 3.9).

In addition to the attenuation mentioned above, high frequencies are also attenuated by the absorbency of humidity in air. Consequently, low frequencies will travel further than high frequencies in open air.

The listening environment gives the sound certain characteristics which also work as distance cues. These effects are divided in two parts; the early

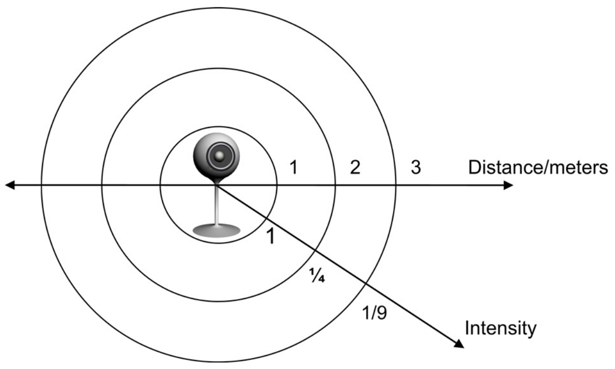


Figure 3.9: Relation between distance and intensity according to the inverse square law.

echoes and the following reverberation. The initial or primary wavefront are known as the echoes and were mentioned in the last section as the precedence effect. It simply says that the ear will localize a sound in the direction of the earliest arriving sound event even if the initial echoes arriving at the other ear are louder. Larger spaces yield larger echoes.

After the initial echoes, a dense fusion of other echoes hits the listener. This fusion is called reverberation, and is the main contributor for revealing information about the auditory environment. Only in anechoic chambers, or in atypical environmental conditions such as within a large, open expanse of snow-covered ground or on a mountain summit, sound sources will be non-reverberant (Begault 2000). As a distance cue, reverberation also plays an important role. The intensity of the primary sound relative to its reverberation is high when the sound source is close to the listener. For more distant sound sources, the ratio between the initial sound and its reverberation is much smaller (Morgan 1998).

3.2 Reproduction of 3D sound

3D sound is conveyed to the listener either through loudspeakers or through headphones. Both methods have their advantages as well as shortcomings which must be coped with. In most cases the most favorable method will depend on the application and setting.
This chapter will examine the underlying techniques for reproducing 3D sound, possible issues regarding the different approaches and the various qualities of loudspeakers and headphones.

3.2.1 Directional filtering

Directional filtering is done by taking a monophonic input signal and filter this to produce a stereo pair of sounds. The filter must use parameters like azimuth angle and elevation to produce the right filtering. After filtering all sounds from different locations, all left and right channels are added to create a stereophonic signal (see figure 3.10).

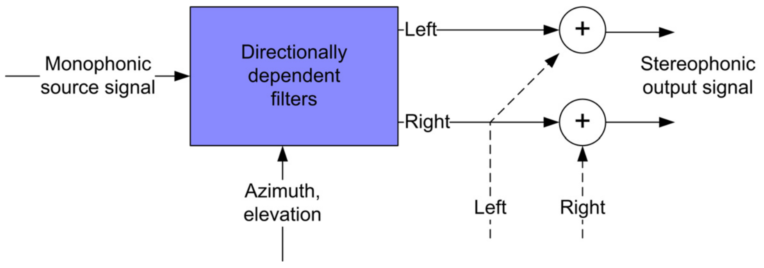


Figure 3.10: Directional filtering.

An example of a filter as depicted in figure 3.5 is the *Finite Impulse Response filters* (FIR), whose coefficients are the head related transfer functions (HRTFs) impulse response (Kendall 1995).

3.2.2 Equalization

Equalization is done to eliminate errors as a result of the reproduction process. One of the potential problems when reproducing sound is the reproduction equipment itself. The characteristics of the equipment, especially the transducers, tend to become superimposed on the output signals. This superimposition should be compensated for. Also the transmission path to the ear drum could be compensated to produce a more correct sound image (Kendall 1995).

3.2.3 Headphone reproduction

Reproducing 3D sound using headphones is probably the most controlled method of representing directional cues, but still some challenges exist. Even though separating left and right sound events is rather easy, the listener often gets the impression that the auditory images only move along the interaural axis inside the head. This gets even more difficult when interpreting sources in the median plane. Because of the absence of time- and intensity-difference cues (ITD and IID), the main directional cue for the median plane is the HRTF. This HRTF may be a non-individual HRTF or a HRTF distorted by for instance a bad equalization. This may be improved either by justifying the equalization of the headphones, or by exaggerate the front/back spectral differences of the HRTFs (Kendall 1995).

When using headphones, another topic which should be given attention is which type to use. As a thumb rule, an *open headphone* is normally to be preferred. In this context open means that it does not affect the radiation impedance as seen from the ear (Kendall 1995).

3.2.4 Loudspeaker reproduction

When using loudspeakers to reproduce 3D sound events, the effect will often be a convincing reproduction of sources situated straight ahead of a person and partly from the sides. It gives however weak reproduction of sources originating from behind (Svensson 2004). The weakness of not supporting high quality 360° representation is probably not the biggest issue when using loudspeakers. A sound wave created at the right speaker intended for the right ear will also reach the left ear, causing crosstalk, which again leads to constructive and destructive interference at the left ear. This is depicted in figure 3.11 (Kendall 1995).

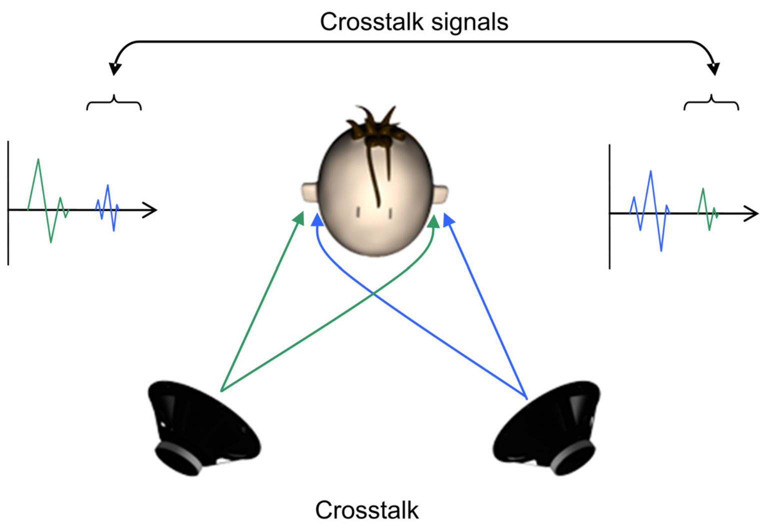


Figure 3.11: Crosstalk signals.

Cross-talk cancellation

Crosstalk cancellation (CTC) is a method to remove crosstalk, which is undesired signals reaching the wrong ear. This cancellation is implemented with a CTC-filter adding inverted signals at both left and right channels. This aims at canceling the crosstalk from the opposite loudspeaker (Svensson 2004).

A CTC-filter is necessary in all binaural reproductions when using loudspeakers. This will create a *sweet-spot*, in which the crosstalk is compensated for and where the listener will get a correct acoustic image.

The loudspeakers will normally be positioned at $\pm 30^\circ$ azimuth from the listener and consequently form an equilateral triangle with the sweet-spot. If the listener moves forward or backwards from the sweet-spot, the distance to each loudspeaker will remain equal, and therefore the crosstalk cancellation will remain intact. However, if the user moves sideways away from the sweet-spot, the cancellation will be less precise and eventually deteriorate. This could be avoided if the listener's position was known by for instance tracking, and used to update the CTC-filter.

3.2.5 Multi-loudspeaker techniques

Another approach to increase the quality of a loudspeaker reproduction system is by using a so-called *multi-loudspeaker technique*. By using more than two loudspeakers, it is possible to increase the size of the sweet-spot. The well-known standard Dolby Digital 5.1 and also Dolby Digital 7.1 use 6 and 8 loudspeakers respectively. However, they do not use any HRTFs to calculate the output. Instead they use conventional panning, which means they play the same sound on all speakers but at different intensity. This implies that this solution will need a lot of loudspeakers to perform satisfying, especially if the sound sources are allowed to be positioned above/below the horizontal plane.

In addition to surround-systems based on panning, some other techniques have been developed. Both Sensaura (*Sensaura website* 2005) and Creative (*Creative website* 2005) have developed reproduction systems using HRTFs and CTC-filters on 4 or more speakers. Such a solution is quite demanding when it comes to computational power since each speaker need a unique HRTF and a separate CTC-filter. The result though will be a rather large sweet-spot with very good localization possibilities compared to e.g. a reproduction system with only 2 loudspeakers.

Wavefield synthesis is a method based on the principle that if one can reproduce the particle velocity over a closed surface, the sound pressure inside the circumscribed volume will be correct. This implies that a large number of loudspeakers must be used to recreate the entire surface (Svensson 2004). Another rather complex technique is *ambisonics*, a method of recording information about a sound field and reproduce it over some form of loudspeaker array to give the impression of hearing a true three dimensional sound image. Even tough ambisonics offers less restrictions than wavefield synthesis when it comes to the number of speakers required, the number will soon grow large if a high quality reproduction is demanded. The fact that the ambisonics encoding (recording) and decoding are completely independent is however a big advantage (Svensson 2004).

3.3 Auditory displays

The term *Auditory display* was first introduced in 1994 by Kramer as an embryonic[1] (Cohen 1994). The name derives from the attempt to use sound to present information that previously have been communicated through visual aids. In the paper Monitoring Background Activities in Auditory Display, Cohen (Cohen 1994) summarizes the following reasons for adopting sound to enlighten the users:

- Audio does not take up screen space.
- Audio easily fades into the background, but users are alerted when it changes.
- People can process audio information while simultaneously engaged in an unrelated task (e.g. listening to music whilst editing a paper).
- The well-known cocktail party effect (the ability to selectively attend to one conversation in the midst of others in a crowded room) allows users to monitor multiple background processes via the audio channel, as long as the sounds attributed to each process can be distinguished.

The research on Auditory displays is mainly divided into two different areas (Vickers 1999): The use of audio in the interface and the use of sound in visualization (see figure 3.12).

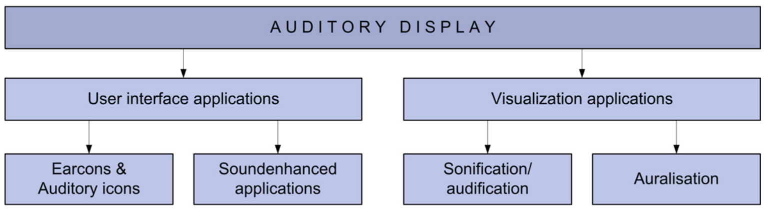

AUDITORY DISPLAY
User interface applications
Visualization applications
Earcons & Auditory icons
Soundenhanced applications
Sonification/ audification
Auralisation

Figure 3.12: Taxonomy of auditory display.

The applications regarding visualization can further be subdivided into two categories called *Sonification/Audification* and *Auralisation*. This concerns how to represent data, and also how to improve the understanding of a software state through auditory displays. The category User interface applications covers the types of auditory displays where the user interface is improved by using acoustic aids, such as the sound of a button being clicked, or a program being closed. Even though a separation is made of the different types of auditory displays, it is clear that some applications can be included in several categories. It is also worth mentioning that some studies use different definitions than those presented here. This taxonomy will however be used to structure the further explanations.

[1]Something that is embryonic is at a very early stage of its development

3.3.1 Sonification and audification

Sonification is the process where certain properties of a dataset or events are mapped to different sounds, preferably related to the semantic of the event or dataset. The range of the areas where this has been applied is vast, and is probably the most popular venue of auditory display research.
Audification is a process quite similar to sonification, but instead of mapping events to different sounds, the data are played directly. Kramer explains audification as *"The direct conversion of data to sounds"*(Kramer 1994). This may sound strange, but could for instance be implemented by scaling seismic data up until their values lie in the audible frequency range.

3.3.2 Auralisation

Auralisation is a term often used as a synonym to sonification, but Vickers (Vickers 1999) chooses to separate these as shown in the taxonomy in figure 3.12. While sonification concerns generic data, auralisation is more about representing programs and algorithms, often directly associated with internal items of a program.
CAITLIN (the Computer Audio Interface to Locate Incorrect Nonsense) by (Vickers 1999) is a system using the auralisation technique which was developed to assist novice Pascal programmers in debugging their code.

3.3.3 Earcons and auditory icons

Earcons is a term introduced in (Blattner, Sumikawa & Greenberg 1989), and is a definition of simple musical structures composed from so-called motifs. These musical structures are mapped to computer objects, operations and interactions (Vickers 1999). When using earcons, two different approaches could be chosen: Hierarchical or compound. The hierarchical method could build an earcon hierarchy from motifs, using the root earcon's motif as a basis for the lower levels. By changing certain properties, like pitch or timbre, the structure of the parent earcon can be preserved even though a new and distinguishable earcon is produced. An example of this could be an earcon associated with the state *warning*, consisting of a motif with a structure of a single pitch (e.g. a middle C). By changing the timbre of the root motif several subcategories of *warning* could be created. For instance could *warning* $\Rightarrow$ *high temperature* could sound like a trumpet, while *warning* $\Rightarrow$ *low temperature* could sound like an organ.
The compound approach associate different earcons to different objects or operations, a use these to build a meaningful representation of the current situation. For example, if *open program* was represented by an organ sound, and a specific program was represented by a three-note tuba sound, the sequence of opening this specific program would be represented by an organ sound followed by a three-note tuba sound.

Since no obvious relation between the earcon and the associated object or operation is required, this relation has to be learned by the listener. An alternative to earcons, where natural, everyday sounds are used, is auditory icons. The term was first suggested by Gaver (Gaver 1986), and represent sounds which imitate the information or action being presented. For example, the auditory icon for deleting a file could be the sound of something being dumped in a trash can. Further information about Gaver's work is found in chapter 5.
Even tough little or no training is necessary to use auditory icons, finding the right sounds can often be difficult. As a result, it is often necessary to use more metaphorical and less analogical sounds, causing the need of memorizing these discrete sounds. If the number of different sounds grows large, one needs some sort of hierarchical structure. Otherwise remembering all the relations may be difficult. This is probably the reason why auditory icons have been left behind, compared to earcons, when it comes to formal research.

3.4 Virtual Reality Modeling Language

The Virtual Reality Modeling Language (VRML) is a file format for describing interactive 3D objects and worlds. VRML is designed to be used on the Internet, intranets, and on local client systems. VRML is also intended to be a universal interchange format for integrated 3D graphics and multimedia (*VRML97, ISO/IEC 14772-1* 1997). Lukka defines VRML with these describing words: *"VRML is intended to be for virtual reality what HTML is for text - a structured, standard, cross-platform format for static or interactive hyperlinked content."* (Lukka 1999). A VRML file contains a scene graph which are processed by a program known as a browser. An example of such a file is depicted in code example 1.

The resulting 3D graphics produced by example 1 is illustrated in figure 3.13.

3.5 Open Audio Library

Open Audio Library (OpenAL) is a cross-platform library for producing multi-channel audio output. This allows simulations of 3D arrangements with multiple sound sources and a listener. The library is developed by Creative Technologies Ltd and Loki Entertainment Software, and is provided under the Lesser GNU Public License (*LGPL* 2005). The OpenAL API is easily integrated with OpenGL, as both the coding style and conventions resembles the OpenGL API. Even though OpenAL originally was designed for audio in games, it is now appropriate for many audio applications (*Loki website: OpenAL Specification and Reference* 2005).

Example 1 A VRML scene graph.

```
#VRML V2.0 utf8
   Transform {
   translation 0 1 0
   children [
      Shape {
         appearance Appearance {
            material Material {
               diffuseColor 0 0.8 0
            }
         }
         geometry Sphere {
            radius 0.5
         }
      }
   ]
}
```

The OpenAL functionality revolves around three main kinds of objects: sources, buffers and listeners. A buffer is filled with audio data, whereas a source describes an instance of a buffer, a position from which a certain buffer emanates. The sources can be controlled by setting parameters such as position, velocity, direction and angles for cones that determine how the sound is traveling. All parameters are set relative to the listener, which is the object defining the user's position in the 3D arrangement.

OpenAL contains functions to support the use of extensions, such as DirectSound3D and EAX.

3.6 Coin3D

Coin3D is a set of software libraries developed by Systems in Motion (SIM 2005*a*) for 3D graphics software development. The API is built on top of OpenGL and is compliant with the Open Inventor 2.1 API. Coin3D comes with libraries for interfacing Coin with the native Microsoft Windows GUI, Trolltech's QT, Xt/Motif on X Windows and the native Mac Os X GUI. It also includes libraries for import and export of various standard file formats. Coin3D is based on a scene graph (derived from the Open Inventor API) structure and the data used in the model is stored as nodes in this structure. An example of such is depicted in example 2. The resulting 3D graphics is shown in figure 3.14.

The resemblance between a Coin3D scene graph and a VRML scene graph

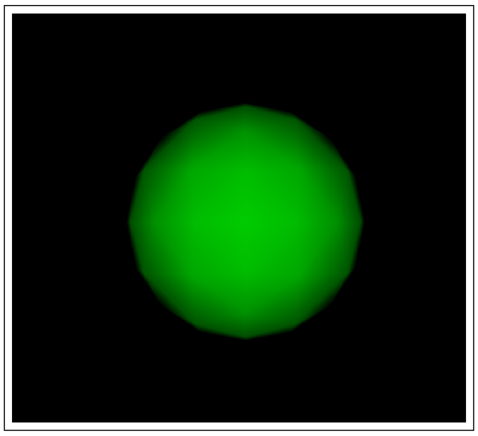


Figure 3.13: Example of 3D graphics produced by the scene graph in example 1.

Example 2 A Coin3D scene graph.

```
#Inventor V2.1 ascii
DEF Root Separator {
   Transform {
      translation 0 1 0
   }
   Material {
      diffuseColor 0 0.80000001 0
   }
   Sphere {
      radius 0.5
   }
}
```

is evident (ref. section 3.4). This is due to the fact that they both derives from the Open Inventor API.
The Coin3D source code used to produce the scene graph in example 2, and hence figure 3.14, is illustrated in example 3.

In addition to the 3D graphic support, Coin3D also supports modeling and rendering of 3D sound. The library contains different classes to support this. These classes are described in the following sections.

SoAudioDevice

SoAudioDevice is the class used to control the audio device. Coin3D uses OpenAL to render audio, meaning that a runtime library of OpenAL must be installed in addition to Coin3D to enable sound support. Configuring

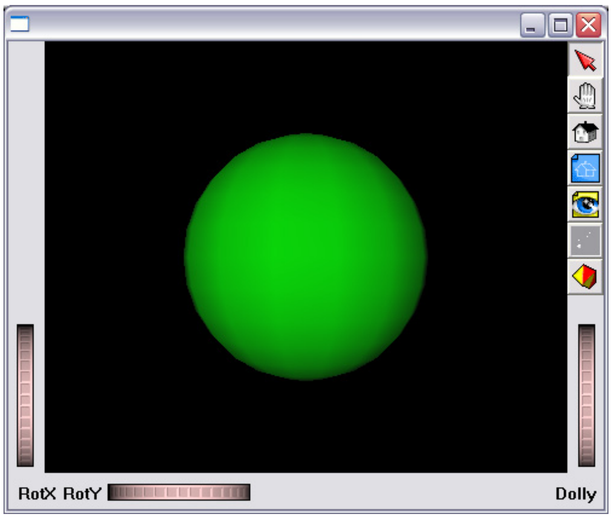


Figure 3.14: Example of Coin3D graphics produced by scene graph in example 2.

OpenAL according to a specific speaker configuration can easily be done by using DirectSound3D as an extension to OpenAL. The available configurations will thus be dependent on the DirectSound3D driver and will be transparent to both Coin and OpenAL. Examples of configurations which can be used are 5.1 and 7.1-channel systems originally made to support the respectively Dolby Digital surround formats.

SoVRMLSound

The SoVRMLSound class is used to represent a sound source (node). As the class name indicates, it is a reimplementation of the VRML97 standard (*VRML97, ISO/IEC 14772-1* 1997). Or more specifically, as the following section will explain, it is a partial reimplementation of the VRML97 standard.
The VRML97 standard defines sound sources in a different way than e.g. OpenAL, and it is therefor necessary to introduce the reader to some new concepts.

- **Geometry**
  The geometry of a sound node is defined by two ellipsoids; an inner ellipsoid and an outer ellipsoid as depicted in figure 3.15. In the inner ellipsoid there is no attenuation of the loudness. The attenuation between the two ellipsoids is linear.

- **Intensity**
  The intensity adjust the loudness of the sound emitted by the sound

Example 3 Example of Coin3D source code.

```
SoSeparator * root = new SoSeparator;
   root->setName("Root");
SoSphere * sphere = new SoSphere;
   sphere->radius = 0.5f;
SoMaterial * sphereMat = new SoMaterial;
   sphereMat->diffuseColor.setValue(0.0f, 0.8f, 0.0f);
SoTransform * sphereTrans = new SoTransform;
   sphereTrans->translation.setValue(0.0, 1.0, 0.0);

root->ref();
root->addChild(sphereTrans);
root->addChild(sphereMat);
root->addChild(sphere);

viewer->setSceneGraph(root);
viewer->show();
```

node. The value ranges from 0.0 to 1.0, where 0.0 is used when no sound is emitted. The intensity defines the loudness within the inner ellipsoid.

- **Direction**
  The direction vector stretches from the location of the sound source and along the longest diagonal of the ellipsoids.
- **Min/Max**
  As depicted in figure 3.15, the minBack, maxBack, minFront and maxFront defines the relative position of the ellipsoids' focus points relative to the location.

SoVRMLAudioClip

The SoVRMLAudioClip is a container for audio data that can be referenced by a sound node (SoVRMLSound). The container supports audio formats like the wavefile format in uncompressed PCM format and Ogg Vorbis[2].

3.7 CSound

CSound is a programming language designed and optimized for sound rendering and signal processing. Created in 1985 by Barry Vercoe, CSound is

[2]Ogg Vorbis: An open source patent-free audio compression format.

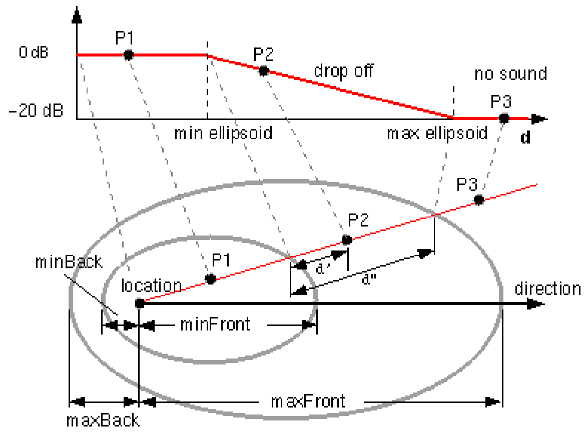


Figure 3.15: Sound node geometry.

one of the most widely used software for sound synthesis. It supports several sound synthesis methods, analysis and resynthesis, support for room simulation and 3D modeling, and physical and mathematical instruments modeling (Boulanger 2005).

CSound is based on four components; the compiler, the orchestra file, the score file, and the output sound (as depicted in figure 3.16). The compiler works by first translating a set of text-based instruments, found in the orchestra file, into a computer data-structure that is machine-resident. Then it transforms these user-defined instruments by interpreting a list of note events and parameter datas that the program "reads" from: a text-based score file, a sequencer-generated MIDI file, a real-time MIDI controller, real-time audio, or a non-MIDI devices such as a normal keyboard or a mouse. Related to the work in this assignment, a text-based score file would be the desired input type.

Depending on the speed of the computer (and the complexity of the instruments in the orchestra file), the performance of the compiler can either be auditioned in real-time, or written directly into a file (raw, wave, aiff or IRCAM) on the hard disk (Boulanger 2005).

3.8 Snøhvit Simulator

The Snøhvit Simulator (SnøhvitSIM), is developed by Statoil with Systems In Motion and Geodata as sub contractors. SnøhvitSIM is developed using the awarded Geo2000 system. It is a virtual model of the Snøhvit facility, including the Melkøya process plant and the subsea installations. The user interacts and navigates within the simulator by a mouse. Most objects are interactive and have certain actions attached to them, like *Open* or *Goto*. Two snapshots from the simulator are presented in figure 3.17.

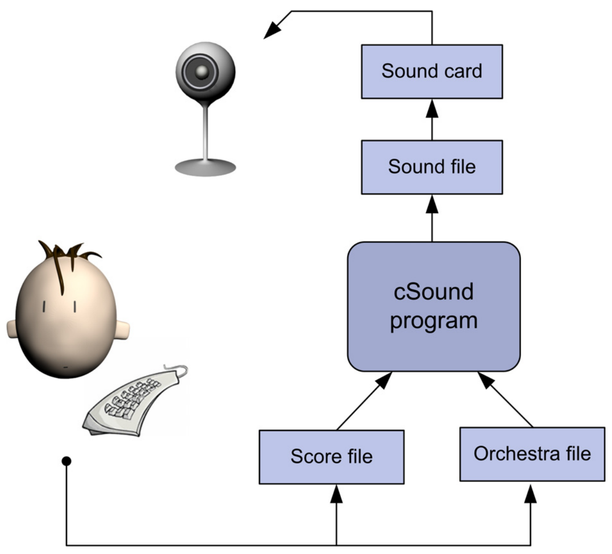


Figure 3.16: Base components of CSound.

The Geo2000 platform consists of several components which are interconnected through various interfaces. Details regarding some of the components will now be described.

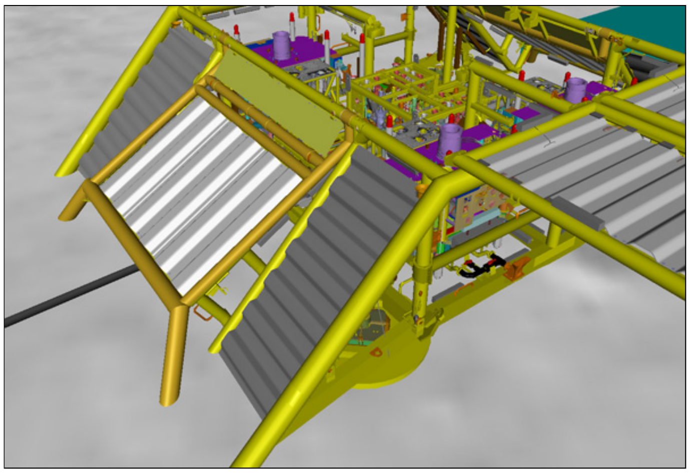
OpenGL is a low-level graphics library specification.

Coin3D is a high-level 3D graphics library.

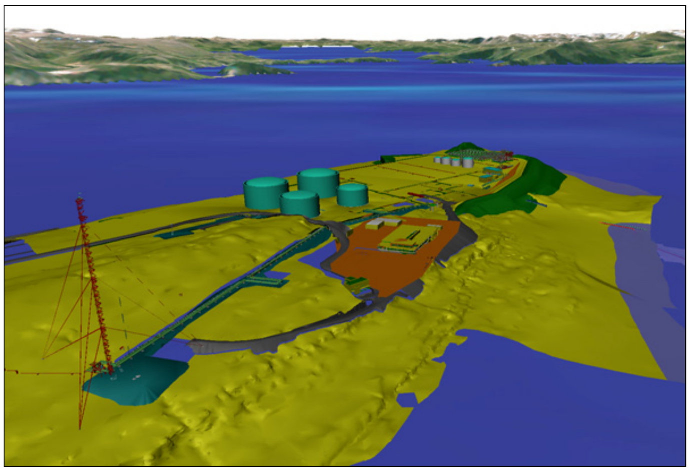
Scenery is used for real-time visualization of huge terrain databases, and is a product of Systems in Motion. The terrain database is built according to a quadtree structure, which enables different advantageous techniques, like continuous level of detail (CLOD) and popping-avoidance (SIM 2005*b*). The combination of Scenery and Coin3D provides the ability to view large terrain databases in real-time together with almost any other type of geometric data.

Scheme is a script language, and is incorporated in the Geo2000 platform using the Guile library. Using Guile to develop wrappers for internal Geo2000 C++ functions, these functions can easily be called from Scheme scripts. The usage of Scheme in Geo2000 is an architectural decision, providing flexibility for e.g. the user interface (Thomassen 2005).

An overview of the previously described components, and how they are related to each other, is depicted in figure 3.18.



(a) Subsea template.



(b) Melkøya process plant.

Figure 3.17: Snapshots from the Snøhvit Simulator.

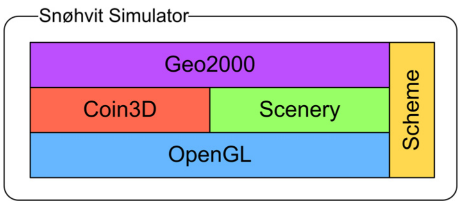


Figure 3.18: Overview of SnøhvitSIM software components.

Chapter 4

Situation today

Alarm management systems used in operational environments, such as control-rooms, have during the last decades evolved in many areas. From manual interaction and mimic boards to highly advanced computer systems as we see today. Despite this, the conceptual design of the user interfaces remains the same as it did ten years ago.

This chapter is based on interviews and visits made by the author. This includes interviews with control-room operators, former control-room management, alarm management system designers, in addition to participation at several lectures regarding the topic. Reports from this work are documented in appendix A to D.

4.1 Alarm management systems

The systems which are used today, also called Distributed Control Systems (DCS), consists mainly of 2D schematic process diagrams. The process diagrams contain real-time data, and are used to investigate and control a plant or facility. An example is given in figure 4.1.

All alarms are normally listed in separate windows, but are are also represented in the process diagrams as text or highlighted objects. The complexity of these diagrams are rather high, and the investigation often requires switching between many diagrams. Information gathered from Statoil tells that the alarm-rate may rise up to several hundreds an hour. Even though this may not be the usual scenario, it is obvious that there exist challenges related to the user interface.

Until now, most of the development has revolved around finding new and effective layouts for process diagrams. It has undoubtedly been made great progress here, and today's layout seem far more intuitive than earlier versions. Some system vendors have also started to look into 3D graphics and virtual models. Although this is still on a very early development-stage, it shows that the interest is present. The initiatives taken so far, aims towards

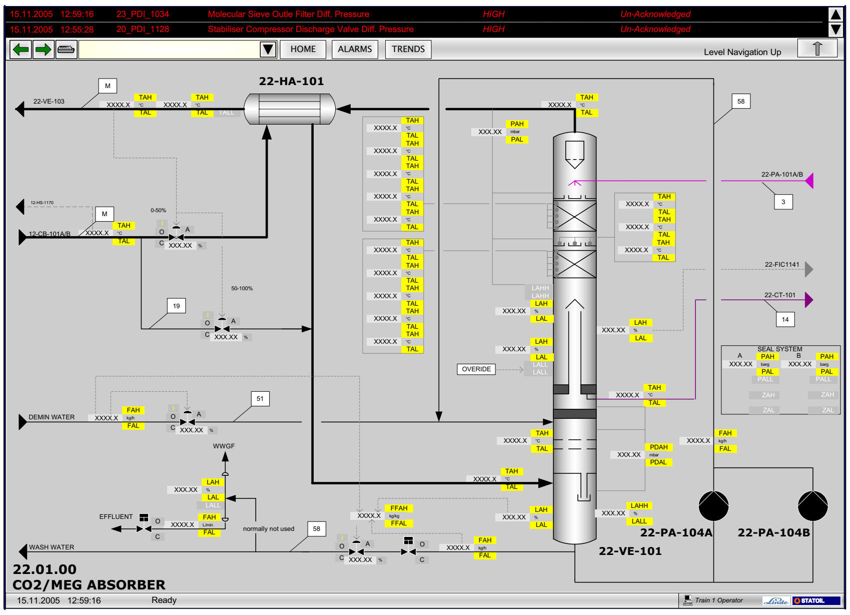

```
15.11.2005  12:59:16   23_PDI_1034   Molecular Sieve Outle Filter Diff. Pressure            HIGH   Un-Acknowledged
15.11.2005  12:55:28   20_PDI_1128   Stabiliser Compressor Discharge Valve Diff. Pressure   HIGH   Un-Acknowledged
```

Figure 4.1: Example of an alarm management system process diagram from Snøhvit DCS. Courtesy of Statoil.

systems like remote operations and collaborative environments. Whether this technology also will be included in alarm management systems or not is unclear.

According to the objectives of this master thesis, it has also been investigated to what degree sound is used in control rooms and alarm management systems. The findings have been rather slim, showing a minor degree of audio utilization. The most common configurations use sound only to catch the operators attention. This is usually achieved by simple and monotonous sounds whenever new alarms occur. However, the feedback given from operators and others involved with alarm management systems, shows a great interest in further development in this area.

4.2 Challenges

Even though much effort is spent to improve user interfaces, these interfaces does not solve the underlying cause of the challenge; the alarms. It is the author's beliefs that the biggest challenge regarding alarm management systems is to reduce the amount of alarms. This can be done in several ways, and the best approach is probably a combination of many of them.
The ideal situation would be to avoid all alarms from happening. Instrumentation failures, component breakdowns and inaccurate readings are all

incidents which could be avoided if the equipment were maintained correctly. Still, some alarms will always occur. Taking into consideration the highly advanced processes which often are involved, it is unlikely that these will ever run fully automated and without any human intervention. In other words, it is plausible to believe that some sort of alarm management system always will be required. However, this does not mean that all alarms must be presented to an operator. Installing some sort of intelligent alarm filter could possibly retain unnecessary alarms from the operator, giving him or her more time to focus on other alarms. The filter could for instance be instructed to remove alarms which came as an expected consequence of another alarm. It could also constantly be monitoring all alarms and easily maintain a prioritized list of all alarms according to criticality. Other supporting applications, such as trend analyzers, could also be implemented in this solution. This would improve the quality of the proactive work, and eventually reduce the amount of alarms even more.
Finally, when the amount of alarms have been significantly reduced, it is important that the remaining alarms are mediated through an effective interface. Taking into consideration today's trends, with fewer persons operating more complex and time-demanding tasks, is like putting more wood on the fire. By utilizing the possibilities within sound, through conscious or subconscious mediation, a more effective interface could be developed.

It is important to notice that the alarm management system only fills one fraction of the systems used to support the operators. Their challenge is to achieve full situation awareness (SA), which is not gained through the alarm management system alone. Improving this would nevertheless also improve the overall situation.

Chapter 5

Related work

This chapter describes some of the work that has been done in fields that relate to the objectives of this master thesis.

5.1 Alarm management systems

Ian Nimmo, the president and founder of User Centered Design Services has more than 35 years experience within the fields of user interfaces and control room design (Nimmo 2005*c*). In his work, he develops a framework to describe supervisory control (Nimmo 2005*b*). This framework outlines distinct intervention activities that occur during an abnormal situation, categorized in different stages. This framework is illustrated in figure 5.1.

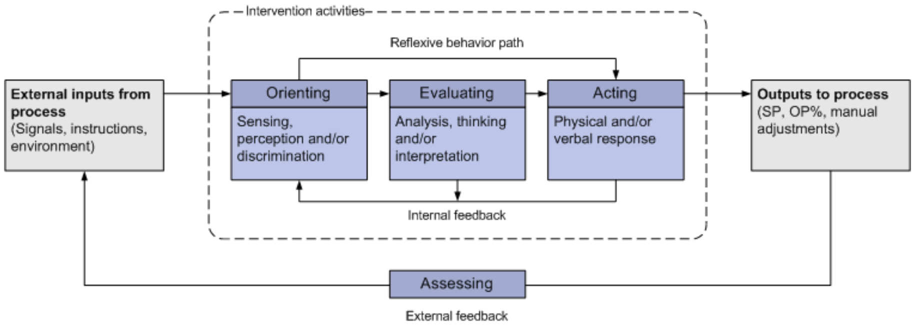

Intervention activities
Reflexive behavior path
External inputs from process
(Signals, instructions, environment)
Orienting
Sensing, perception and/or discrimination
Evaluating
Analysis, thinking and/or interpretation
Acting
Physical and/or verbal response
Outputs to process
(SP, OP%, manual adjustments)
Internal feedback
Assessing
External feedback

Figure 5.1: Framework defined by Ian Nimmo showing intervention activities during an abnormal situation.

Using this framework in a real context can help to pinpoint possible problems and areas of improvement within an operations team or related to its external conditions. In relevance to the objectives of this master thesis, the first stage (orienting) is the most interesting. This stage involves perceptual discrimination of an anomaly, i.e. an alarm, in the supervised process.
According to Nimmo, several external contributors may affect this activity

and cause human intervention failure. These contributors are among others: information overload, inappropriate detail, navigation problems, distracting environment and missing information. Creating a new design for an alarm management system, as will be suggested in this master thesis, should include an evaluation and a list of measures to avoid these. However, according to the scope and limited amount of time available, this can not be achieved.

By participation at various workshops, much information is gathered regarding today's alarm management systems and future development. This has been very interesting and has provided valuable information and references used throughout this report. Summaries are included in appendix A and appendix D.

5.2 Auditory display

The expanding usage of auditory displays and the need to convey new types of information with sounds, leads to a demand for descriptions of best practice. Matt Adcock *et al.* address this problem by describing good solutions to common problems in so-called design patterns (Adcock & Barrass 2004). In his work he presents six new prototype design patters for auditory displays, including patterns for system monitoring and situational awareness. The patterns are written to be easily understood and consists of the following sections: Context, problem, forces, solution, rationale and examples. An example from the situational awareness-pattern, defines one of the problems like this: *"Without appropriate support, the user is not likely to perceive and understand all the relevant environmental activity."*. The solutions are maintained by user groups, which constantly updates the information regarding new and upcoming problems.
Even though the design patterns have been sparsely used in this master thesis, they provide a very useful tool when developing new auditory displays.

Another research project (Kazem, Noyes & Lieven 2003), exploring different designs of auditory displays, is performed by Kazem *et al.* The project investigates design guidelines and whether a background auditory environment can be used to increase a pilot's situational awareness. Situational awareness is a term used to describe the operator's ability to perceive and understand, as well as predict future events within their operating environment (Endsley 1994).
The conclusions made from Kazem's work is very positive, and it is found that the technology has a potential to enhance the pilot situation awareness and reduce peripheralisation. This can be achieved by means of an background auditory display using spatialized sound (3D sound). The design is meant to provide an impression of the state of the aircraft relative to the outside world as well as an indication of basic aircraft system state. This is also verified by Ian Nimmo in his work on operator situation awareness and

the impact on control room design (Nimmo 2005*b*).

One of the initiators behind the terms auditory icons and auditory displays is William Gaver. In his application SonicFinder (Gaver 1989) sound was added to the components of the Apple Macintosh's graphical user interface. Files were represented by wood-based noises, applications by metallic noses and folders by paper-like sounds. Various modifications of the basic audio parameters were made to represent different features of the interface. For instance, the larger a file, the deeper its sound. The result from Gaver's project has been an inspiration to many new projects and also to further development in new areas.
Another application worth mentioning is the ARKola simulation, also developed by Gaver (Gaver, Smith & O'Shea 1991). In this application the semantics of different machines used to produce soft drinks where mapped to different analogical sounds.

In co-operation with a local radio station, Hermann *et al.* have performed a pilot project using sonification to render and present auditory weather forecasts (Hermann, Drees & Ritter 2003). The sonifications include auditory markers for certain relevant time points, expected weather events like thunder, snow or fog and several auditory streams to summarize the temporal weather changes during the day. The data set used to generate this sonification consisted of 9 different dimension, including temperature, humidity, rainfall, fog etc. Several different designs were tried, from mere pitch- and level modified sound streams to the usage of auditory icons. The finally used sonification used a multi-stream of metaphoric auditory icons. The auditory icons were carefully selected to avoid masking effects and interference. An example of a sonification is depicted in figure 5.2.

Mediating multi-dimensional data sets is one of the challenges in this master thesis. Even though the application area used by Hermann is quite different, the idea of using multiple metaphorical auditory icons may be worthwhile further exploration.

5.3 Sound design

Auditory warnings in particular have enjoyed comprehensive investigation in many application areas. Researchers have illustrated the ability to design highly informational warnings conveying an appropriate level of urgency and tackling user confusion and compliance issues (Kazem et al. 2003).

In (Catchpole, McKeown & Withington 2004), the authors identify three important forms of information in an auditory warning: what (semantic), where (location) and when (perceived urgency). These forms are all addressed in the design suggested by Catchpole *et al.*. The results coming from a series of test concluded that auditory warning pulses may be designed using simultaneous notched noise and tonal signatures. The benefits

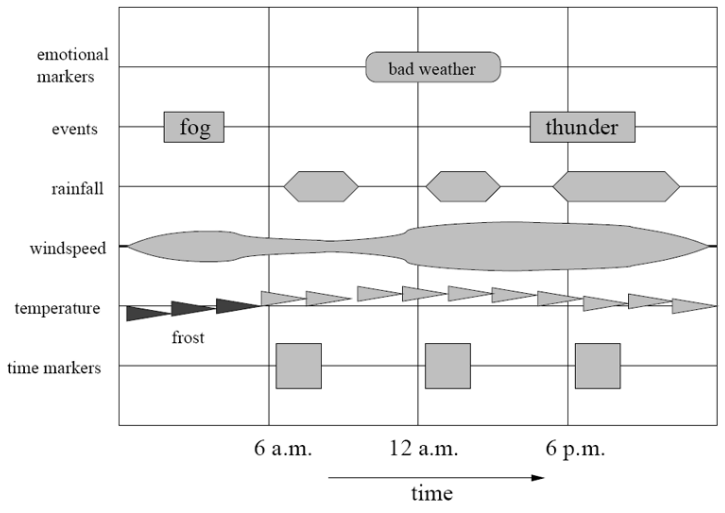


Figure 5.2: Illustration of the weather forecast sonification used by Hermann *et al.*

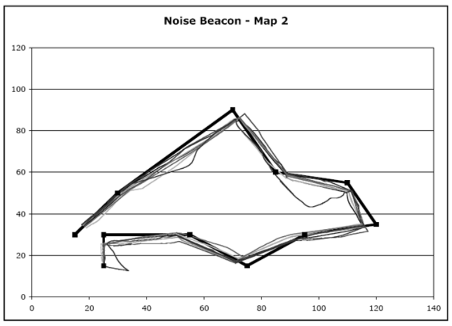
achieved by this were highly localizeable warnings due to the broadband noise. Further, the signals were highly manipulable in accordance to the urgency of the event and they formed discriminable auditory warning which mapped well to the mediated event.
This work have some interesting results. Using a combination of broadband noise and tonal signatures fulfills the demands for both localization and mediation of the warning. Replacing the tonal signature with an auditory icon is an interesting combination which should be investigated closer.

As already have been mentioned several times in this chapter, auditory icons have been used in many settings to convey information about system events by analogy with everyday events. This may be due to the fact that these sounds have the potential to be understood more quickly and easily than abstract sounds. Graham has in compared auditory icons with conventional sound warnings (Graham 1999). The study was performed for an in-vehicle collision avoidance application and focused on three main measures: reaction time, number of inappropriate responses and subjective ratings. The results showed that the participants responded significantly faster to auditory icons, but that auditory icons also produced more false-positive reactions. Due to these results, Graham stresses the fact that auditory icons may be *perceived by the listener to be 'gimmicky' or semi-serious.* Auditory icons should therefor be used sparingly and perhaps in combination with other types of auditory warnings. This could be abstract sounds, earcons or speech, each of which has its particular advantages and disadvantages. This somewhat pessimistic comment of Graham's should not be forgotten. Huge and easy accessible sound collections enables very creative designs, which does not necessarily perform according to the designers intention.

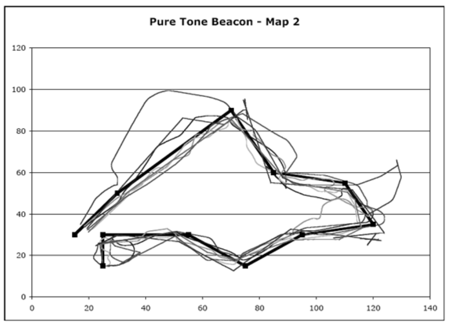
More research on the perception of acoustic warnings have been done. From a psychophysical point of view, Guillaume *et al.* have evaluated different psychological conditions that affects the perceived degree of urgency in acoustic alarms (Guillaume, Drake, Rivenez, Pellieux & Chastres 2002). It has also been studied whether or not former research on acoustic parameters in urgency perception are valid. Guillaume *et al.* confirms these results according to their own findings: *"The sequences perceived as the most urgent were fast, had a high pitch varying temporarily in a random way, irregular harmonics and a fast onset ramp. The less urgent sequences had a low rate, a quite low pitch progressively falling over time, regular harmonics and slow onsets."*
When it comes to the listeners reaction and the influence of his or hers psychological condition, the efficiency in inducing perception of urgency were reduced when the subjects were under a high workload. However, learning is pointed out as an important element to compensate for this. If the link between the sequence and alarm notion or urgency was reinforced, less attention was found to be required. With respect to this master thesis, these results stress the importance of operator training if an auditory display were included in the alarm management system.

So far, much work regarding the design of auditory icons have been mentioned. One topic which has got less attention, is the navigational issues related to these sounds. In (Catchpole et al. 2004) it is found that broadband noise can be used to improve the localization cues of an auditory warning. Walker *et al.* takes this research much further and investigates the performance achieved using different types of beacon sounds (Walker & Lindsay 2003). A series of subjects were assigned to conditions with differing auditory beacons and assessed on how quickly and efficiently they were able to navigate through a series of virtual environments. The research also included measurements on beacon type effectiveness and the impact of training. Figure 5.3 show the results of the participants using two different beacons.

The studies of Walker *et al.* lead to three main points. First, practice proved to has a major effect on performance. Second, different beacon sounds lead to markedly different performance. A 1 second broadband noise burst centered on 1 kHz gave the overall best results. The two other beacons used were a pure sine wave and a sonar pulse. The last point showed that using an auditory navigation system is most effective in cases were the user are not constrained to a tight path such as corridors or sidewalks. Even though this is an interesting finding, it is of little interest in the context of this master thesis. The two first points however adds up to the conclusions that training and broadband sounds are important keywords when working with localization and navigation in auditory environments.



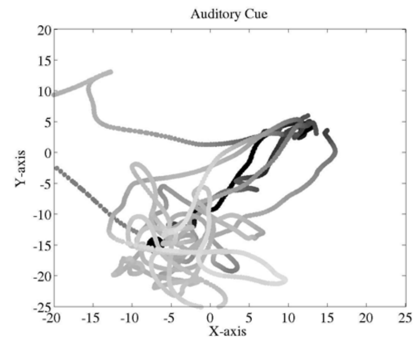
(a) Noise beacon.



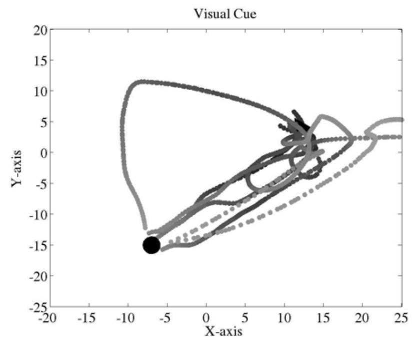
(b) Pure tone beacon.

Figure 5.3: Paths traveled by participants using different beacons in study by Walker *et al.*

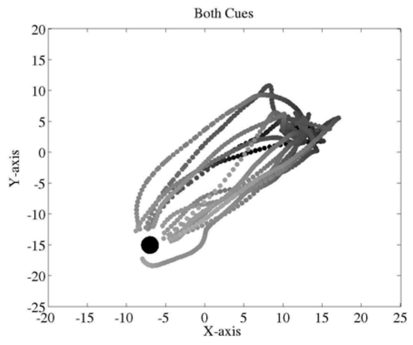
In (Gröhn, Lokki & Takala 2003), similar research is performed with a somewhat different focus. Gröhn *et al.* have carried out a navigation test in a spatially immersive virtual environment. The task of the participants was to find a series of gates while they navigated through a track guided by auditory and/or visual cues. The objective of the project was to find which combination of cues gave the best results. Audio-visual navigation was not surprisingly the most efficient. The projection of the results is depicted in figure 5.4. Further analysis of the travel paths indicated that auditory cue was utilized in the beginning to locate the next gate, and visual cue was the most important in the final approach to the gate.
According to the objectives of this master thesis, these results are very useful. Indicating a new alarm with a spatialized sound will probably have good effect on the user's perception of the alarm's origin.



(a) Auditory cue.



(b) Visual cue.



(c) Both cues.

Figure 5.4: 2D projection of the navigation paths in research by Gröhn *et al.* Coordinates of the starting point (triangle): [12, 4], and end point (dot): [-7, -15].

Part III

Own Contribution

Chapter 6

Task specification

6.1 A revisit to the purpose of the task

As described in chapter 2, a new system used to localize alarms and problems shall be developed. This system will be based on SnøhvitSIM, developed by Statoil with Systems In Motion and Geodata as sub contractors. The system will utilize the virtual model contained in the SnøhvitSIM, and add new functionality upon the existing framework. This is done to meet the new system's requirements regarding alarm management, including both alarm localization and mediation of the facility's overall status. The implemented prototype will consequently be an extension to SnøhvitSIM, and is from now on referred to as Alarm Management Extension (AMEX).

6.2 Requirements

As SnøhvitSIM still is in development, and therefore does not contain any realtime data, the input-data for the prototype must be simulated. The simulated data will be implemented as a part of the prototype.

6.2.1 Definitions

The following definitions are used in the requirement specification.

Prototype is the system to be implemented (AMEX).

User is the person using the prototype.

Object is a virtual component, e.g. a flange or a valve.

Facility is the sum of all objects, i.e. the entire virtual model.

6.2.2 Functional requirements

1. The prototype must detect occurring alarms and:
    (a) Inform the user
    (b) Identify the virtual object which caused the alarm
    (c) Be able to move the camera[1] to the placing of the alarm/object
    (d) Highlight the object
    (e) Attach an audible signal to the object representing the alarms':
        i. Type
        ii. Severity
    (f) Provide audible information which can be used to localize the alarm
    (g) Provide the user with relevant information about the object
2. The prototype must provide information about the overall status of the facility, including:
    (a) Visual representation
    (b) Audible representation

6.2.3 Non-functional requirements

1. The prototype must be implemented in the SnøhvitSIM framework
2. The prototype must have an user-friendly and intuitive interface

6.3 Priorities

The main focus of this master thesis has been to demonstrate the possibilities regarding the usage of virtual models and 3D sound in operational control-rooms. Hence, it has been deemed most appropriate to focus on the design of the prototype, and the sound design in specific. The outcome will be a proof of concept for inspiration rather than a base for further development. The priorities in the making of the prototype have been:

1. Design
2. Functionality
3. Architecture

[1]the camera refers to the users point-of-view in the virtual model

It should be noted that this would not have been the case if the outcome was to be an permanent extension to SnøhvitSIM. However, this is not the case in this study. The prototype will be used for demonstration-purposes only, and should therefor be easy to use and to set up. Resulting from this, the primary goals in the design of the prototype have been:

- Illustrate possibilities
- Use of auditory display
- Ease of use

Chapter 7

Architecture

7.1 Overall architecture

The overall architecture of the prototype consists of different units and is based on the specifications given in chapter 6. As illustrated in figure 7.1, the *Prototype Core* provides information to and from the SnøhvitSIM framework and works as a connection link between the other logical units in the prototype. The *Alarm Generator* generates random alarms on objects stored in the *Object Information unit*. In case of a fully operational system, the *Alarm Generator* would be replaced by realtime data from the facility. The *Audio Engine* and the *GUI Engine* is responsible for presenting information to the user when an incident occurs. These units will use functionality from the SnøhvitSIM framework.

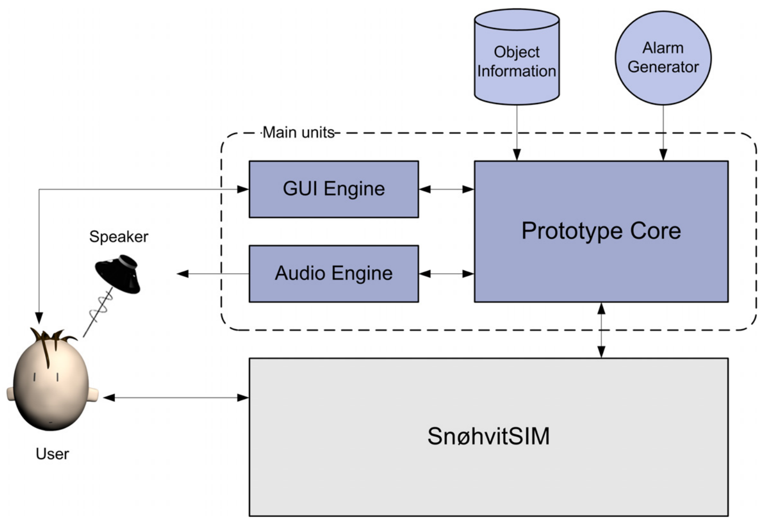


Figure 7.1: Overall architecture of prototype.

As the system will use 3D sound, a suitable speaker-setup is also comprised as a part of the prototype.

Further details on the main units are given in the following sections.

7.2 Prototype Core

As the name indicates, the prototype core contains the most crucial functionalities and logic of the prototype. The relationship between the core and the other prototype units is illustrated in figure 7.2

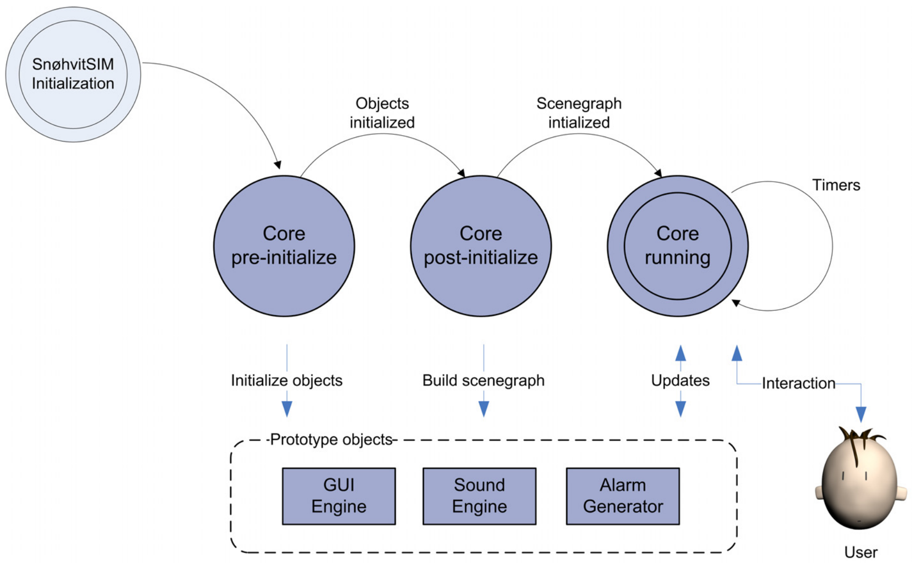


Figure 7.2: Schematic overview of prototype core.

The initialization process will be done in two steps. The first step, *Core pre-initialization*, takes care of creating all objects. This is initiated by the SnøhvitSIM framework. The second step, *Core post-initialization*, is also initiated by SnøhvitSIM, but subsequent to the SnøhvitSIM scenegraph creation. This procedure is due to the architecture of the SnøhvitSIM framework.
After the initialization, all actions are controlled by initiatives from either the user or core timers. The core timers will at certain intervals request new alarms. After receiving a new alarm, both the sound engine and the GUI engine will be updated. User-specific initiatives will be controlled by event-driven functions, which respond on either mouse or keyboard input.
As mentioned earlier, the prototype core will also be responsible for the interface towards the SnøhvitSIM framework.

7.3 GUI Controller

The GUI Controller will be responsible for keeping and updating the graphical user interface relevant to the prototype. This includes both virtual elements and elements in the HUD, used to project information outside the virtual environment, such as alarm-information, pop-up boxes etc.
All initiatives comes from the prototype core, which use public functions to send and receive relevant information to and from the GUI Controller.

7.4 Audio Controller

The Audio Controller resembles the GUI Controller, but instead of keeping and updating graphical elements it takes care of all audio elements. Starting, stopping and altering sounds are done on initiatives from the prototype core.

Chapter 8

Design

The requirements specification and system architecture have previous been presented in chapter 6 and 7 respectively. This chapter aims at giving a detailed view of the prototype implementation, including both hardware and software descriptions. A discussion is included in section 8.1 to justify the chosen approach described in the succeeding sections.

8.1 Discussion

8.1.1 Organizational challenges

Referring to appendix B and appendix D, many challenges must be met if this prototype is to be fully implemented in an operational environment. The suggested solution provides a whole new way of operating the control-rooms, and consequently it also affects the operators and their working routines. Moving from a 2D workspace, to a fully 3D virtual environment is a big change, and will for instance require training. In accordance with standard software development procedures, all involved users should participate in the specification and validation-process. This will hopefully avoid the feeling of enforcement, and instead provide an ownership to the system. It is also important for the users to fully trust the systems and its stability. A good way to achieve this is to give the operators time to get used to the system, and avoid forcing it onto them (appendix D.4.4).

8.1.2 Scope of the new system

An important aspect which should be carefully considered, is the usability and benefits which comes with virtual reality and 3D sound. As far as the author knows, there has been very few studies which looks into whether these technologies are fit to cover the entire alarm management process or not. This has also been confirmed in dialog with Ian Nimmo, president and founder of User Centered Design Services (Nimmo, 21.05.2005, pers. comm.).

Perhaps this new system eventually should be designed as an supportive tool in addition to today's 2D systems, instead of a full replacement.
There are no obvious answers to this question, but it is tempting to ask why virtual environments so far have been very little used in control rooms (section D.4.4).
However, the scope of the suggested prototype is not to replace the systems used in the control-rooms today. It will be a showcase to illustrate new possibilities regarding identification and localization of problems and alarms. As this only represent a fraction of today's functionality, much work still remains.

8.1.3 Simulation

As mentioned in section 6.2, the SnøhvitSIM does not receive any real-time data from the facility yet. It is still under development, and some sort of simulation will therefor be needed to evaluate the prototype. To keep this simulation within the scope of the master thesis, it is suggested to select a small part of the subsea equipment where a set of fake alarms are generated. It will be emphasized to demonstrate the possibilities and design of the prototype, more than keeping the simulation fully realistic. However, the simulated alarms will be discussed with experienced staff to retain the prototype's integrity.

8.1.4 Hardware

In a real implementation, choosing the right hardware is much about finding the best possible configuration within budget. However, when developing a prototype, things normally change. Important aspects of a prototype are for instance portability, ease of operation and set-up. The hardware should be relatively flexible, and in many cases it must also be within a smaller budget than in a full implementation. At the same time, developing a prototype using less quality components will reduce the quality of the output and consequently reduce the effect of the prototype. In other words, choosing hardware for a prototype is about making sacrifices in one way or another.
As mentioned previously in this chapter, the prototype which shall be developed also includes a 3D sound speaker system. Several available technologies exist (see section 3.2), and the choice of technology will strongly affect the software implementation. For this prototype, a fairly cheap off-the-shelf surround sound system will be used. This gives great portability at a reasonable price, but with some limitations on the 3D sound quality. However, the author believes that for this prototype, the result will be convincing enough.

8.1.5 Audio interface

A great part of this master thesis is to design and use good audio. This may seem quite straight forward, but there are a lot of things to take into consideration. It is important to remember that the audio and sound itself is not the essential part, but the information provided by the audio is. The intention for using audio is to reduce the information overload to the operators, and provide a new and redundant media for communication to improve the operator's situation awareness. Therefore, the audio must be designed to be very efficient and intuitive at the same time.

The auditory environment considered here is intended to enhance and reinforce, but not compete with, the visual cues available to the operator. This implies a multi-modal display of information (Kazem et al. 2003). It is important that the more or less redundant information presented by the auditory display, does not lead to ambiguity.

The audio used in the prototype can be divided in three main parts.

Alarm sounds Inform about new alarms

Beacon sounds Provide localization cues for alarms

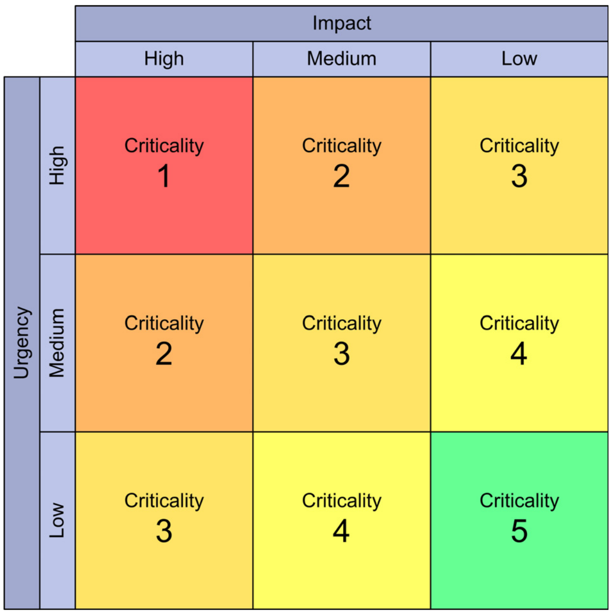
Status sounds Inform about the facility's overall status

The different types of audio will now be discussed regarding its characteristics, like type of sound, duration and usage.

Alarm sounds

The alarm sounds are similar to those alarms sounds which are used in today's alarm management systems. A sound will occur each time a new alarm occurs. However, there are some major differences. One of the aspects experienced at Tjeldbergodden (see appendix B), was the ongoing alarm-sounds. At the beginning these sounds were very stressful, but eventually they all became noise which did not catch one's attention. Hence, they had in general little effect on the operators. Besides, the alarm sounds were only in a small degree used to mediate information about the alarms. This was due to the fact that the operators saw a great potential in this feature.
In this prototype, the alarms will be categorized according to its type and criticality. Each type of alarm will have a specific sound, and some kind of altering effect will be used proportional with the criticality. Reverberation is an example of such an effect. This would cause a perception of distance when the alarm is less severe. Even though the alarm sounds will be spatially located in the 3D-model, this will not interfere with the manually added reverberation. This is due to the absence of a reverberation engine in the prototype. All sounds will however loose intensity proportional with the distance between the operator and the alarm's origin.

Classification of each alarm's criticality is an extensive task which is not within the scope of this master thesis. However, using an impact-urgency matrix as depicted in figure 8.1 can be a rational approach.



		Impact		
		High	Medium	Low
Urgency	High	Criticality 1	Criticality 2	Criticality 3
	Medium	Criticality 2	Criticality 3	Criticality 4
	Low	Criticality 3	Criticality 4	Criticality 5

Figure 8.1: Impact-urgency matrix for calculation of criticality, ranging from 1 to 5.

The impact could further be subdivided in a consequence-probability matrix. These matrices should be calculated individually in cooperation with technical staff such as field operators or control room operators.
The sounds used for each type of alarm should be some sort of a metaphorical auditory icon. It is not practical to use one sound for each alarm, since this will produce an enormous list of different sounds. The alarms must therefor be categorized, where each category has a specific sound attached. This leads to another challenge; how to categorize the alarms. The author has learned that some control-rooms have categorized the alarms according to which operator is responsible for it (appendix D). As the operators normally are responsible for different sections of the facility, *section* would be the category keyword. Such a solution offers a rather coarse subdivision of the alarms, and could possibly be improved by decreasing the scope of each keyword. After discussing this particular challenge with the operators at Tjeldbergodden (appendix B), it was suggested to use *system* as a dividing keyword. At Tjeldbergodden, the facility is divided into several systems; air separation unit, methanol plant, gas receiving station, utilities, etc. This approach requires that the facility are divided into a reasonable number of physical or logical systems. If the number grows to large, the quantity of different sounds will probably become too extensive and cause the operators

to loose overview.
The alarms mentioned so far are first and foremost addressing one type of alarms, known as *production alarms*. These alarms concerns the production processes of the facility. This could for instance be a pressure drop in a pipeline, or an open valve which does not respond to a given action. In addition, two more major types of alarms exist (appendix C and appendix A). *Safety-critical alarms* are one important type of alarms which must be mediated. These alarms occurs when either human or environmental damage is at stake. A third category is *notifications*, which hardly can be defined as alarms but still contain essential information to the operators. These two categories differ from the production alarms in several ways. First, in contrast to the production alarms, these alarms does not have a variable criticality. Safety critical alarms are naturally always critical, while notifications contains nothing but information about expected events. This leads to another disparity. As the outcome of these alarms either require full attention (safety-critical alarms) or just a glance with the eyes (notifications), the audible alarm for these should differ from the production alarms. Using only an altering effect to separate a production alarm within a system from a safety-critical alarm, may not be enough. Separate auditory icons can therefor be used for both safety-critical alarms and notifications, without considering which system is involved. Since the criticality for these alarms are constant, the alarm sounds used for these alarms could be static and without any variable altering effect. This leads to a categorization and design as summarized in table 8.1.

Type	System	Sound design	
		Auditory icon	Criticality / effects
Safety-critical alarms	All	Icon #0	Static [ ]
Production alarms	System #1	Icon #1	Dynamic [1, 5]
	System #2	Icon #2	Dynamic [1, 5]
	..	..	..
Notifications	All	Icon #9	Static [ ]

Table 8.1: Superior categorization and design of alarm sounds according to type and system.

Beacon sounds

The beacon sounds are intended to provide localization cues to the operator. This means that he or she could turn on the beacon on a specific alarm to hear its origin. The design and operation mode of this feature is challenging. For instance; what should it sound like? Should there be any special acoustic effects modulating the sound? Should there be many active beacons at once?

If so, how could they be partitioned?
Since this prototype's intention is to illustrate possibilities, it is probably satisfactory to only allow one beacon. The sound should be played in continuous pulses, and designed such that it provides good localization cues. Hence, a spectrally rich sound would be preferable (Morgan 1998). Using spectrally rich sounds for beacons instead of pure tones or speech, is also the conclusion of Tran (Tran, Letowski & KS 2000) and Walker (Walker & Lindsay 2003). To avoid misconceptions, the beacon will use a sound without relevance to the alarm sounds.
To increase the localization cues, it is possible to modulate the sound according to the operators position. This could be done either by altering the pitch or the playback-speed of the sound. For instance could the pitch increase when the operator was facing the right direction. Combined with the distance-controlled intensity already used in the virtual environment, this could provide good results.

Status sounds

The status sound mediates the facility's superior status. It represents a summary of all active alarms, which is reflected in a *status level.* The sound which is associated with each level will mainly be created using consonance and dissonance. This will hopefully lead to a system which sounds nice and consonant when everything is OK. This effect could be increased by adding sounds acting on the operators emotions, such the sound of an explosion, fire or chirping of birds.
Experiences from Tjeldbergodden (appendix B) shows that the operators in general are well informed about the current status of the facility. This is also confirmed by the interview of Per Ivar Karstad (appendix C). This information does not support the usage of a status sound. Even though it could be an interesting feature, it does not mediate any new information to the operators and should consequently be eliminated. Looking ahead, this scenario may change. Implementing intelligent alarm-filtering is an initiative from many major oil-producing companies (appendix D). This would hopefully reduce the amount of alarms in the control-room, which reduces the operator's ability to possess an overview. A similar feature to the suggested status sound could prevent this from happening. Other application areas are also possible. Forwarding this status sound to other personnel at the facility could be useful. This could for instance be the field operators, the shift leader or the operational management. It could also be played in rooms used by the operators, like the kitchen or bathroom. This would prevent the operators from loosing overview if they have to leave their desks.

8.1.6 Visual interface

The visual interface of the prototype will mainly consist of the already existing interface of the SnøhvitSIM (see section 3.8). However, some new elements will be necessary to support and control the alarm management system. As mentioned in section 6.3, the design of the visual interface is not considered the most important part of the prototype. Therefore, it will first and foremost be designed on a need-to-have basis. In a full implementation, or if more time were available, the visual interface would have to be more throughly considered.
Feedback from for instance Tjeldbergodden (see appendix B) fortified the importance of a visual list of all active alarms. This will be emphasized in the prototype, and it should contain relevant information about the alarms. In addition to this, detailed information about each alarm should be available. This could be done in several ways, but utilizing the virtual objects through animations would probably be the best way. An alternative solution is to present this information through an information board.

8.1.7 Interaction

It is important for the prototype to provide effective and intuitive interaction. The intended users of the system will at times work under hectic circumstances, and should therefor have easy access to all necessary commands.
Due to the restricted scope of this master thesis and the complex framework of SnøhvitSIM, the interaction will use a mixture of mouse and keyboard for interaction. An ideal solution would probably allow more interaction through a mouse (or similar devices), since this prevents the user from moving his/her hands so much. Despite this, the keyboard interaction should still be easy to grasp through information in the visual interface.

8.2 Simulation

The simulation for the prototype is situated on a subsea template, more exactly template N in the south-west corner of the Snøhvit area (see figure 8.2). The template is considered a suitable test-bed for the simulation, with a decent amount of details and infrastructure. A set of alarms have been pre-arranged for the prototype, all of them related to the subsea-system (system 18). The selection has been made in collaboration with Christian Salbu Aasland who works in the process control division at Statoil Research Center (Aasland, 28.05.2005, pers. comm.). The alarms are depicted in table 8.2, and will appear in random order and at random intervals in the prototype.

Due to lack of available 3D models, only the supporting steel structures of the templates are available in SnøhvitSIM AMEX. The wellheads placed

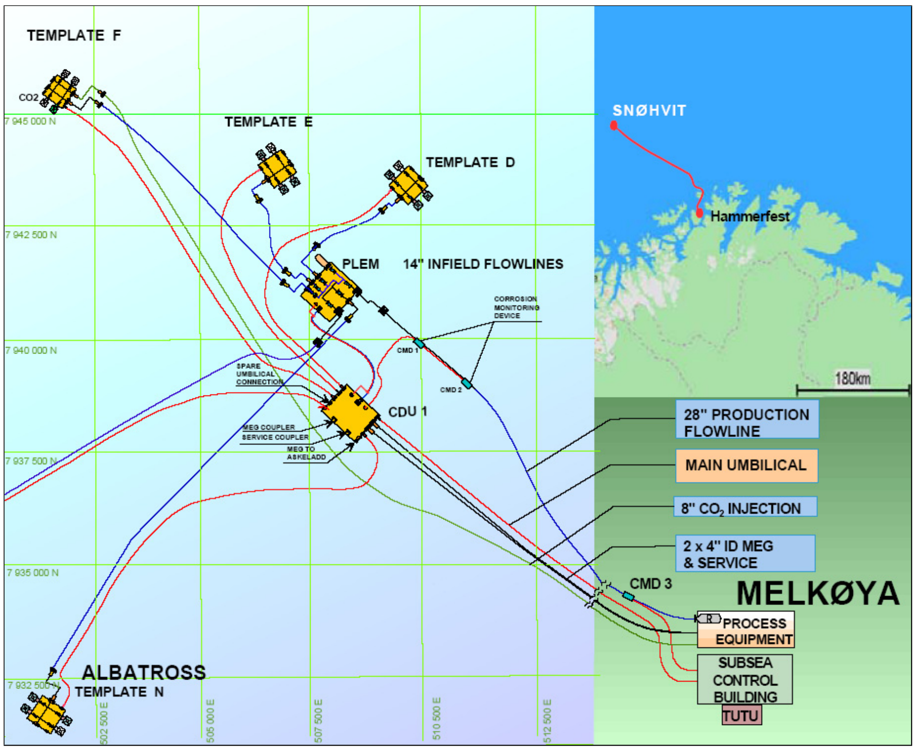


Figure 8.2: Snøhvit field layout.

Type *			ID	System	Details
SC	P	N			
X			0	Subsea	• Information: SCSSV failure • Criticality: n/a
	X		1	Subsea	• Information: Pipe, high pressure • Criticality: 3
	X		2	Subsea	• Information: Pipe, high temperature • Criticality: 1
		X	3	Subsea	• Information: Hatch open • Criticality: n/a

* SC: Safety critical alarms, P: Production alarms, N: Notifications

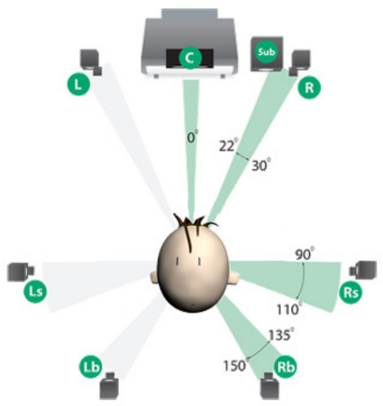
Table 8.2: Pre-arranged alarms used in the prototype.

inside these structures, containing all the sensor technology, is not included. It has therefore been difficult to get a correct spatial placing of the alarms within the template. Parts of the steel structure have consequently been used instead of the real objects. For a demonstration purpose, this should be no problem.
Further details regarding the simulation are described in the remaining sections of this chapter.

8.3 Hardware

The reproduction system used for this prototype is a Creative GigaWorks S750 speaker system (figure 8.3(a)), combined with a Create Audigy 4 Pro soundcard. This solution provides 8 separate channel-output, including one channel dedicated to the sub-woofer. The satellite speakers will be placed around the listener in a configuration as depicted in figure 8.3(b)

(a) Creative GigaWorks S750 7.1 speaker system used in prototype.



(b) Speaker placement of a 7.1 speaker system.

Figure 8.3: Speaker system and configuration used in the prototype.

As already mentioned in the previous discussion, this is not an optimal solution for 3D sound reproduction. However, the quality of the panning is found to be sufficient for demonstration purposes. In addition to this, an open headphone is included in the hardware-setup for single-user mode.
The hardware will be installed in a suitable room at Statoil's Research Center, Rotvoll. An illustration of the intended room is included in figure 8.4. The satellite speakers are placed in the ceiling.

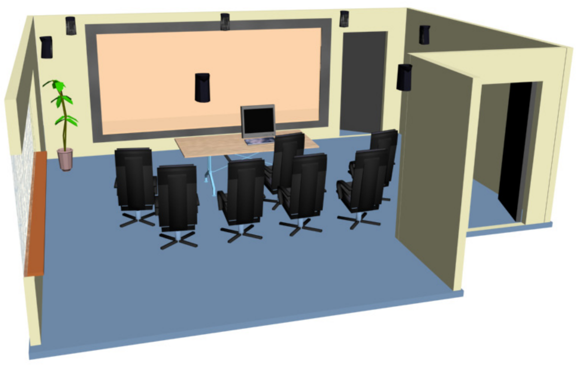


Figure 8.4: Model of room where hardware will be installed. Courtesy of Knut-Olav Fjell, Statoil.

8.4 Audio interface

The audio design describes the the auditory display which will be used in this prototype. It will consist of different elements, which together will fulfill the audible requirements given in section 6.2. The elements are categorized according to the type of information they are mediating.

8.4.1 Alarm sounds

As mentioned in the previous discussion, each new alarm will trigger an alarm sound. The design suggested here will at the first level categorize all alarms according to its type. This *includes safety-critical alarms*, *production alarms* and *notifications*. Further, the production alarms will be subdivided into different systems, where each system is a physical or logical part of the facility. In this case, only the subsea-system will be involved. The alarms will also be given a criticality-factor, resulting in an alteration of the auditory icon. It has been discussed to use reverberation as an effect to modify the auditory icon according to its criticality. In order to fortify the difference even more, the sounds envelope will also be changed. This envelope can be described by a so-called ADSR envelope, as depicted in figure 8.5. The envelope shows the sound's amplitude level at different stages of its duration. By increasing the attack-rate of a sound, the initial part will sound more abrupt, and consequently more severe.

It has also been emphasized to make the sounds sufficiently different to avoid masking and misunderstandings. However, no scientific approach has been used to confirm this. To achieve an optimal and intuitive meaning of

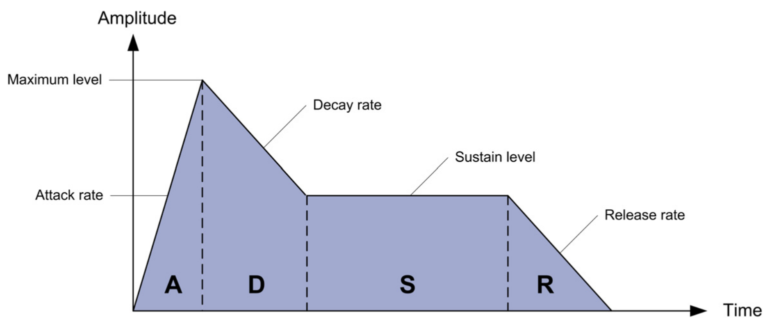


Figure 8.5: The four phases of an ADSR envelope.

the auditory icons, they should also be evaluated and tested by operational staff.This has not been performed in this master thesis.
According to table 8.1, three different auditory icons are needed. Among these, the auditory icon used on *production alarms* will have 5 different variations, each representing a criticality level. All auditory icons are created using cSound alone or in combination with sampled sounds. A description of all these auditory icons are presented in the following subsections.

Notifications

The notifications in general does not represent any danger, and neither should the related auditory icon. The sound of an aqualung[1] has been selected to represent a new notification. The sound does not imitate or relate to any critical situations, and it fits nicely into the marine subsea environment. As the frequency spectrum depicted in figure 8.6 shows, the auditory icon spans a broad band of frequencies. This insures sufficient localization cues for the user.

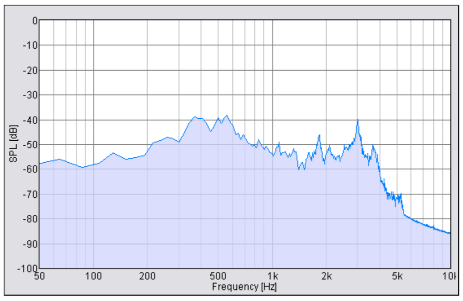
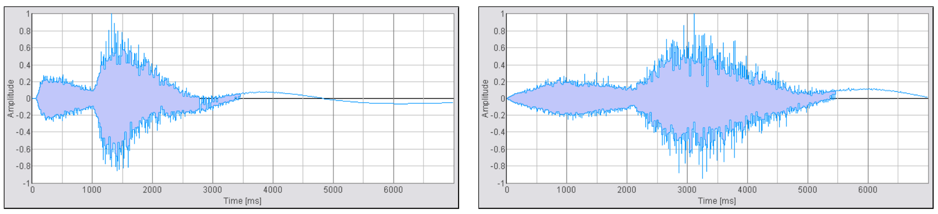


Figure 8.6: Frequency spectrum of notification alarm (aqualung).

[1]a device that lets divers breathe under water

Production alarms

According to the pre-definition of alarms (table 8.2), two different production alarms are needed: (1) High temperature (criticality 1) and (2) high pressure (criticality 3), where the latter is the least severe alarm. The auditory icon used to generate both these alarm sounds is composed by two different samples. The first sample is an air-burst, and the second is the sound of an explosion. These two sounds are put together differently, according to the criticality. In (1), the second sound starts after 1 second. In addition to this, the attack-rate is set high and the reverberation is decreased. In (2), a 2 second interval separates the two sounds, and the attack-rate and reverberation is opposite of (1). This design gives (1) more severe characteristics, with a shorter interval and more abrupt changes. The impulse response of (1) and (2) are depicted in figure 8.7(a) and figure 8.7(b), respectively. These figures shows the difference in interval, attack-rate and reverberation.
It is important to notice that the criticality of an alarm may change during its existence. If a measuring value exceeds its limits and keeps getting worse, it would lead to a more severe criticality-factor which also should affect the design of the sounds. This has not been considered in this prototype.



(a) Most severe production alarm (1). (b) Least severe production alarm (2).

Figure 8.7: Impulse response of production alarms.

Safety-critical alarms

The auditory icon used for the safety-critical alarm is similar to the production alarm, composed by two different samples. The first sample is the sound of a horn used in submarines. The second is an explosion slightly different from the one used in the production alarm. The intention of choosing the sound of a horn, is to use something quite different from other sounds to get the user's immediate attention. The explosion is used to indicate that something is wrong. The explosion is shaped with a high attack-rate and short reverberation (high release-rate). The impulse response of this auditory icon is presented in figure 8.8. As the figure shows, the explosion starts after approximately 2.25 seconds.

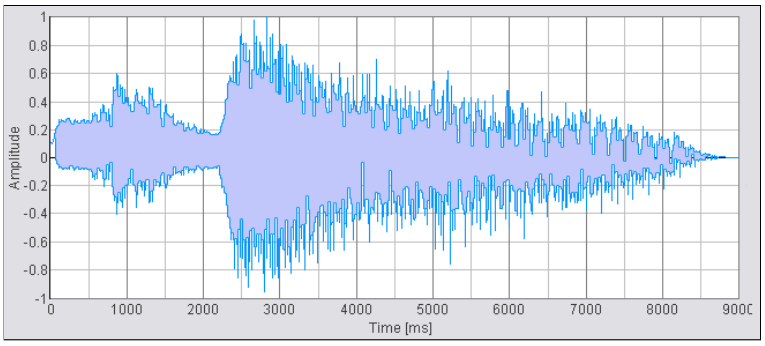


Figure 8.8: Impulse response of safety-critical alarm.

Having defined all alarm-sounds, table 8.2 in section 8.2 can be updated as shown in table 8.3.

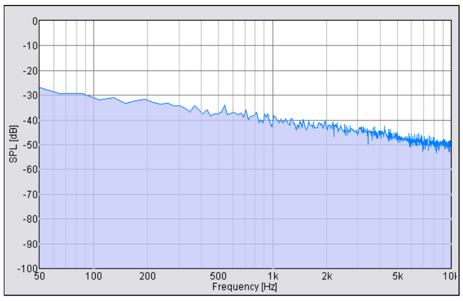
Type *			ID	System	Details	
SC	P	N				
X			0	Subsea	• Information: • Criticality: • Auditory icon:	SCSSV failure n/a Horn + Explosion
	X		1	Subsea	• Information: • Criticality: • Auditory icon: • Attack-rate: • Release-rate:	Pipe, high pressure 3 Airburst + Explosion Low (0.8 sec) Low (2.5 sec)
	X		2	Subsea	• Information: • Criticality: • Auditory icon: • Attack-rate: • Release-rate:	Pipe, high temperature 1 Airburst + Explosion High (0.1 sec) High (1.0 sec)
		X	3	Subsea	• Information: • Criticality: • Auditory icon:	Hatch open n/a Water / aqualung

* SC: Safety critical alarms, P: Production alarms, N: Notifications

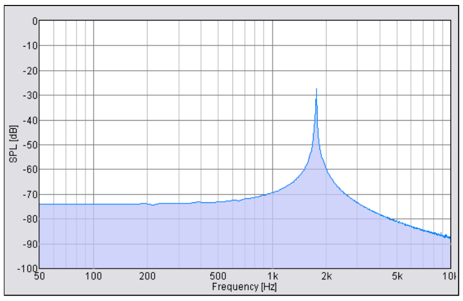
Table 8.3: Detailed design of pre-arranged alarms used in the prototype.

8.4.2 Beacon sounds

The most important thing when designing the beacon sound, is to make sure that it provides good localization cues. Work by Catchpole *et al.* (2004), Walker *et al.* (2003), Morgan (1998) and Tran *et al.* (2000) shows that using broadband signals provides the best results. A frequency analysis of two different sounds, pink noise, and a pure sine wave, is depicted in figure 8.9. The figure illustrates the differences between a spectrally rich broadband signal (noise) and a narrowband signal (pure sine wave), figure 8.9(a) and figure 8.9(b) respectively. The narrowband signal has its energy concentrated around a specific frequency (1,8kHz).



(a) Pink noise.



(b) A pure sine wave.

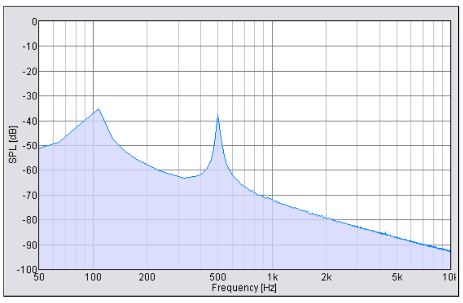
Figure 8.9: Difference between broadband and narrowband signals.

Even though noise may prove to be an appropriate choice, it may have enervating effects on the listener. To give a more pleasant sound, a compromise will be needed. The sound which are designed, using the cSound software, imitates a gong or a bell. It is composed by sine waves of different frequencies, and altered by vibrato[2] and reverberation. The main frequency is set to 500 Hz. The sound has also been shaped by an envelope to provide better localization. This is done by making the initial touch more abrupt, similar to the ADSR-envelope described in figure 8.5.

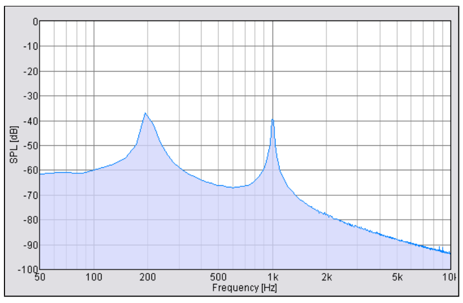
In addition to provide spatial information about its related alarm, the beacon must also help the operator to navigate effective in the virtual environment. To achieve this, the sound will be altered according to the operators direction. When the operator are facing the beacon, the sound's pitch and speed will be doubled. When facing the opposite direction, the pitch and speed is set to normal (1.0). A linear transformation will be used for the directions in between. Increasing the pitch and speed is a common way of indicating a nearby target and should be intuitive for the user to understand. It is important to notice that when the pitch are increased, the sound's energy will be centered at higher frequencies. According to Walker *et al.* (2003), the best results are achieved around 1 kHz, which coincide with the outcome in this case. In other words, the localization cues should improve as the user closes in on the desired direction.

[2] A smooth and repeated changing of the pitch up and down from the regular musical pitch, often done by singers.

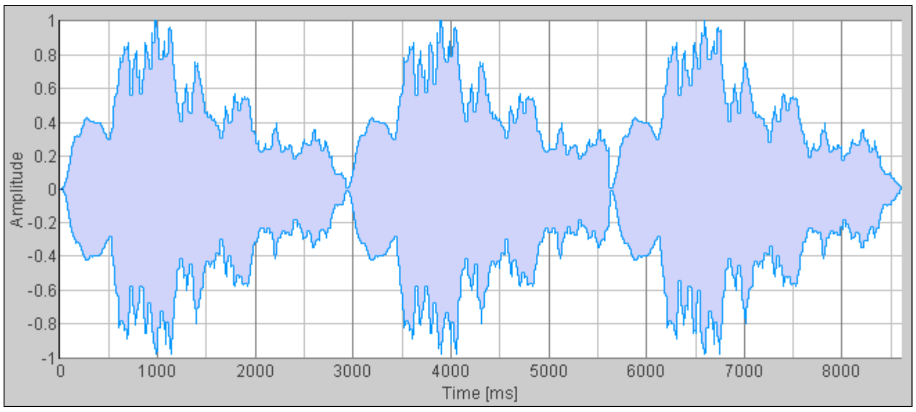
The beacon sound is depicted in figure 8.10. While figure 8.10(a) and figure 8.10(b) illustrates the frequency spectrum at pitch 1.0 and 2.0, figure 8.10(c) shows the impulse response of three consecutive strokes of the bell.



(a) Frequency spectrum at pitch 1.0.



(b) Frequency spectrum at pitch 2.0.



(c) Impulse response of three consecutive strokes.

Figure 8.10: Frequency spectrum and impulse response of the beacon sound.

8.4.3 Status sounds

The status sound mediates information about the overall status of the facility. The status has been defined on three different levels according to the number of active alarms. The status levels range from level 0 to level 2, where level 2 is the most critical. There are numerous ways of defining when to escalate the status level. A common approach is to generate a sum by adding together each alarm's criticality-factor. This gives a system where a high number of critical alarms will escalate the status level more than the same number of non-critical alarms. For this prototype, only 4 different alarms will be used, and the escalation routines are therefor somewhat simplified. The configuration used for the prototype is depicted in table 8.4.

Status level	Escalation range	Playback frequency
Status level 0	[0, 1] alarms	Every 10. min
Status level 1	[2, 3] alarms	Every 5. min
Status level 2	[4, 4] alarms	Every 2.5 min

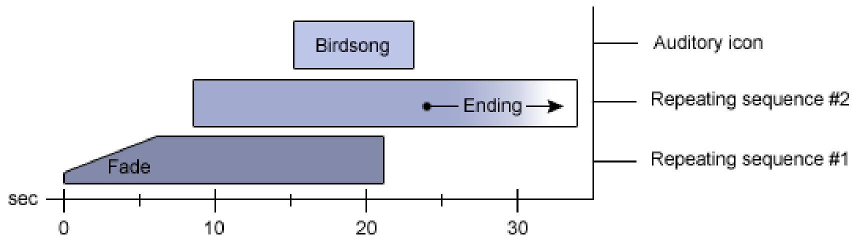
Table 8.4: Configuration of status sounds.

As table 8.4 shows, the playback frequency for the status sound changes according to its level. This means that when the situation gets more critical, the status sound is played more often. Which frequency that is most suitable at different status levels has not been an objective for careful consideration. In the case of a full implementation, more testing will be needed. It should be noted that the frequency has been set high due to demonstration considerations.
Each of the sounds used to mediate a status level, will be composed by adding together different sequences. A sequence consists of a series of tones with different frequencies and timber. According to the criticality of the sound, dissonance will be used to create a sense of danger. This will also be combined with various auditory icons which acts on the operators feelings. The suggested design is explained in figure 8.11.

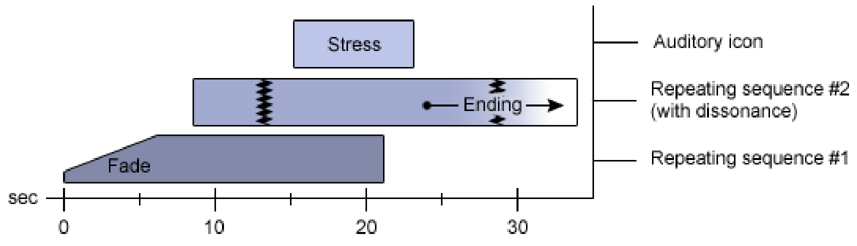
Each of the different designs is composed by three parts; an auditory icon (or a combination of several), and two different repeating sequences. The arguments for each part will :

Repeating sequence#1 A steady and consonant sequence of four different frequencies which are meant to prepare the listener for the subsequent information. It contains no information about the status level, and is only used to catch one's attention.
Frequencies: [200, 300, 400, 100] ↺ 10

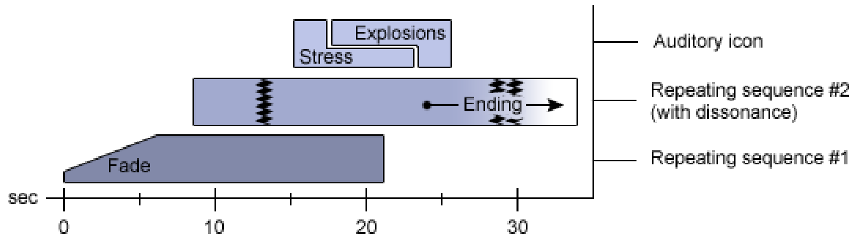
Repeating sequence#2 A more savage sequence of four different frequencies followed by a 10 second ending. The sequence is mainly consonant, but dissonant frequencies are used in the sound for level 1 and 2. The location of the dissonant frequencies are marked with a vertical wavy bar.
The opening of the sequence is meant to illustrate or imitate a query, which could be verbally translated to a series of *"Is everything OK?"*. This query is answered by a consecutive auditory icon, responding on the question.
Frequencies: [880, 1100, 780, 970] ↺ 4 + [880, 1100, 580, 370, 180]



(a) Sound design for status level 0



(b) Sound design for status level 1



(c) Sound design for status level 2

Figure 8.11: Design of sounds used for different status levels.

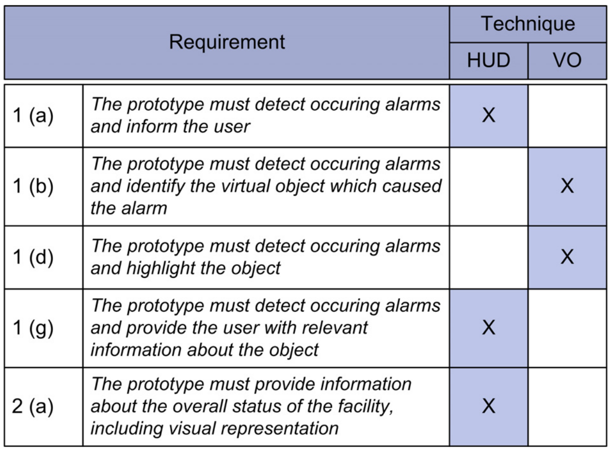
Auditory icon Different auditory icons are used to arouse different feelings in the listener:

- **Birdsong** is used to mediate that everything is OK.
- **Stress** is a fluctuating sound with rapid switches between left and right channel. Is is used to mediate stress.
- **Explosions** is the most severe sound, used to mediate danger. Consists of a series of consecutive explosions.

The status sound will be reproduced without any spatial location, hence normal stereo-sound reproduction.

8.5 Visual interface

Section 6.2 defined several requirements regarding the visual interface of the prototype. Even though all visual information could be created as virtual elements, this may not be the best solution. It has therefore been decided to utilize the HUD to present textual information when this is appropriate. The visual design has been divided in two different parts; HUD and virtual objects (VO).
Figure 8.12 repeats the requirements regarding the visual interface and denotes what type of technique are used to fulfill it. Details regarding each type is described in the following subsections. In addition to the require-



Requirement		Technique	
		HUD	VO
1 (a)	*The prototype must detect occuring alarms and inform the user*	X	
1 (b)	*The prototype must detect occuring alarms and identify the virtual object which caused the alarm*		X
1 (d)	*The prototype must detect occuring alarms and highlight the object*		X
1 (g)	*The prototype must detect occuring alarms and provide the user with relevant information about the object*	X	
2 (a)	*The prototype must provide information about the overall status of the facility, including visual representation*	X	

Figure 8.12: Techniques used to fulfill the visual requirements.

ments described in figure 8.12, the non-functional requirements states that the prototype must have an user-friendly and intuitive interface. This will be emphasized during the design.
It should be noted that it is not within the scope of this master thesis to

perform the needed user-testing which are required when designing a graphical user interface (GUI). More effort should be invested to assure that the right information is presented, and in the right way.

8.5.1 Head up display

As already mentioned, the Head up display (HUD) will be used for textual presentations. The HUD is a transparent layer placed in front of the user, similar to a car's windscreen. This means that independently of the users location within the virtual environment, the HUD will always be visible. Some problems comes with this technique. When putting text upon the HUD, it may be difficult to read, as the background color changes whenever the user moves or rotates. To cope with this, a semitransparent background will be used behind the text to make it more visible. In addition to this, the fontstyle will be outlined to give an even better contrast.
According to the requirements, three types of textual information are needed: (1) New alarm, (2) a list of all alarms and (3) detailed information about an alarm. This information will be presented in separate boxes as a result of an event or user input. For instance, when the system registers the occurrence of a new alarm, it should respond by displaying (1). If this event also enables new and event-specific functionality, this should be presented to the user (more about functionality in section 8.6). Details regarding all three types of information-boxes, including sketches, are presented in table 8.5. Some additional comments are included in the following sections.

New alarm

As described in figure **??**, this information should be displayed whenever a new alarm occurs. At this point, it is not suitable to give all details about the alarm. The operator needs to make a decision whether or not immediate attention is needed. The content of the information box will therefor be limited to the name of the alarm, together with its impact and urgency. Some logic will also be included regarding what types of commands that are available, depending on the criticality of the alarm. More details about this is described in section 8.6.

All alarms

The list or log of all occurred alarms will always be visible in the HUD. The list will be sorted by each alarm's timestamp, and the current status level of the facility will be presented at the bottom of the information-box. Whenever an alarm is taken care of, the color of the text will change from white to gray. This gives the operator an easy overview over the alarms that still is in need of attention. The information presented in the alarm list is

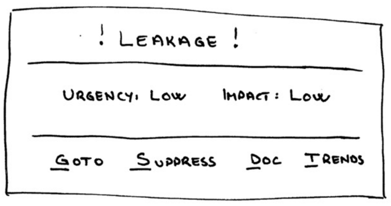
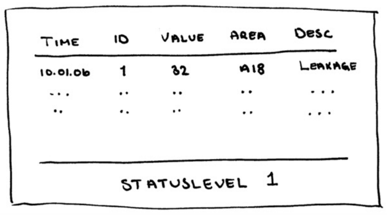
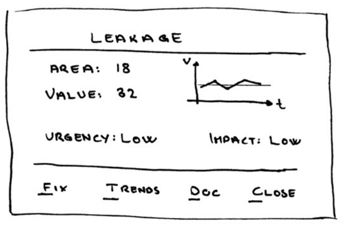
Type	Contents		Sketch
(1) New alarm (on new alarm)	Message	• Name • Urgency • Impact	
	Commands	• Goto • Suppress/ Confirm • Documentation • Trends	
(2) All alarms (always)	Message	• Time • ID • Value • Area • Description • Statuslevel	
	Commands	• n/a	
(3) Alarm detail (on user input)	Message	• Name • Urgency • Impact • Area • Value	
	Commands	• Fix • Trends • Documentation • Hide information	

Table 8.5: Details regarding each type of textual information displayed in the HUD.

selected in accordance with experiences from the alarm-management system used at Tjeldbergodden (appendix B).

Alarm detail

The detailed description of an alarm are activated on a user request. Its intention is to assist the user in troubleshooting, and consequently it should give as much information about the alarm as possible. For instance, using graphs could be a useful way of illustrating the alarm-object's measuring value. However, this kind of functionality is beyond the scope of this master thesis and is therefor not included.

8.5.2 Virtual objects

The virtual objects are used to identify the origin of alarms within the virtual environment. This is solved by highlighting the objects which have an alarm attached to it. The highlighting is accomplished by gradually altering the object's color to red, and then back to its original color. The frequency of alteration is twice each second. Since the object may be hidden behind or inside other objects, the remaining structures are made semitransparent. An example showing an opaque object within a semitransparent template is illustrated in figure 8.13.

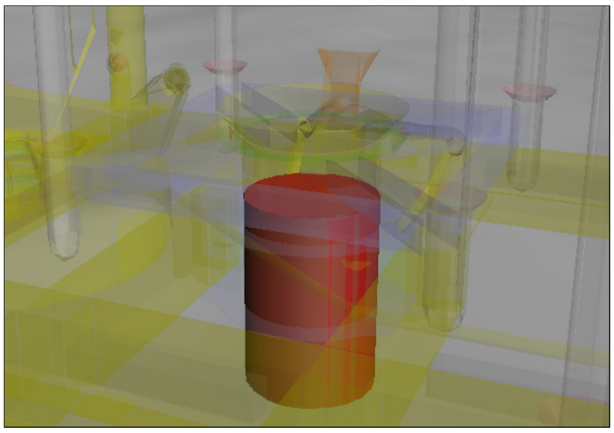


Figure 8.13: An example of an highlighted object located within a semi-transparent template.

8.6 Interaction

In order to be effective, the tool must be easy to operate. In this prototype this is accomplished by means of keyboard commands. While some commands will be universal, meaning that they work regardless of state, others will only be available at certain situations. An example of this is the commands made available at the occurrence of a new alarm. If a non-critical alarm occurs, for instance a notification, it should be possible to confirm or suppress the alarm without any further actions. However, if the alarm is a safety-critical alarm, suppression should not be available.
As already mentioned, most functionality in the prototype is controlled by the keyboard. In cases where the commands are made available by an event, like the showing of an information-box, the commands are indicated by placing brackets around the letter used to control the command. An example of this is *[C]onfirm*, where C is the key used to confirm.
A list of all available commands are listed in table 8.6. Figure 8.14 shows the same information in a tree-structure.

In addition to the commands mentioned above, the prototype also offers all interaction possibilities made available by the SnøhvitSIM framework. A description of these is found in the SnøhvitSIM user guide.

Command	Functionality	Type / activating event
# *	Goto alarm with id #	Universal
Shift + # *	Turn on beacon on alarm with id #	Universal
Ctrl + # *	Turn off beacon on alarm with id #	Universal
G	Goto alarm, moves camera to alarm	On showing of **New alarm**
S	Suppress, mark the alarm as fixed	On showing of **New alarm,** if the alarm is of type production-alarm
C	Confirm, mark the alarm as fixed	On showing of **New alarm**, if the alarm is of type notification
D	Show documentation of alarm-object	On showing of **New alarm**
T	Show trends of alarm-object	On showing of **New alarm**
F	Fix the alarm	On showing of **Alarm detail**
T	Show documentation of alarm-object	On showing of **Alarm detail**
D	Show trends of alarm-object	On showing of **Alarm detail**
C	Close the information-box	On showing of **Alarm detail**
Mouse click on object	Shows **Alarm Detail** of object	Object is highlighted, i.e. there is an alarm on the object

* # refers to the ID of an alarm

▨ Not implemented due to limitations of scope

Table 8.6: List of all available commands in prototype.

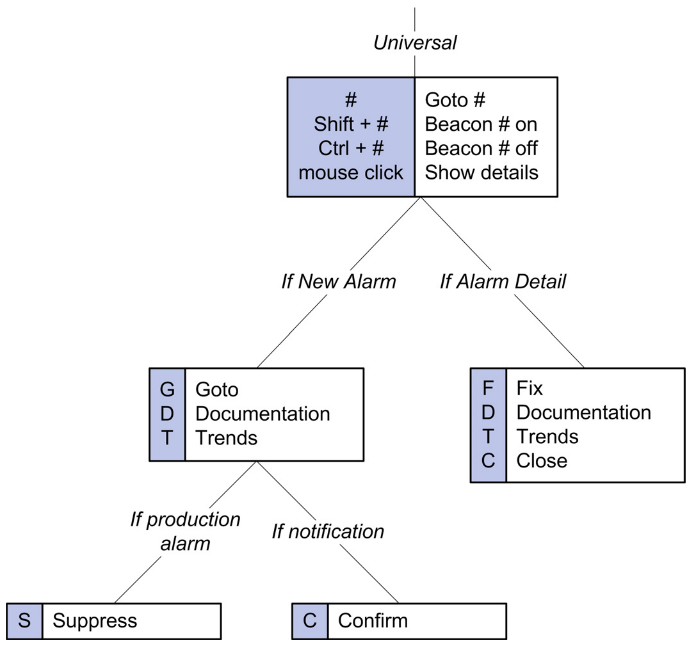


Figure 8.14: Tree-structure presentation of all commands in prototype.

Chapter 9

Implementation

The prototype has been implemented using the design described in chapter 8. According to the the priorities listed in section 6.3, it has first and foremost been important to make the prototype work. Due to limitations in time, this prioritizing has resulted in less effort invested in creating a neat and structured code. It is also worth mentioning that parts of the functions have been hardcoded[1]. Even though this works fine in the prototype, such an approach would not be possible in a system using real-time data.

This chapter will describe details from the implementation of the prototype, AMEX.

9.1 Development environment

AMEX has been developed on a Dell Latitude X200 laptop, using several different tools. The code is mainly written in C++, using Microsoft Visual Studio .NET as an editor. Compilation and debugging has been done in Cygwin, a Linux-like environment for Microsoft Windows. This is due to the design and limitations of the SnøhvitSIM framework. All virtual objects are generated in 3D studio MAX 6, and exported to AMEX using the VRML file format. As already mentioned in chapter 8, the cSound programming language has been widely used when generating sounds and auditory icons. While some of the sounds are fully generated using this language, some has also used prerecorded samples as input.
AMEX has been tested both on the laptop and in a visualization-room at Statoil's facilities at Rotvoll. The visualization-room is equipped with more appropriate hardware for this type of applications, including a $2 * 4$ meter visual display and the sound-system suggested in section 8.3.

[1]data or values inserted directly into the code/program, where it cannot be easily modified

9.2 Implementation of requirements

Table 9.1 and 9.2 lists the requirements defined in section 6.2 with additional comments on how they were implemented in AMEX.

Nr	Requirement[*] [**]	Implemented	Comment
1(a)	Inform the user	yes	Informed through audible and visual means
1(b)	Identify the virtual object which caused the alarm	yes	Each alarm is attached to a virtual object
1(c)	Be able to move the camera to the placing of the alarm/object	yes	Using goto-functionality or manual by mouse
1(d)	Highlight the object	yes	The object gets an altering color
1(e)	Attach an audible signal to the object representing the alarms' type and severity	yes	The auditory icon represent both the alarm's system and criticality
1(f)	Provide audible information which can be used to localize the alarm	yes	Each alarm has a beacon sound attached to it
1(g)	Provide the user with relevant information about the object	yes	An information-box is available for all objects with alarms, showing static information. A dynamic presentation is not implemented due to limitations in time.
2(a)	Visual representation	yes	Textual representation in the HUD
2(b)	Audible representation	yes	A dynamic sound played regularly which represents the state of the facility

* *Initial text for requirement 1(x): "The prototype must detect occurring alarms and:"*

** *Initial text for requirement 2(x): "The prototype must provide information about the overall status of the facility, including:"*

Table 9.1: Implementation of functional requirements

Nr	Requirement	Implemented	Comment
1	The prototype must be implemented in the SnøhvitSIM framework	yes	AMEX is an extension to SnøhvitSIM
2	The prototype must have an user-friendly and intuitive interface	yes	According to the author's judgment

Table 9.2: Implementation of non-functional requirements

9.3 Overview

Figure 9.1 shows an overview of AMEX. To keep the figure on a reasonable level of detail, only a subset of the functionality is included. As the legend explains, the colored circles represents different components, their functions and methods.

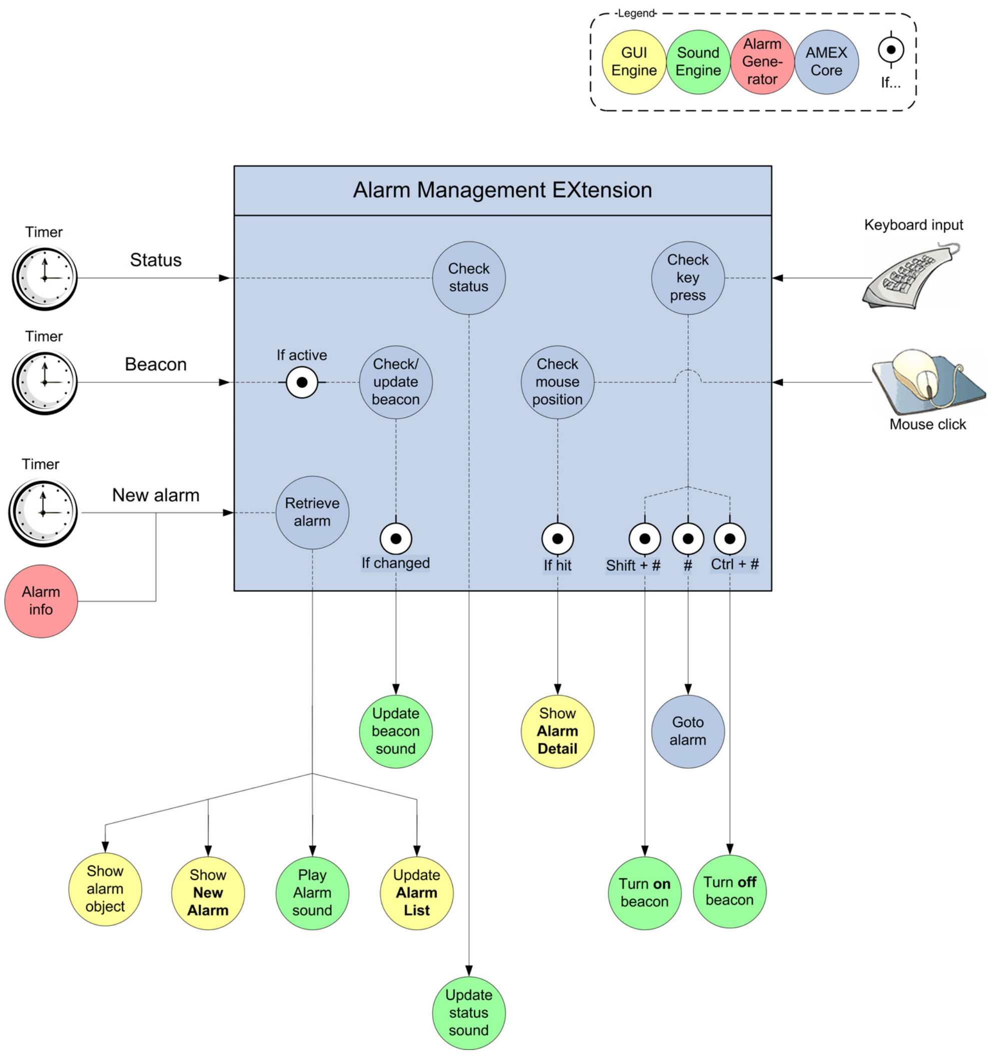


Figure 9.1: Subset of functionality and dataflow in AMEX, illustrating the relation between the four components; AMEX Core, GUI Engine, Sound Engine and Alarm Generator.

As figure 9.1 shows, the initiating events of AMEX are either timers or user input (mouse or keyboard). Two of the timers are used to check and update the status- and beacon sound. The third timer is used to trigger a

new alarm. This timer would be removed if real-time data from the facility was available.
The mouse- and keyboard input is interpreted by the AMEX Core. After checking the input, the commands are forwarded to the different components responsible for the actual event. For instance, if Shift + 3 is pressed, the AMEX Core tells the Sound Engine to turn on the beacon related to alarm with id = 3.

A detailed description of the essential functionality is given in the following sections. Parts of the source code will be used when necessary in the explanations. For the interested reader, the entire source code is included in appendix G. It is worth mentioning that the AMEX Core initializes the other components as objects, naming them according to their initials: Sound Engine = se, GUI Engine = ge and Alarm Generator = ag. The same denotation will be used in the remaining of this chapter.

9.4 Alarms

As have been described in chapter 8, there are four pre-defined alarms in AMEX. The alarms are created by ag at a random order every 60th second. The creation of each alarm is initiated by a timer, as illustrated in figure 9.1. Each alarm is represented by a C++ -struct. The struct contains values like id, criticality, value, description, position etc. The struct is shown in example 4.

Example 4 Struct of an alarm

```
struct alarmstruct{
   SbTime * timestamp;     // timestamp of alarm
   int id;                 // alarm id
   int value;              // static value
   int area;               // area
   int criticality;        // static criticality
   char * description;     // textual description
   char * description2;    // textual description
   char * impact_urgency;  // textual description
   float position[3];      // position of alarmobject
   double utm[3];          // position of beacon
   float location[3];      // position for Goto()
   float rotation[3];      // rotation vector for Goto()
   float radians;          // amount rotation for Goto()
};
```

9.4.1 Retrieving alarms

The core::retrieveAlarm() function showed in figure 9.1 is initiated when a new alarm occurs. The function forwards information about the new alarm to ge and se which updates the audible and visual information. This includes highlighting the alarm's object, updating the alarm list and the overall status and playing the alarm sound.

9.4.2 Alarm visualization

Alarms are visualized both through virtual objects and by textual presentations in the HUD. Both types are constructed by ge at start-up, and added to the scenegraph. To control whether or not these nodes are visible to the user, they are placed under so-called SoSwitch-nodes. These nodes can be switched on and off, showing and hiding the underlying objects. At start-up, only the alarm list is visible to the user.

The four different virtual objects, one for each alarm, are displayed whenever their belonging alarm is active. The objects are extracted from the template used in the simulation, and modified to increase the performance of AMEX. The VRML-code representing the object is then manually added an interpolator which changes the color in runtime. The interpolator uses RGB-values to gradually change the color from one to another. In AMEX, the color changes from the object's original color to red and back, once each second. An example of such a virtual object is seen in example 5.

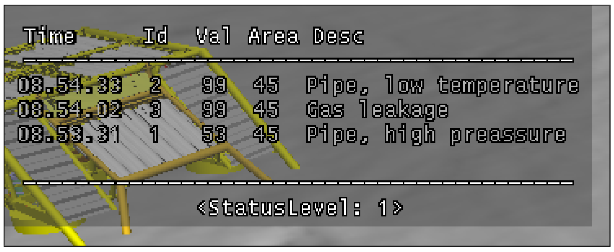
The HUD, which is used to show 2D textual information, is located within the SnøhvitSIM framework. AMEX utilizes this by defining several inventor-files (similar to VRML) and add these to the HUD. Each inventor-file represents an information-box, and consists of a semitransparent background in addition to the textual information. The content is updated by different update-functions provided by ge, one for each of the information boxes. This procedure can be described by the following steps.

1. An update is requested from core
2. ge stores the update information in a local string-array, where each element in the array represents a line in the information-box
3. ge searches the scenegraph, and locates the proper information-box
4. ge passes the entire string-array to the node describing the information-box

An illustration of an information-box is depicted in figure 9.2. Even though this implementation does not provide a high quality graphical layout, it is a very flexible solution suitable for prototyping. In a real implementation, a different approach should be considered.

Example 5 Example of virtual object

```
#VRML V2.0 utf8
#Object definition
DEF AlarmObject1 Transform {
   translation 4.871 -1.979 -2.871
   children [
      Shape {
         appearance Appearance {
            material DEF changeableColor Material {
               diffuseColor 0.42 0.75 0.46
            }
         }
         geometry DEF AlarmObject1_object Cylinder {
            radius 0.5 height 2.83
         }
      }
   ]
}
#Timer
DEF TIME TimeSensor {
   cycleInterval  2
   startTime      0
   stopTime      -1
   loop           TRUE
}
#Color changer
DEF colorChanger ColorInterpolator {
   key [0, .25, .5, .75, 1]
   keyValue [0.42 0.75 0.46, 1.0 0.1 0.1,
             0.42 0.75 0.46, 1.0 0.1 0.1, 0.42 0.75 0.46]
}
#Commands
ROUTE TIME.fraction_changed TO colorChanger.set_fraction
ROUTE colorChanger.value_changed TO
      changeableColor.set_diffuseColor
```



```
Time        Id   Val  Area Desc
-----------------------------------------------
08.54.33    2    99   45   Pipe, low temperature
08.54.02    3    99   45   Gas leakage
08.53.31    1    53   45   Pipe, high preassure
-----------------------------------------------
            <StatusLevel: 1>
```

Figure 9.2: Information-box with semitransparent background placed in the HUD.

9.4.3 Alarm sounds

Similar to the visual elements, all sound-nodes are created at start-up and added to SoSwitch-nodes. The different alarm-sounds are placed at the same coordinates as the related alarm-object, and are played once when the alarm occurs. The sound node geometry is defined by two ellipsoids (see figure 3.15), where the outer ellipsoid depends on the users distance d_u to the alarm. The radius r_o of the outer ellipsoid, which in this case is a circle, is set according to (9.1).

$$r_o = \begin{cases} 75 & \text{if } d_u < 75 \\ d_u + 20 & \text{if } d_u > 55 \end{cases} \tag{9.1}$$

As can be seen from (9.1), r_o will be constant whenever the user is within 55 meters of the alarm. If the user is farther away, r_o is set to $d_u + 20$. The addition of 20 is performed to reduce the chance of the sound being silenced if the user moves away from the alarm. The inner circle's radius r_i is set to 8.0, meaning that the sound's intensity is constant when the user comes within 8 meters of the object. Example 6 shows how this is implemented, including switching on the parent SoSwitch-node.

Example 6 Updating sound node geometry

```
//Setting sound node geometry
soundVRMLSound0->maxFront = std::max(75.0f, distance + 20.0f);
soundVRMLSound0->maxBack = std::max(75.0f, distance + 20.0f);
soundVRMLSound0->minFront = 8.0f;
soundVRMLSound0->minBack = 8.0f;

//Switching on
soundSwitch0->whichChild.setValue(SO_SWITCH_ALL);
```

9.5 Beacon sounds

Whenever a beacon is activated, core::updateBeacon() is responsible for updating the beacon's pitch and sound node geometry according to the users orientation and position. This is controlled by a timer which calls the function every 0.2 second. The pitch is altered in an inverse ratio with the angle (θ) between the two vectors defined by the user's direction (u) and the direction to the alarm object (v) . θ is defined by the dot product (also called scalar product) of u and v:

$$\cos(\theta) = \frac{v \cdot u}{|v||u|} \tag{9.2}$$

Since both u and v are normalized, the denominator in (9.2) is eliminated. In AMEX this is implemented as depicted in example 7.

Example 7 Calculate dot product and update beacon

```
float dotProd = lookatVector.dot(directVector);
se.updateBeaconPitch(BEACON_ACTIVE, acos(dotProd), distance);
```

As illustrated in example 7, the actual update is performed by se::updateBeaconPitch(). Since the pitch only is defined in [1.0, 2.0], a transformation is needed on the parameter sent from core::updateBeacon(). The implemented transformation place emphasis on the lower values, resulting in an increased pitch for low values of θ. The transformation and update of sound clip are described in example 8.

Example 8 Transform and update pitch on soundclip

```
float pitchModified = ((pitch/3.14f) * -1.0f) + 2.0f;
beaconClipX->pitch = pitchModified;
```

As can be seen from example 7, an additional parameter called distance is sent to se::updateBeaconPitch(). This is used to control the sound node geometry of the beacon. Referring to figure 3.15, the maxBack-value (m_b) and maxFront-value (m_f) of the beacon is set according to (9.3).

$$m_b = m_f = \begin{cases} 1.1 * 75 & \text{if } d_u < 75 \\ 1.1 * d_u & \text{if } d_u > 75 \end{cases} \tag{9.3}$$

d_u refers to the user's distance to the beacon (i.e. the alarm). In other words, if the user moves more than 75 meters away from the alarm, the size of the outer ellipsoid is increased. The user will consequently always hear the beacon. The reason for multiplying the distance with 1.1, is to avoid that the sound disappears due to the update-frequency when the user moves away.

9.6 Status sounds

Opposite to the other sounds, the status sounds have no specific location within the virtual modal, and are therefore not spatialized. Hence, they are reproduced using normal stereo without any directional effects applied. The intensity of the sound is constant within 1 kilometer from the template.
As described in section 8.4.3, the status sounds are played at different intervals t, controlled by the current status level s.

$$t = \begin{cases} 4 * 150 \text{ sec.} = 10 \text{ min.} & \text{if } s = 0 \\ 2 * 150 \text{ sec.} = 5 \text{ min.} & \text{if } s = 1 \\ 1 * 150 \text{ sec.} = 2.5 \text{ min.} & \text{if } s = 2 \end{cases} \tag{9.4}$$

Equation (9.4) can be described by the exponential function (9.5), which are used in the implementation.

$$t = 150(\frac{1}{2}s^2 - \frac{5}{2}s + 4) \qquad \text{for } s \in [0, 2] \tag{9.5}$$

9.7 User interaction

The user can interact with the model either through the mouse or the keyboard. This is accomplished by implementing a so-called *event-callback*, and adding this to the SnøhvitSIM framework. The event-callback is then called whenever a new event occurs, such as a mouse-click. The event-callback used in AMEX, `core::amex_event_cb()`, handles both keyboard-events and mouse-events. The keyboard-events are handled rather easily, as illustrated in example 9.

Example 9 Handling a keyboard-event

```
if (event->isOfType(SoKeyboardEvent::getClassTypeId())) {

   //Casting event to keyboard-event
   keyboardEvent = (SoKeyboardEvent*) event;
   //Key G: Goto alarm
   if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                          SoKeyboardEvent::G)) {
      ...
   }
}
```

The mouse-events are used to display the details about an alarm (see figure 9.1). This is done by clicking on a highlighted object. Since Coin3D does not provide such functionality, a new routine has been developed. The routine consists of the following steps:

1. Register when the left mouse-button is pressed
   - Save the position p_1 of the mouse
2. Register when the left mouse-button is released
   - Save the position p_2 of the mouse
3. Check if the mouse has been moved ($|p_1 - p_2| < limit$)

- If yes, show details about the alarm if there is a highlighted object projected on the screen at p_2
- If no, abort

Checking whether a highlighted object is projected at p_2 is a non-trivial task. One solution is to send an imaginary ray along the z-axis from p_2 and check which object that first intersects its path. This is supported by Coin3D using a so-called SoRayPickAction-node. The node, which are used in AMEX, returns the scenegraph-path to the first object it intersects. Searching this path gives a potential match with one of the active alarm-objects.
As already stated, evaluating mouse-interaction in a three-dimensional environment is not trivial. An excerpt from the source code illustrates this in example 10.

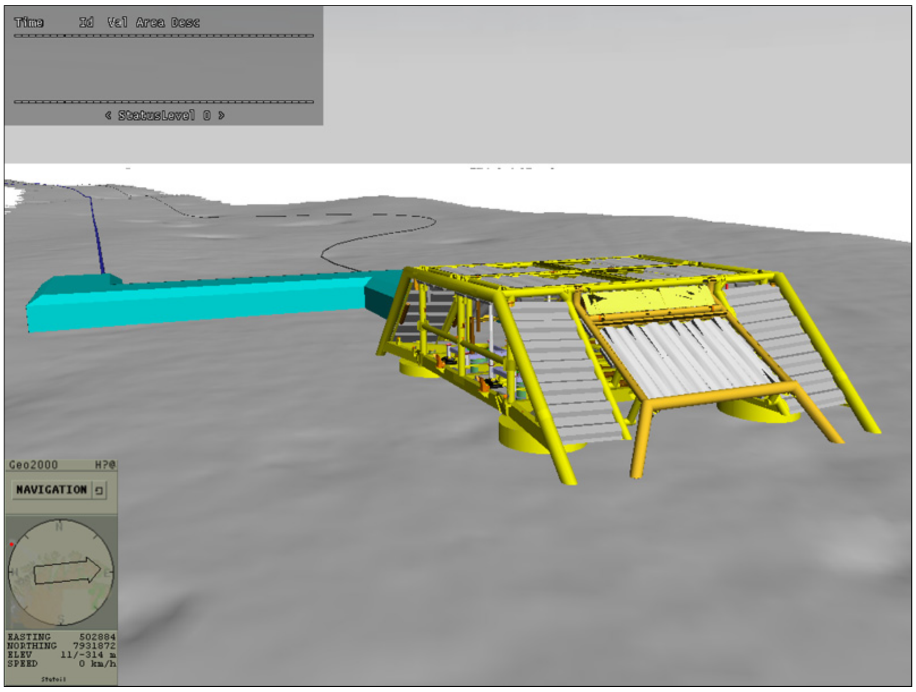
Example 10 Handling a mouse-event

```
//Check mouse button press
if(SoMouseButtonEvent::isButtonPressEvent (
   mouseButtonEvent, SoMouseButtonEvent::BUTTON1)) {
   eventPositionDown = event->getPosition();
//Check mouse button release
} else if(SoMouseButtonEvent::isButtonReleaseEvent (
   mouseButtonEvent, SoMouseButtonEvent::BUTTON1)) {
   eventPositionUp = event->getPosition();

   //Check if mouse is moving
   if ((abs(eventPositionDown[0] - eventPositionUp[0]) < 3) &&
       (abs(eventPositionDown[1] - eventPositionUp[1]) < 3)) {
      //Ray picking
      SoRayPickAction rayPicker(currviewer->getViewportRegion());
      rayPicker.setPoint(event->getPosition());
      rayPicker.apply(currviewer->getSceneGraph());
      //Finding object
      SoFullPath * pathToPickedObject;
      const SoPickedPoint *pickedPoint=rayPicker.getPickedPoint();
      if (pickedPoint != NULL) {
         pathToPickedObject=(SoFullPath*) pickedPoint->getPath();
         SoNode * node = pathToPickedObject->getTail();
         //Check if object is an alarm object
         if (node->getName() == SbName("amex_alarm_object_0")) {
            ...
         }
      }
   }
}
```

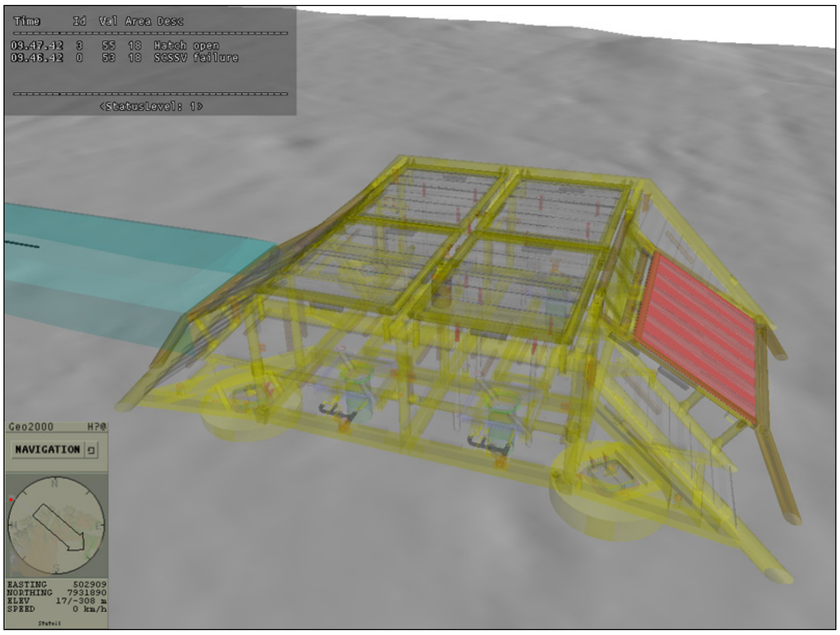
9.8 Final solution

Screenshots of the final solution of AMEX is depicted in figure 9.3, figure 9.4 and figure 9.5.



```
EASTING      502884
NORTHING    7931872
ELEV      11/-314 m
SPEED        0 km/h
```

Figure 9.3: Screenshot of AMEX, initial view.

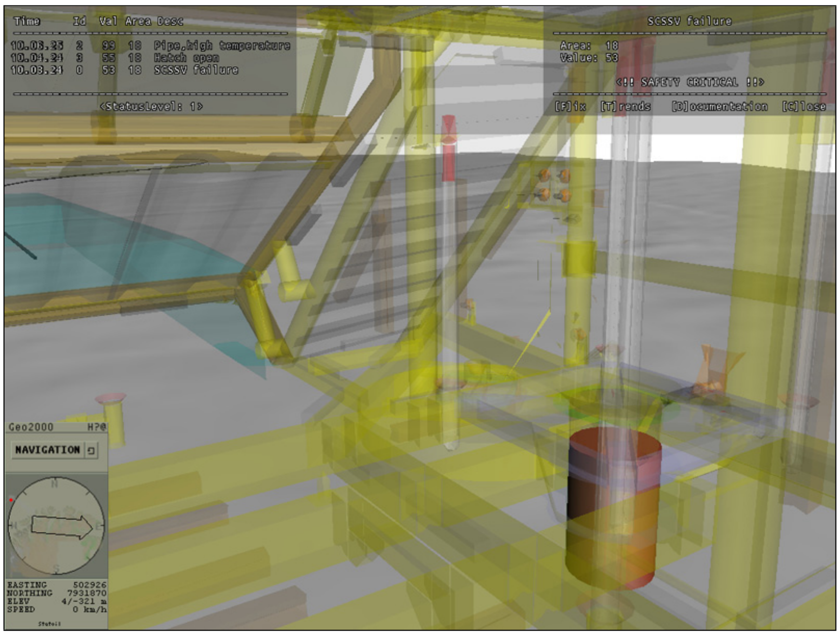


```
Time      Id  Val Area Desc
------------------------------------------
09.47.42  3   55   18  Hatch open
09.46.42  0   53   18  SCSSV failure

------------------------------------------
             <StatusLevel: 1>
```

```
EASTING      502909
NORTHING    7931890
ELEV     17/-308 m
SPEED       0 km/h
```

Figure 9.4: Screenshot of AMEX, overview of template.



```
Time      Id  Val Area Desc
------------------------------------------
10.06.25  2   99   18  Pipe,high temperature
10.04.24  3   55   18  Hatch open
10.03.24  0   53   18  SCSSV failure
------------------------------------------
             <StatusLevel: 1>
```

```
                 SCSSV failure
------------------------------------------
Area:   18
Value:  53
           <!! SAFETY CRITICAL !!>
------------------------------------------
[F]ix  [T]rends   [D]ocumentation   [C]lose
```

```
EASTING      502926
NORTHING    7931870
ELEV      4/-321 m
SPEED       0 km/h
```

Figure 9.5: Screenshot of AMEX, close-up of one alarm.

Chapter 10

Evaluation

This chapter evaluates the strengths and weaknesses of AMEX and its fundamental concept. The evaluation is mainly based on discussions with Statoil employees Vidar Hepsø (Hepsø, 20.06.2005, pers. comm.) and Kjell Bjerkeli (Bjerkeli, 21.06.05, pers. comm.). While Hepsø is a social anthropologist, working in the process control division at Statoil Research Center, Bjerkeli has 15 years experience as a management technician. He is now working as an instructor at Statoil's control-room simulator, which is used to train operators. The selection of subjects is deliberate to evoke as many issues and strengths as possible.
The evaluation is structured in three sections; (1) Scope, (2) strengths and (3) weaknesses. (1) evaluates whether or not AMEX, and its underlying concept, is a proper platform for further development towards an operative alarm management system. The different components developed in this master thesis, seen in a separate view, is commented in (2) and (3). The auditory display is an example of such. Detailed discussions about the chosen design are given in chapter 8, and are generally not repeated here.

10.1 Scope

Both Hepsø and Bjerkeli questioned the possibilities of AMEX as a replacement of today's alarm management systems. The idea of using a virtual model to control a system, may be difficult due to its mode of presentation. While alarm management systems normally are designed as decision support systems, virtual models are more suitable for exploration. Being able to navigate through a facility is a great feature, but does not necessarily give the needed overview. Despite being able to move quickly from the location of one alarm to another, the troubleshooting in AMEX may be very ineffective. It is analogous to giving the overall alarm management responsibility to the field operators, and let them hurry around in the facility to find the cause of an alarm. Having said that, similar solutions to AMEX could still give a valuable contribution to today's systems. This could for instance be in relation with the safety-critical alarms, such as fire or gas leakage. In these situations, it is important for the operator to know as much as possible about the area surrounding the alarm. This could for example be knowledge about nearby personnel, hazardous areas and fresh air supplies. Using AMEX, or similar systems, this information could be provided very efficient and accurate. Such visual and audible communication could also be beneficial where non-experts are involved in troubleshooting or maintenance of the facility. Hepsø mentions several areas where this could be useful: Vendor-communication, trend analysis, training of personnel and virtual surveys or status-reports of the facility. The latter could be solved by aggregating alarm-information to a higher level and altering e.g. a template's color according to its overall state. Such information would be useful for the operative and administrative management, onshore support centers (OSC), and at briefings during shift changeovers.

10.2 Strengths

The auditory display developed in this master thesis is according to the feedback, the biggest strength related to a control-room setting. Using both audible and visual aids can provide richer and more effective information than visual presentations alone. Bjerkeli mentions the design of the alarm-sounds as very interesting. Even though more studies and adjustments are needed, it shows a potential feature which relatively easily could be implemented in today's alarm management systems. Also, the overall status sound could be an important contribution for future systems. Even though most operators today are well informed about the overall state of the facility, the development goes towards fewer and more mobile personnel. Providing such information to field operator could help him to get a better understanding of the current situation.

10.3 Weaknesses

As already mentioned, several weaknesses are identified in the fundamental idea behind AMEX. This relates especially to the possibilities of using this type of tool for decision support in a control-room. Bjerkli also remarks the limited human ability of remembering and separating different sounds. As an alternative to the implemented design in AMEX, he suggests a coarser subdivision of production-alarms. Dividing the alarms in *hydrocarbon-bearing systems*, *utilities* and *fire/gas-systems*, could be an alternative approach.
The biggest challenge or weakness, though, is not related to AMEX or its components directly. Both Hepsø and Bjerkeli identifies the huge amount of alarms presented to the operators as a serious problem in today's control-

rooms. Bjerkeli confirms that at certain critical points, several hundred, maybe even thousand, alarms are trigged. At this magnitude, no system can provide a perspicuous image to the operators. Developing a system or a filter which can reduce the amount of alarms reaching the control-room, should clearly be brought into focus by the industry.
One should also consider how a system, based on new types of communication, can be implemented in the organization. Many operators are used to their standard routines and procedures, and changing these are not something which should be based on enforcement. This is especially important, given the essential role of the operators.

Part IV

Conclusion

Chapter 11

Conclusion and further work

In this master thesis, a prototype has successfully been developed according to the objectives of the study. It demonstrates feasible techniques to improve an operator's overview of a facility, by means of a multi-modal virtual environment. This includes improved operator awareness and immediate mediation of the whereabouts and severity of new alarms. Experience gained throughout the study indicates a need and craving for new and effective alarm management systems. This prototype does not solve this challenge, but it is a showcase for future development, providing potentially very effective solutions for tomorrow's alarm management system.

11.1 Contribution

The development of AMEX includes an auditory display which provides (1) spatial information about new alarms and (2) information about the overall state of the facility. The system also offers (3) beacons to help navigating within the virtual environment. The modes of operation and the design of the auditory icons used in the auditory display, has also been a part of the contribution. Together with the visual presentation and the suggested hardware, AMEX illustrates possibilities for further development of alarm management systems.

11.2 Future work

The author has great believes in further development of the concepts presented in this master thesis. However, more studies are needed to ensure usability, and a proper scope of the system. This goes for both the visual and audible design, and in particular the auditory display. Extensive testing and inclusion of experienced personnel in the design-phase will be essential. A deeper study of the domain is also required to make sure all constraints are allowed for, including an evaluation of the magnitude and characteristics of the alarms. This evaluation could further be used as a basis for developing an alarm-filter which reduces the amount of alarms that are presented to the operator. Even though it goes beyond the scope of this master thesis, it has been identified as a requirement in case of an operative implementation.

Bibliography

Adcock, M. & Barrass, S. (2004). Cultivating design patterns for auditory displays. *in* 'Proceedings of ICAD 04-Tenth Meeting of the International Conference on Auditory Display'. URL visited 23.04.05: `http://www.icad.org/websiteV2.0/Conferences/ICAD2004/posters/adcock_barrass.pdf`.

Albertalli, B. (2003). An intuitive interface for mixing multi-track audio in three dimensions. Master's thesis. University of Miami.

Begault, D. R. (2000). 3-d sound for virtual reality and multimedia. Technical report. Ames Research Center, NASA. Moffett Field, California. URL visited 24.03.05: `http://human-factors.arc.nasa.gov/ihh/spatial/papers/pdfs_db/Begault_2000_3d_Sound_Multimedia.pdf`.

Bishton, D. (2003). Human computer interaction. *in* 'Lecture notes in Multimedia Systems'. URL visited 01.02.05: `http://www.soc.staffs.ac.uk/cah1/immcit/usability.ppt`.

Blattner, M., Sumikawa, D. & Greenberg, R. (1989). Earcons and icons: Their structure and common design principles. *in* 'Human-Computer Interaction'. Vol. 4. pp. 11–44.

Blauert, J. (1974). *Räumliches Hören. Also available as translation by Allen, John S. (1983) Spatial Hearing.* S. Hirzel Verlag.

Boulanger, D. R. (2005). *Official Csound Website.* cSounds. URL visited 10.03.05: `http://www.csounds.com`.

Catchpole, K. R., McKeown, J. D. & Withington, D. J. (2004). Localizable auditory warning pulses. *in* 'Ergonomics'. Vol. 47. pp. 748–771.

Cohen, J. (1994). Monitoring background activities. *in* G. Kramer, ed., 'Auditory Display: Sonification, Audification, and Auditory Interfaces'. Addison-Wesley, Reading, MA.

Creative website (2005). URL visited 01.06.05: `http://www.creative.com`.

Endsley, M. (1994). Situation awareness in dynamic human decision making: Theory. *in* R. Gilson, D. Garland & J. Koonce, eds, 'Situation Awareness in Complex Systems'. Embry-Riddle Aeronautical University Press,. pp. 27–58.

Gardner, W. G. (1999). 3d audio and acoustic environment modeling. *Electronic Music and Audio Guide.* URL visited 04.06.05: `http://www.sonicspot.com/guide/3daudio.html`.

Gaver, W. (1986). Auditory icons: Using sound in computer interfaces. *in* 'Human-Computer Interaction'. Vol. 2. pp. 167–177.

Gaver, W. W. (1989). The sonicfinder: An interface that uses auditory icons. *in* 'Human-Computer Interaction'. Vol. 4. pp. 67–94.

Gaver, W. W., Smith, R. B. & O'Shea, T. (1991). Effective sounds in complex systems: The arkola simulation. *in* 'CHI'91'.

Graham, R. (1999). Use of auditory icons as emergency warnings: evaluation within a vehicle collision avoidance application. *in* 'Ergonomics'. Vol. 42. pp. 1233–1248.

Gröhn, M., Lokki, T. & Takala, T. (2003). Comparison of auditory, visual, and audio-visual navigation in a 3d space. *in* 'Proceedings of the 2003 International Conference on Auditory Display'. URL visited 26.05.05: `http://www.icad.org/websiteV2.0/Conferences/ICAD2003/paper/49%20Grohn.pdf`.

Guillaume, A., Drake, C., Rivenez, M., Pellieux, L. & Chastres, V. (2002). Perception of urgency and alarm design. *in* 'Proc of the 8th Int. Conf. on Auditory Display'. Int. Community on Auditory Display. URL visited 31.05.05: `http://www.icad.org/websiteV2.0/Conferences/ICAD2002/proceedings/04_AGuillaume.pdf`.

Hermann, T., Drees, J. M. & Ritter, H. (2003). Broadcasting auditory weather reports - a pilot project. *in* E. Brazil & B. Shinn-Cunningham, eds, 'Proc. of the 9th Int. Conf. on Auditory Display'. Int. Community on Auditory Display. Boston University Publications. Boston, MA, USA. pp. 208–211. URL visited 26.05.05: `http://www.techfak.uni-bielefeld.de/ags/ni/publications/media/HermannDreesRitter2003-BAW.pdf`.

Kazem, M. L. N., Noyes, J. M. & Lieven, N. J. (2003). Design considerations for a background auditory display to aid pilot situation awareness. *in* 'Proceedings of the 2003 International Conference on Auditory Display'. URL visited 26.05.05: `http://www.icad.org/websiteV2.0/Conferences/ICAD2003/paper/22%20Kazem.pdf`.

Kendall, G. (1995). A 3-d sound primer: Directional hearing and stereo reproduction. *Computer Music Journal* pp. 23–46. URL visited 12.02.05: `http://music.northwestern.edu/classes/3D/pages/sndPrmGK.html`.

Kramer, G. (1994). Some organizing principles for representing data with sound in auditory displays. *in* 'Studies in the Science of Complexity'. Santa Fe Institute. Addison-Weasley. pp. 185–222.

LGPL (2005). URL visited 01.06.05: `http://www.gnu.org/copyleft/lesser.html`.

LoBrutto, V. (1994). *Sound-on-film: interviews with creators of film sound.* Praeger Paperback.

Loki website: OpenAL Specification and Reference (2005). URL visited 05.03.05: `http://www.openal.org/oalspecs-specs/`.

Lukka, T. (1999). An introduction to vrml. *Linux Journal.* URL visited 05.06.05: `http://www.linuxjournal.com/article/3085`.

Montan, N. (2002). An audio augmented reality system. Master's thesis. KTH, Royal Institute of Technology, Stockholm, Department of Microelectronics and Information Technology. URL visited 12.02.05: `http://vvv.it.kth.se/docs/Reports/DEGREE-PROJECT-REPORTS/030409-Nils-Montan.pdf`.

Morgan, C. R. (1998). Circumfusion: A composition for real-time computer music spatialization system. Technical report. Collin County Community College. URL visited 20.05.05: `http://iws.ccccd.edu/cmorgan/diffuser/`.

Nimmo, I. (2005*a*). Alarm management and graphics projects. Technical report. User Centered Design Services, LLC.

Nimmo, I. (2005*b*). Operator situation awareness and the impact on control room design. Technical report. User Centered Design Services, LLC.

Nimmo, I. (2005*c*). *User Centered Design Services, LLC Website.* User Centered Design Services, LLC. URL visited 10.05.05: `http://www.mycontrolroom.com`.

NOU (2000). Åsta-ulykken, 4. januar 2000. Technical report. Investigating Commission of the Åsta accident.

Rudi, J. (2000). *Kort innføring i lydens fysikk og akustikk, pskoakustikk, digital innspilling og lydbehandling.* Institutt for musikk og teater, UiO. URL visited 18.02.05: `http://www.notam02.no/~joranru/InnforingLyd/`.

Sensaura website (2005). URL visited 01.06.05: `http://www.sensaura.com`.

Shaw, M. (2003). Writing good software engineering research papers. *in* 'Proc of the 25th Int. Conf. on Software Engineering'. IEEE Computer Society. pp. 726–736.

SIM (2005*a*). *Coin3D developer page.* URL visited 25.02.05: `http://www.coin3d.org`.

SIM (2005*b*). *SIM Scenery.* URL visited 22.03.05: `http://www.sim.no/products/Scenery/`.

Sim, S. E. (2004). Research methodology for software. *in* 'Seminar in Information and Computer Science'. URL visited 04.02.05: `http://www.ics.uci.edu/~ses/teaching/ics280/`.

Sjøvoll, H. (2004). 3d sound as a psychoacoustic aid in virtual models.

Strutt, J. L. R. (1907). On our perception of sound direction. *Phil. Mag.* **13**, 214–232.

Svensson, P. (2004). 3d sound and sound in multimedia applications. *in* 'Lecture notes in TT1, Fall 2004'. URL visited 31.01.05: `http://www.tele.ntnu.no/akustikk/fag/ttt1/kursmateriell`.

Thomassen, C. A. (2005). Use of augmented reality in geosimulators. Master's thesis. Norwegian University of Science and Technology, Department of Computer and Information Science.

Tran, T., Letowski, T. & KS, A. (2000). *Evaluation of acoustic beacon characteristics for navigation tasks.* Vol. 43. Ergonomics. pp. 807–27.

Vickers, P. (1999). CAITLIN: Implementation of a Musical Program Auralisation System to Study the Effects on Debugging Tasks as Performed by Novice Pascal Programmers. PhD thesis. Loughborough University, Loughborough.

VRML97, ISO/IEC 14772-1 (1997). URL visited 01.06.05: `http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-IS-VRML97WithAmendment1/`.

Walker, B. N. & Lindsay, J. (2003). Effect of beacon sounds on navigation performance in a virtual reality environment. *in* 'Proceedings of the 2003 International Conference on Auditory Display'. URL visited 26.05.05: `http://www.icad.org/websiteV2.0/Conferences/ICAD2003/paper/50%20Walker2%20-%20navigation.pdf`.

Wallach, H., Newman, E. B. & Rosenzweig, M. R. (1949). The precedence effect in sound localization. *The American Journal of Psychology* **62**, 315–36.

Part V

Appendix

Appendix A

Workshop: VR, Halden

This report gives a summary of the different topics presented at the 2005 VR Workshop 02.-03.03.05.

A.1 Summary

The workshop gave a broad presentation on different aspects regarding the industrial usage of virtual and augmented reality. Some keywords concludes the workshop:

Interactivity and simplicity: VR is not intended for computer specialists and should be presented in a natural and intuitive manner.

Collaborative: Environments and objects needs to be shared among users. This also includes sharing point of view and to include different expertise in an environment.

Usability: It is necessary to bring VR into an industrial setting and assess the usability regarding the industrial activities and the results (time, money, security etc.).

Open standards: Further development of VR needs open standards and joint development to succeed.

The workshop has also given the author many interesting and useful suggestion regarding his master thesis. This information will be used in his further work.

A.2 Introduction

In September 1998 and November 2001, the Halden Project organized successful workshops focusing on the application of virtual reality technology to topics of interest to the process industry. The third Halden workshop

on virtual reality applications took place 2nd - 3rd March, 2005 in Halden, Norway.
The main topics of the 2005 VR workshop titled *VR in the Future Industrial Workplace: Working Together - Regardless of Distance* were:

- Design
- Operations and Maintenance
- Training
- Engineering VR Systems

The workshop was divided into sessions reflecting the main topics of the workshop. The sessions were organized as a combination of presentations, demonstrations, and discussions.

A.3 Intention of visit

The intention of the visit was bipartite: First, to participate at presentations and get an update on recent research and development in the fields of virtual and augmented reality. Second, to receive feedback on the author's own ideas regarding the usage of virtual models and 3D sound in a control-room setting.

A.4 Presentations and demonstrations

The sessions were chaired by Pascal Laureillard, EDF and covered a wide fields of applications mainly from the industry of oil production and nuclear power plants:

Design: component design, assembly process, control-room, building design, scientific visualization

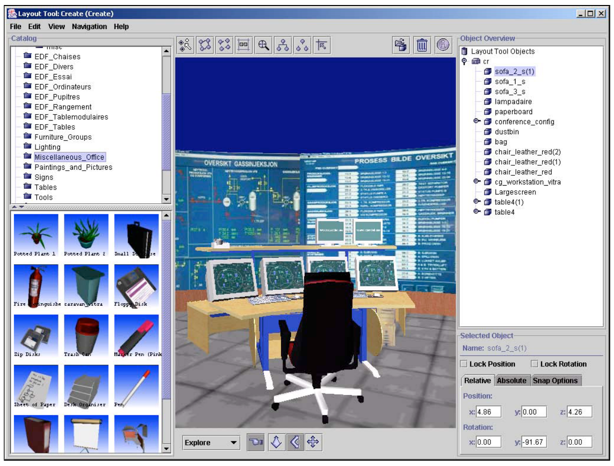
Operations and maintenance: collaborative teamwork, power plant maintenance (moving packages, dose-rates, inspection)

Training: refueling operations, maintenance operations

Even though the exact usage of virtual reality may differ in the different industries, the ideas and concepts proved to be rather similar and relevant for all participants.
A full summary of all presentations is not included in this report. A few have been selected according to the authors area of interest.

A.4.1 CREATE 2

CREATE 2, Control Room Engineering Advanced Toolkit Environment, is a suite of tools for designing and testing room layouts. It supports an iterative design process with multiple participants and provides version management for tracking design iterations. The toolkit uses interactive 3D technology to enable designers to rapidly prototype and test designs against guidelines and recommendations (see figure A.1). CREATE was designed specifically



Name: sofa_2_s(1)

Position: x: 4.86 y: 0.00 z: 4.26

Rotation: x: 0.00 y: -91.67 z: 0.00

Figure A.1: Layout tool of CREATE 2.

to support control room design and testing tasks and have been used by IFE in several real projects.
The toolkit is developed in Java3D and runs on all standard Internet-browsers. More information is available at http://www.ife.no/vr.

A.4.2 The LNPP refueling machine training simulator

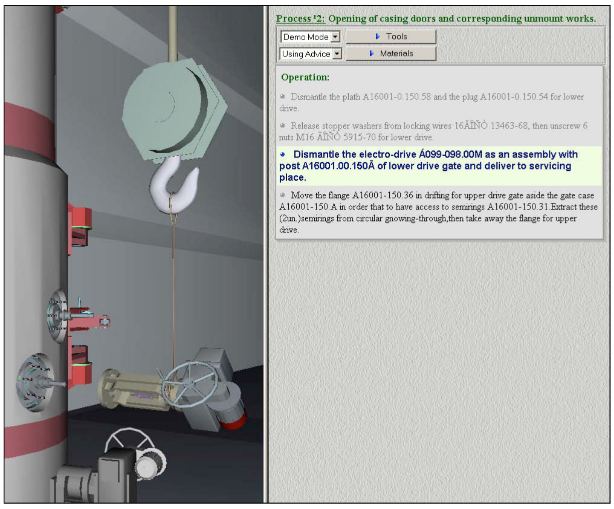
The LNPP (Leningrad Nuclear Power Plant) training simulator has been developed by IFE in co-operation with LNPP with the following objectives:

- Improve safety at LNPP by more effective training

  1. The refueling operation done using the refueling machine
     -Using a simulator system
  2. The safety critical maintenance procedures on the refueling machine
     -Using a procedural training system (see figure A.2)

- Enhance the qualifications of LNPP personnel

- Introduce new LNPP employees and visitors to the refueling machine



Process ¹2: Opening of casing doors and corresponding unmount works.

Operation:

- Dismantle the plath A16001-0.150.58 and the plug A16001-0.150.54 for lower drive.
- Release stopper washers from locking wires 16ÃÎÑÒ 13463-68, then unscrew 6 nuts M16 ÃÎÑÒ 5915-70 for lower drive.
- **Dismantle the electro-drive Á099-098.00M as an assembly with post A16001.00.150Ã of lower drive gate and deliver to servicing place.**
- Move the flange A16001-150.36 in drifting for upper drive gate aside the gate case A16001-150.A in order that to have access to semirings A16001-150.31.Extract these (2un.)semirings from circular gnowing-through,then take away the flange for upper drive.

Figure A.2: Procedural documentation of the procedure training system.

The simulator has proved to be very efficient to train personnel both for refueling and maintenance. The Norwegian government will support the project for further development towards new functions and scenarios.

A.4.3 CollabVE

The CollabVE project was presented by Michael Louka, IFE and is a software library to enable quick and easy development of collaborative virtual environments. It is primarily used as a test-bed for studying concepts and implementation strategies that affect the usability of multi-user systems. The library supports multiple collaboration strategies and is for instance used to perform usability studies, using a shared virtual control room environment. The users are represented in the virtual environments as avatars and the communication is done through gestures, text, text-to-speech, audio, video and file sharing. An example of a virtual environment is depicted in figure A.3.

A.4.4 Discussion

During the Workshop several useful discussions were held with other participants and employees at IFE.

Jette Lundtang Paulsen, Risø National Laboratory, Denmark

Paulsen is working at the Risø National Laboratory in Denmark and is currently assigned to a project called Safesound. Her participation covers the evaluation and usability-testing of the systems developed. The project is

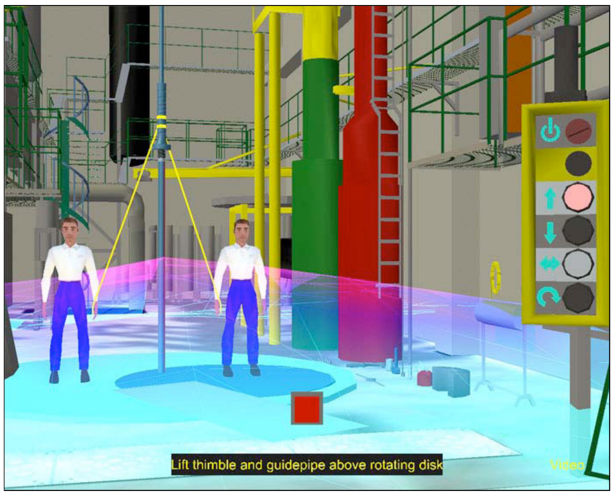


Figure A.3: Example for virtual environment with avatars in CollabVE.

sponsored by European Commission's GROWTH program (GRD1 2001). Safesound aims at improving safety during aircraft's ground and flight operation by means of enhanced audio functionalities. Project's enhanced audio functionalities are 'Direct Voice Input' (DVI), 'Direct Voice Output' (DVO) and distance speech capture.
The project also includes using 3D sound to warn the pilots about nearby planes to avoid dangerous situations.

Alf Ove Braseth, IFE Halden

Alf Ove Braseth is currently working as a research scientist at IFE Halden in the fields of control-room systems design and user interfaces. Braseth has earlier been working as an operator at a drilling platform.
Braseth gave the following comments on a presentation of the authors work and ideas:

- Alarms can be divided in two types:
  - Production alarms
  - Safety critical alarms
- If one alarm occurs, there will be a natural development with some expected alarms following the initial one. If for instance a valve does not open, it is expected that the pressure will rise/drop at some locations. These expected alarms should not be warned with an audible alarm.
- Overall sound-design:

- **Production alarms**
  Every 15. minute give an indication of the current status. The audio is generated based on the current level of status. A level is defined as a given number on non-critical alarms regarding the production. Maybe there should be a signal whenever it occurs, not repeating - just an indication.
- **Safety critical alarms**
  Usage of auditory icons could be effectfull.- but not repeatedly. All other sounds should be silenced.
- **Overall status**
  Every x. minute - play some sound to indicate the overall status. The idea is good.

A.4.5 Social

In the evening, all participants attended to a 17th century dinner at Fredriksten Fortress, followed by a private concert with the Halden band El Corazon (see figure A.4). This social event lead to many interesting conversations and discussions with other participants.

Figure A.4: Social event with 17th century dinner at Fredriksten Fortress.

Appendix B

Study: Tjeldbergodden

This trip-report was written after visiting the control-room at Tjeldbergodden 13.04.05.

B.1 Summary

The trip to Tjeldbergodden was made to get a broader picture of today's work-situation of operators in a control-room environment. The focus has been how the alarm management is performed and whether or not there exist room of improvement by means of virtual models and 3D sound compared to today's system.
The observations reveals potential improvements regarding the presentation of alarms to the operator, both visually and audible. There is, however, no clear-cut solution on how to integrate a virtual model with today's routines and way's of working. Whether this is caused by practical limitations or lack of innovative thinking is not clear.

B.2 Introduction

The Tjeldbergodden complex in mid-Norway comprises a methanol plant, an associated pipeline transport system and receiving terminal for gas from the Heidrun field, and an air separation facility. The complex was built in the early nineties and is today operated by Statoil.
The visit to Tjeldbergodden was arranged with responsible of extern relationships, Frank Sinnes, and took place the 13th of April 2005 from 11:00 until 17:00.

B.3 Intention of visit

The intention of the visit was to observe the way operators work in an operational control-room and to give the author a broader picture of their situation, problems and challenges. The main focus has been on how alarms and alarms are presented and taken care off by the operators.

B.4 Observations

The control-room at Tjeldbergodden contains two main workplaces, one for the methanol plant and one for the air separation facility. Each workplace is operated by at least one person. The work is supervised by the shift-leader seated in an office nearby. In addition to this, 7-8 field-operators are assigned to different areas of the facility and are working on orders from the control-room operators.

B.4.1 Equipment in the control-room

The control-room (illustrated in figure B.1) consists of following main elements:

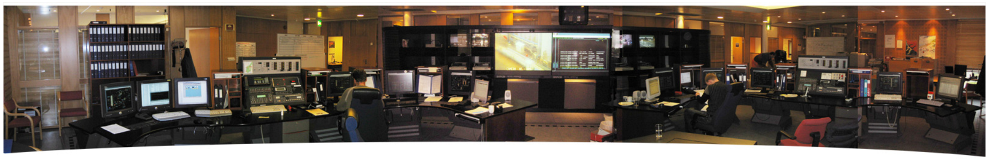


Figure B.1: Control-room at Tjeldbergodden.

- Billboard with list of alarms
- Terminals controlling the facility by means of process diagrams
- Computers for regular work, email, Internet etc.
- CCTV, visual surveillance of possible dangerous situations, tankers, gate etc.
- Radio communication
- Whiteboard with schedule of tankers
- Emergency shutdown panels
- Masterdocuments of facility used to identify and localize objects
- Access to kitchen and suite

When an alarm occurs in the facility, a new line appears on the large display combined with an audible alarm. This means that both operators are informed of all alarms, even though their area of interests and responsibility are limited. Therefor both operators working in the control-room have a rather extensive overview of the condition of the facility.

B.4.2 Environment

The environment in the control-room is normally relaxing and without stress. Other personnel are often dropping by for a chat or coffee and much time are spend doing non-relevant work. The alarms occurring are mostly non-critical and the work are more or less routine. This is mainly a consequence of the character of this type of facility. In a different type of industry, the situation could be different.
The level of noise in the control-room is rather high and may have different sources, some which have no relevant purpose:

- Voices/chat from personnel present
- Radio communication
- The terminals used for controlling the plant produce a beep every time the mouse is clicked
- Audible alarm
- Phones

If an extended use of audible signals was going to be integrated in this specific control-room, some measures should be taken to reduce the amount of background noise.

B.4.3 Alarm management

Whenever an unexpected event occurs in the facility, an audible alarm is played and a new record is created and presented as a new line in the list of alarms. The list is available at both workplaces and also displayed at the billboard in the center of the control-room (see figure B.2). This list is the basis of the operators work-day and is monitored closely.

Each new line consists of different parameters as depicted in fig B.3: A timestamp, identification-number of the component, the current measuring value causing the alarm, a keyword describing the function of the component and the name of the area where the component is located.

All new alarms are initially marked as unconfirmed. As long as an alarm is not confirmed by an operator, the area-field of the alarm-line blinks. Unconfirmed alarmed are thus easy to spot on the billboard. Even though an alarm in most cases are caused by an unexpected event, some alarms occurs as an result of an expected event. This could be a valve being opened, an engine starting etc. These alarms are notifications from the system and do not need any intervention from the operator. However, these alarms still contains important information and must therefor be confirmed by an operator.

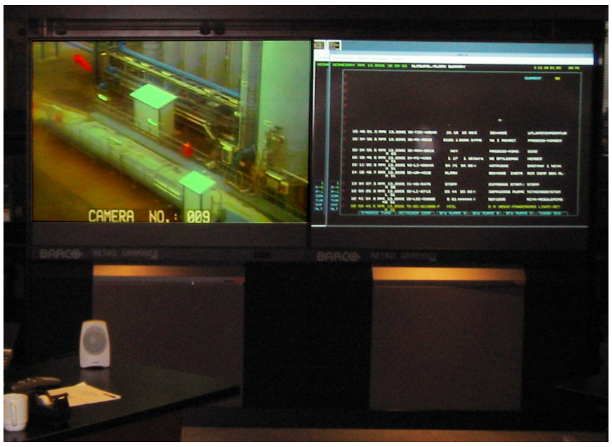
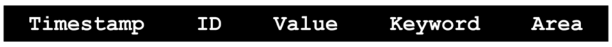


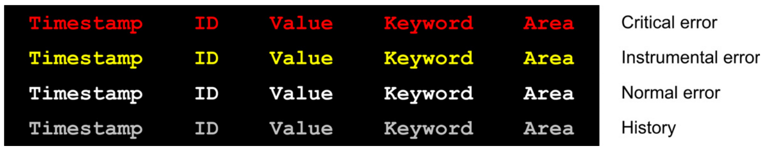
Figure B.2: Billboard with list of alarms and CCTV.



Timestamp	ID	Value	Keyword	Area

Figure B.3: Structure of lines in the alarm list.

Each line is colored according to the current status of the alarm (see figure B.4. When the values causing an alarm returns to normal, the line turns gray (history) and eventually disappears.



Timestamp	ID	Value	Keyword	Area	Critical error
Timestamp	ID	Value	Keyword	Area	Instrumental error
Timestamp	ID	Value	Keyword	Area	Normal error
Timestamp	ID	Value	Keyword	Area	History

Figure B.4: Coloring of lines in the alarm list.

The audible alarm which, indicates a new alarm, changes according to the impact of the alarm. A critical error produce a louder signal than a normal event. However, as the amount of background noise in the control-room may be rather high, it is not necessarily very easy to hear the difference between a critical and a non-critical alarm. Another important observation is that the amount of alarms occurring, which may be several each minute, soon turn the alarm into "background-noise". The author experienced this after approximately one hour in the control room.

B.4.4 Alarm solving

When an alarm occurs it is normally because a measuring value is outside normal range. This could have several reasons, and in some cases it is necessary for the operator to intervene the process to get the situation back to

normal. Finding the root-cause of an alarm is not necessarily an easy job and the solution is seldom listed in a manual. It is something which requires experience and high amount of knowledge of the facility and the processes involved. The investigation performed may include looking at trends, checking related elements and processes, history, current work performed in the area and previous alarms or notifications. It could even include checking external factors like the weather outside.
Some alarms require that a field-operator is sent out to investigate. This is controlled by the control-room operator, and requires thus that he/she has some first-hand experience with the facility.

B.4.5 New control-room system

In cooperation with ABB AS, Statoil Tjeldbergodden is currently working on a project to develop a new control-room system. The system will replace the operational system used today and will improve the usability and provide a more integrated and standardized functionality. The new system are based on an object-aspect-principle, meaning that an object may be looked upon from different aspects. New aspects can easily be linked with an object through standard Microsoft Windows protocols and could for instance be the object's documentation, trend-curve, visual image, history, current measuring-value etc. The user interface is built on the same elements as today's system, a graphical process-diagram of the facility.
The new system is planned to be fully integrated during a main-audit in 2007.

B.5 Challenges

The operators at Tjeldbergodden control-room was asked to identify any challenges or possible problems with today's solution which gave the following answers:

- The graphical user-interface of the operational terminals could be improved and extended with more relevant information.
- The audible alarm is monotonous and does not reflect any information about the type of alarm.
- The operators need first-hand experience with the facility to know anything about distances and dimensions of the elements displayed in the process-diagrams.
- Each operator must deal with too many monitors and keyboards/mice.
- There are too many alarms occurring.

- The alarms are only sorted according to the timestamp, and not according to urgency and/or impact.

B.6 Feedback

The operators was verbally presented to the authors ideas regarding replacing today's control-room systems with virtual models assisted by 3D sound. 3 operators listed up the following comments:

- The new design based on virtual models is interesting and could provide new functionality compared to today's system.
- Today's process-diagrams are vital to the operator and can not fully be replaced by a virtual model.
- The new control-room system being developed in cooperation with ABB AS could easily provide information to a virtual representation.
- It is unclear whether or not the authors new design could provide improved functionality compared to today's system.

B.7 Thoughts and ideas

Some thoughts and ideas have been documented as an result of the visit:

- The operators have a rather extensive overview of the facility' condition. It is therefor uncertain what benefits would come from an audible system informing about the current status-level. If such a system is to be used, it may be good to use a short and concise sound.
- Whenever one of the operators are leaving the control-room, he/she must always carry a radio in case of emergency. An audible system installed in nearby rooms, i.e. the kitchen or bathroom, could keep the operators informed even though they are not present.
- It is difficult to avoid the usage of process-diagrams to be able to control the facility. Replacing today's solution without using the process-diagrams requires innovative thinking and training of today's operators.
- Solving an alarm requires investigation, often including elements at great distance from the origin of the alarm. This implies that a virtual model of the facility may be a less suitable tool than today's process-diagrams. Is there a way to interconnect a virtual model and process-diagram?

- The alarms should be filtered before they are presented to the operators in such a way that less important alarms (notifications) are separated from the real alarms. This filter should also be capable of identifying relations between alarms.
- The audible alarm could easily be designed to reflect information about the type of alarm occurring.

Appendix C

Interview: Per Ivar Karstad

Per Ivar Karstad works as a project manager at Statoil Research Center, Rotvoll. This interview took place in informal settings and resulted in the following comments:

- Operators are often occupied with other tasks than monitoring the facility. Communication and keeping track of with the technical staff are one example. Because of this, audio should be used to inform the operators of occurring alarms when they are not sitting at the desk.
- The operators should be able to prioritize which alarm to work with. Otherwise, some intelligence must be developed to analyze the alarms regarding urgency and impact prior to the trigging of alarms
- An operator should at all times know the status of the facility. Therefor - an 'overall status'-sound may not be necessary. However, if it is desirable to reduce staff in some ways - and 'overall status'-sound could help.
- For intelligent alarm-handler: An alarm may be critical/non critical if combined with some others. This should be checked in advance of presentation to the operators..
- It is very important to inform the operators about safety-critical alarms.

Appendix D

Seminar: Alarm management and control rooms

This report gives a summary of the different topics presented at the alarm management and control room seminar in Stavanger, 10.-11.05.05.

D.1 Summary

Through the variety of lectures presented at the seminar, several interesting challenges, solutions and concepts were discussed. Despite the differences, most lectures revolved around achieving an optimal situation awareness for the operators. Even though much research and effort are invested in this area, the development regarding alarm management systems has yet to see the revolutionary step. The author finds this peculiar, and believes that virtual reality will become a natural part of control rooms in the near future. The seminar has also given the author much useful information regarding his master thesis. This includes both literature and contacts.

D.2 Introduction

The seminar aimed at giving a brief introduction and raise consciousness regarding the design of control rooms and alarm management systems.
The lead speaker of the seminar were Ian Nimmo (see figure D.1), President and founder of User Centered Design Services (Nimmo 2005*c*). Nimmo is a well-known lecturer within control-room design and has more than 35 years of experience in this field. His lectures revolved around Operator Situation Awareness and his experiences from best-practice projects. Apart from this, several lecturers from different Norwegian industry companies presented their own experiences and projects regarding the topic of interest. Some vendors was also invited to show how their solutions address the problems regarding alarm management. This included both solutions which are available today,

and future technology under development. The seminar, consisting solely of lectures, lasted for two days and had over 80 participants.

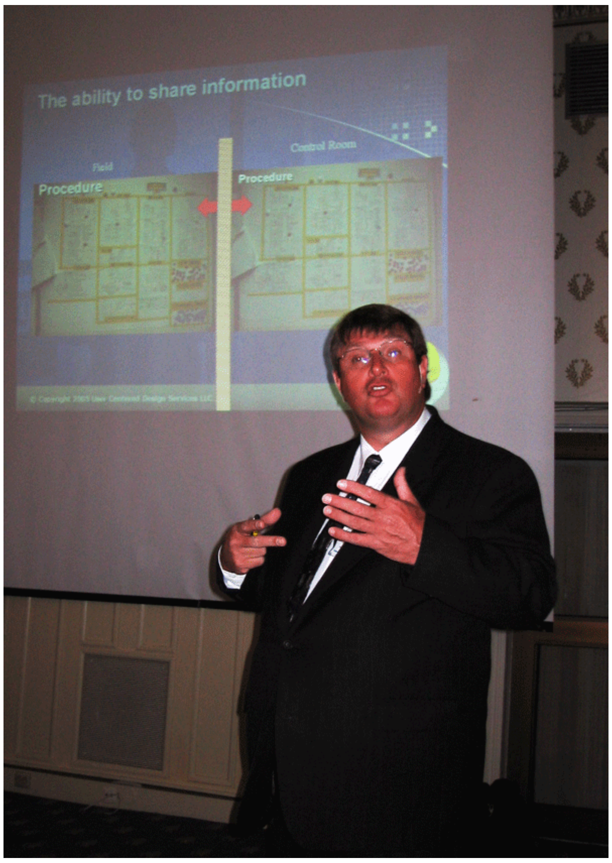


Figure D.1: Ian Nimmo, lead speaker of the seminar.

D.3 Intention of visit

The intention of the visit was mainly to get an overview of the variety of today's alarm-management systems and their enabling technologies.

D.4 Presentations and demonstrations

The different lectures covered many aspects regarding control rooms and alarm management systems, and can be divided into the following categories:

- Operator Situation Awareness
- Best practices

- Control room design
  - Use of Virtual Reality
  - Guidelines and regulations
- Alarmphilosophy
- Existing systems and experiences
- Today's and future technology

A full summary of all presentations is not included in this report. A few have been selected according to the authors area of interest.

D.4.1 Operator Situation Awareness

Ian Nimmo from the User Centered Design Services LLC, held an interesting lecture about the importance of operator situation awareness and what conclusions he has made during his research.
Nimmo defines good situation awareness as achieving high performance in preventing abnormal situations and designing a work environment to provide good situation awareness and a proactive operating stance. The term *situation* reflects the changing states of a process plant, also including minor and planned changes like process control moves and maintenance work. In order to find what elements influence the situation awareness, Nimmo has defined a framework which outlines distinct intervention activities that occur during an abnormal situation (see figure D.2). Using this framework in a real con-

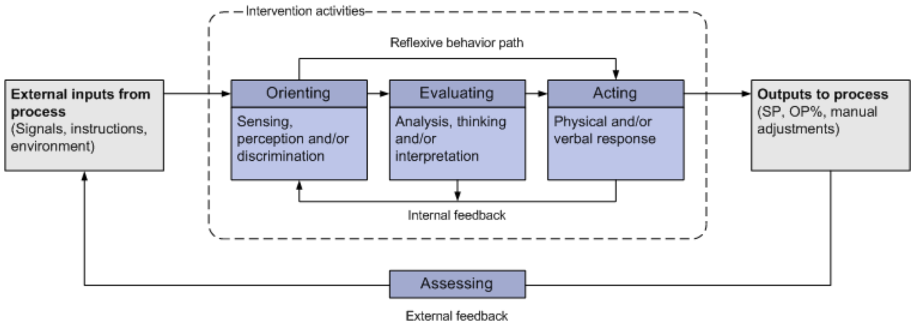
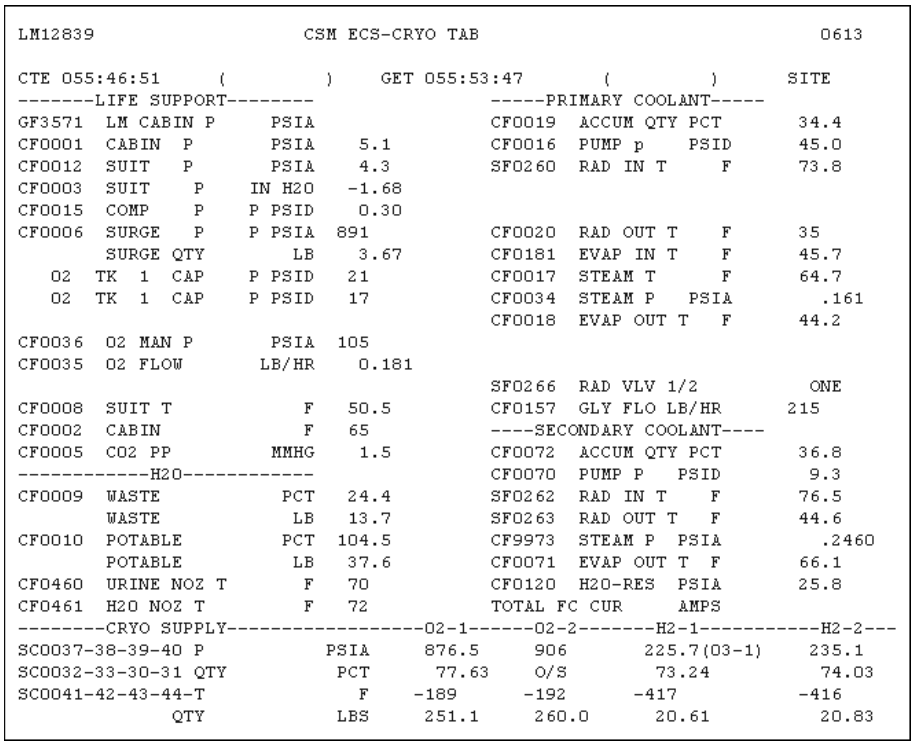

Intervention activities
Reflexive behavior path
External inputs from process
(Signals, instructions, environment)
Orienting
Sensing, perception and/or discrimination
Evaluating
Analysis, thinking and/or interpretation
Acting
Physical and/or verbal response
Outputs to process
(SP, OP%, manual adjustments)
Internal feedback
Assessing
External feedback

Figure D.2: Intervention activities during an abnormal situation.

text can help to pinpoint possible problems and areas of improvement in the given setting. Nimmo also talks about alarm management and how an user interface should be designed to provide good operator situation awareness. Among others, he stresses the fact that data only becomes information when it is shown in context. An example which illustrates this is the Apollo XIII accident in 1966. One of the operators monitoring the spaceship failed to obtain the readings from an oxygen tank as the pressure raised above critical limits. This eventually lead to a massive explosion and damages causing the whole mission to be aborted. A screen capture from the console which was monitored is depicted in figure D.3. As the figure illustrates, there is clearly room for improvement in designing this user-interface. However, since the



```
LM12839                   CSM ECS-CRYO TAB                               0613

CTE 055:46:51     (         )    GET 055:53:47       (         )      SITE
-------LIFE SUPPORT--------                -----PRIMARY COOLANT-----
GF3571  LM CABIN P     PSIA                CF0019  ACCUM QTY PCT       34.4
CF0001  CABIN  P       PSIA    5.1         CF0016  PUMP p    PSID      45.0
CF0012  SUIT   P       PSIA    4.3         SF0260  RAD IN T     F      73.8
CF0003  SUIT    P    IN H2O   -1.68
CF0015  COMP    P    P PSID    0.30
CF0006  SURGE   P    P PSIA  891           CF0020  RAD OUT T    F      35
        SURGE QTY        LB    3.67        CF0181  EVAP IN T    F      45.7
   O2  TK  1  CAP    P PSID   21           CF0017  STEAM T      F      64.7
   O2  TK  1  CAP    P PSID   17           CF0034  STEAM P   PSIA       .161
                                           CF0018  EVAP OUT T   F      44.2
CF0036  O2 MAN P       PSIA  105
CF0035  O2 FLOW       LB/HR    0.181
                                           SF0266  RAD VLV 1/2          ONE
CF0008  SUIT T            F   50.5         CF0157  GLY FLO LB/HR      215
CF0002  CABIN             F   65           ----SECONDARY COOLANT----
CF0005  CO2 PP         MMHG    1.5         CF0072  ACCUM QTY PCT       36.8
------------H2O------------                CF0070  PUMP P   PSID        9.3
CF0009  WASTE           PCT   24.4         SF0262  RAD IN T    F       76.5
        WASTE            LB   13.7         SF0263  RAD OUT T   F       44.6
CF0010  POTABLE         PCT  104.5         CF9973  STEAM P  PSIA        .2460
        POTABLE          LB   37.6         CF0071  EVAP OUT T  F       66.1
CF0460  URINE NOZ T       F   70           CF0120  H2O-RES  PSIA       25.8
CF0461  H2O NOZ T         F   72           TOTAL FC CUR     AMPS
--------CRYO SUPPLY------------------O2-1------O2-2-------H2-1-----------H2-2---
SC0037-38-39-40 P          PSIA     876.5     906        225.7(O3-1)    235.1
SC0032-33-30-31 QTY         PCT      77.63    O/S         73.24          74.03
SC0041-42-43-44-T             F    -189      -192       -417            -416
              QTY           LBS     251.1     260.0       20.61          20.83
```

Figure D.3: Screen captures of the ECS console used to monitor the oxygen tank.

Apollo XIII accident, much work has been done on developing user interface design elements. Nimmo lists the following elements as the key solution concepts and innovations:

- Single, integrated view
- Windows management and layout
- Navigation scheme
- Visual coding schemes, use of color and shapes
- Interaction objects
- Contextual menus
- Task view organization

The lecture was based on two papers by Ian Nimmo ((Nimmo 2005*b*) and (Nimmo 2005*a*)).

D.4.2 Alarm philosophy in Statoil

Svein Louis Bersaas, working as a senior teamleader at Statoil ASA Kårstø NG, gave a presentation of Statoil's alarmphilosophy. The reason for establishing such an philosophy was tripartite:

- High alarm rate
- Little or wrong usage of alarm prioritizing
- Little usage of alarm suppression

The alarmphilosophy will be used, together with an ongoing gap analysis, to increase the consciousness and effort around safety and quality aspects of alarm management systems. Each of Statoil's facilities will create such a philosophy based on the following template:

1. Intention
2. Central Control Room (CCR) alarm functions
3. Human factor design principles
4. Alarm performance requirements
5. Documentation of alarm management
6. Alarm design

Several examples of content were presented by Bersaas, followed by a description of the future initiatives.
The author finds Statoil's alarmphilosophy a very useful tool for creating guidelines and descriptions of the alarm management process. Even though the real value lies in the contents, the alarmphilosophy as a framework has great benefits.

D.4.3 Future technology

Different vendors of alarm management systems, including ABB, Siemens, Honeywell, Kongsberg Maritime and Emerson Process Management, were invited to demonstrate their products and plans of development. In general, these presentations gave little or no indication of any revolutionary designs. Apart from upgraded alarm management functionality, the user interface were still based on process diagrams and 2D-graphics. ABB however, did show a video of a supposedly new type of management tool based on a virtual model of the plant. Details regarding this tool was not available as this system was on very early stage.

D.4.4 Thoughts and reflections

Throughout the seminar some thoughts and reflections were noted, for further use in the author's master thesis.

- There has been a great improvement on 2D user interfaces of alarm management systems, due to much research the last 10-15 years. However, little effort has been put into exploring the usability of virtual reality and 3D models.
- There is a close resemblance between the IT Infrastructure Library[1] (ITIL) and the way the whole alarm management process should be designed. This field should be investigated further.
- Several guidelines for control rooms and alarm management systems exist, including EMMUA (Alarm management guidelines) and guidelines from the Norwegian Petroleumstilsynet.
- Using alarm management systems requires that the operator trusts the system and its stability. Such maturity requires time and cannot be forced onto the operators.
- None of the other participants had any experience of using sound to communicate meta-information about alarms, such as type-description. However, it is quite common to increase the amplitude in accordance with the alarm's urgency/severity. Some control-rooms also used sound to indicate which operator was responsible for the occurring alarm, by allowing the operator to use his favorite sounds.
- A virtual representation could be used to visualize alarms by automatically displaying the object causing the alarm. The virtual representation doesn't need to be interactive as it is not used in investigation and/or analysis. This approach could be a suitable approach to increase the operator's tolerance of new technology (virtual models).

[1]ITIL is a best-practice framework for IT service support and delivery

Appendix E

Installation Guide

Due to restrictions given from Statoil, SnøhvitSIM is not enclosed this report. This also goes for AMEX, since it is a built-in part of the SnøhvitSIM application.

For demonstrations of SnøhvitSIM AMEX, Bjørn Sæther at Statoil Research Center should be contacted.

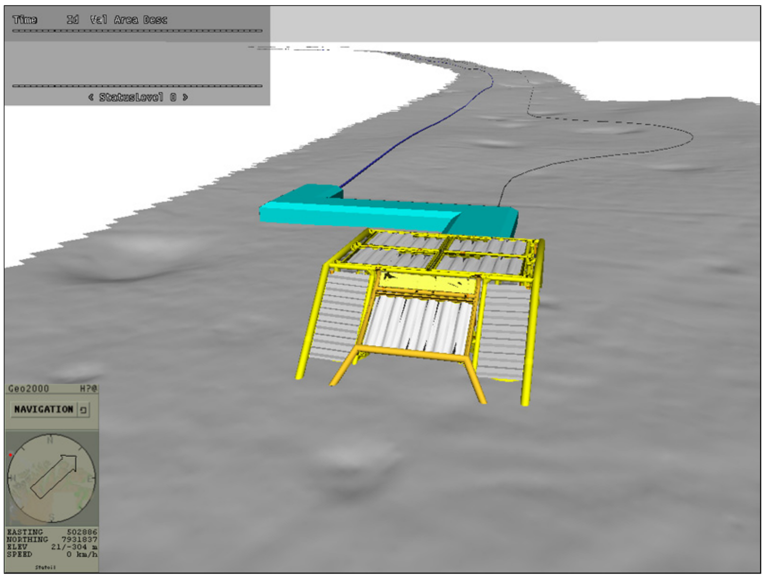
The source code and media files (sounds, 3D-models etc) are included on the enclosed CD (see appendix H).

Appendix F

User guide

Before starting SnøhvitSIM AMEX, make sure the volume of the speakers are set to a proper level, and that they are configured for 7.1 reproduction (if available).

After the application is started, the screen should look similar to figure F.1. The application is now in simulation-mode, and alarms will appear at random order and intervals.



```
EASTING      502886
NORTHING    7931837
ELEV     21/-304 m
SPEED       0 km/h
```

Figure F.1: Initial view of SnøhvitSIM AMEX.

F.1 Interaction

SnøhvitSIM AMEX is mainly controlled by keyboard commands. A list of available commands, including when they are available, is described in table F.1.

Command	Functionality	Type / activating event
F# *	Goto alarm with id #	Universal
Shift + F# *	Turn on beacon on alarm with id #	Universal
Ctrl + F# *	Turn off beacon on alarm with id #	Universal
G	Goto alarm, moves camera to alarm	On showing of **New alarm**
S	Suppress, mark the alarm as fixed	On showing of **New alarm,** if the alarm is of type production-alarm
C	Confirm, mark the alarm as fixed	On showing of **New alarm**, if the alarm is of type notification
D	Show documentation of alarm-object	On showing of **New alarm**
T	Show trends of alarm-object	On showing of **New alarm**
F	Fix the alarm	On showing of **Alarm detail**
T	Show documentation of alarm-object	On showing of **Alarm detail**
D	Show trends of alarm-object	On showing of **Alarm detail**
C	Close the information-box	On showing of **Alarm detail**
Mouse click on object	Shows **Alarm Detail** of object	Object is highlighted, i.e. there is an alarm on the object

* # refers to the ID of an alarm.
ID 0 = F10, ID 1 = F1, ID 2 = F2, ID 3 = F3

Not implemented due to limitations of scope

Table F.1: List of all available commands in prototype.

A few comments on the use of SnøhvitSIM AMEX is described in the following sections.

F.1.1 SnøhvitSIM user guide

An early release of the SnøhvitSIM documentation and user guide is included on the enclosed CD (see appendix H).
Tip: Closing the application is done by pressing ESC, ESC.

F.1.2 Keyboard corrections

Due to bugs in the SnøhvitSIM framework, using Shift and Ctrl combined with the numerical keys is not working properly. These updates are included in table F.1.

Appendix G

Source code

G.1 amex.cpp

Listings of the sourcecode of amex.cpp.

```
//AMEX-classes
#include "amexGUIEngine.h"
#include "amexSoundEngine.h"
#include "amexAlarmGenerator.h"
#include "amex.h"

//Viewer
#include "../viewer/CLODViewer.h"
#include "../guile/guile.h"
#include "../guile/guilestuff.h"

//Various
#include <SmallChange/nodes/PickCallback.h>
#include <Inventor/sensors/SoTimerSensor.h>
#include <Inventor/SbTime.h>
#include <Inventor/SbDPLine.h>
#include "properties.h"
#include <Inventor/fields/SoSFFloat.h>
#include <Inventor/lists/SbList.h>
#include <Inventor/SbRotation.h>
#include "viewer/CLODViewer.h"
#include <SmallChange/nodes/UTMCamera.h>
extern CLODViewer * currviewer;
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/SbLinear.h>
#include <math.h>

//Events & picking
#include "../misc/eventgrabber.h"
#include <Inventor/events/SoButtonEvent.h>
#include <Inventor/events/SoMouseButtonEvent.h>
#include <Inventor/events/SoKeyboardEvent.h>
#include <Inventor/actions/SoRayPickAction.h>
#include <Inventor/SoPickedPoint.h>
#include <Inventor/SoPath.h>

//Menus
#include "../nodes/ShadowText2.h"

//Sensors and timers
#include <Inventor/sensors/SoTimerSensor.h>


amexGUIEngine ge;                /** < Graphical User Interface Engine */
amexSoundEngine se;              /** < Sound Engine */
amexAlarmGenerator ag;           /** < Alarm Generator */
SbVec2s eventPositionDown;       /** < Position of event on button down*/
SbVec2s eventPositionUp;         /** < Position of event on button up*/

int incidentCount;               /** < Number of alarms */
int statusLevel;                  /** < Statuslevel */
int gotoAlarm;                    /** < Which object to goto from popup-window */
int alarmBoardActive;             /** < Object currently display at alarm board */
int alarmObjectMap[4];            /** < Mapps objects to occurance in alarmlist */
int alarmActiveMap[4];            /** < Which alarms are active/fixed*/
int status_counter;               /** < How long since last status-sound */

char alarm001[50];                /** < Container for 1st alarm */
char alarm002[50];                /** < Container for 2nd alarm */
char alarm003[50];                /** < Container for 3rd alarm */
char alarm004[50];                /** < Container for 4th alarm */
char status[50];                  /** < Container for statuslevel */
char heading[50];                 /** < Heading for popup */

bool SOUND_ENABLED = true;        /** < Flag to activate sound */
bool POPUP_ACTIVE = false;        /** < True when popup is displayed */
bool ALARM_BOARD_ACTIVE = false;  /** < True when alarm-board is displayed */
int BEACON_ACTIVE = -1;           /** < Which beacon is enabled */
int STATUS_TIMERCONTROL = 150;    /** < Freq. of status sound */

float X_POSITION = 502898.367475894;
float Y_POSITION = 7931858.305316999;
float Z_POSITION = -325.1879939097652;

/**
* Initialize and creates various nodes after scenegraph is loaded
* @return true
*/
SbBool amex_post_init(void) {

   //MOVE CAMERA TO INITIAL POS
      setCameraToAlarm(99);

   //BUILD SCENEGRAPH
      SoSeparator * AMEX01Sep = (SoSeparator *) SoNode::getByName("AMEX01");
      SoSeparator * AMEX_TEMPLATE_LOCATOR_Separator =
         (SoSeparator *) SoNode::getByName("AMEX_TEMPLATE_LOCATOR");
      //Add alarm-elements
         ge.buildElementGraph(AMEX_TEMPLATE_LOCATOR_Separator);
      //Add structures and templates
         ge.buildSceneGraph(AMEX_TEMPLATE_LOCATOR_Separator);
      //Add sounds;
         se.initialize(0, AMEX_TEMPLATE_LOCATOR_Separator);
         se.buildSoundGraph();
         se.buildBeaconGraph();
         se.buildStatusGraph();

   //START TIMERS
      //Alarm-timer
         srand(static_cast<unsigned>(time(0)));
         SoTimerSensor * alarmTimer = new SoTimerSensor();
            alarmTimer->setFunction(retrieveAlarm);
            alarmTimer->setInterval(60.0);
            alarmTimer->schedule();
      //Beacon-timer
         SoTimerSensor * beaconTimer = new SoTimerSensor();
            beaconTimer->setFunction(updateBeacon);
            beaconTimer->setInterval(0.2f);
            beaconTimer->schedule();
      //Status-timers
         SoTimerSensor * statusTimer = new SoTimerSensor();
            statusTimer->setFunction(checkStatusSound);
            statusTimer->setInterval(10.0f);
            statusTimer->schedule();
      //Running checkStatus in beginning
         UTMCamera * cam = (UTMCamera*) currviewer->getCamera();
         SbVec3d cameraLoc = cam->utmposition.getValue();
         se.updateStatusGraph(0, cameraLoc[0]-X_POSITION,
                        cameraLoc[1]+2.0f-Y_POSITION,
                        cameraLoc[2]-Z_POSITION);

   return true;
}

/**
* Retrieves new alarms and communicates with GUIEngine and SoundEngine
* @param *data data (not used)
* @param *sensor sensordata (not used)
*/
void retrieveAlarm(void * data, SoSensor * sensor) {
   ag.callback();

   if (ag.incidentCount > incidentCount) {
      incidentCount++;
```

```
    SbString timeString;

    switch(ag.incidentCount) {
       case 1:
          //Update Alarm list
             timeString = ag.incidentArray[incidentCount-1].timestamp->
                format("%h.%m.%s");
             sprintf(alarm001, "%s  %i   %i  %i  %s", timeString.getString(),
                ag.incidentArray[ag.incidentCount-1].id,
                ag.incidentArray[ag.incidentCount-1].value,
                ag.incidentArray[ag.incidentCount-1].area,
                ag.incidentArray[ag.incidentCount-1].description);
             alarmObjectMap[ag.incidentArray[ag.incidentCount-1].id] = 0;
          //Set alarm active
             alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id] = 1;
          //Update alarmlist
             updateAlarmList(incidentCount);
          //Show Popup
             gotoAlarm = ag.incidentArray[ag.incidentCount-1].id;
             sprintf(heading, "!! %s !!",
                ag.incidentArray[ag.incidentCount-1].description2);
             POPUP_ACTIVE = true;
             ge.updatePopup(bool(true), heading,
                ag.incidentArray[ag.incidentCount-1].impact_urgency,
                ag.incidentArray[ag.incidentCount-1].criticality);
          break;
       case 2:
          //Update Alarm list
             timeString = ag.incidentArray[incidentCount-1].timestamp->
                format("%h.%m.%s");
             sprintf(alarm002, "%s  %i   %i  %i  %s", timeString.getString(),
                ag.incidentArray[ag.incidentCount-1].id,
                ag.incidentArray[ag.incidentCount-1].value,
                ag.incidentArray[ag.incidentCount-1].area,
                ag.incidentArray[ag.incidentCount-1].description);
             alarmObjectMap[ag.incidentArray[ag.incidentCount-1].id] = 1;
          //Set alarm active
             alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id] = 1;
          //Update alarmlist
             updateAlarmList(incidentCount);
          //Show Popup
             gotoAlarm = ag.incidentArray[ag.incidentCount-1].id;
             sprintf(heading, "!! %s !!",
                ag.incidentArray[ag.incidentCount-1].description2);
             POPUP_ACTIVE = true;
             ge.updatePopup(bool(true), heading,
                ag.incidentArray[ag.incidentCount-1].impact_urgency,
                ag.incidentArray[ag.incidentCount-1].criticality);
          break;
       case 3:
          //Update Alarm list
             timeString = ag.incidentArray[incidentCount-1].timestamp->
                format("%h.%m.%s");
             sprintf(alarm003, "%s  %i   %i  %i  %s", timeString.getString(),
                ag.incidentArray[ag.incidentCount-1].id,
                ag.incidentArray[ag.incidentCount-1].value,
                ag.incidentArray[ag.incidentCount-1].area,
                ag.incidentArray[ag.incidentCount-1].description);
             alarmObjectMap[ag.incidentArray[ag.incidentCount-1].id] = 2;
          //Set alarm active
             alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id] = 1;
          //Update alarmlist
             updateAlarmList(incidentCount);
          //Show Popup
             gotoAlarm = ag.incidentArray[ag.incidentCount-1].id;
             sprintf(heading, "!! %s !!",
                ag.incidentArray[ag.incidentCount-1].description2);
             POPUP_ACTIVE = true;
             ge.updatePopup(bool(true), heading,
                ag.incidentArray[ag.incidentCount-1].impact_urgency,
                ag.incidentArray[ag.incidentCount-1].criticality);
          break;
       case 4:
          //Update Alarm list
             timeString = ag.incidentArray[incidentCount-1].timestamp->
                format("%h.%m.%s");
             sprintf(alarm004, "%s  %i   %i  %i  %s", timeString.getString(),
                ag.incidentArray[ag.incidentCount-1].id,
                ag.incidentArray[ag.incidentCount-1].value,
                ag.incidentArray[ag.incidentCount-1].area,
                ag.incidentArray[ag.incidentCount-1].description);
             alarmObjectMap[ag.incidentArray[ag.incidentCount-1].id] = 3;
          //Set alarm active
             alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id] = 1;
```

```
            //Update alarmlist
               updateAlarmList(incidentCount);
            //Show Popup
               gotoAlarm = ag.incidentArray[ag.incidentCount-1].id;
               sprintf(heading, "!! %s !!",
                  ag.incidentArray[ag.incidentCount-1].description2);
               POPUP_ACTIVE = true;
               ge.updatePopup(bool(true), heading,
                  ag.incidentArray[ag.incidentCount-1].impact_urgency,
                  ag.incidentArray[ag.incidentCount-1].criticality);
            break;
      }

      //Update Status, IncidentElements and Sound/Beacon
         int occuringIncidentId = ag.incidentArray[incidentCount-1].id;
         //Show Alarm object
            ge.updateIncidentElements(bool(true), occuringIncidentId);
         //Update sound
            if(SOUND_ENABLED) se.updateSoundGraph(occuringIncidentId,
               getDistanceToAlarm(occuringIncidentId),
               ag.incidentArray[incidentCount-1].position[0],
               ag.incidentArray[incidentCount-1].position[1],
               ag.incidentArray[incidentCount-1].position[2]);
            if(SOUND_ENABLED) se.updateBeaconGraph(bool(false),
               occuringIncidentId, 0.0f ,
               ag.incidentArray[incidentCount-1].position[0],
               ag.incidentArray[incidentCount-1].position[1],
               ag.incidentArray[incidentCount-1].position[2]);
         //Update status
            updateStatus();
   } else {
      //Do nothing
   }

}

/**
* Updates internal alarm-information and communicates with GUIEngine
* @param alarmCount Number of alarms
*/
void updateAlarmList(int alarmCount) {
   switch(alarmCount) {
      case 1:
         ge.updateIncidentText(alarmCount, alarm001,
            alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id],
            "", 0,
            "", 0,
            "", 0);
         break;
      case 2:
         ge.updateIncidentText(alarmCount, alarm002,
            alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id],
            alarm001, alarmActiveMap[ag.incidentArray[ag.incidentCount-2].id],
            "", 0,
            "", 0);
         break;
      case 3:
         ge.updateIncidentText(alarmCount, alarm003,
            alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id],
            alarm002, alarmActiveMap[ag.incidentArray[ag.incidentCount-2].id],
            alarm001, alarmActiveMap[ag.incidentArray[ag.incidentCount-3].id],
            "", 0);
         break;
      case 4:
         ge.updateIncidentText(alarmCount, alarm004,
            alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id],
            alarm003, alarmActiveMap[ag.incidentArray[ag.incidentCount-2].id],
            alarm002, alarmActiveMap[ag.incidentArray[ag.incidentCount-3].id],
            alarm001, alarmActiveMap[ag.incidentArray[ag.incidentCount-4].id]);
         break;
   }
}
/**
* Updates status level according to number of incidents.
* [0,0] = level 0
* [1,3] = level 1
* [4,4] = level 2
*/
void updateStatus() {
   switch(ge.visibleAlarmElements) {
      case 0:
         statusLevel = 0;
         break;
      case 1:
            statusLevel = 1;
            break;
        case 2:
            statusLevel = 1;
            break;
        case 3:
            statusLevel = 1;
            break;
        case 4:
            statusLevel = 2;
            break;
    }
    sprintf(status, "<StatusLevel: %i>", statusLevel);
    ge.updateIncidentText(7, status, 0, "", 0, "", 0, "", 0);
}

/**
* Keep's control over when to play the status-sound
* @param *data ...
* @param *sensor ...
*/
void checkStatusSound(void * data, SoSensor * sensor) {
    //Getting user-position
        UTMCamera * cam = (UTMCamera*) currviewer->getCamera();
        SbVec3d cameraLoc = cam->utmposition.getValue();
    //increase counter
        status_counter = status_counter + 10;

    switch(statusLevel) {
        case 0:
            if (status_counter >= STATUS_TIMERCONTROL*4) {
                status_counter = 0;
            }
            break;
        case 1:
            if (status_counter >= STATUS_TIMERCONTROL*2) {
                status_counter = 0;
            }
            break;
        case 2:
            if (status_counter >= STATUS_TIMERCONTROL*1) {
                status_counter = 0;
            }
            break;
    }
    //playing status sound
    if(status_counter==0)se.updateStatusGraph(statusLevel,
        cameraLoc[0]-X_POSITION,
        cameraLoc[1]-Y_POSITION,
        cameraLoc[2]-Z_POSITION);
}

/**
* Updates the beacon-sound's pitch if rotating
* @param *data ..
* @param *sensor ..
*/
void updateBeacon(void * data, SoSensor * sensor) {
    if(BEACON_ACTIVE > -1) {
        //Get information from camera about current position
            UTMCamera * cam = (UTMCamera*) currviewer->getCamera();
            SbRotation camrot = cam->orientation.getValue();
            SbVec3d cameraLoc = cam->utmposition.getValue();
        //Look-at vector
            SbVec3f lookat(0, 0, -1); // init to default view direction vector
            camrot.multVec(lookat, lookat);
            const float * camrotValues = lookat.getValue();
        //Direct vector
            SbVec3d directVec;
            directVec.setValue(
                ag.incidentArray[alarmObjectMap[BEACON_ACTIVE]].utm[0],
                ag.incidentArray[alarmObjectMap[BEACON_ACTIVE]].utm[1],
                ag.incidentArray[alarmObjectMap[BEACON_ACTIVE]].utm[2]);
            directVec -= cameraLoc;
            const double * directValues = directVec.getValue();
            float directDistance = directVec.length();
            directVec.normalize();
            SbVec3f directVecConst;
            directVecConst.setValue(directVec);
        //Find the angle between the two vectors
            float dotProd = lookat.dot(directVecConst);
        //Update Beacon
            se.updateBeaconPitch(BEACON_ACTIVE, acos(dotProd), directDistance);
    }
```

```
}

/**
* Returns the distance to an alarm
* @param id id of alarm
* @return distance
*/
float getDistanceToAlarm(int id) {
   float distance = 30.0f;     //default if alarm do not exist
   for (int i = 0; i<incidentCount ;i++){
      if (ag.incidentArray[i].id == id) {
         SbVec3f * distanceVec = new SbVec3f;
            distanceVec->setValue(ag.incidentArray[i].utm[0],
                                  ag.incidentArray[i].utm[1],
                                  ag.incidentArray[i].utm[2]);

         UTMCamera * cam = (UTMCamera*) currviewer->getCamera();
            SbVec3d cameraLoc = cam->utmposition.getValue();

         distanceVec->operator -=(cameraLoc);
         distance = distanceVec->length();
      }
   }
   return distance;
}


/**
* Moves camera to location of alarm
* @param alarmId id of alarm
*/
void setCameraToAlarm(int alarmId) {
   if (alarmId == 99) {
      //Initial position
      UTMCamera * cam = (UTMCamera*) currviewer->getCamera();
      cam->utmposition.setValue(SbVec3f(502868.0, 7931820.0, -292.916));
      cam->orientation.setValue(0.791249, -0.34784, -0.502923, 1.43665);
   } else {
      for (int i = 0; i<incidentCount ;i++){
         if (ag.incidentArray[i].id == alarmId) {
            UTMCamera * cam = (UTMCamera*) currviewer->getCamera();
            cam->utmposition.setValue(SbVec3f(
               ag.incidentArray[i].location[0],
               ag.incidentArray[i].location[1],
               ag.incidentArray[i].location[2]));
            cam->orientation.setValue(SbRotation(SbVec3f(
               ag.incidentArray[i].rotation[0],
               ag.incidentArray[i].rotation[1],
               ag.incidentArray[i].rotation[2]),
               ag.incidentArray[i].radians));
         }
      }
   }
}


/**
* Internal scheme methode.
*/
static SCM g2k_amex_post_init(void) {
  return gh_bool2scm(amex_post_init());
}

/**
* Event callback handler. Checks which type of event and acts if necessary
* @param *event event
* @param *closure data (not used)
* @return true/false
*/
static SbBool amex_event_cb(const SoEvent * event, void * closure) {
   SoMouseButtonEvent *mouseButtonEvent;
   SoKeyboardEvent *keyboardEvent;
   bool ctrl = event->wasCtrlDown();
   bool shift = event->wasShiftDown();

   if (event->isOfType(SoKeyboardEvent::getClassTypeId())) {
      keyboardEvent = (SoKeyboardEvent*) event;

      if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                            SoKeyboardEvent::F10)) {
         if (ctrl) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;
        } else {
            if(shift) {
                se.updateBeaconGraph(bool(true), 0,
                    getDistanceToAlarm(0), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 0;
            } else {
                setCameraToAlarm(0);
            }
        }
        return TRUE;
    }
    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                          SoKeyboardEvent::NUMBER_0)) {
        if (ctrl) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;

        } else {
            if(shift) {
                se.updateBeaconGraph(bool(true), 0,
                    getDistanceToAlarm(0), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 0;
            } else {
                setCameraToAlarm(0);
            }
        }
        return FALSE;
    }
    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                          SoKeyboardEvent::NUMBER_1)) {
        if (ctrl) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;

        } else {
            if(shift) {
                se.updateBeaconGraph(bool(true), 1,
                    getDistanceToAlarm(1), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 1;
            } else {
                setCameraToAlarm(1);
            }
        }
        return TRUE;
    }
    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                          SoKeyboardEvent::F1)) {
        if (ctrl) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;

        } else {
            if(shift) {
                se.updateBeaconGraph(bool(true), 1,
                    getDistanceToAlarm(1), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 1;
            } else {
                setCameraToAlarm(1);
            }
        }
        return TRUE;
    }

    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                          SoKeyboardEvent::NUMBER_2)) {
        if (event->wasCtrlDown()) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;

        } else {
            if(event->wasShiftDown()) {
                se.updateBeaconGraph(bool(true), 2,
                    getDistanceToAlarm(2), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 2;
            } else {
                setCameraToAlarm(2);
            }
        }
        return TRUE;
    }
    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                    SoKeyboardEvent::F2)) {
        if (event->wasCtrlDown()) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;

        } else {
            if(event->wasShiftDown()) {
                se.updateBeaconGraph(bool(true), 2,
                    getDistanceToAlarm(2), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 2;
            } else {
                setCameraToAlarm(2);
            }
        }
        return TRUE;
    }
    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                    SoKeyboardEvent::NUMBER_3)) {
        if (event->wasCtrlDown()) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;

        } else {
            if(event->wasShiftDown()) {
                se.updateBeaconGraph(bool(true), 3,
                    getDistanceToAlarm(3), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 3;
            } else {
                setCameraToAlarm(3);
            }
        }
        return TRUE;

    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent,
                                    SoKeyboardEvent::F3)) {
        if (event->wasCtrlDown()) {
            se.beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
            se.beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
            BEACON_ACTIVE = -1;

        } else {
            if(event->wasShiftDown()) {
                se.updateBeaconGraph(bool(true), 3,
                    getDistanceToAlarm(3), 0.0f, 0.0f, 0.0f);
                BEACON_ACTIVE = 3;
            } else {
                setCameraToAlarm(3);
            }
        }
        return TRUE;

    }
    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent, SoKeyboardEvent::S)) {
        if (POPUP_ACTIVE) {
            ge.updatePopup(bool(false), "", "", -1);
            ge.updateIncidentElements(bool(false), gotoAlarm);
            POPUP_ACTIVE = false;
            alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id] = 0;
            updateAlarmList(incidentCount);
            updateStatus();
            return TRUE;
        }
    }
    if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent, SoKeyboardEvent::G)) {
        if (POPUP_ACTIVE) {
            setCameraToAlarm(gotoAlarm);
            ge.updatePopup(bool(false), "", "", -1);
            POPUP_ACTIVE = false;
            return TRUE;
         }
      }
      if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent, SoKeyboardEvent::C)) {
         if (POPUP_ACTIVE) {
            //If a notification, it can be "[C]onfirmed".
            ge.updateIncidentElements(bool(false), gotoAlarm);
            ge.updatePopup(bool(false), "", "", -1);
            POPUP_ACTIVE = false;
            alarmActiveMap[ag.incidentArray[ag.incidentCount-1].id] = 0;
            updateAlarmList(incidentCount);
            updateStatus();
         } else if (ALARM_BOARD_ACTIVE) {
            ge.updateAlarmBoard(bool(false), "", "", "", "");
            ALARM_BOARD_ACTIVE = false;
         }
      }
      if (SoKeyboardEvent::isKeyPressEvent (keyboardEvent, SoKeyboardEvent::F)) {
         if (ALARM_BOARD_ACTIVE) {
            ge.updateIncidentElements(bool(false), alarmBoardActive);
            ge.updateAlarmBoard(bool(false), "", "", "", "");
            alarmActiveMap[alarmBoardActive] = 0;
            updateAlarmList(incidentCount);
            ALARM_BOARD_ACTIVE = false;
            updateStatus();
            return TRUE;
         }
      }
   }

   if (event->isOfType(SoMouseButtonEvent::getClassTypeId())) {
      mouseButtonEvent = (SoMouseButtonEvent*) event;

      if(SoMouseButtonEvent::isButtonPressEvent (
         mouseButtonEvent, SoMouseButtonEvent::BUTTON1)) {

         eventPositionDown = event->getPosition();
      } else if(SoMouseButtonEvent::isButtonReleaseEvent (
         mouseButtonEvent, SoMouseButtonEvent::BUTTON1)) {

         eventPositionUp = event->getPosition();
         if (abs(eventPositionDown[0] - eventPositionUp[0]) < 3) {
            //Ray picking
            SoRayPickAction rayPicker( currviewer->getViewportRegion() );
            rayPicker.setPoint(event->getPosition());
            rayPicker.apply(currviewer->getSceneGraph());
            //Finding object
            SoFullPath * pathToPickedObject;
            const SoPickedPoint *pickedPoint = rayPicker.getPickedPoint();
            if (pickedPoint != NULL) {
               pathToPickedObject = (SoFullPath*) pickedPoint->getPath();
               SoNode * node = pathToPickedObject->getTail();
               if (node->getName() == SbName("amex_alarm_object_0")) {
                  printf("Alarm object #0 picked\n");
                  char alarmInfo1[50];
                  char alarmInfo2[50];
                  sprintf(alarmInfo1,
                     " Area:  %i                                    ",
                     ag.incidentArray[alarmObjectMap[0]].area);
                  sprintf(alarmInfo2,
                     " Value: %i                                    ",
                     ag.incidentArray[alarmObjectMap[0]].value);
                  alarmBoardActive = 0;
                  ALARM_BOARD_ACTIVE = true;
                  ge.updateAlarmBoard(bool(true),
                     ag.incidentArray[alarmObjectMap[0]].description2,
                     alarmInfo1,
                     alarmInfo2,
                     " <!! SAFETY CRITICAL !!>");
               } else if (node->getName() == SbName("amex_alarm_object_1")) {
                  printf("Alarm object #1 picked\n");
                  char alarmInfo1[50];
                  char alarmInfo2[50];
                  sprintf(alarmInfo1,
                     " Area:  %i                                    ",
                     ag.incidentArray[alarmObjectMap[1]].area);
                  sprintf(alarmInfo2,
                     " Value: %i                                    ",
                     ag.incidentArray[alarmObjectMap[1]].value);
                  alarmBoardActive = 1;
                  ALARM_BOARD_ACTIVE = true;
```

```
                        ge.updateAlarmBoard(bool(true),
                            ag.incidentArray[alarmObjectMap[1]].description2,
                            alarmInfo1,
                            alarmInfo2,
                            "[Urgency: MED]                [Impact: MED]");
                    } else if (node->getName() == SbName("amex_alarm_object_2")) {
                        printf("Alarm object #2 picked\n");
                        char alarmInfo1[50];
                        char alarmInfo2[50];
                        sprintf(alarmInfo1,
                            " Area:  %i                                   ",
                            ag.incidentArray[alarmObjectMap[2]].area);
                        sprintf(alarmInfo2,
                            " Value: %i                                   ",
                            ag.incidentArray[alarmObjectMap[2]].value);
                        alarmBoardActive = 2;
                        ALARM_BOARD_ACTIVE = true;
                        ge.updateAlarmBoard(bool(true),
                            ag.incidentArray[alarmObjectMap[2]].description2,
                            alarmInfo1,
                            alarmInfo2,
                            "[Urgency: HIGH]               [Impact: HIGH]");
                    } else if (node->getName() == SbName("amex_alarm_object_3")) {
                        printf("Alarm object #3 picked\n");
                        char alarmInfo1[50];
                        char alarmInfo2[50];
                        sprintf(alarmInfo1,
                            " Area:  %i                                   ",
                            ag.incidentArray[alarmObjectMap[3]].area);
                        sprintf(alarmInfo2,
                            " Value: %i                                   ",
                            ag.incidentArray[alarmObjectMap[3]].value);
                        alarmBoardActive = 3;
                        ALARM_BOARD_ACTIVE = true;
                        ge.updateAlarmBoard(bool(true),
                            ag.incidentArray[alarmObjectMap[3]].description2,
                            alarmInfo1,
                            alarmInfo2,
                            "<Notification>");
                    }
                }
            }
        }
    }
    return FALSE;
}

/**
* Initialize class. The method is executed before scenegraph is loaded.
*/
void amex_init() {
    //Add Eventgrabber
    add_event_grabber(amex_event_cb, NULL);

    add_guile_func(gh_new_procedure0_0, "amex-post-init",
                   g2k_amex_post_init);

    incidentCount = 0;
}
```

G.2 amex.h

Listings of the sourcecode of amex.h.

```
#ifndef AMEX_H
#define AMEX_H

#include "../guile/guile.h"
#include <Inventor/SbBasic.h>
#include <Inventor/lists/SbList.h>
#include <Inventor/SbVec3d.h>
#include <Inventor/SbRotation.h>
#include <Inventor/nodes/SoSelection.h>

//Functions
SbBool amex_post_init(void);
void amex_init();
void retrieveAlarm(void * data, SoSensor * sensor);
void updateAlarmList(int alarmCount);
void updateStatus();
void updateBeacon(void * data, SoSensor * sensor);
void checkStatusSound(void * data, SoSensor * sensor);
float getDistanceToAlarm(int id);
void setCameraToAlarm(int alarmId);
static SbBool amex_event_cb(const SoEvent * event, void * closure);

#endif // AMEX_H
```

G.3 amexSoundEngine.h

Listings of the sourcecode of amexSoundEngine.h.

```
#ifndef AMEXSOUNDENGINE_H
#define AMEXSOUNDENGINE_H

#pragma once

#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/sensors/SoTimerSensor.h>
#include <Inventor/nodes/SoSwitch.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/VRMLnodes/SoVRMLAudioClip.h>
#include <Inventor/VRMLnodes/SoVRMLSound.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoRotor.h>
#include <stdlib.h>
#include <ctime>
#include <algorithm>

/**
*  A class generating and updating audio elements
*/
class amexSoundEngine {

//variables
int statusLevel;  /** < statuslevel of facility*/

//functions
public:
   /**
   * Constructor
   */
   amexSoundEngine(void) {}
   /**
   * Destructor
   */
   ~amexSoundEngine(void){ }

   SoSeparator *soundSwitchSep;      /** < Switch for all alarm sounds */
   SoSeparator *beaconSwitchSep;     /** < Switch for all beacon sounds */

   SoSwitch *soundSwitch0;           /** < Switch for alarm sound #0 */
      SoSeparator *soundSep0;
      SoVRMLSound *soundVRMLSound0;
      SoTransform *soundTrans0;
   SoSwitch *beaconSwitch0;          /** < Switch for beacon sound #0 */
      SoSeparator *beaconSep0;
      SoVRMLAudioClip *beaconClip0;
      SoVRMLSound *beaconVRMLSound0;
      SoTransform *beaconTrans0;
   SoSwitch *soundSwitch1;           /** < Switch for alarm sound #1 */
      SoSeparator *soundSep1;
      SoVRMLSound *soundVRMLSound1;
      SoTransform *soundTrans1;
   SoSwitch *beaconSwitch1;          /** < Switch for beacon sound #1 */
      SoSeparator *beaconSep1;
      SoVRMLAudioClip *beaconClip1;
      SoVRMLSound *beaconVRMLSound1;
      SoTransform *beaconTrans1;
   SoSwitch *soundSwitch2;           /** < Switch for alarm sound #2 */
      SoSeparator *soundSep2;
      SoVRMLSound *soundVRMLSound2;
      SoTransform *soundTrans2;
   SoSwitch *beaconSwitch2;          /** < Switch for beacon sound #2 */
      SoSeparator *beaconSep2;
      SoVRMLAudioClip *beaconClip2;
      SoVRMLSound *beaconVRMLSound2;
      SoTransform *beaconTrans2;
   SoSwitch *soundSwitch3;           /** < Switch for alarm sound #3 */
      SoSeparator *soundSep3;
      SoVRMLSound *soundVRMLSound3;
      SoTransform *soundTrans3;
   SoSwitch *beaconSwitch3;          /** < Switch for beacon sound #3 */
      SoSeparator *beaconSep3;
      SoVRMLAudioClip *beaconClip3;
      SoVRMLSound *beaconVRMLSound3;
      SoTransform *beaconTrans3;

   SoSeparator *statusAllSep;        /** < Separator for status sounds */
```

```
SoSwitch *statusSwitch0;          /** < Switch for status sound #0*/
   SoSeparator *statusSep0;
   SoVRMLSound *statusVRMLSound0;
   SoVRMLAudioClip * statusClip0;
   SoTransform *statusTrans0;
SoSwitch *statusSwitch1;          /** < Switch for status sound #1*/
   SoSeparator *statusSep1;
   SoVRMLSound *statusVRMLSound1;
   SoVRMLAudioClip * statusClip1;
   SoTransform *statusTrans1;
SoSwitch *statusSwitch2;          /** < Switch for status sound #2*/
   SoSeparator *statusSep2;
   SoVRMLAudioClip * statusClip2;
   SoVRMLSound *statusVRMLSound2;
   SoTransform *statusTrans2;

/**
* Set status level
* @param level new status level
*/
void setStatus(int level) {
   statusLevel = level;
}

/**
* Get status level
* @return current level
*/
int getStatus() {
   return statusLevel;
}

/**
* Initialize sound engine and related nodes
* @param level inital status level
* @param rootSeparator root of scenegraph
*/
void initialize(int level, SoSeparator * rootSeparator) {
   printf("Initializing soundEngine..\n");
   statusLevel = level;

   soundSwitch0 = new SoSwitch;
   soundSep0 = new SoSeparator;
   beaconSwitch0 = new SoSwitch;
   beaconSep0 = new SoSeparator;

   soundSwitch1 = new SoSwitch;
   soundSep1 = new SoSeparator;
   beaconSwitch1 = new SoSwitch;
   beaconSep1 = new SoSeparator;

   soundSwitch2 = new SoSwitch;
   soundSep2 = new SoSeparator;
   beaconSwitch2 = new SoSwitch;
   beaconSep2 = new SoSeparator;

   soundSwitch3 = new SoSwitch;
   soundSep3 = new SoSeparator;
   beaconSwitch3 = new SoSwitch;
   beaconSep3 = new SoSeparator;

   soundSwitchSep = new SoSeparator;
      soundSwitchSep->addChild(soundSep0);
         soundSep0->addChild(soundSwitch0);
      soundSwitchSep->addChild(soundSep1);
         soundSep1->addChild(soundSwitch1);
      soundSwitchSep->addChild(soundSep2);
         soundSep2->addChild(soundSwitch2);
      soundSwitchSep->addChild(soundSep3);
         soundSep3->addChild(soundSwitch3);
   beaconSwitchSep = new SoSeparator;
      beaconSwitchSep->addChild(beaconSep0);
         beaconSep0->addChild(beaconSwitch0);
      beaconSwitchSep->addChild(beaconSep1);
         beaconSep1->addChild(beaconSwitch1);
      beaconSwitchSep->addChild(beaconSep2);
         beaconSep2->addChild(beaconSwitch2);
      beaconSwitchSep->addChild(beaconSep3);
         beaconSep3->addChild(beaconSwitch3);

   //Status
      statusSwitch0 = new SoSwitch;
      statusSep0 = new SoSeparator;
```

```
        statusSwitch1 = new SoSwitch;
        statusSep1 = new SoSeparator;
        statusSwitch2 = new SoSwitch;
        statusSep2 = new SoSeparator;

        statusAllSep = new SoSeparator;
           statusAllSep->addChild(statusSep0);
              statusSep0->addChild(statusSwitch0);
           statusAllSep->addChild(statusSep1);
              statusSep1->addChild(statusSwitch1);
           statusAllSep->addChild(statusSep2);
              statusSep2->addChild(statusSwitch2);

    //Adding all to root
        rootSeparator->addChild(statusAllSep);
        rootSeparator->addChild(soundSwitchSep);
        rootSeparator->addChild(beaconSwitchSep);
}

/**
* Builds scenegraph for status sound, using nodes initialized
* in soundEngine::initialize()
* @see initialize()
*/
void buildStatusGraph() {

    //Status 0
        //Transform
           statusTrans0 = new SoTransform;
        //AudioClip
        statusClip0 = new SoVRMLAudioClip;
           statusClip0->loop = false;
           statusClip0->url = "clips/statuslevel_0.wav";
           statusClip0->startTime = SbTime::getTimeOfDay();
        //Sound
        statusVRMLSound0 = new SoVRMLSound;
           statusVRMLSound0->intensity=0.25f;
           statusVRMLSound0->spatialize=false;
           statusVRMLSound0->maxFront=1000.0f;
           statusVRMLSound0->maxBack=1000.0f;
           statusVRMLSound0->minFront=900.0f;
           statusVRMLSound0->minBack=900.0f;
           statusVRMLSound0->source = statusClip0;
        //Add to switch
           statusSwitch0->addChild(statusTrans0);
           statusSwitch0->addChild(statusVRMLSound0);

    //Status 1
        //Transform
           statusTrans1 = new SoTransform;
        //AudioClip
        statusClip1 = new SoVRMLAudioClip;
           statusClip1->loop = false;
           statusClip1->url = "clips/statuslevel_1.wav";
           statusClip1->startTime = SbTime::getTimeOfDay();
        //Sound
        statusVRMLSound1 = new SoVRMLSound;
           statusVRMLSound1->intensity=0.25f;
           statusVRMLSound1->spatialize=false;
           statusVRMLSound1->maxFront=1000.0f;
           statusVRMLSound1->maxBack=1000.0f;
           statusVRMLSound1->minFront=900.0f;
           statusVRMLSound1->minBack=900.0f;
           statusVRMLSound1->source = statusClip1;
        //Add to switch
           statusSwitch1->addChild(statusTrans1);
           statusSwitch1->addChild(statusVRMLSound1);

    //Status 2
        //Transform
           statusTrans2 = new SoTransform;
        //AudioClip
        SoVRMLAudioClip * statusClip2 = new SoVRMLAudioClip;
           statusClip2->loop = false;
           statusClip2->url = "clips/statuslevel_2.wav";
           statusClip2->startTime = SbTime::getTimeOfDay();
        //Sound
        statusVRMLSound2 = new SoVRMLSound;
           statusVRMLSound2->intensity=0.5f;
           statusVRMLSound2->spatialize=false;
           statusVRMLSound2->maxFront=1000.0f;
           statusVRMLSound2->maxBack=1000.0f;
           statusVRMLSound2->minFront=900.0f;
           statusVRMLSound2->minBack=900.0f;
```

```
            statusVRMLSound2->source = statusClip2;
        //Add to switch
            statusSwitch2->addChild(statusTrans2);
            statusSwitch2->addChild(statusVRMLSound2);

    //Adding to parent
        statusSwitch0->whichChild.setValue(SO_SWITCH_NONE);
        statusSwitch1->whichChild.setValue(SO_SWITCH_NONE);
        statusSwitch2->whichChild.setValue(SO_SWITCH_NONE);
}

/**
* Updates status sound position and type of sound to be played
* @param statusLevel current status level. Used to decide which sound
* @param x x-position
* @param y y-position
* @param z z-position
* @see buildStatusGraph()
*/
void updateStatusGraph(int statusLevel, float x, float y, float z) {
    switch(statusLevel) {
        case 0:
            statusSwitch1->whichChild.setValue(SO_SWITCH_NONE);
            statusSwitch2->whichChild.setValue(SO_SWITCH_NONE);

            statusTrans0->translation.setValue(x, y, z);
            statusClip0->startTime = SbTime::getTimeOfDay();
            statusSwitch0->whichChild.setValue(SO_SWITCH_ALL);
            break;
        case 1:
            statusSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            statusSwitch2->whichChild.setValue(SO_SWITCH_NONE);

            statusTrans1->translation.setValue(x, y, z);
            statusSwitch1->whichChild.setValue(SO_SWITCH_ALL);
            statusClip1->startTime = SbTime::getTimeOfDay();
            break;
        case 2:
            statusSwitch0->whichChild.setValue(SO_SWITCH_NONE);
            statusSwitch1->whichChild.setValue(SO_SWITCH_NONE);

            statusClip0->startTime = SbTime::getTimeOfDay();
            statusTrans2->translation.setValue(x, y, z);
            statusSwitch2->whichChild.setValue(SO_SWITCH_ALL);
            break;
    }
}


/**
* Builds scenegraph for beacon sound, using nodes initialized
* in soundEngine::initialize()
* @see initialize()
*/
void buildBeaconGraph() {
    //BEACON0
        //Transform
            beaconTrans0 = new SoTransform;
        //AudioClip
            beaconClip0 = new SoVRMLAudioClip;
                beaconClip0->loop = TRUE;
                beaconClip0->url = "clips/incident_beacon.wav";
                beaconClip0->startTime = SbTime::getTimeOfDay();
        //Sound
            beaconVRMLSound0 = new SoVRMLSound;
                beaconVRMLSound0->intensity=1.2f;
                beaconVRMLSound0->source = beaconClip0;
        //Adding to Switch
            beaconSwitch0->addChild(beaconTrans0);
            beaconSwitch0->addChild(beaconVRMLSound0);

    //BEACON1
        //Transform
            beaconTrans1 = new SoTransform;
        //AudioClip
            beaconClip1 = new SoVRMLAudioClip;
                beaconClip1->loop = TRUE;
                beaconClip1->url = "clips/incident_beacon.wav";
                beaconClip1->startTime = SbTime::getTimeOfDay();
        //Sound
            beaconVRMLSound1 = new SoVRMLSound;
                beaconVRMLSound1->intensity=1.2f;
                beaconVRMLSound1->source = beaconClip1;
        //Adding to Switch
```

```
            beaconSwitch1->addChild(beaconTrans1);
            beaconSwitch1->addChild(beaconVRMLSound1);

    //BEACON2
        //Transform
            beaconTrans2 = new SoTransform;
        //AudioClip
            beaconClip2 = new SoVRMLAudioClip;
                beaconClip2->loop = TRUE;
                beaconClip2->url = "clips/incident_beacon.wav";
                beaconClip2->startTime = SbTime::getTimeOfDay();
        //Sound
            beaconVRMLSound2 = new SoVRMLSound;
                beaconVRMLSound2->intensity=1.2f;
                beaconVRMLSound2->source = beaconClip2;
        //Adding to Switch
            beaconSwitch2->addChild(beaconTrans2);
            beaconSwitch2->addChild(beaconVRMLSound2);

    //BEACON3
        //Transform
            beaconTrans3 = new SoTransform;
        //AudioClip
            beaconClip3 = new SoVRMLAudioClip;
                beaconClip3->loop = TRUE;
                beaconClip3->url = "clips/incident_beacon.wav";
                beaconClip3->startTime = SbTime::getTimeOfDay();
        //Sound
            beaconVRMLSound3 = new SoVRMLSound;
                beaconVRMLSound3->intensity=1.2f;
                beaconVRMLSound3->source = beaconClip3;
        //Adding to Switch
            beaconSwitch3->addChild(beaconTrans3);
            beaconSwitch3->addChild(beaconVRMLSound3);
}

/**
* Builds scenegraph for beacon sound, using nodes initialized
* in soundEngine::initialize()
* @param id beacon-id
* @param pitch basis for pitch
* @param distance users distance to beacon
*/
void updateBeaconPitch(int id, float pitch, float distance) {
    float pitchModified = ((pitch/3.14f)*-1.0f)+2.0f;
    switch(id) {
        case 0:
            beaconClip0->pitch = pitchModified;
            if (distance > 75) {
                beaconVRMLSound0->maxFront = distance * 1.1f;
                beaconVRMLSound0->maxBack = distance * 1.1f;
            } else {
                beaconVRMLSound0->maxFront = 75.0f * 1.1f;
                beaconVRMLSound0->maxBack = 75.0f * 1.1f;
            }
            break;
        case 1:
            beaconClip1->pitch = pitchModified;
            if (distance >75) {
                beaconVRMLSound1->maxFront = distance * 1.1f;
                beaconVRMLSound1->maxBack = distance * 1.1f;
            } else {
                beaconVRMLSound1->maxFront =75.0f * 1.1f;
                beaconVRMLSound1->maxBack =75.0f * 1.1f;
            }
            break;
        case 2:
            beaconClip2->pitch = pitchModified;
            if (distance >75) {
                beaconVRMLSound2->maxFront = distance * 1.1f;
                beaconVRMLSound2->maxBack = distance * 1.1f;
            } else {
                beaconVRMLSound2->maxFront =75.0f * 1.1f;
                beaconVRMLSound2->maxBack =75.0f * 1.1f;
            }
            break;
        case 3:
            beaconClip3->pitch = pitchModified;
            if (distance >75) {
                beaconVRMLSound3->maxFront = distance * 1.1f;
                beaconVRMLSound3->maxBack = distance * 1.1f;
            } else {
                beaconVRMLSound3->maxFront =75.0f * 1.1f;
                beaconVRMLSound3->maxBack =75.0f * 1.1f;
            }
            break;

        }
    }

    /**
    * Updates beacon sound position, amplitude and whether it is played or not
    * @param enable enables the specific beacon sound
    * @param id sound-ID (e.g. alarm-id)
    * @param distance distance from beacon to camera
    * @param x x-position
    * @param y y-position
    * @param z z-position
    * @see buildStatusGraph()
    */
    void updateBeaconGraph(bool enable, int id, float distance,
                                     float x, float y, float z) {
        printf("Distance: %f\n", distance);
        switch(id) {
            case 0:
                if(enable) {
                    beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
                    beaconVRMLSound0->maxFront = distance * 1.1f;
                    beaconVRMLSound0->maxBack = distance * 1.1f;
                    beaconVRMLSound0->minFront = 4.0f;
                    beaconVRMLSound0->minBack = 4.0f;
                    beaconSwitch0->whichChild.setValue(SO_SWITCH_ALL);
                    printf("ID: %i - GO!\n", id);
                } else {
                    beaconTrans0->translation.setValue(x, y+0.2f, z);
                }
                break;

            case 1:
                if(enable) {
                    beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
                    beaconVRMLSound1->maxFront = distance * 1.1f;
                    beaconVRMLSound1->maxBack = distance * 1.1f;
                    beaconVRMLSound1->minFront = 4.0f;
                    beaconVRMLSound1->minBack = 4.0f;
                    beaconSwitch1->whichChild.setValue(SO_SWITCH_ALL);
                    printf("ID: %i - GO!\n", id);
                } else {
                    beaconTrans1->translation.setValue(x, y+0.2f, z);
                }
                break;

            case 2:
                if(enable) {
                    beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch3->whichChild.setValue(SO_SWITCH_NONE);
                    beaconVRMLSound2->maxFront = distance * 1.1f;
                    beaconVRMLSound2->maxBack = distance * 1.1f;
                    beaconVRMLSound2->minFront = 4.0f;
                    beaconVRMLSound2->minBack = 4.0f;
                    beaconSwitch2->whichChild.setValue(SO_SWITCH_ALL);
                    printf("ID: %i - GO!\n", id);
                } else {
                    beaconTrans2->translation.setValue(x, y+0.2f, z);
                }
                break;

            case 3:
                if(enable) {
                    beaconSwitch0->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch1->whichChild.setValue(SO_SWITCH_NONE);
                    beaconSwitch2->whichChild.setValue(SO_SWITCH_NONE);
                    beaconVRMLSound3->maxFront = distance * 1.1f;
                    beaconVRMLSound3->maxBack = distance * 1.1f;
                    beaconVRMLSound3->minFront = 4.0f;
                    beaconVRMLSound3->minBack = 4.0f;
                    beaconSwitch3->whichChild.setValue(SO_SWITCH_ALL);
                    printf("ID: %i - GO!\n", id);
                } else {
                    beaconTrans3->translation.setValue(x, y+0.2f, z);
                }
                break;
    }
}


/**
* Builds scenegraph for alarm sounds, using nodes initialized
* in soundEngine::initialize()
* @see initialize()
*/
void buildSoundGraph() {

    //Sound for alarm#0
        //Transform
        soundTrans0 = new SoTransform;
        //AudioClip
        SoVRMLAudioClip * soundClip0 = new SoVRMLAudioClip;
            soundClip0->loop = false;
            soundClip0->url = "../common/models/clips/safety_critical.wav";
            soundClip0->startTime = SbTime::getTimeOfDay();
        //Sound
        soundVRMLSound0 = new SoVRMLSound;
            soundVRMLSound0->source = soundClip0;
            soundVRMLSound0->intensity=1.5f;
        //Adding to Switch
        soundSwitch0->addChild(soundTrans0);
        soundSwitch0->addChild(soundVRMLSound0);

    //Sound for alarm#1
        //Transform
        soundTrans1 = new SoTransform;
        //AudioClip
        SoVRMLAudioClip * soundClip1 = new SoVRMLAudioClip;
            soundClip1->loop = false;
            soundClip1->url = "../common/models/clips/alarm03.wav";
            soundClip1->startTime = SbTime::getTimeOfDay();
        //Sound
        soundVRMLSound1 = new SoVRMLSound;
            soundVRMLSound1->source = soundClip1;
            soundVRMLSound1->intensity=1.3f;
        //Adding to Switch
        soundSwitch1->addChild(soundTrans1);
        soundSwitch1->addChild(soundVRMLSound1);

    //Sound for alarm#2
        //Transform
        soundTrans2 = new SoTransform;
        //AudioClip
        SoVRMLAudioClip * soundClip2 = new SoVRMLAudioClip;
            soundClip2->loop = false;
            soundClip2->url = "../common/models/clips/alarm01.wav";
            soundClip2->startTime = SbTime::getTimeOfDay();
        //Sound
        soundVRMLSound2 = new SoVRMLSound;
            soundVRMLSound2->source = soundClip2;
            soundVRMLSound2->intensity=1.4f;
        //Adding to Switch
        soundSwitch2->addChild(soundTrans2);
        soundSwitch2->addChild(soundVRMLSound2);

    //Sound for alarm#3
        //Transform
        soundTrans3 = new SoTransform;
        //AudioClip
        SoVRMLAudioClip * soundClip3 = new SoVRMLAudioClip;
            soundClip3->loop = false;
            soundClip3->url = "../common/models/clips/water.wav";
            soundClip3->startTime = SbTime::getTimeOfDay();
        //Sound
        soundVRMLSound3 = new SoVRMLSound;
            soundVRMLSound3->source = soundClip3;
            soundVRMLSound0->intensity=1.1f;
        //Adding to Switch
        soundSwitch3->addChild(soundTrans3);
        soundSwitch3->addChild(soundVRMLSound3);
}

/**
* Updates alarm sound position and amplitude
* @param id sound-ID (e.g. alarm-id)
* @param distance distance from beacon to camera
* @param x x-position
* @param y y-position
* @param z z-position
* @see buildStatusGraph()
```

```
    */
    void updateSoundGraph(int id, float distance, float x, float y, float z) {
        switch(id) {
            case 0:
                soundTrans0->translation.setValue(x, y, z);
                soundVRMLSound0->maxFront = std::max(75.0f, distance + 20.0f);
                soundVRMLSound0->maxBack = std::max(75.0f, distance + 20.0f);
                soundVRMLSound0->minFront = 8.0f;
                soundVRMLSound0->minBack = 8.0f;
                soundSwitch0->whichChild.setValue(SO_SWITCH_ALL);
                break;
            case 1:
                soundTrans1->translation.setValue(x, y, z);
                soundVRMLSound1->maxFront = std::max(75.0f, distance + 20.0f);
                soundVRMLSound1->maxBack = std::max(75.0f, distance + 20.0f);
                soundVRMLSound1->minFront = 8.0f;
                soundVRMLSound1->minBack = 8.0f;
                soundSwitch1->whichChild.setValue(SO_SWITCH_ALL);
                break;
            case 2:
                soundTrans2->translation.setValue(x, y, z);
                soundVRMLSound2->maxFront = std::max(75.0f, distance + 20.0f);
                soundVRMLSound2->maxBack = std::max(75.0f, distance + 20.0f);
                soundVRMLSound2->minFront = 8.0f;
                soundVRMLSound2->minBack = 8.0f;
                soundSwitch2->whichChild.setValue(SO_SWITCH_ALL);
                break;
            case 3:
                soundTrans3->translation.setValue(x, y, z);
                soundVRMLSound3->maxFront = std::max(75.0f, distance + 20.0f);
                soundVRMLSound3->maxBack = std::max(75.0f, distance + 20.0f);
                soundVRMLSound3->minFront = 8.0f;
                soundVRMLSound3->minBack = 8.0f;
                soundSwitch3->whichChild.setValue(SO_SWITCH_ALL);
                break;
        }
    }
};

#endif
```

G.4 amexGUIEngine.h

Listings of the sourcecode of amexGUIEngine.h.

```
#ifndef AMEXGUIENGINE_H
#define AMEXGUIENGINE_H

#pragma once
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoSwitch.h>
#include <Inventor/nodes/SoPointLight.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/nodes/SoLightModel.h>
#include <Inventor/nodes/SoCamera.h>
#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoTexture2.h>
#include <Inventor/nodes/SoTexture2Transform.h>

#include <Inventor/VRMLnodes/SoVRMLBox.h>
#include <Inventor/VRMLnodes/SoVRMLInline.h>
#include <Inventor/vrmlnodes/SoVRMLGroup.h>
#include <Inventor/vrmlnodes/SoVRMLtransform.h>
#include <Inventor/vrmlnodes/SoVRMLBillboard.h>

#include <Inventor/misc/SoAudioDevice.h>
#include <Inventor/VRMLnodes/SoVRMLAudioClip.h>
#include <Inventor/VRMLnodes/SoVRMLSound.h>
#include <Inventor/VRMLNodes/SoVRMLText.h>

#include <Inventor/SoDB.h>
#include <Inventor/nodes/SoText3.h>
#include <Inventor/nodes/SoText2.h>
#include <Inventor/nodes/SoAsciiText.h>
#include <Inventor/nodes/SoScale.h>

//Menus
#include "../nodes/ShadowText2.h"

/**
 *  A class generating and updating graphical elements
 */
class amexGUIEngine {
private:

public:
   SoSwitch * templateNormalSwitch;        /* Switch for opaque structure */
   SoSwitch * templateTransparentSwitch;   /* Switch for transparent struct.*/
   SoSwitch * element0_Switch;             /* Switch for alarm-object #0 */
   SoSwitch * element1_Switch;             /* Switch for alarm-object #1 */
   SoSwitch * element2_Switch;             /* Switch for alarm-object #2 */
   SoSwitch * element3_Switch;             /* Switch for alarm-object #3 */
   const char * menu[8];                   /* Container for alarm list*/
   const char * menuGrey[8];               /* Container for alarm list,grey text*/
   const char * popup[5];                  /* Containter for popup menu*/
   const char * alarm[8];                  /* Container for alarm info board*/
   int visibleAlarmElements;               /* Counter of visible alarm elements */

   /**
   * Constructor
   */
   amexGUIEngine(void) {
         menu[0] = "Time      Id   Val  Area  Desc                    ";
         menu[1] = "---------------------------------------------------";
         menu[2] = "";
         menu[3] = "";
         menu[4] = "";
         menu[5] = "";
         menu[6] = "---------------------------------------------------";
         menu[7] = "< StatusLevel 0 >";

         menuGrey[0] = "";
         menuGrey[1] = "";
         menuGrey[2] = "";
         menuGrey[3] = "";
         menuGrey[4] = "";
         menuGrey[5] = "";
         menuGrey[6] = "";
         menuGrey[7] = "";
```

```
        alarm[0] = "   !! New Alarm !!";
        alarm[1] = "---------------------------------------------";
        alarm[2] = "  Object blapp                               ";
        alarm[3] = "";
        alarm[4] = "";
        alarm[5] = "[Urgency: HIGH]                 [Impact: LOW]";
        alarm[6] = "---------------------------------------------";
        alarm[7] = "[F]ix   [T]rends    [D]ocumentation   [C]lose";

        popup[0] = "!! Pipe, low temperature !!";
        popup[1] = "--------------------------------";
        popup[2] = "Urgency:HIGH   Impact:LOW";
        popup[3] = "--------------------------------";
        popup[4] = "[G]oto [S]uppress [D]oc [T]rend";

        visibleAlarmElements = 0;
    }
    /**
    * Destructor
    */
    ~amexGUIEngine(void) {}

    /**
    * Update scenegraph and sets structures to transparent if enable==true
    * @param transparent sets template/structures to transparent if true
    */
    void updateSceneGraph(bool transparent) {
        if(transparent==true) {
            templateNormalSwitch->whichChild.setValue(SO_SWITCH_NONE);
            templateTransparentSwitch->whichChild.setValue(SO_SWITCH_ALL);
        } else {
            templateNormalSwitch->whichChild.setValue(SO_SWITCH_ALL);
            templateTransparentSwitch->whichChild.setValue(SO_SWITCH_NONE);
        }
    }

    /**
     * Update alarm information board
     * @param enable enable alarm info board
     * @param string1 textline#0 in alarm info board
     * @param string2 textline#2 in alarm info board
     * @param string3 textline#3 in alarm info board
     * @param string4 textline#5 in alarm info board
     */
    void updateAlarmBoard(bool enable, const char string1[],
                               const char string2[],
                               const char string3[],
                               const char string4[]) {
        SoSwitch * AMEX_ALARM_BOARD_SWITCH =
            (SoSwitch *) SoNode::getByName("amex_alarm_overlay_switch");
        if(enable) {
            ShadowText2 * AMEX_ALARM_BOARD_TEXT = (
                ShadowText2 *) SoNode::getByName("amex_alarm_board_text");
                alarm[0]=string1;
                alarm[2]=string2;
                alarm[3]=string3;
                alarm[5]=string4;

            AMEX_ALARM_BOARD_TEXT->string.setValues(0, 8, alarm);
            AMEX_ALARM_BOARD_SWITCH->whichChild.setValue(SO_SWITCH_ALL);
        } else {
            AMEX_ALARM_BOARD_SWITCH->whichChild.setValue(SO_SWITCH_NONE);
        }
    }


    /**
     * Update popup
     * @param enable enable popup box
     * @param heading heading of popup box
     * @param string text string of popup box
     */
    void updatePopup(bool enable, const char heading[],
                    const char string[], int criticality) {
        SoSwitch * AMEX_POPUP_SWITCH =
            (SoSwitch *) SoNode::getByName("amex_popup_overlay_switch");
        if(enable) {
            ShadowText2 * AMEX_POPUP_TEXT =
                (ShadowText2 *) SoNode::getByName("amex_popup_text");
            popup[0] = heading;
            popup[2] = string;

            switch(criticality) {
```

```
        case 0:
              popup[4] = "[G]oto [C]onfirm [D]oc [T]rend";
           break;
        case 1:
        case 2:
        case 3:
           popup[4] = "[G]oto [S]uppress [D]oc [T]rend";
           break;
        case 99:
           popup[4] = "[G]oto [D]oc [T]rend";
           break;
        }

        AMEX_POPUP_TEXT->string.setValues(0, 5, popup);
        AMEX_POPUP_SWITCH->whichChild.setValue(SO_SWITCH_ALL);

     } else {
        AMEX_POPUP_SWITCH->whichChild.setValue(SO_SWITCH_NONE);
     }
  }

  /**
   * Update alarm objects and superimposed information board
   * @param enable enable showing of object
   * @param id alarm-ID
   */
  void updateAlarmElements(bool enable, int id) {
     switch(id) {
        case 0:
           if (enable) {
              element0_Switch->whichChild.setValue(SO_SWITCH_ALL);
              visibleAlarmElements++;
           } else {
              element0_Switch->whichChild.setValue(SO_SWITCH_NONE);
              visibleAlarmElements--;
           }
           break;
        case 1:
           if (enable) {
              element1_Switch->whichChild.setValue(SO_SWITCH_ALL);
              visibleAlarmElements++;
           } else {
              element1_Switch->whichChild.setValue(SO_SWITCH_NONE);
              visibleAlarmElements--;
           }
           break;
        case 2:
           if (enable) {
              element2_Switch->whichChild.setValue(SO_SWITCH_ALL);
              visibleAlarmElements++;
           } else {
              element2_Switch->whichChild.setValue(SO_SWITCH_NONE);
              visibleAlarmElements--;
           }
           break;
        case 3:
           if (enable) {
              element3_Switch->whichChild.setValue(SO_SWITCH_ALL);
              visibleAlarmElements++;
           } else {
              element3_Switch->whichChild.setValue(SO_SWITCH_NONE);
              visibleAlarmElements--;
           }

           break;
     }
     printf("Number of alarm elements %i\n", visibleAlarmElements);
     if (visibleAlarmElements == 1) {
        //Make surrounding structure transparent
        updateSceneGraph(bool(true));
     } else if (visibleAlarmElements == 0) {
        updateSceneGraph(bool(false));
     }
  }

  /**
   * Update list of alarms
   * @param line line number [0, 4], [7]
   * @param string1 textline (see code for details)
   * @param string2 textline (see code for details)
   * @param string3 textline (see code for details)
   * @param string4 textline (see code for details)
   * @param active1 trigger (see code for details)
   * @param active2 trigger (see code for details)
```

```
 * @param active3 trigger (see code for details)
 * @param active4 trigger (see code for details)
 */
void updateAlarmText(int line, const char string1[], int active1,
                     const char string2[], int active2,
                     const char string3[], int active3,
                     const char string4[], int active4) {
   ShadowText2 * AMEX_ALARMLIST_TEXT =
      (ShadowText2 *) SoNode::getByName("amex_menu_text");
   ShadowText2 * AMEX_ALARMLIST_TEXT_GREY =
      (ShadowText2 *) SoNode::getByName("amex_menu_text_grey");

   switch(line) {
      case 1:
         if(active1) {
            menu[2]=string1;
            menuGrey[2]="";
         } else {
            menu[2]="";
            menuGrey[2]=string1;
         }
         break;
      case 2:
         if(active2) {
            menu[3]=string2;
            menuGrey[3]="";
         } else {
            menu[3]="";
            menuGrey[3]=string2;
         }

         if(active1) {
            menu[2]=string1;
            menuGrey[2]="";
         } else {
            menu[2]="";
            menuGrey[2]=string1;
         }
         break;
      case 3:
         if(active3) {
                  menu[4]=string3;
            menuGrey[4]="";
         } else {
            menu[4]="";
            menuGrey[4]=string3;
         }

         if(active2) {
            menu[3]=string2;
            menuGrey[3]="";
         } else {
            menu[3]="";
            menuGrey[3]=string2;
         }

         if(active1) {
            menu[2]=string1;
            menuGrey[2]="";
         } else {
            menu[2]="";
            menuGrey[2]=string1;
         }
         break;
      case 4:
         if(active4) {
            menu[5]=string4;
            menuGrey[5]="";
         } else {
            menu[5]="";
            menuGrey[5]=string4;
         }

         if(active3) {
                  menu[4]=string3;
            menuGrey[4]="";
         } else {
            menu[4]="";
            menuGrey[4]=string3;
         }

         if(active2) {
            menu[3]=string2;
            menuGrey[3]="";
            } else {
                menu[3]="";
                menuGrey[3]=string2;
            }

            if(active1) {
                menu[2]=string1;
                menuGrey[2]="";
            } else {
                menu[2]="";
                menuGrey[2]=string1;
            }
            break;
        case 7:
            menu[7]=string1;
            break;
    }

    AMEX_ALARMLIST_TEXT->string.setValues(0, 8, menu);
    AMEX_ALARMLIST_TEXT_GREY->string.setValues(0, 8, menuGrey);

}

void buildElementGraph(SoSeparator * rootSeparator) {
    //alarm #0
        element0_Switch = new SoSwitch();
            element0_Switch->setName("element0_Switch");
            SoSeparator * element0_Sep = new SoSeparator;
                element0_Sep->setName("element0_Sep");
                SoTransform * element0_Trans = new SoTransform;
                    element0_Trans->scaleFactor.setValue(1.0f, 1.0f, 1.0f);
                SoInput element0_in;
                SoVRMLGroup *element0_VRML = new SoVRMLGroup;
                    element0_in.openFile("element_A.wrl",FALSE);
                    element0_VRML = SoDB::readAllVRML(&element0_in);
            element0_Sep->addChild(element0_Trans);
            element0_Sep->addChild(element0_VRML);
        element0_Switch->addChild(element0_Sep);
        element0_Switch->whichChild.setValue(SO_SWITCH_NONE);

    //alarm #1
        element1_Switch = new SoSwitch();
            element1_Switch->setName("element1_Switch");
            SoSeparator * element1_Sep = new SoSeparator;
                element1_Sep->setName("element1_Sep");
                SoTransform * element1_Trans = new SoTransform;
                    element1_Trans->scaleFactor.setValue(1.0f, 1.0f, 1.0f);
                SoInput element1_in;
                SoVRMLGroup *element1_VRML = new SoVRMLGroup;
                    element1_in.openFile("element_D.wrl",FALSE);
                    element1_VRML = SoDB::readAllVRML(&element1_in);
            element1_Sep->addChild(element1_Trans);
            element1_Sep->addChild(element1_VRML);
        element1_Switch->addChild(element1_Sep);
        element1_Switch->whichChild.setValue(SO_SWITCH_NONE);

    //alarm #2
        element2_Switch = new SoSwitch();
        element2_Switch->setName("element2_Switch");
            SoSeparator * element2_Sep = new SoSeparator;
                element2_Sep->setName("element2_Sep");
                SoTransform * element2_Trans = new SoTransform;
                    element2_Trans->scaleFactor.setValue(1.0f, 1.0f, 1.0f);
                SoInput element2_in;
                SoVRMLGroup *element2_VRML = new SoVRMLGroup;
                    element2_in.openFile("element_B.wrl",FALSE);
                    element2_VRML = SoDB::readAllVRML(&element2_in);
            element2_Sep->addChild(element2_Trans);
            element2_Sep->addChild(element2_VRML);
        element2_Switch->addChild(element2_Sep);
        element2_Switch->whichChild.setValue(SO_SWITCH_NONE);

    //alarm #3
        element3_Switch = new SoSwitch();
        element3_Switch->setName("element3_Switch");
            SoSeparator * element3_Sep = new SoSeparator;
                element3_Sep->setName("element3_Sep");
                SoTransform * element3_Trans = new SoTransform;
                    element3_Trans->scaleFactor.setValue(1.0f, 1.0f, 1.0f);
                SoInput element3_in;
                SoVRMLGroup *element3_VRML = new SoVRMLGroup;
                    element3_in.openFile("element_C.wrl",FALSE);
                    element3_VRML = SoDB::readAllVRML(&element3_in);
            element3_Sep->addChild(element3_Trans);
            element3_Sep->addChild(element3_VRML);
         element3_Switch->addChild(element3_Sep);
         element3_Switch->whichChild.setValue(SO_SWITCH_NONE);

      //Adding to root
         rootSeparator->addChild(element0_Switch);
         rootSeparator->addChild(element1_Switch);
         rootSeparator->addChild(element2_Switch);
         rootSeparator->addChild(element3_Switch);
   }

   /**
    * Builds scenegraph for structures, templates, pipelines etc.
    * @param *rootSeparator root of scenegraph
    */
   void buildSceneGraph(SoSeparator * rootSeparator) {

   //Template - normal
      templateNormalSwitch = new SoSwitch();
      SoSeparator * templateSep = new SoSeparator;
            SoInput template_in;
            SoVRMLGroup *templateVRML = new SoVRMLGroup;
               template_in.openFile("template_n_12.wrl",FALSE);
               templateVRML = SoDB::readAllVRML(&template_in);
         templateSep->addChild(templateVRML);
      templateNormalSwitch->addChild(templateSep);
      templateNormalSwitch->whichChild.setValue(SO_SWITCH_ALL);

   //Template - transparent
      templateTransparentSwitch = new SoSwitch();
         SoSeparator * templateTransSep = new SoSeparator;
            SoInput templateTrans_in;
            SoVRMLGroup *templateTransVRML = new SoVRMLGroup;
             templateTrans_in.openFile("template_n_12_transparent.wrl",FALSE);
             templateTransVRML = SoDB::readAllVRML(&templateTrans_in);
         templateTransSep->addChild(templateTransVRML);
      templateTransparentSwitch->addChild(templateTransSep);
      templateTransparentSwitch->whichChild.setValue(SO_SWITCH_NONE);

   //Adding to root
      rootSeparator->addChild(templateTransparentSwitch);
      rootSeparator->addChild(templateNormalSwitch);

   }

};
#endif
```

G.5 amexAlarmGenerator.h

Listings of the sourcecode of amexAlarmGenerator.h.

```
#ifndef AMEXALARMGENERATOR_H
#define AMEXALARMGENERATOR_H

#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <vector>
#include <Inventor/sensors/SoTimerSensor.h>
#include <Inventor/SbTime.h>
#include <Inventor/SbString.h>

/**
 *  A class which generates alarms
 */
class amexAlarmGenerator {
private:
   struct alarmstruct{
      SbTime * timestamp;              // timestamp of alarm
      int id;                          // alarm id
      int value;                       // static value
      int area;                        // area
      int criticality;                 // static criticality
      char * description;              // textual description
      char * description2;             // textual description
      char * impact_urgency;           // textual description
      float position[3];               // position of alarmobject
      double utm[3];                   // UTM-position
      float location[3];               // position for Goto()
      float rotation[3];               // rotation vector for Goto()
      float radians;                   // amount roatation for Goto()
   };

   int alarmActive[5];                 /**< Array of active alarms */
   static const int MAX_ALARM = 5;     /**< Maximum number of alarms */

   /**
   * Generates a random number
    * @param lowest_number lower limit of random range
    * @param highest_number upper limit of random range
    * @return random number
    */
   int random_range(int lowest_number, int highest_number) {
      if(lowest_number > highest_number){
         std::swap(lowest_number, highest_number);
      }
      int range = highest_number - lowest_number + 1;
      return lowest_number + int(range * rand()/(RAND_MAX + 1.0));
   }

public:
   int alarmCount;                        /**< Number of active alarms */
   alarmstruct alarmArray[MAX_ALARM];     /** < Containerarray for alarm-structs */

   /**
   * Constructor
   */
   amexAlarmGenerator(void) {
      alarmCount = 0;
      for (int i = 0; i<5; i++){alarmActive[i] = 0;}
   }
   /**
   * Destructor
   */
   ~amexAlarmGenerator(void) {
   }

   /**
   * Generates an alarm and adds to alarmArray
    */
   void callback() {
      int tmpId = -1;
      int tmpValue = -1;
      int tmpArea = -1;
      int tmpCriticality = -1;
      char * tmpDescription;
      char * tmpDescription2;
    char * tmpImpact_urgency;
    float tmpPosition[3] = {0.0f, 0.0f, 0.0f};    //position of sound
    double tmpUTM[3] = {0.0, 0.0, 0.0};            //UTM position
    float tmpLocation[3] = {0.0f, 0.0f, 0.0f};    //used in goto()
    float tmpRotation[3] = {0.0f, 0.0f, 0.0f};    //used in goto()
    float tmpRadians = 0.0f;

    switch (random_range(0, 3)) {
       case 0:
          //Thick pipe
          if (alarmActive[1] == 0) {
             alarmActive[1] = 1;
             alarmCount++;
             tmpId = 0;
             tmpValue = 53;
             tmpArea = 18;
             tmpCriticality = 99;
             tmpDescription = "SCSSV failure          ";
             tmpDescription2 = "SCSSV failure";
             tmpImpact_urgency = "<!! SAFETY CRITICAL !!>";
             tmpPosition[0] = 4.871f;
             tmpPosition[1] = 1.0f;
             tmpPosition[2] = -2.871f;
             tmpUTM[0] = 502933.417353;
             tmpUTM[1] = 7931867.259912;
             tmpUTM[2] = -324.391401;
             tmpLocation[0] = 502926.0;
             tmpLocation[1] = 7931870.0;
             tmpLocation[2] = -321.297;
             tmpRotation[0] = 0.486865;
             tmpRotation[1] = -0.516464;
             tmpRotation[2] = -0.70442;
             tmpRadians = 1.96918;
          }
          break;

       case 1:
          //inlet
          if (alarmActive[3] == 0) {
             alarmActive[3] = 1;
             alarmCount++;
             tmpId = 1;
             tmpValue = 99;
             tmpArea = 18;
             tmpCriticality = 3;
             tmpDescription = "Pipe, high pressure ";
             tmpDescription2 = "Pipe,high pressure ";
             tmpImpact_urgency = "Urgency:MED     Impact:MED";
             tmpPosition[0] = 0.2519;
             tmpPosition[1] = 2.321;
             tmpPosition[2] = -21.9;
             tmpUTM[0] = 502946.561913;
             tmpUTM[1] = 7931879.642961;
             tmpUTM[2] = -322.139355;
             tmpLocation[0] = 502990.0;
             tmpLocation[1] = 7931880;
             tmpLocation[2] = -306.416;
             tmpRotation[0] = 0.538709;
             tmpRotation[1] = 0.504049;
             tmpRotation[2] = 0.675076;
             tmpRadians = 1.89165;
          }
          break;

       case 2:
          //Thin pipe
          if (alarmActive[2] == 0) {
             alarmActive[2] = 1;
             alarmCount++;
             tmpId = 2;
             tmpValue = 99;
             tmpArea = 18;
             tmpCriticality = 1;
             tmpDescription = "SCSSV failure          ";
             tmpDescription = "Pipe,high temperature";
             tmpDescription2 = "Pipe, high temperature";
             tmpImpact_urgency = "Urgency:HIGH   Impact:HIGH";
             tmpPosition[0] = -2.44;
             tmpPosition[1] = 4.8;
             tmpPosition[2] = 0.7029;
             tmpUTM[0] = 502927.024922;
             tmpUTM[1] = 7931872.017507;
             tmpUTM[2] = -321.567013;
             tmpLocation[0] = 502922.0;
```

```
                    tmpLocation[1] = 7931870;
                    tmpLocation[2] = -320.373;
                    tmpRotation[0] = 0.790075;
                    tmpRotation[1] = -0.348702;
                    tmpRotation[2] = -0.504171;
                    tmpRadians = 1.43812;
                }
                break;

            case 3:
                if (alarmActive[0] == 0) {
                    alarmActive[0] = 1;
                    alarmCount++;
                    tmpId = 3;
                    tmpValue = 55;
                    tmpArea = 18;
                    tmpCriticality = 0;
                    tmpDescription = "Hatch open             ";
                    tmpDescription2 = "Hatch open";
                    tmpImpact_urgency = "<Notification>";
                    tmpPosition[0] = 0.01124f;
                    tmpPosition[1] = 1.594f;
                    tmpPosition[2] = 11.27f;
                    tmpUTM[0] = 502920.970775;
                    tmpUTM[1] = 7931866.416976;
                    tmpUTM[2] = -322.075169;
                    tmpLocation[0] = 502909.0;
                    tmpLocation[1] = 7931890.0;
                    tmpLocation[2] = -307.689;
                    tmpRotation[0] = -0.211038;
                    tmpRotation[1] = 0.498211;
                    tmpRotation[2] = 0.840981;
                    tmpRadians = 3.82565;
                }
                break;
        }

        if (tmpArea != -1) {
            SbTime * currentTime = new SbTime(SbTime::getTimeOfDay());
            SbString string;
            string = currentTime->format("%h.%m.%s");
            this->alarmArray[alarmCount-1].timestamp =
                                         new SbTime(SbTime::getTimeOfDay());
            this->alarmArray[alarmCount-1].id = tmpId;
            this->alarmArray[alarmCount-1].value = tmpValue;
            this->alarmArray[alarmCount-1].area = tmpArea;
            this->alarmArray[alarmCount-1].criticality = tmpCriticality;
            this->alarmArray[alarmCount-1].description = tmpDescription;
            this->alarmArray[alarmCount-1].description2 = tmpDescription2;
            this->alarmArray[alarmCount-1].impact_urgency = tmpImpact_urgency;
            this->alarmArray[alarmCount-1].position[0] = tmpPosition[0];
            this->alarmArray[alarmCount-1].position[1] = tmpPosition[1];
            this->alarmArray[alarmCount-1].position[2] = tmpPosition[2];
            this->alarmArray[alarmCount-1].utm[0] = tmpUTM[0];
            this->alarmArray[alarmCount-1].utm[1] = tmpUTM[1];
            this->alarmArray[alarmCount-1].utm[2] = tmpUTM[2];
            this->alarmArray[alarmCount-1].location[0] = tmpLocation[0];
            this->alarmArray[alarmCount-1].location[1] = tmpLocation[1];
            this->alarmArray[alarmCount-1].location[2] = tmpLocation[2];
            this->alarmArray[alarmCount-1].rotation[0] = tmpRotation[0];
            this->alarmArray[alarmCount-1].rotation[1] = tmpRotation[1];
            this->alarmArray[alarmCount-1].rotation[2] = tmpRotation[2];
            this->alarmArray[alarmCount-1].radians = tmpRadians;
        } else {
            printf("\n");
        }

    }

};

#endif
```

Appendix H

Enclosed CD

The enclosed CD contains the following directories:

\Data\Models	(the 3D models used in AMEX)
\Data\Sounds	(the sounds used in AMEX)
\Data\Sourcecode	(the sourcecode of AMEX)
\Report	(this report in .pdf)