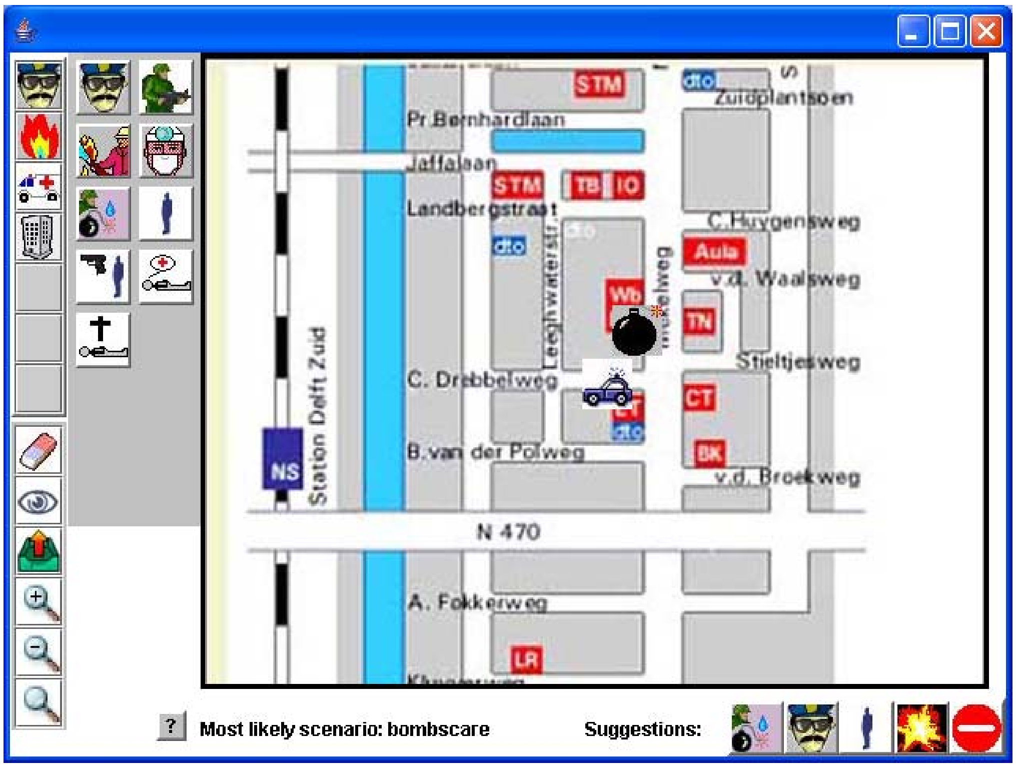
ISME

Icon based System for Managing Emergencies



Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department Mediamatica / Man-Machine-Interaction

TUDelft

Delft University of Technology

Master Thesis of:
Paul Schooneman
Student number: 9745257
Date: January 2005

Graduation committee

Dr. Drs. L.J.M. Rothkrantz
Dr. ir. C.A.P.G van der Mast
ir. F. Ververs
ir. B. Tatomir
Dr. S. Oomes

Abstract

Since the terrorist attacks on 9/11 the issue about the collaboration of emergency services has become increasingly important. One of the conclusions is that better communication between the different services is needed. This motivated us to try a different modality for communication. The modality we have chosen is a graphical one, which is very useful when only a limited set of concepts need to be represented. This thesis describes how iconic communication can be applied to the field of Emergency Management.

We have designed and implemented a prototype application, ISME, which is suited for emergency services to communicate with each other, using a map and icons. Our main focus is on the graphical user interface, and the intelligence of the system. The system is designed as a client server application, where the client is focussed on the interface and the server concentrates on the intelligence. We use a Jess knowledge and rule base to provide a consistent world model at all times, while we represent the concepts in XML files. The interface and network is implemented in Java.

ISME gives the users the possibility to report about what they observe by placing icons on a map. The maps will be send to the server, which fuses the multiple observations and constructs a new world model of it. Besides the world model, the server also sends information about the most likely scenario, and it will suggest icons that are expected in the world model but are not placed yet.

Keywords: icon, communication, map, emergency, crisis, interface, Jess, world model

Preface

This thesis is the result of my graduation project at the Man Machine Interaction group of the faculty Electrical Engineering, Mathematics and Computer Science at the Delft University of Technology. This research is done as part of the Combined Systems [Comb] group at DECIS LAB, a collaboration of Delft University of Technology, University of Amsterdam, THALES Nederland, and TNO. This collaboration is focussed on the research of decision support systems, seeking to order information in complex and chaotic situations.

Acknowledgements

First of all I would like to thank my supervisor Leon Rothkrantz for inspiring me to do my research. Our weekly sessions always brought new ideas or different approaches. He was also a great help in ordering this report.

Next I would like to thank my fellow students for always giving me useful input and assistance. They are, in no particular order: Iulia Tatomir, Marcel van Velden, Paul Klapwijk, Stefan Strijdhaftig, Jan Chau and Bogdan Tatomir (MMI staff).

Last, but certainly not least, I would like to thank my parents, sister and especially my girlfriend for always supporting me, and listening to my stories, trying to comprehend even the most technical problems I encountered.

Table of Contents

Table of Figures

Chapter 1: Introduction

Since the terrorist attacks on September 11th 2001 the issue about the collaboration of emergency services has become increasingly important. One of the conclusions is that better communication between the different services is needed. Another important issue on that day was the total breakdown of the communication infrastructure, immediately after the attack.

Because our MMI department is doing research on, among other topics, multi modal interfaces and AI, it seemed promising to develop an intelligent system with a new modality for communication, using a wireless, ad-hoc network.

Before this project was started there had been some research about chatting with icons, in particular in crisis situations [Tat03]. The basic idea was to create a laguage, which is universal, easy to use and easy to learn. Human communication is based on exchange of ideas or concepts. An idea originating in the mind of the 'sender' is first converted to a string of words. The receiver processes the string of words and tries to understand the underlying ideas of the sender. Conversion, translation and interpretation introduce miscommunication. The challenge is to define a way of communication based on a direct exchange of ideas or concepts. That's why we start with a basic set of concepts, visualized by icons.

In a communication system for emergency services it is important to be able to talk about locations, hence we decided it could be useful to incorporate a map of the surroundings. To communicate about geometrical positions of objects, we will use local maps of the world. Now observers have to position icons on a map, so we have two kinds of communication:

1) Using strings of icons
2) Locating icons on a local map of the world

The proposed goal of this thesis is a system that is based on the second kind of communication. It can be used by the emergency services to keep each other up to date about what's going on in a particular area, e.g. a city, by placing icons on the map and sending them to each other. For now we will ignore the need of the wireless capabilities of the system [Kla05], and focus on the interface and intelligence.

1.1 Project Overview

What we would like to achieve is to get a structured World Model from a real life crisis situation. A World Model is composed of objects, characteristic features of the objects, and relations between the objects. Every observer has his own World Model. Police officers, firemen and laymen have different views of the world. An observer will look at the situation that is going on, from this he will form his own ideas of what is happening. When you get just a glimpse of a situation, and recognize certain aspects, the brain will automatically start to make assumptions about what is going on. The brain will construct its own model about the situation, the mental world model. Thoughts like these are based on what he observes, but also on his background knowledge.

We may also assume that observers are positioned differently in time and space. Not all observers will be able to see the same things because they report at another time, or from another place. What we want to do is mould these mental models into a computer system. To do this we need to make the mental model more concrete, so that it can be stored in a structured way. Next we want to fuse the different reports into one shared World Model, see Figure 1.

The agent in the field observes what is going on in the Real World and forms his own Mental World Model of the situation. He then wants to report his thoughts with the report tool of the system. The tool will only be able to handle structured information; concepts represented by icons. Thus the reporter has to concretise his ideas in icons, that he can then place on the map. Then the Structured World Model gets send to a central server, which collects reports of all the agents in the field. This server will fuse the ideas and form its own structured world model that gets send back to the agents, along with suggestions that the agent might have forgotten to report. The agent will see these suggestions, forcing him to observe the situation again, to see if he missed anything.

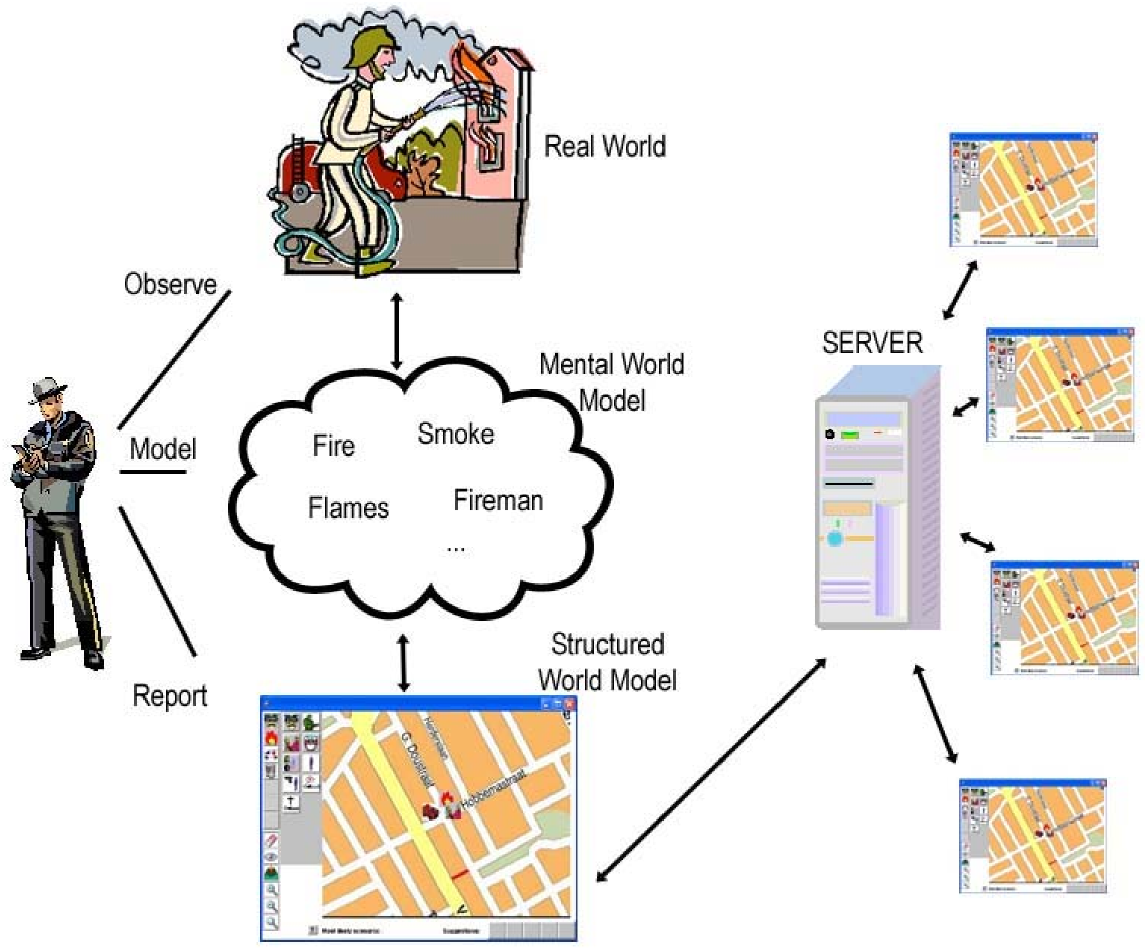


Figure 1 Project Overview

For an intelligent communication system like this we will have to look into several aspects:

- First of all we have to *define a World Model*, different sets of icons, corresponding to different crisis situations, and a priori information about characteristics and relations between the objects. What icons will we need? How are these icons related to each other, and what are the specific characteristics of each?

- Then there needs to be the *interface* of the report tool. The interface should provide a clear structure for the communication. What kind of information should be reported to the system, and what kind of information should be distributed back to the users? How will the information be represented to the user?

- A next aspect is the *intelligence* of the system, in particular in the fusion of the different reports. How does the system handle double or missing information? How should it deal with contradicting information? How does it keep its world model unambiguous and up to date? How does the system handle time and dynamic events?

- Another issue is the *security* of the system. Since it will be based on wireless communication, how can we prevent outsiders to intercept information? Should all information be send to all the users, and if not, how do we define different roles of users? How can we prevent the server from going down? And if it does go down, how can we prevent losing the information?

1.1.1 Define a World Model

To define a World Model we will need information about what concepts play a part in crisis situations and how the concepts are related to each other. It is important that the icons will be expressive enough to represent the needed information, and that the vocabulary of the system is extendible and editable. One of the first problems we encountered during the project is that the emergency services are very conservative with providing information. The information we have used is therefore for a big part originating from emergency scenarios as described in news articles. Although we tried to define the concepts that can occur in a crisis situation to our best knowledge, the reality of what the emergency services would actually use as concepts may be somewhat different. The information about icons should therefore be stored as flexible as possible, to be able to cope with different, more or other icons, or different scenarios.

1.1.2 Interface

As said in the introduction we will be communicating by the means of placing icons on a map. Since we are developing it for crisis situations we want the clients to run on handheld computers. This means the application has to fit a certain dimension, making it rather compact. It's important for the interface that icons can be easily selected and placed on the map. The icons should therefore be grouped in a logical way, so one can find the needed icon quickly. It is also important that the icons can be placed with some accuracy, which can be obtained by developing zoom functions for the map. A last issue is that the interface should be able to deal with the flexible information about the icons. There is little use in making the information flexible, but the interface not.

1.1.3 Intelligence

We want a system that is intelligent, in the sense that it can deal with icons that are received multiple times, cope with icons that might be missing and icons that might be placed wrongly. If icons are received multiple times, the system should be able to handle this on its own, without human interference. In the other cases however it is very hard and dangerous to let the system delete, modify or add icons to its world model without human approval. Therefore the system should be able to ask for feedback to the users, if it detects that errors might have occurred. Again these errors are based on our own beliefs about crisis situations, and might be different from reality. Taking this in mind, we want the Artificial Intelligence of the system to be dynamic and easily adjustable. Another issue about the intelligence of the system is its ability to cope with time. Specific events may happen in a strict order.

1.1.4 Security

Since the application is going to run on handheld computers, there will be wireless communication. Wireless communication can quite easily be intercepted, allowing people to 'listen' to what is happening, and maybe worse, to actively send (wrong) information to the system, making the system worthless. This can be solved by encrypting the information. Another security issue is the failure of the server. We don't want a system with only a central server, because we could lose all information when it goes down. Therefore we want to have all the information distributed amongst the clients, simulating a central server.
It might also be necessary for the different users to receive different kind of information. Maybe ambulance personnel is not interesting in certain events that are important for the police or the fir department. This could be solved by giving the different users different roles, which determines what information will be send to them.

1.2 Problem Description

In the ideal case we would have a system that has its information safely distributed among different clients. The system would have a clear interface that is easy enough to not make mistakes, but complex enough to handle difficult and unexpected situations. The system would at all times have a correct and up to date world model, that automatically adds missing information, alters wrong information and deletes excessive information. Furthermore we want a system that can easily be extended and altered, both on its vocabulary and its intelligence.

Taking in account that not everything can be handled in a single thesis work, the problem we are trying to solve in this particular thesis work is focussed on the interface and intelligence, and is defined as follows:

Design and implement a system that is suited for iconic communication in a crisis situation, using a map of the surroundings, which is expressive enough to handle complex and unexpected situations, yet intuitive enough to use without making (a lot of) errors. The system should be intelligent enough to assemble and maintain a correct and up to date world model. It should detect possible errors in the form of missing, double and wrongly placed icons. Furthermore the system should be dynamic in the sense that new concepts and rules can easily be added.

In chapter 3.1 we will elaborate on this problem description, splitting it up in workable components, which the final result will be tested against.

Chapter 2: Related Work

In this chapter we will describe some related work that was studied before starting the design of the system. We will first discuss some background information about icons, then we will look at an emergency system that has been designed by the Dutch government, and finally we will take a look at a system that uses icons to communicate.

2.1 About Icons

Because we will be using icons to communicate we will first present some information about icons. This section will tell something about the history of icons, followed by some information about the modern use of icons. See [Bea94], [Cha02], [Dor94], [Jon96], [Mea91], [Mea94], [NRC03], [Ric94], [Shn98].

2.1.1 The History of Icons

Icons are graphical symbols representing a concept or thing in reality. The term icon has been adapted from the Russian word ikon, which is a religious painting or statue. Icons have been around for a very long time, as early as the middle ages complex iconic systems have been used, for example to denote systems of astrological signs. It may even be argued that the ancient Egyptians were using icons as a language. They may not have called them icons, but they did communicate using graphics.

In the 1930s Otto Neurath developed Isotype, a system for communication which uses stylised graphics within a two-dimensional syntax. Neuraths work ranges from a very specific example of how a complex idea can be conveyed graphically, to a proposal for an international set of iconic images.

In the 1950s, Charles Bliss developed a set of atomic icons that represent basic objects in the world, and their features. These can be combined to form complex icons that map on to the set of words found in natural languages. Figure 2 shows how we can construct a symbol for telephone using: mouth-ear-language-electricity-telephone

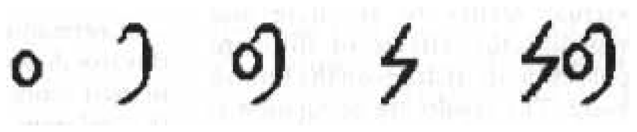


Figure 2 The construction of the Bliss symbol for telephone

The work of Bliss has some resemblance with the work of linguist Anna Wierzbicka, who claims to be able to describe any concept with using only 61 different words. The combination of these atomic words lead to a new concept, just as the atomic pictures of Bliss lead to a new concept. Although Wierzbicka does not use icons, the possibility of mapping her atomic words to atomic icons seems interesting.

The iconic languages were not all as successful as their developers might have hoped for, but they do show that there are distinct advantages in a communication based on graphical icons.

Our ability to learn or to recall the meaning of a sign seems to be greatly enhanced to the point where we may not need to be told what the sign represents or to explicitly learn its meaning. There might also be some advantage in the efficiency of using icons over natural language in the sense that difficult concepts might be represented by only a small number of icons, as opposed to many more words. Furthermore our ability to recognize icons does not depend on the natural languages we know, suggesting that iconic systems may be a way to overcome linguistic differences. Note that this does not imply that icons are also culturally independent. [Colin Beardon]

2.1.2 Modern use of icons

Within the computing context the word icon is used to denote a small graphical representation of a program, resource, state, option or window. As such, icons form an important part of the Graphical User Interface (GUI).

An ideal icon language wouldn't need any explanation, the intuition of the user, based upon his life experience, should be enough to immediately understand it. Of course, this is not a very realistic goal. Just as any language, icon language is something that does need some training. Most people already have some training in recognizing icons however, because icons can be found anywhere. In many public places they are used extensively, for example to indicate the toilets, or to show where the emergence exits are. A lot of icons are used in traffic signs, they point out if you are allowed to overtake other cars, if a road is one way only, or is a dead end.

The challenge in designing icons is that they should be as easy as possible to learn, as easy as possible to remember, and as easy as possible to recognize. Therefore icons should be designed with the following criteria in mind:

- Graphically clear
- Semantically unambiguous
- Cultural independent
- Simple

To make them *graphically clear* is straightforward, make the icons so that they resemble a concept in the real world, and keep them *simple*. Too much detail cannot be shown clearly in 32x32 pixels. *Semantically unambiguous* means that the icon only represents one concept, and that concept is only represented by that particular icon. Don't make two icons for the same concept, don't make one icon for 2 concepts. *Cultural independence* is harder to achieve, try to make the icons independent of any cultural background information. An easy example of doing this in a wrong way is using the road sign in Figure 3 in Great Britain, where this sign would mean that it is forbidden to be overtaken by other cars.

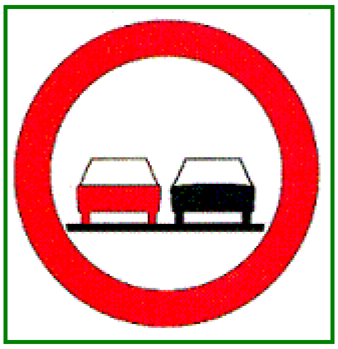


Figure 3 Cultural dependent icon

If icons are for some reason not clear to the user, they can be explained by written or spoken text. This has, however, a very big drawback. One of the most important advantages of icon use will be neglected in this way; icons can be used to communicate with people that don't speak each others natural language. If the explanation of an icon is given in a natural language, somebody who doesn't speak that language isn't able to use it.

A better way, perhaps, to explain icons that are not very clear is to animate them. Especially on computers this is very convenient to do and some icons are animated already. In most operating systems you see an animation when you copy files from one map to another, for example. Animated icons should only be used if normal 'static' icons will not suffice. In addition, most animation should only be used for the explanation of the icon; otherwise, you will become very distracted by all the moving images on your screen.

There are three styles of icons that are commonly used, see Figure 4:

1) Silhouette style; this one is very straight forward and clear, the drawback is that it is somewhat limited in the range of things it can represent.
2) Three-quarter top view; this style is very informative, but it requires some visual understanding.
3) Realistic style; this one is easy to recognize, but it is not very generalizing.

Although the use of these different styles makes it possible to select the best one for each icon, it is not recommended to use a mix of different styles, as it can be confusing.

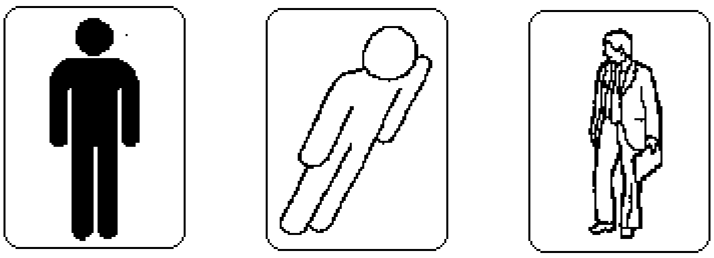


Figure 4 Three styles of icons

In systems that have a lot of different functions, it is not easy to design an icon for each function. To improve the recognition of the different icons, they may be divided in subsets. For example one could use a single group icon for representing surfaces, and have as subset different icons for representing circular and rectangular surfaces. The division of icons in subsets can also improve the overview and layout of certain applications.

The icons we are going to use in our system are all designed in a silhouette style. Since we use colours the icons may seem to fit in a realistic style as well. Because in a crisis situation we want as much information as possible, an icon on the map is often not descriptive enough. To add extra information we have tried a visual approach at first. We have tried 3 ways to visualize combinations of icons.

Transparent icons

By making the icons background transparent we can just place them on top of each other. For some icons this works, but a lot of icons cannot be recognized anymore, especially when more than 2 icons are used in the combination. For example a fire in a building is shown as follows in Figure 5. This is not a good visualisation, although there are only two icons used it becomes unclear what's shown in the first icon.

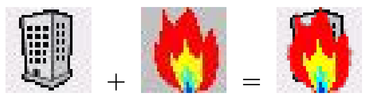


Figure 5 Transparent icons

Alternating icons

In this version the combinations will be visualized by alternating the icon every time unit. For example a building is shown for one second, then it gets alternated by one second of flames. This type of visualization is already a big improvement on the transparency, but still has some drawbacks. If there are a lot of combined icons on the map, the whole map is blinking, which is very distracting.

Another drawback we found after implementing and testing it is that some combinations are shown very poorly. If we would select a *car*, a *crash*, and another *car*, it would just show a car for two time units and a crash for one time unit. This is not nice because it looks like there is just one car in this case. Of course there can be worked around this problem, there could be certain rules that would not allow two identical icons in one combination. The example combination could then be rephrased like *2*, *car*, *crash*.

If a very big combination is made, another drawback can be found. It takes the user several seconds to see the meaning of the icons because they are not shown simultaneously. They would have to wait for the whole message to come by, before they understand the meaning. We would rather have an instant overview with a single look at the map.

The final drawback about this implementation is that the application becomes pretty slow when a lot of combinations are made. The map keeps getting repainted every time unit, and this is of course really computer time intensive. Because the application is designed to eventually run on a handheld, this might prove to be a problem. It may be clear from the above that this is not the type of visualisation we are looking for.

Stacking icons

In this version the icons simply get stacked on top of each other. The advantage is that this is a very easy solution, because each icon will simply be placed shifted some pixels up and right of the original icon. Also the icons used in the combination can still be recognized and they are shown simultaneously. The only disadvantage is that the stacks take up more space than the other versions. This can be worked around by setting a maximum size of a combination. The stacking version is shown in Figure .

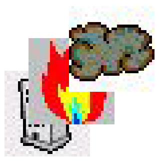


Figure 6 Stacking icons

Later on during the project we have decided to not make use of any of these combinations after all. When the zooming in and out of the map had been implemented, it became clear that we did not need any stacking at all, because the icons can be placed very close to each other already. An additional reason to abandon the idea of visual representation of additional information is that certain types of additional information are very hard to represent with an icon. If we want to say something about the status of a policeman, we use terms as *busy, waiting, wounded*. It is much easier to give this information in a natural language. That's why we implemented a way to add and view information by clicking on the icon and adjust some of its attributes.

2.2 C2000

In the Netherlands a new digital radio network for the communication of emergency services is being developed [C2000]. It is called C2000, and its goal is to maximally facilitate the communication between the fire-brigades, ambulance services, police-brigades and military police. The mobile communication between these emergency services should be supported and improved. The system should guarantee fast and secure communication, make communication between different emergency services possible, and help improve the safety of the emergency personnel.

The need for a reliable communication system for these services is high. Not only for the day to day activities, but also in case the different services need to cooperate with each other, for example in crisis situations as in Enschede, where a fire occurred in a firework deposit. The emergency services themselves are closely involved in the development if C2000. In 1996 the first steps were undertaken to develop it, and the system is (was) supposed to be in full operation at the end of 2003. In Figure 7 the design of the C2000 network is shown.

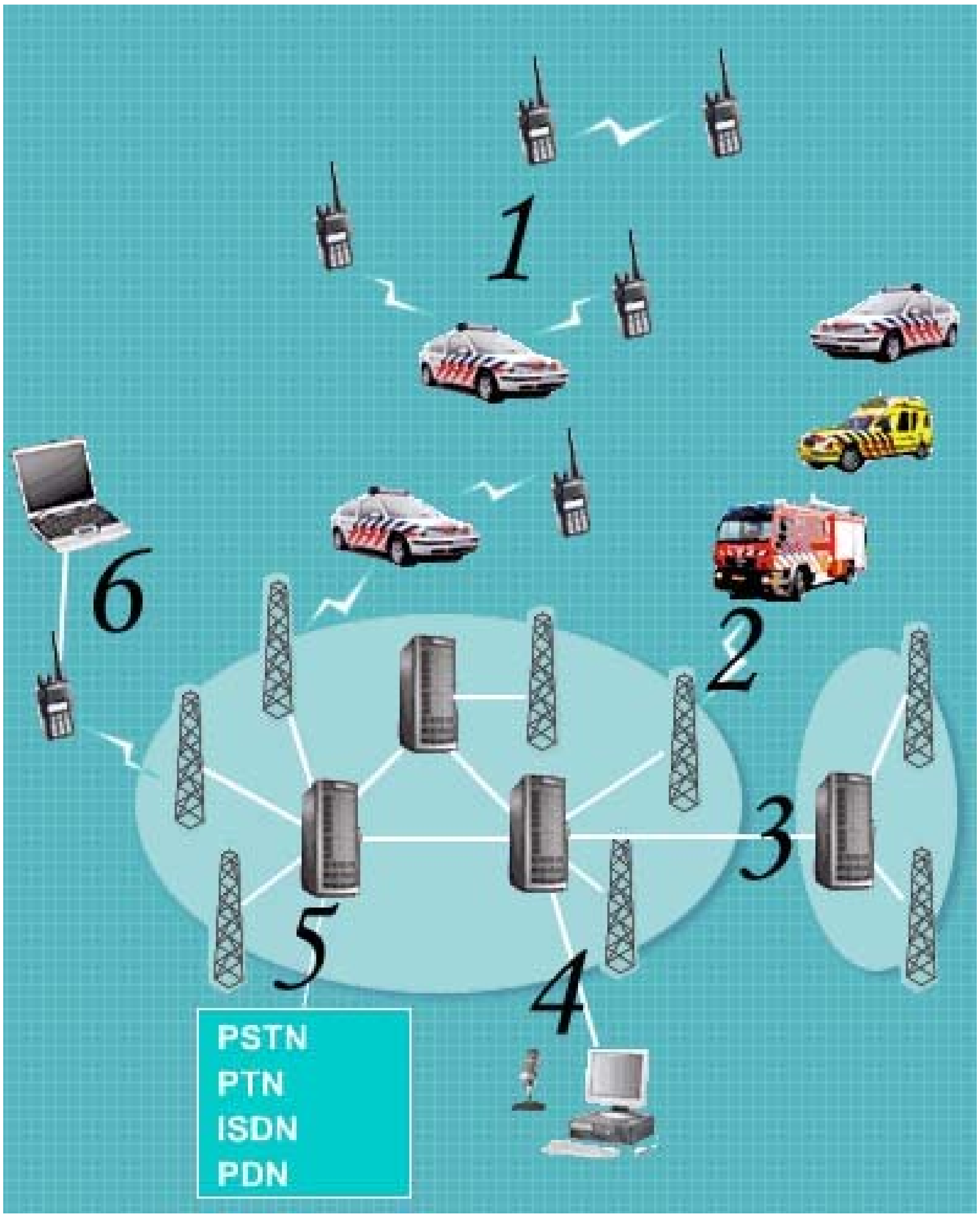


Figure 7 Overview of C2000

The numbered components are explained below:

1) Direct Mode Operation. DMO makes it possible for car phones and walkie-talkies to communicate with each other directly, without making use of the network.
2) Air Interface. Communication of a mobile station takes place using electromagnetic waves, with a transmitter mast, or with another mobile station via the Air Interface.
3) Inter System Interface. Multiple TETRA networks can be linked using the Inter System Interface. This is important for international communication.
4) Direct link with the central emergency room.
5) Gateways. The gateways make it possible to link the system to other external networks, such as the public telephone network, or the Nationale Noodnet (national emergency net).
6) Peripheral Equipment Interface. The PEI supports communication between laptops and mobile stations, such as car phones.

2.2.1 Advantages

C2000 makes the communication among emergency services fast, simple and reliable. This one national system will replace almost 100 local systems that are currently used by the different services. The digital network has big advantages over the old analogue systems:

- C2000 is suitable for multidisciplinary communication, whereas this was impossible with the old systems.
- C2000 is designed in a way that is easy to secure, making it virtually impossible to eavesdrop on it.
- C2000 has a national coverage, whereas the old systems only have regional coverage.
- C2000 has a much better sound quality for speech.
- C2000 is very suitable for data communication.
- All car phones and walkie-talkies are provided with emergency buttons.
- C2000 supports communication with foreign co-workers, improving provided services near the borders.

2.2.1.1 A joint radio network with national coverage

C2000 is a big improvement on the current situation. At this time all regional organisations of the four emergency services have their own networks in use, adding up to almost 100 networks, spread out all over the Netherlands. This makes communication among them very difficult. However, in the case of a big calamity, like the disaster with the fireworks deposit in Enschede, good communication can save lives. On top of this, the old networks are all analogue and outdated. Because C2000 is one national network, the communication will be fast, simple and reliable. It is believed that in the long term a lot of money can be saved on the acquisition of hard and software because it will be standardized. Money will also be saved by having a joint education of the personnel. Furthermore the network is believed to provide an excellent basis for future technologies, because of its state of the art technology.

2.2.1.2 A high level of security

All car phones and walkie-talkies get provided with an emergency button. When this button is pressed a connection is made directly with the central emergency room. The operator can then automatically listen along with what’s going on. C2000 has an excellent quality of speech. The system is designed in a way that it can not fail because of excessive use, what happens with ‘normal’ communication such as the telephone when a lot of people use the network at the same time. Think about New Years Eve, when everybody is trying to call their families at the same time, the system then gets overloaded and fails to do its job. Furthermore the system is secured against eavesdroppers. While people with special scanners can now freely hear everything that’s being said over the analogue systems, this can’t happen in the new system because every conversation will be secured automatically.

2.2.1.3 An open European standard

C2000 is based on the European TETRA standard. TETRA stands for TErrestrial Trunked RAdio. Just like GSM is the standard for mobile telephony, TETRA will be the standard for emergency services. The standard is designed with the cooperation of the industries, ensuring that a customer is not dependant on only one supplier, but can purchase TETRA equipment at several suppliers. The TETRA standard is based on the latest digital technologies and is continued to be improved. This will ensure the systems based on the standard will be fit for the future. Most countries will be using, besides the TETRA standard, the same frequency. This will allow for international cooperation.

The infrastructure for the C2000 system has been delivered in July 2004 and at this moment some regions are working with it. The system is expected to be up and running in the whole of the Netherlands at the end of 2004.

2.3 Iconic Communication

A closely related project involving icons and emergency situations was done by Iulia Tatomir [Tat03]. The goal of this project was to create an application that allows its users to communicate with each other using an international 'language', icons.

First the graphical user interface was developed, which is designed to be as easy as possible to use. Because its is designed to be a simulation for a PDA application, big constraints on the applications dimension were imposed. The interface is shown in Figure 8

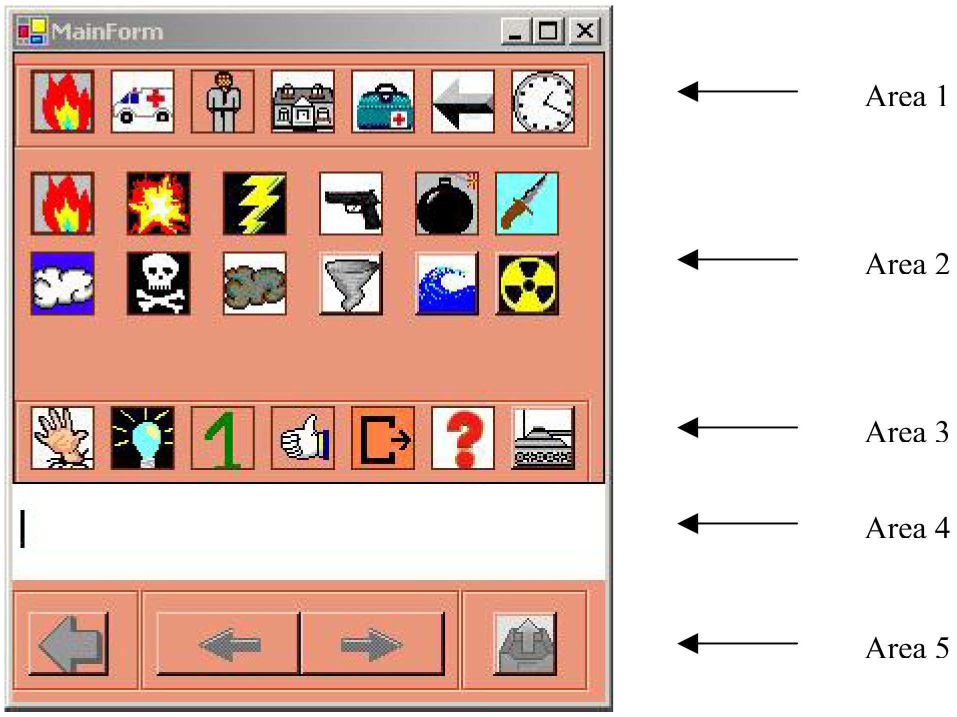


Figure 8 Interface for iconic communication

The application is divided in 5 areas

Area 1: This is where the main categories are represented by their defined icon. The categories are Crisis, Cars, People, House, First Aid, Directions and Time.

Area 2: This is where the icons of the chosen category are displayed. The first icon in this area is always the icon that represents the index icon for the category. When an icon in this area is being clicked on, it will appear at the cursor position in Area 4.

Area 3: This area is an extension of Area 1. The categories in this area are Human Actions, Information, Numbers, Yes/No, Special Signs, Intonation, Military.

Area 4: This is the area where the selected icons are placed to form the sentence that the user wants to send. There is room for 7 icons, which should be long enough for a sentence in icons.

Area 5: This last area is to edit the sentence. The middle left and right arrows are for navigating through the sentence, it will move the cursor through the sentence. The big left arrow is used for deleting an icon. The delete works as the backspace button on a keyboard, so it deletes the icon before the cursor position. The last button is used for sending the sentence.

The main part of the research in this project was done on defining the grammar of the icon sentence. Not all combinations of icons form correct sentences.

The grammar that was developed is a context free grammar, based on Chomsky's hierarchy. A grammar is basically a set of rules that defines how grammatically correct sentences can be formed. A grammar is formally defined as a quadruple G = {N, T, S, P} with:
N – a finite set of non terminal symbols
T – a finite set of terminal symbols
S – a special goal or start or distinguished symbol
P – a finite set of production rules

The union of the sets N and T form the vocabulary of the grammar and should not intersect. The final grammar used in the application is defined as follows:

```
S = negation A | number B | adjective C | noun R | verb E | adverb H
A = number B | O
O = verb E | P
P = verb | B
B = adjective C | C
C = noun R | noun
R = sign | D
D = negation F | F
F = verb | G
G = verb E | E
E = adverb | I
I = adverb H | H
H = number J | J
J = adjective K | K
K = noun L | noun
L = adverb
```

Figure 9 Grammar of the Iconic Communication system

When an icon is selected, the sentence gets parsed and the system decides which icons can follow. This is extended by the interface, by only making the icons that fit correctly in the sentence selectable.

The similarities of the work by Tatomir and this project are of course the use of icons and the context of a crisis environment. As we worked together for some time we made some agreements on which icons to use for which concepts. A lot of icons in her work will thus be seen again in this project. The categories of icons are different however, because on a map only *nouns* can be placed. There is no use for verbs, numbers, adjectives, and so on.
While the idea of defining full sentences with icons seems promising it is not very applicable in combination with a map. The only sentences we will need are of the form:

```
There is a <noun: icon> at position <number: x, number:y>
```

Chapter 3: Global Design

In this chapter we will discuss the requirements and constraints of the system, we will see how the system is built up from different components, what the responsibilities of these components are, and how they are designed.

3.1 Requirements and Constraints

When developing any system, there are some requirements and constraints that needs to be taken in mind. We will take the problem description to split up the different requirements into workable components.

Design and implement a demonstrator for a system that is suited for iconic communication in a crisis situation, using a map of the surroundings (1), which is expressive enough to handle complex and unexpected situations (2), yet intuitive enough to use without making (a lot of) errors (3). The system should be intelligent enough to assemble and maintain a correct and up to date world model (4). It should detect possible errors in the form of missing, double and wrongly placed icons (5). Furthermore the system should be dynamic in the sense that new concepts and rules can easily be added (6).

The requirements that can be found in this problem description are discussed below:

1) To make the system suited for iconic communication in a crisis situation we need to have icons that represent concepts in a crisis, and maps of the surroundings to place the icons on.
2) To make the system expressive enough we will need different categories of icons for people, events, transportation and buildings. Within these categories several icons are needed for more specific information. Just placing a man on the map doesn't do much good, if it's not specified he is e.g. a fireman. Even when there is a good icon to represent the concept, we will need a way to add even more information. That's why the icons will have several attributes. In the case of flames these attributes will be the size of the flames, the intensity and the status (increasing, decreasing, under control).
3) To prevent a lot of errors, the GUI should be intuitive and easy to use. It should be clear which category and icon is selected and they should be added to the map with just clicking on the location. To provide extra information, an attribute window will pop up where the values of the attributes can be given. The values can be selected out of a small list, this decreases the chance of making a wrong selection, and eliminates the chance to make an illegal selection. When icons are placed, the user should be able to delete them again, or to inspect or alter its attributes. To prevent placing icons on the wrong location, the user should be able to easily zoom in and out of the map, to be able to place the icon exactly where it should be.
4) In order to assemble and maintain a world model we will collect all information at one server. To create a world model out of this information we could store it all in some sort of database, which will have to be kept consistent at all times. We will choose a client-server implementation for this, where the many clients are the reporters of the crisis, and the server is the part that keeps a consistent world model and distributes all information among its clients.

5) To detect missing, double and wrongly placed icons we need some intelligent agent, that constantly works on the information that's being gathered. This means it will have to work on the database of collected information.
6) In order to make the system dynamic, it's useful to store all its information about icons and their rules in some sort of database. The entries of this database should be easy to edit, and the database should be extendible. If we keep this database in separate files that are read by the system on start up, it's possible to adjust rules, and icons in a way that does not require the entire system to be recompiled.

We are going to make a demonstrator for a real system, which, except for the server, has to run in the field during a crisis situation. This means that the type of hardware we need should be usable in such situations. It is impossible to let the eventual system run on laptop computers, let alone normal PC's. The system has to run on handheld computers, and thus the demonstrator has to take into account the constraints which come with that.

The PDA where the application will be based on, the Sharp Zaurus SL-C760 (Figure 10), has a maximum resolution of 640x480 pixels, a 400 MHz Intel XScale PXA255 processor, and 62 MB of RAM. The constraints that follow from this are the resolution and the limited speed of the client. That's why we need the server to do all the time intensive calculations, and make the client as light weight as possible. Furthermore we need to use a programming language that can run on the Zaurus.

Figure 10 Sharp Zaurus SL-C760

Another constraint of using a handheld computer is that the system should be designed for wireless communication. Because this is much slower than a normal network, we need to keep the data traffic to an absolute minimum.

3.2 Java, Jess and XML

Following from the requirements and constraints we have made a choice on which programming language(s) to use.

The main programming language we will use is Java. There are some advantages of Java that made us decide to use it. The first, and most important, is that Java is platform independent, meaning that it should have little problems running on a handheld, or the server. The second important reason to use Java is that there is a lot of knowledge available about how to handle problems that may arise. A lot of people are familiar with the programming language, both internally at our faculty and outside of it. Many help forums are available as well.

The storage of the world model of the server and the intelligence of the server are both combined in one component, Jess [JESS]. Jess stands for Java Expert System Shell. In short Jess is an expert system that works with facts, and rules that are automatically triggered when the conditions are met. Jess is actually the Java version of CLIPS [CLIPS], with some added functionality to cooperate better with other Java classes. More about Jess and how it works is explained in section 3.5. The fact that Jess is written in Java made the choice of using it easy. It should give little problem to embed the Jess component in the rest of the application, written in Java.

The last part of the system is the storage of icons and the rules that apply to them. We have chosen XML files for this part. They are very light weight, and easy to edit. Yet they have constraints about how information is stored. The form of the XML files can be defined in special Data Type Declaration files, as will be explained in section 3.4. Another advantage is that XML files will be relatively easy to read by Java.

3.3 Overview

As said the system will work with one server and multiple clients. The clients collect the information about their surroundings on their map and send it to the server. The server, in turn, will make a consistent world model of all the gathered information and send it back to the clients. The clients can not interact with each other directly, but all communication will be done via the server.

The basic input of our system are the observations of the users. To reduce the ambiguity and to come up with a shared view of the world, the system will provide the set of icons. Nevertheless, the observations can still be somewhat ambiguous because of the following reasons:

- Observers miss objects in the scene; they can overlook overlook certain events or have a different view on what is important to report.

- Observers are remote in time. Crisis events occur and develop over time, so the observations are time dependent.

- Observers are remote in location. Observers are positioned at different locations and can see different things, or from a different angle.

- Observers can either report with the use of the map, or just with icon strings (when these systems are integrated). Both types of messages are supposed to be consistent and complementary.

In Figure 11 an overview of how the different components interact is shown. The XML files contain all information about the icons and scenarios. They also contain parameters that influence the rules in the Server. The server is mainly done in Jess, there is only some Java functionality that lets the server communicate with the clients, and lets it read in the XML files. The Client is done entirely in Java, it gets information about what icons should be in the GUI from the XML files, and then dynamically builds up the interface. A change in the XML files will cause other icons to appear in the client, and other icons and rule parameters will be used in the server as well. When the XML files are changed, both server and client need to start up again, in order to let the changes affect them.

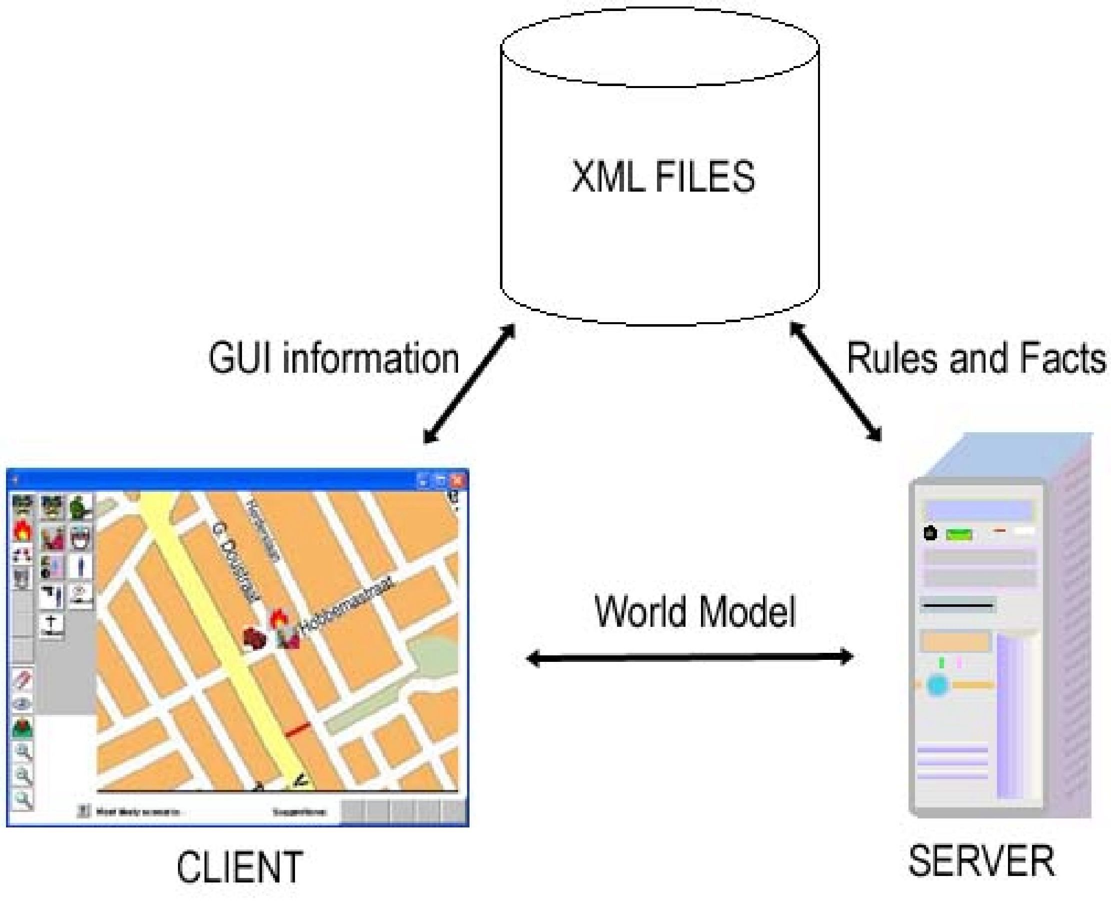


Figure 11 Overview of the components

Besides sending back the world model to the clients, the server will also send some information about scenarios and will suggest some icons that can be placed next.

The scenario information will consist of the predefined scenarios and the probability values that the server has added to them. The values are not in percentages, but are a number that the intelligence of the system will award to each scenario. The higher the number awarded, the more likely it is that the scenario is happening. The scenario information is to give the users of the clients a quick idea of what is happening around them, and to make better predictions of expected icons.

The suggestions for new icons are some feedback for the user. The server has some expectations of what icons it will be receiving next. If an icon is missing, the server will notice this by the information of the XML files. If a fireman is reported, but there is no fire truck, the system will be likely to suggest placing one. The suggestions for placing new icons come from both individual icon relations (icon:flames → icon:smoke) and relations between scenarios and icons (scenario:bomb scare → icon:bomb). Providing suggestions like these, we believe, will add to a more accurate report. Things the reporter might have missed, or did not find important enough to report will be more likely to be reported now. The user will be actively looking for the concept represented by the icon of the suggestion.

3.4 Design of the XML Files

The XML files will form the basis of all knowledge in the system. In the first place, all the used icons are defined in these files. The icons will be summed up, and each icon will have its own attributes, as defined in these files. Furthermore these files will contain parameters that influence the intelligent behaviour of the system as a whole. Finally the XML files will contain information about emergency scenarios.

XML stands for eXtensible Markup Language and is a language to define structured information. It is very lightweight, as opposed to a traditional database, and can be edited relatively easy. All XML files should be *well-formed*, meaning that they should obey to certain grammar rules of XML. This will lower the likelihood of making errors while editing or creating the file. Besides this protection, the structure of the document can be further constrained to be *valid*, meaning the file should obey a predefined structure. These structures can be defined in Document Type Declaration file. This file exactly defines what structures are allowed.

In DTD files is defined exactly what structures in the XML file are allowed. XML files use, just as HTML, tags that separate the structure from the data. A tag is an indication of what information will follow, at the end of the information will be a closing tag. Tags and information can be nested. An example to clear this up:

```
<book>
      <title>Artificial Intelligence: a modern approach</title>
      <author>Russell</author>
      <author>Norvig</author>
</book>
```

Figure 12 Example XML fragment

Everything from the beginning to the end of a tag pair, is called an element. In the example in Figure 12 we have the following elements: book, title, author.

Elements can thus be nested. Note that elements may contain multiple of the same sub elements, while a book has only one title, it may have more than one author. This structure can exactly be defined in DTD files. The DTD file that defines the structure of the example is shown below:

```
<!ELEMENT book (title, author*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

Figure 13 Example DTD

Every element has to be specified in the DTD, and it defines how each element is build up. From Figure 13 we can see that every book has one title, and zero or more authors. The title and the author elements are both specified to contain #PCDATA, this is Parsed Character Data, which means it can contain an arbitrary string of characters.

In our system we will need two separate XML files. The first one contains all the icons and the inter icon relations. The DTD file is defined as follows:

```
<!ELEMENT iconlist (group*)>
<!ELEMENT group (icon*)>
<!ELEMENT icon (icon_name,icon_image,slot*,next_icon*,previous_icon*)>
<!ELEMENT icon_name (#PCDATA)>
<!ELEMENT icon_image (#PCDATA)>
<!ELEMENT slot (slot_name, slot_value*)>
<!ELEMENT slot_name (#PCDATA)>
<!ELEMENT slot_value (#PCDATA)>
<!ELEMENT next_icon (icon_name, chance, timespan)>
<!ELEMENT previous_icon (icon_name, chance, timespan)>
<!ELEMENT chance (#PCDATA)>
<!ELEMENT timespan (#PCDATA)>
```

Figure 14 DTD of the icon XML file

A valid XML file, as defined by Figure 14, has an element *iconlist*, which is the main structure. We want to have the icons divided in categories, that's why we make one big list, which can contain multiple *groups*. The *iconlist* can contain zero or more *groups*, as indicated by the *. Each *group*, in turn can contain zero or more *icons*. Each *icon* contains exactly one *icon_name*, one *icon_image*, zero or more *slot*s, zero or more *next_icon*s, and zero or more *previous_icon*s. The icons name and image are defined as #PCDATA, an arbitrary string of characters. The *slot*s of an icon is where its attributes are stored. It contains the name of the attribute and the possible values the attribute can take on. The inter icon relations are defined in the *next_icon* and *previous_icon* fields. The fields have the name of the next, respectively previous icon, a chance and a timespan. The chance defines how likely it is that if the icon is placed, a *next_icon* or *previous_icon* is required as well. The higher the number in the chance slot, the more likely. The timespan is currently not used, but can add extra information about how much time may pass until the relationship 'expires'.

```
- <iconlist>
  - <group>
    - <icon>
        <icon_name>policeman</icon_name>
        <icon_image>./icons/people/policeman.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>busy</slot_value>
          <slot_value>idle</slot_value>
          <slot_value>wounded</slot_value>
        </slot>
      - <previous_icon>
          <icon_name>policecar</icon_name>
          <chance>2</chance>
          <timespan>1</timespan>
        </previous_icon>
      </icon>
    </group>
</iconlist>
```

Figure 15 Fragment of the icon XML file

The second XML file we will use contains the information about scenarios. Its DTD is given below.

```
<!ELEMENT scenariolist (scenario*)>
<!ELEMENT scenario (scenario_name, icon*)>
<!ELEMENT icon (icon_name, chance, slot*)>
<!ELEMENT icon_name (#PCDATA)>
<!ELEMENT slot (slot_name, slot_value*)>
<!ELEMENT slot_name (#PCDATA)>
<!ELEMENT slot_value (#PCDATA)>
<!ELEMENT chance (#PCDATA)>
```

Figure 16 DTD of the scenario XML file

As we can see the *scenariolist* contains zero or more *scenario*s, each consisting of multiple *icon*s. In the *icon* fields is the name of the icon is defined, with its *chance* of being in the scenario, and possible *slot*s that are relevant. Below, in Figure 17 is a small sample of the scenariolist.xml file.

```
- <scenariolist>
  - <scenario>
      <scenario_name>shooting</scenario_name>
    - <icon>
        <icon_name>policeman</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>roadblock</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>victim</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>policecar</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>ambulance</icon_name>
        <chance>3</chance>
      </icon>
    + <icon>
    + <icon>
    </scenario>
  </scenariolist>
```

Figure 17 Fragment of the scenario XML file

For a complete overview of the used XML files see Appendix B: XML Files. To understand how we made the inter icon relations, and scenario icon relations, we refer to Appendix A: Scenarios.

3.5 Jess

Jess is the expert shell in which we will program our expert system to add the systems intelligent behavior. First we will provide some information about how Jess works, after that we will discuss the design of our expert system.

3.5.1 About Jess

Jess is a rule-based expert system shell made in Java. This means that Jess's purpose is to continuously apply a set of if-then statements, the rules, to a set of data, the knowledge base. The user can define his own rules to make his particular expert system. Jess rules are of the form:

```
A
B
→
C
```

Figure 18 Structure of Jess rules

Which means that when A and B are statements that are both true, C will be made true as well. An example of a rule and its explanation:

```
(defrule library-rule
      (book (name ?X) (status late) (borrower ?Y))
      (borrower (name ?Y) (address ?Z))
=>
      (send-late-notice ?X ?Y ?Z))
```

Translation:

```
Library rule:
If
      A late book exists, with name X, borrowed by someone
named Y
And
      The address of borrower Y is known to be Z
Then
      Send a late notice to Y at Z about book X
```

Figure 19 Example Jess rule

The book and borrower information would be found in the knowledge base. The knowledge base is a collection of facts about the world. In the knowledge base we will be using, the placed icons will be a big part of the knowledge base, and they will be the facts that are added and removed. Together the placed icons will form the world model. The attributes, or slots, that the facts are allowed to have are defined in statements called deftemplates. An example of a deftemplate for a book is shown in Figure 20. Actions like send-late-notice are user defined functions that can be either in the Jess language (deffunctions) or in Java (Userfunctions).

```
(deftemplate book
       (slot name)
       (slot author)
       (slot ISBN_number)
       (slot status)
       (slot borrower))
```

Figure 20 Example Deftemplate

A typical expert system has a fixed set of rules while the knowledge base changes continuously. However it's an empirical fact that, in most expert systems, much of the knowledge base is also fixed from one rule operation to the next. Although new facts are being added and old ones get removed all the time, the percentage of facts that change per time unit is rather small. For this reason the obvious implementation of an expert system is rather inefficient. The obvious implementation would require keeping a list of rules, and continuously cycle through that list to see if any rules left hand side (LHS) has been made true by checking each rule against the knowledge base. This is very inefficient, since most of the results of each cycle will be the same in the next cycle. Since the knowledge base is fairly stable, every cycle the same facts are checked against the same LHSs of the rules. The complexity this algorithm gives is of the order O(RF^P). Where R is the number of rules, F the number of facts, and P the average number of patterns per rule LHS. This increases dramatically if P increases. This is not a good solution for any expert system.

Jess instead uses a very efficient method called the Rete algorithm. This algorithm was the basis for a whole generation of expert system shells: OPS5, its descendant ART, and CLIPS. In the Rete algorithm, the inefficiency as described above is solved by remembering past test results across iterations in the rule loop, meaning only new facts are tested against any rule LHSs. Additionally new facts will only be tested against the rule LHSs to which they are most likely to be relevant. As a result the complexity drops to O(RFP), which is linear in the size of the knowledge base.

The Rete algorithm is implemented by building a network of nodes, each representing one or more tests on a rule LHS. Facts that are added are processed through this network of nodes. At the bottom of the network the nodes that represent individual rules. When a fact filters all the way down the network, it has passed all the tests of a particular rule, and this set becomes an activation. The RHS of the associated rule will be fired if the activation is not first invalidated by the removal of one or more facts that make the activation set incomplete.

There are two kind of nodes in the network: one-input and two-input nodes. One-input nodes perform tests on individual facts, while two-input nodes perform tests across facts and perform the grouping function.

If we have these two rules, they can be compiled into the network of figure 21:

```
(defrule example-1                    (defrule example-2
       (x)                                   (x)
       (y)                                   (y)
       (z)                             =>)
 =>)
```

The nodes marked x?, y?, and z? test if a fact contains the given data, while the nodes marked + remember all facts and fire whenever they have received data from both their left and right inputs. To run the network, Jess presents new facts to each node at the top of the network when they are added to the knowledge base. Each node takes input from above and sends its output downwards. A single input node generally receives a fact from above, applies a test to it, and if the test passes, sends the fact downwards to the next node. If the test fails, the one-input nodes simply don't do anything. The two-input nodes have to integrate facts from their left and right inputs. They must remember all facts presented to them and attempt to group facts arriving from their left input with facts arriving from their right input, to make up complete activation sets. A two-input node therefore has a left memory and a right memory.

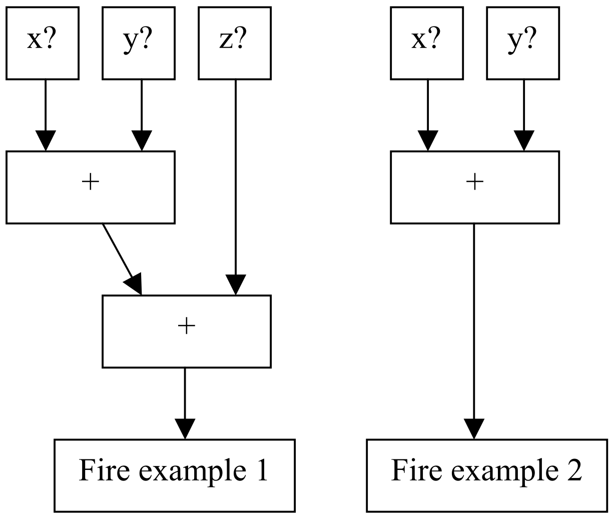


Figure 21 Network version 1

Its convenient to divide the network into two logical components. The single input nodes comprise the pattern network, and the two-input nodes form the join network. There are two simple optimisations that can make the Rete algorithm even better. The first is to share nodes in the pattern network. In the network in Figure 21 there are 5 single input nodes, while there are only 3 distinct ones. We can adjust the network to share the double nodes, as can be seen in figure 22.

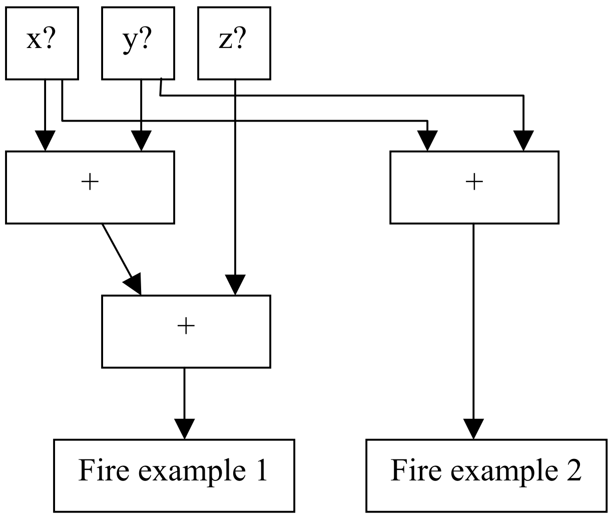


Figure 22 Network version 2

But this is obviously not the only redundancy in the network. We see that there is an identical two-input node in the join network, which is integrating x, y pairs. When we share that node as well we get to the situation in Figure 23.

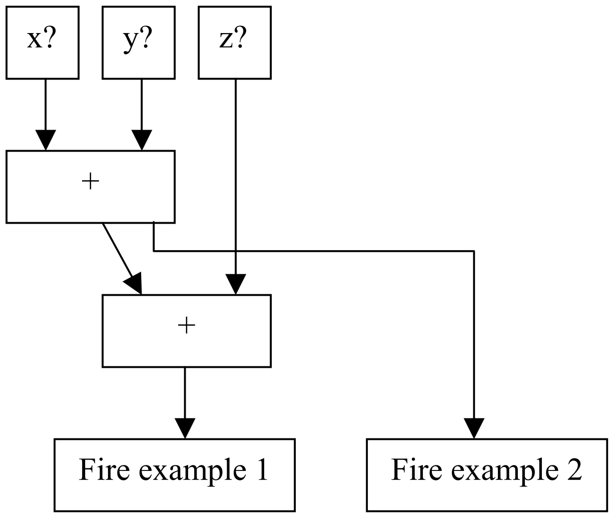


Figure 23 Network version 3

3.5.2 Design of the Jess Component

Our first problem in designing an expert system is to make the knowledge we need explicit. A common way for knowledge elicitation is to interview experts [Cha05] or to get the knowledge from documentation and manuals. The focus of our research is more on the design and implementation of a report tool. To test our system we defined the rules based on common knowledge about cricises. The rules and concepts were in our case extracted from the news articles in Appendix A.

It can be questioned if a rule based approach is appropriate for our system. A common AI procedure is to start with a deterministic rule based system and as a next step to take a probabilistic approach and to design a Bayesian Belief Network. This will be further discussed in the Recommendations. We have chosen an incremental, prototype based approach, to first prove that the concept of such a system is worth further investigation.

The Jess component will contain the systems World Model, based on reports from the clients. The knowledge base of Jess will be kept consistent and up to date by constantly running the rule base on it. When we look at the functionality of the Jess component, we can see that it works as a virtual blackboard, with some intelligent agent that keeps it clean, consistent and up to date. This agent is defined in the rule base. Every client can write information on it, the reported input is then written down on it without question. Then the agent does its work by performing certain functions on it. The resulting new world model is then send back to the clients. In Figure 24 an overview of the Jess component is given.

As can be seen, the clients input will consist of new facts, deleted facts and modified facts. The facts are in this case obviously the placed, deleted and modified icons. When input is received, the rule base will be applied. There are rules for added icons, deleted icons, modified icons and doubly placed icons.

The rule for adding new icons will add the icon to the list of placed icons that the knowledge base keeps track of. Deleting facts will delete the specified fact, and update the list of placed icons again. The modify rule will search the fact that need modification and edit it. The rule that searches for doubly placed icons will scan the knowledge base for icons that are very similar and combine them into one fact. This rule only applies if the double icons are in the near vicinity of each other and are not reported by the same client. The thought behind this is if one client sends two icons that are close to each other, there probably are two distinct icons, while if two clients both send the same icon, there is probably just one.

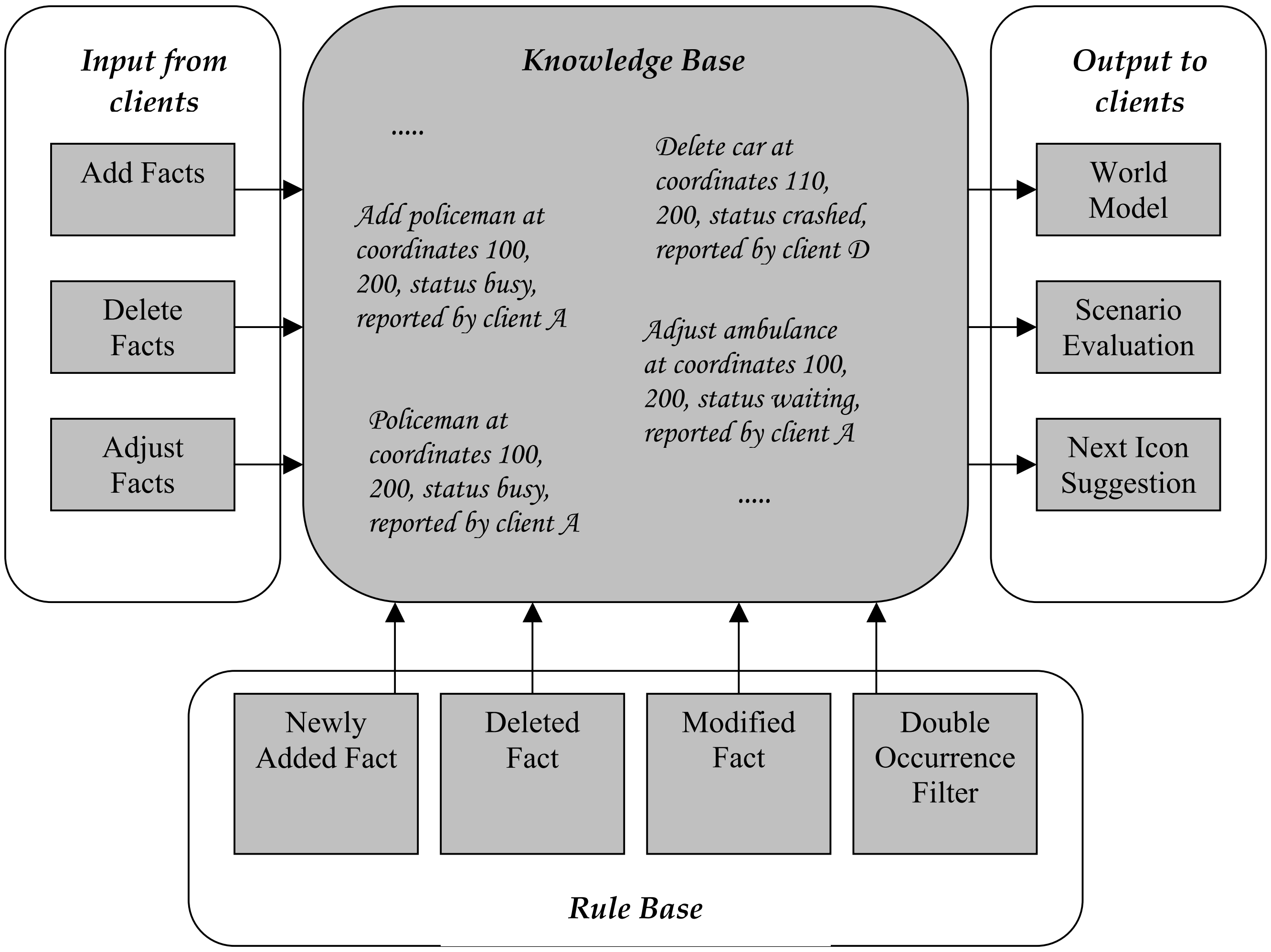

Input from clients
Add Facts
Delete Facts
Adjust Facts
Knowledge Base
.....
Delete car at coordinates 110, 200, status crashed, reported by client D
Add policeman at coordinates 100, 200, status busy, reported by client A
Adjust ambulance at coordinates 100, 200, status waiting, reported by client A
Policeman at coordinates 100, 200, status busy, reported by client A
.....
Output to clients
World Model
Scenario Evaluation
Next Icon Suggestion
Newly Added Fact
Deleted Fact
Modified Fact
Double Occurrence Filter
Rule Base

Figure 24 Overview of the expert system

After the rule base is applied to the knowledge base the resulting world model is send to the clients, together with the scenario information and next icon suggestions. These last two outputs are acquired by performing a function on the knowledge base that calculates the values for the scenarios and suggestions. The Jess components are discussed in more detail in Chapter 4: Implementation

3.6 Design of the Java Component

The Java component in the system is responsible for everything besides what's included in the XML and Jess components. This means Java is responsible for the interface of the clients, the network, and the integration of the XML and Jess components. We will discuss each of these in this chapter, and in some more detail in Chapter 4: Implementation.

3.6.1 Graphical User Interface

The Graphical User Interface (GUI) of the client is made entirely in Java. The icons that will be used, however, are extracted from the XML files. This means the GUI should be as dynamic as possible. Adding a new icon in the XML files should not result in a problem during the initialisation of the GUI.

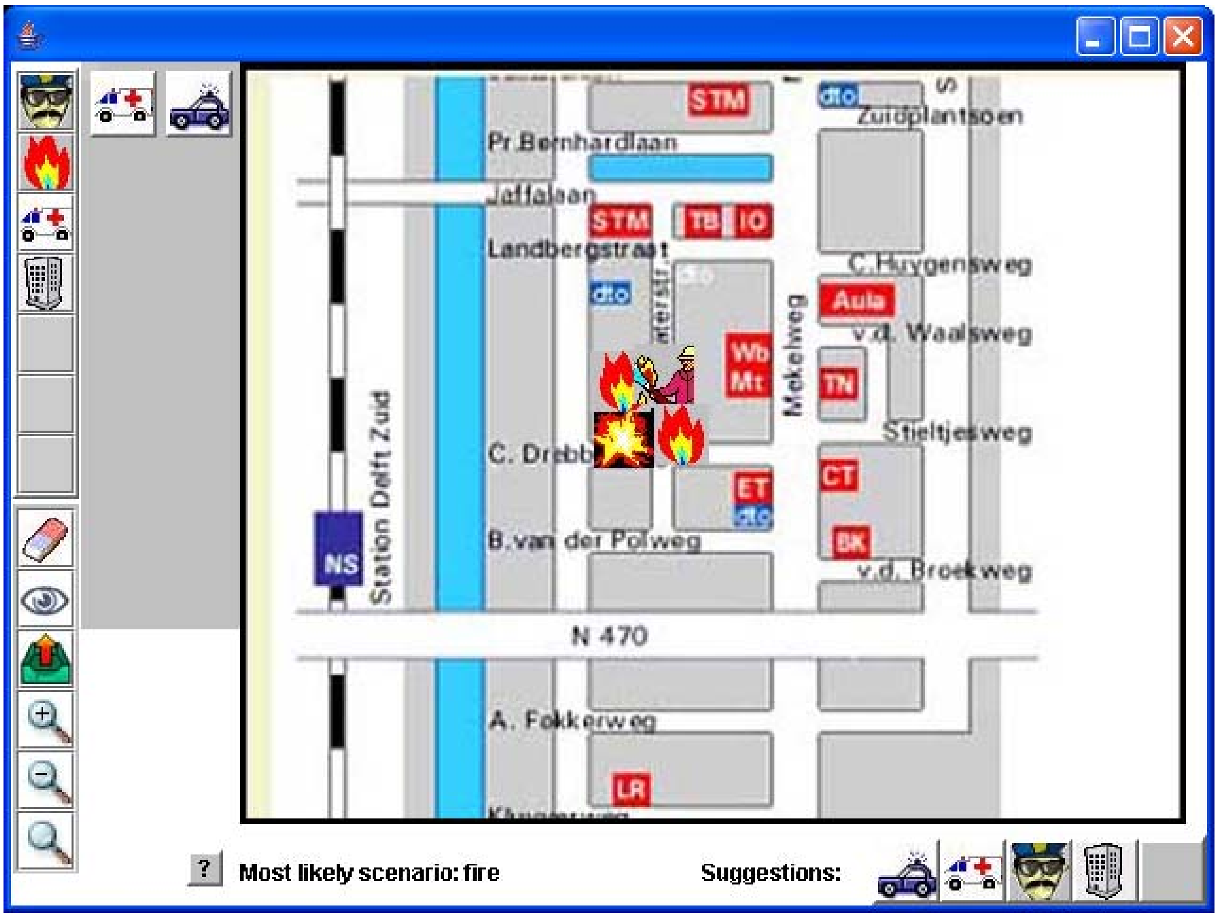
As is stated in the projects problem description, the goal of the system is to communicate with icons using a map. So obviously the icons and the map need to have their place in the GUI. Besides these elements, there also need to be some placed reserved for *control* elements. Things that fall in this category are a delete tool, an inspect option, the ability to send your information to the server, and some zooming possibilities.

Since the clients will be receiving some information about the scenario that they are finding themselves in, there needs to be some room reserved for displaying this information as well. Finally we need some place to display any suggested icons. After many prototypes, the final interface of the main screen that we developed can be seen in Figure 25.

As said before, the application is developed to fit on a handheld computer, and therefore has a resolution of 640x480 pixels. Most of this space is filled up with the map. We tried to set aside as much room as possible for the actual map. This is important because the map should give a direct overview of what is going on.

The top left toolbar is reserved for the icon categories. Although there are only 4 categories in the figure, there is room for 3 extra categories. These can be added to the XML files, and will then automatically be displayed in the GUI. In the grey area right of the categories are the icons that are in this category. This area will open up when a category is selected. There is room for 14 icons per category, so effectively 98 different icons can be used in the application. When an icon is selected to be placed a red border appears around the icon.

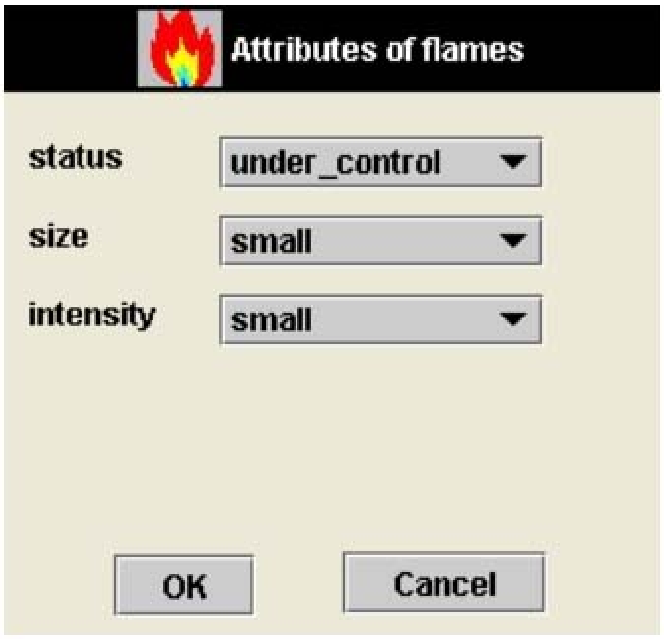
When an icon is selected and the location on the map is being clicked on, a new window opens in which the attributes of the icon can be defined (Figure 26). The application can support a maximum of 5 different attributes per icon, which are defined in the XML file.



Most likely scenario: fire

Suggestions:

Figure 25 The GUI of the report tool



Attributes of flames

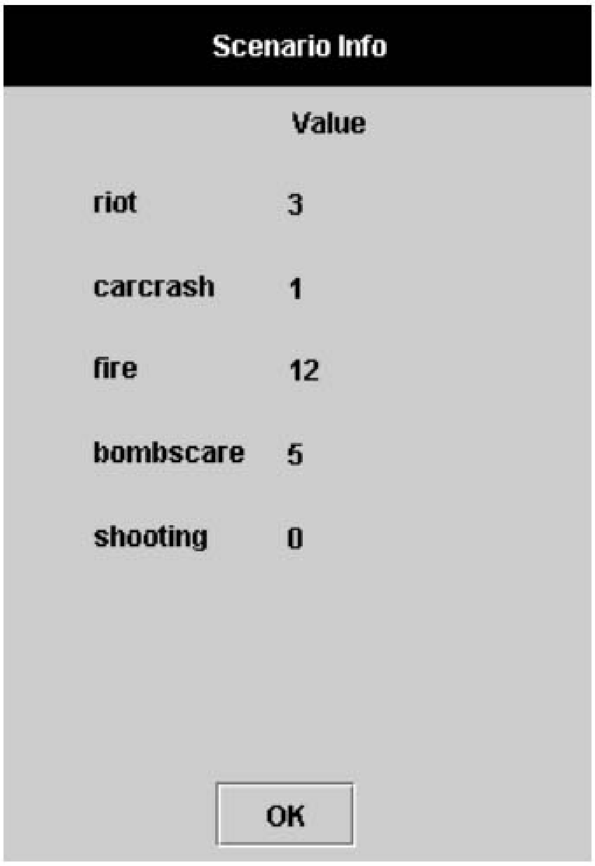
status	under_control
size	small
intensity	small

Figure 26 Attribute window

The bottom left toolbar is reserved for the *control* options. Here the options to delete and inspect are located. When they are selected while an icon on the map is clicked on, it either deletes the icon, or shows its attributes. This latter option is especially relevant when there are icons on the map that were added by other clients. The next button is to send your information to the server and to get an updated world model back.

Finally this toolbar contains some options for zooming. The buttons marked with a + and – magnifying-glass will zoom in and out, centred around the current middle of the map. There are 7 predefined zoom levels. The next zooming function allows the user to drag over the map. The area that will be shown is defined by the top left of the dragged area and the bottom right area of this area.

On the left side below the map is a label that shows the scenario information. There is a button before this label, marked with a question mark, which will bring up a new screen with information about the scenarios. There is room for 7 scenarios. Per scenario a value is printed that is the probability the server has given the scenario. The higher the value, the higher the probability of the scenario. The values are not percentages but custom values, made up by the information in the XML scenario file.



	Value
riot	3
carcrash	1
fire	12
bombscare	5
shooting	0

Figure 27 Scenario Window

The last part of the GUI consists of the icon suggestions of the server. These suggestions are placed on the right side below the map. The best 5 suggestions are shown here, and can be selected to be placed just like the icons on the left.

3.6.2 The Network

Java will also be used for implementing the network. The topic of this thesis is not on networking, that's why the application just simulates a network on a single computer. If the IP addresses are adjusted it's easy to make it a real network over multiple computers. Since we are using java for the network connections, it should be easy to integrate it with JADE [JADE], which seems to be the most promising framework for mobile networks at the moment.

As explained before, the clients send their information to the server, and receive their information from the server, they do not interact with the other clients directly. The client sends its information in three Vectors. The first Vector contains all the icons that were placed since the last time information was send. This prevents the client from sending the same information over and over again. Besides generating less network traffic, this also helps in performance of the server because there is less information to process. After sending the new icons, the client also sends which icons were deleted. Since only new icons were send before, there is no way for the server to figure out if there are icons missing since the last time the client sent data. Finally the client sends a Vector containing any adjusted icons. This Vector is very similar to the first Vector, it contains icons and not just the changes. Since the server knows which Vector will be received, it is clear that on receiving this Vector changes have to be made.

At the other side, the server is going to send information back, after updating its world model. There are two different possibilities of the data that is being send. The client will specify what kind of data it wants. The first type is *send*. This will be used the first time a client connects to the server. The data being send in this case is the map of the surroundings in question, and the icons that are already placed, followed by the scenario information and the icon suggestions. The second type is *update*, this type differs from the first in the fact that the map is not being send to the client. After all, the client already has the map. The icons, scenario information and icon suggestions are still send of course.

3.6.3 Integration

Java is also responsible for the integration of Java, Jess and XML. The XML file about icons is read into Java, and based on the information in them the GUI gets build. The server will read in both the icon information and the scenario information from the XML files. Based on this information the knowledge base and rule base of the Jess engine is build. The integration of the Jess component, is basically importing the right Jess classes and filling the knowledge and rule base dynamically based on the information that was read from the XML files. More information about how this integration is realised is presented in Chapter 4.

Chapter 4: Implementation

In this chapter we will discuss in detail how the system is built. We will show at the hand of diagrams how the different parts of the system work. The most important classes, functions and algorithms will be discussed in detail. Furthermore the Jess component will be explained.

4.1 UML

UML stands for Unified Modelling Language and is a system of diagrams that can specify how systems work. System development focuses on three different models of the system [Bru00]:

1) The functional model is represented in UML by Use Case Diagrams, which specifies the systems functionality from a users perspective.
2) The object model is represented by Class Diagrams and describes the structure of the system in terms of objects, attributes, associations, and operations.
3) The dynamic model is represented by sequence diagrams, state chart diagrams, and activity diagrams. These describe the internal behaviour of the system.

In the next sections we will describe the system according to these three models, and will use one UML representation per model. First we will discuss the Use Case Diagrams, then the Class Diagram, and we will conclude with Sequence Diagrams of some important parts of the system.

4.1.1 Use Case Diagram

Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on an external view of the system. A use case describes a function provided by the system that yields visible results for an actor. An actor describes any entity that interacts with the system (e.g. a user, another system, the systems physical environment).

In the Use Case Diagram in Figure 28 the use case for the reporters is shown. They can add icons, delete icons, inspect or modify icons, view the world model, view the suggestions, view the scenario information, zoom, and send and receive. Some tasks, such as adding icons require some additional actions. To add an icon the icon that needs to be placed has to be selected, a location has to be given and the attributes have to be specified. To delete an icon the icon has to be specified first. To modify or inspect an icon the icon has to be specified, then the current attributes will be given, after which the new attributes can be specified.

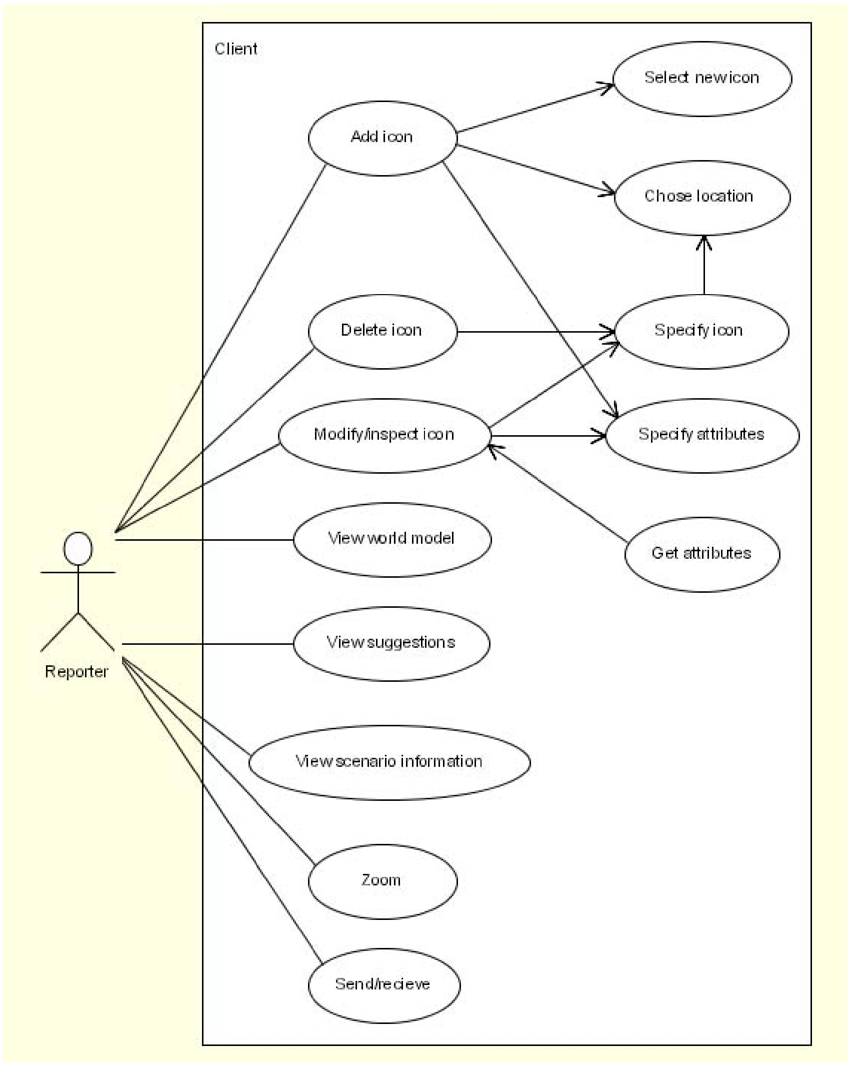


Figure 28 Use case diagram for Reporter

In the next Use Case Diagram (Figure 29) the use case for the client is shown. Just like the reporter gets and inputs information to the client, the client interacts with the server. The client can either send or receive information from the server. Sending will be split up in 3 stages: Send the newly placed icons, send the icons that were modified, and send the icons that were deleted. Receiving from the server is also split up in 3 stages: Receive the icons, receive the information about the scenarios, and receive the suggestions for icons that can be placed.

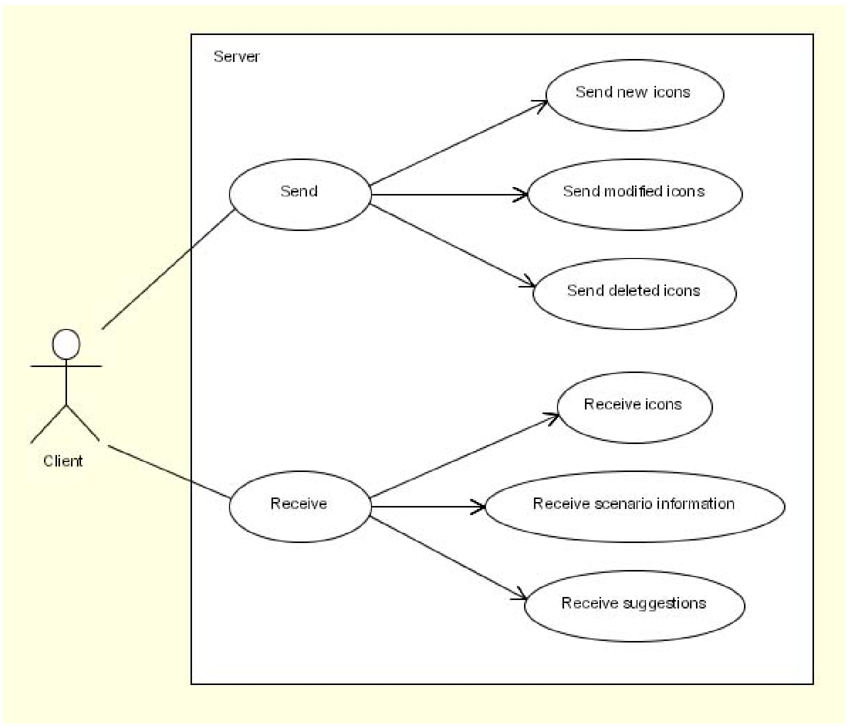


Figure 29 Use case diagram for Client

4.1.2 Class Diagrams

Class Diagrams are used to describe the structure of the system. Classes are abstractions that specify the common structure and behaviour of a set of objects. Objects are instances of classes that are created, modified and destroyed during the execution of the system. An object has a state which includes the values of its attributes and its relationships with other objects. Java programs are build up in classes already and therefore it is easy to create a Class Diagram of it.

In figure 30 the simple Class Diagram for the system is shown. For a more detailed Class Diagram we refer to appendix D: Detailed Class Diagram.

The directed relations in this figure can be read as: x has a y. For example IconApplication has a MyXMLReader. The undirected relations are actually directed in both ways. IconApplication has a ScenarioWindow and ScenarioWindow has an IconApplication. In this case the obvious relation is that the IconApplication, which is the main part of the client, has a ScenarioWindow. The other way around is true as well, because a ScenarioWindow has to know who its parent is to give back the focus to the parent when the window is closed.

Note that the IconApplication and the MapServer do not have a direct relationship. The IconApplication does not have a MapServer and vice versa. They communicate through the network, via a Socket class. More about this in the next section about Sequence Diagrams.

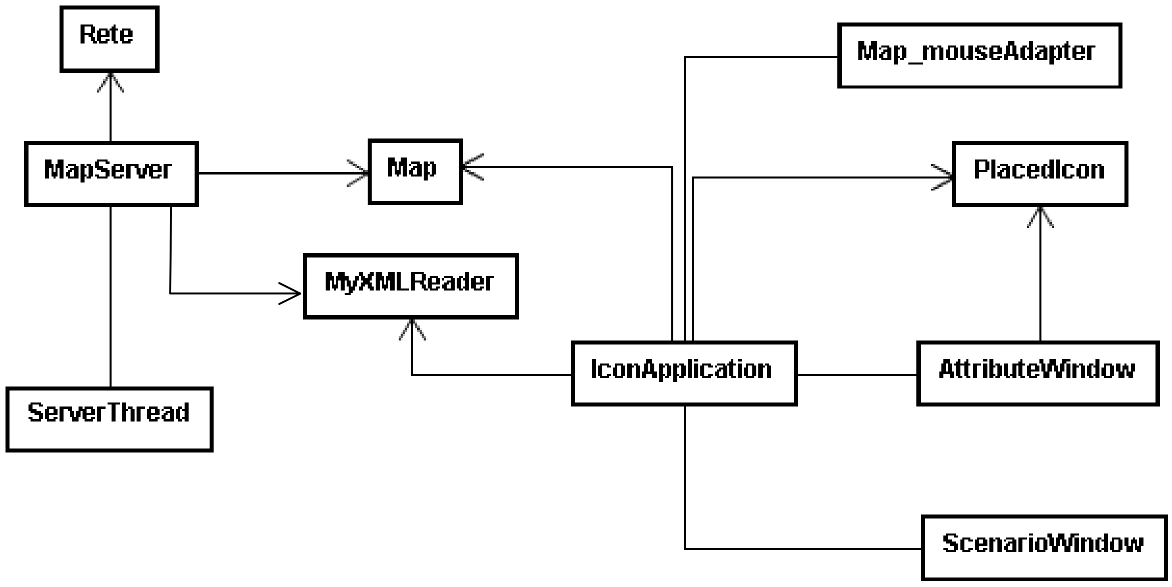


Figure 30 Class diagram of ISME

When we start with the class that has the highest connectivity, IconApplication, we see that this main part of the client has a ScenarioWindow is which the scenario information will be displayed, an AttributeWindow which is used to specify or show the attributes of a selected icon. It also has a PlacedIcon which is a class that defines everything we need to know about an icon that is placed (location, attributes, etc.). Furthermore the IconApplication has a MyXMLReader to help it read in the information from the XML files. It also has a Map_mouseAdapter, which is a class that handles the clicks that were made on the map. And finally it has a Map. Map is a class that stores the background image, the icons that were placed, etc. and it has some functions for zooming.

The MapServer class also has a Map. This is not the same instance as the Map on the IconApplication, but via the network these two instances will be kept synchronized. The ServerThread is the class that handles the connections with the iconApplications. Each serverThread instance will support the connection with one iconApplication, but the mapServer can have many different serverThreads (in fact, as many as the computational power of the server can handle). Finally there is the Rete class. This is the main class for the Jess component, all the reasoning is done via this class. More detail about how the different classes interact is given in the next section.

4.1.3 Sequence Diagrams

Sequence Diagrams are used to represent the flow of events in a system. The objects in the system interact with each other by sending messages. When a message is received this results in some action at the receiving end. Actions that will be executed are operations that may result in the sending of new messages to other objects. Arguments may be passed along with the message to give more detail of what actions need to be undertaken. After the first diagram we will only show the non trivial Sequence Diagrams.

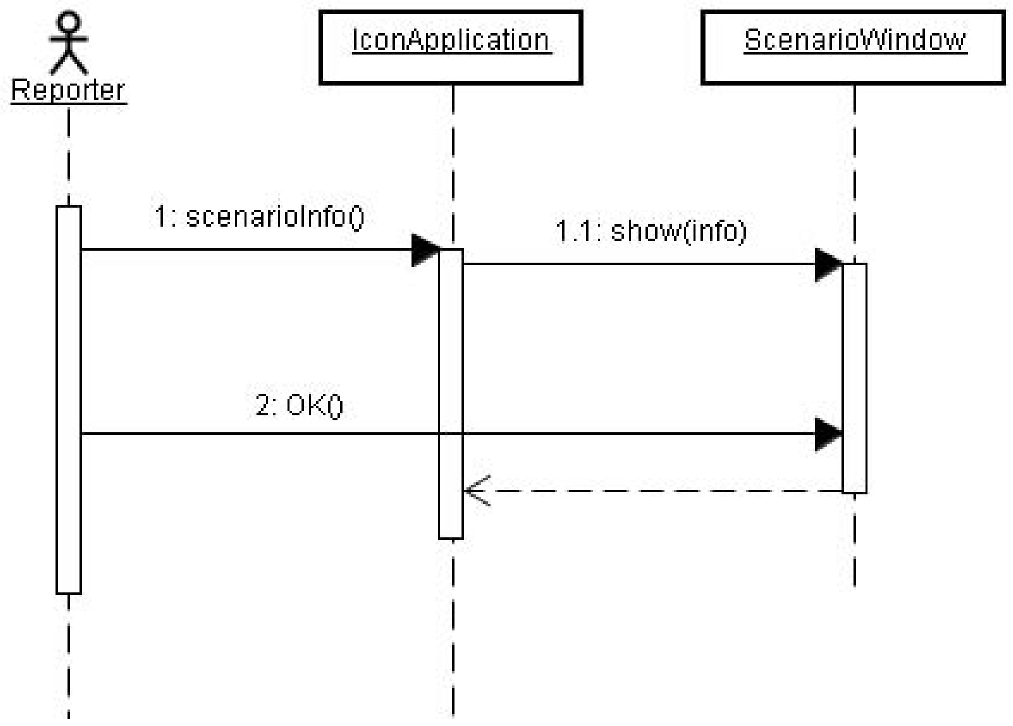


Figure 31 Sequence diagram of showing the scenario window

The first Sequence Diagram in Figure 31 shows how scenario information is shown to the user. The reporter sends a message to get the information to the IconApplication, after that the IconApplication sends a request to show the information. The ScenarioWindow will then show it and wait for the reporter to send an OK message. After that the flow returns to the IconApplication.

Figure 32 shows how icons can be added and modified on the local map. To add a new icon the reporter has to select which icon to place. After that the reporter clicks on the map to indicate where the icon needs to be placed. Note that in reality this happens by the use of the Map_mouseAdapter, but since adding that class does not give any new relevant information, we decided not to show it in this figure, to make it less complex. After the map is clicked on, the AttibuteWindow is told to show the selected icons information. The reporter then selects the appropriate attributes after which the AttributeWindow tells the map to add the icon.

To inspect an already placed icon the reporter has to activate the inspect mode. After clicking on the map (again handled by the Map_mouseAdapter) the Map will decide if there is an icon placed at the specified location. If there is one, the map gets information about which icon is placed. Then the AttributeWindow gets a message to show this icon and the reporter can adjust it, after which the map gets told to modify the icon.

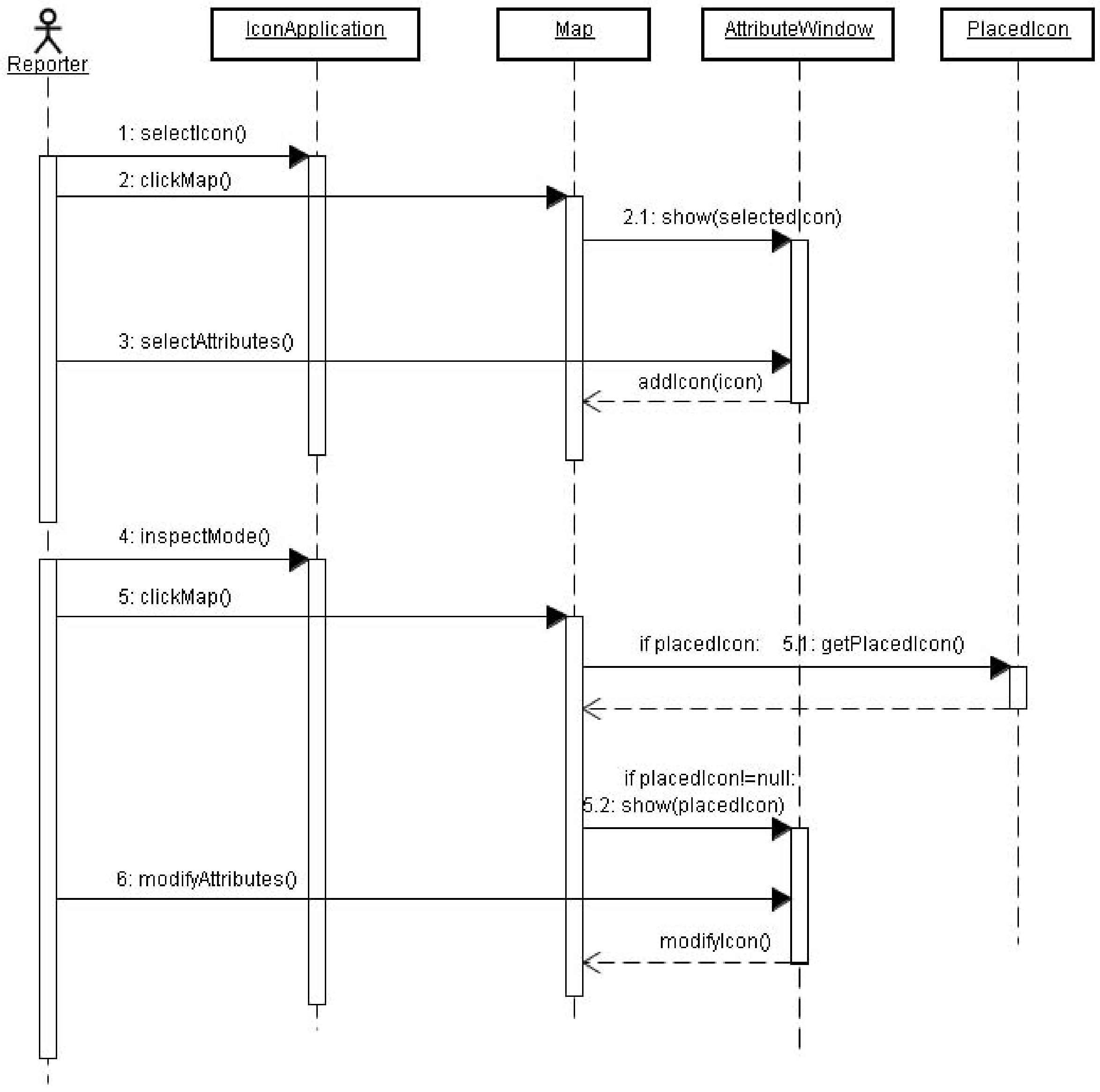


Figure 32 Sequence diagrams of adding and modifying icons on the map

To delete an icon (Figure 33) the flow of events is pretty much the same as those of adding and modifying icons on the map. First the delete mode is activated, then the map is clicked on, and if there is an icon at the specified location, that icon is retrieved and deleted from the map.

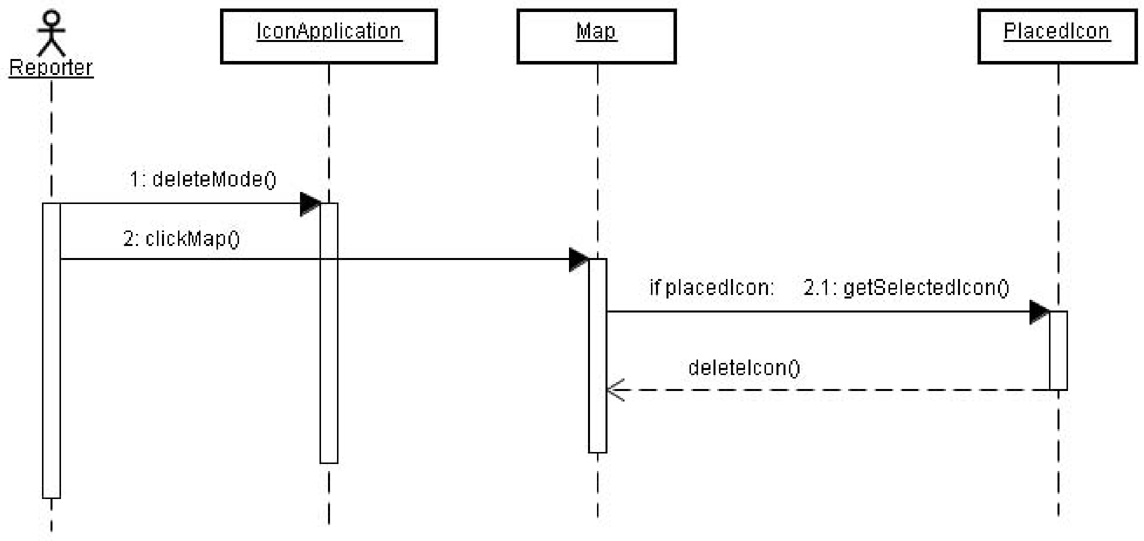
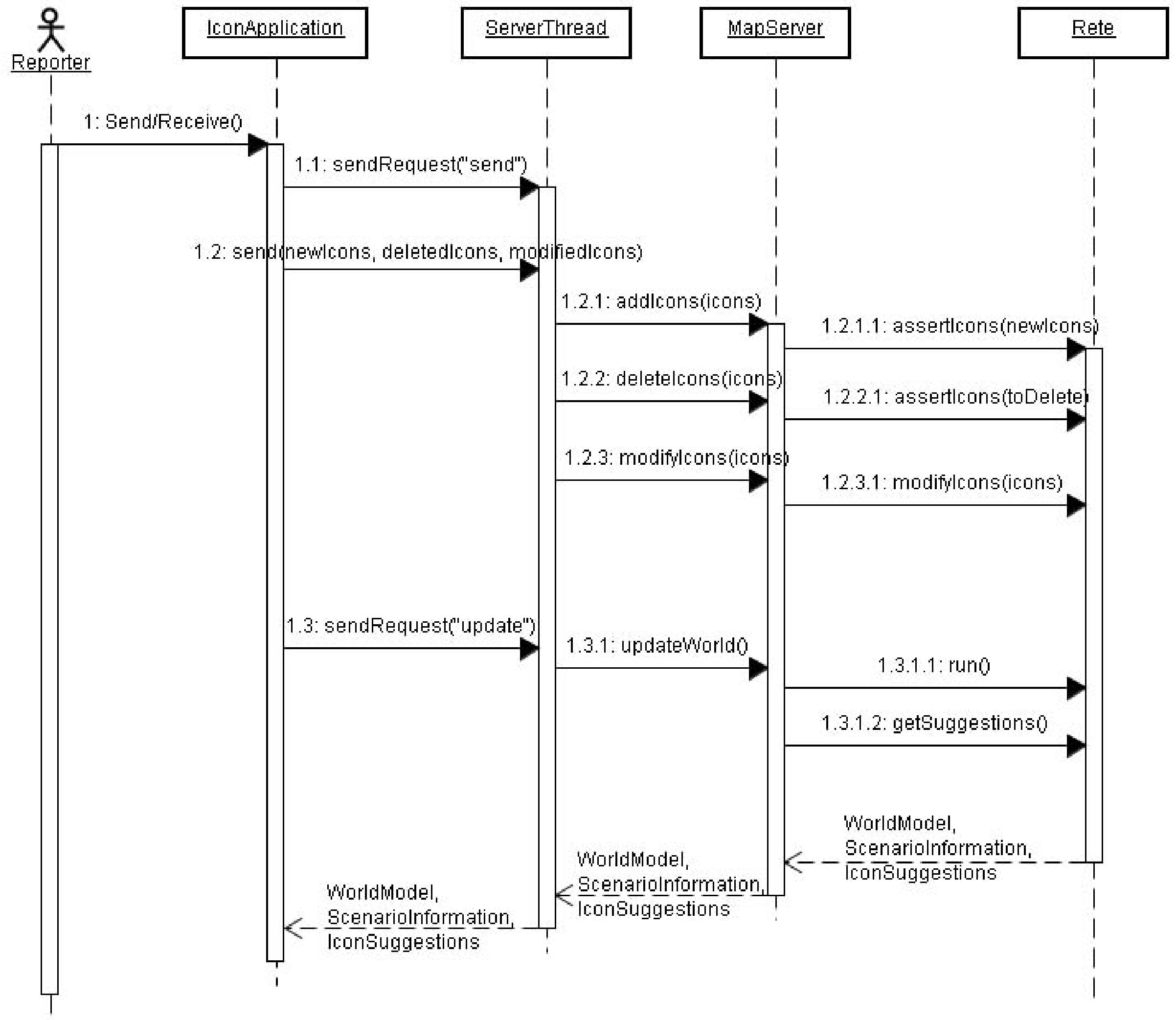


Figure 33 Sequence diagram for deleting placed icons

In Figure 34 we can see the flow of events that happen when the reporter presses the send/receive button. The IconApplication begins with sending a simple String to the ServerThread. The communication between the IconApplication and the ServerThread is not directly, but via Sockets. Both classes use a Socket to communicate over the network with each other. The reason we use a ServerThread to handle the communication, instead of using the MapServer itself directly, is that a MapServer can have multiple ServerThreads that can run at the same time. The advantage of this is that more than one client can communicate with the server at the same time, instead of having to wait for the connection to be free.

The simple String that is send to the ServerThread is to indicate what the client wants from the server. In this case it is “send”, indicating that we want to send our information to the server. The ServerThread reads the request and knows what to do now. In this case it knows more information will be send, namely three Vectors with new icons, deleted icons and modified icons. After the Vectors are received the ServerThread will pass the Vectors on to the MapServer with the request to handle them. The MapServer will then add the information to the Jess knowledge base, represented by the Rete class.

After the information has been send and processed the IconApplication will ask for an update. The ServerThread pass this request on to the MapServer by telling it to update the world. The MapServer in turn will tell the Rete to run, meaning it will match all its facts to the rules. A new world model is the result. The MapServer also requests for the suggestions of the server, both scenario information and suggested icons. This information gets passed back to the IconApplication, where the reporter can see it.



Reporter
IconApplication
ServerThread
MapServer
Rete
1: Send/Receive()
1.1: sendRequest("send")
1.2: send(newIcons, deletedIcons, modifiedIcons)
1.2.1: addIcons(icons)
1.2.1.1: assertIcons(newIcons)
1.2.2: deleteIcons(icons)
1.2.2.1: assertIcons(toDelete)
1.2.3: modifyIcons(icons)
1.2.3.1: modifyIcons(icons)
1.3: sendRequest("update")
1.3.1: updateWorld()
1.3.1.1: run()
1.3.1.2: getSuggestions()
WorldModel, ScenarioInformation, IconSuggestions
WorldModel, ScenarioInformation, IconSuggestions
WorldModel, ScenarioInformation, IconSuggestions

Figure 34 Sequence diagram of sending to and receiving from the server

4.2 Jess

In this section we will elaborate on the inner workings of the Jess component. When we look back to Figure 24 we see what functionality needs to be added. All the input that comes from the client will be asserted as facts to the knowledge base. Even for icons that need to be deleted, we will assert a fact. The rule base should decide to throw it away or not with its rules for newly added facts, deleted facts and modified facts.

First we will see how we can add facts to our knowledge base, how icons will be deleted, and how modifications take place. After that we will explain how the double icons filter works. We will continue with the way the server determines the scenario, and finally we will explain how the system can give suggestions for next icons.

4.2.1 Adding Facts to Jess.

When a client sends its new information, this should be added to the Jess knowledge base. To do this we will add it as a fact. The knowledge base is the collection of all facts that have been inputted to it. In Jess, there are three kinds of facts: *ordered facts*, *unordered facts*, and *definstance facts.*

Ordered facts are simply lists, where the first field (the *head* of the list) acts as a sort of category for the fact. Here are some examples of ordered facts:

```
(policeman 100 200 idle Paul)
(flames 120 300 expanding huge huge Paul)
```

Figure 35 Example of ordered facts

Ordered facts are useful, but they are unstructured. We want a bit more organization. In object-oriented languages, *objects* have named *fields* in which data appears. Unordered facts offer this capability (although the fields are traditionally called *slots.*) When we rewrite the previous ordered facts to unordered facts we get:

```
(policeman (x 100) (y 200) (status idle) (supporter Paul)
(flames (x 120) (y 300) (status expanding) (intensity huge)
```

Figure 36 Example of unordered facts

before you can create unordered facts, you have to define the slots they have using the `deftemplate` construct:

```
(deftemplate <deftemplate-name> [extends <classname>] [<doc-comment>]
     [(slot <slot-name> [(default | default-dynamic <value>)]
                          [(type <typespec>))]*)
```

Figure 37 Deftemplate construct

The `<deftemplate-name>` is the head of the facts that will be created using this template. There may be an arbitrary number of slots. Each `<slot-name>` must be an atom. The `default` slot qualifier states that the default value of a slot in a new fact is given by `<value>`; the default is the atom `nil`. The 'default-dynamic' version will evaluate the given value each time a new fact using this template is asserted. The 'type' slot qualifier is accepted but not currently enforced by Jess; it specifies what data type the slot is allowed to hold. Acceptable values are ANY, INTEGER, FLOAT, NUMBER, ATOM, STRING, LEXEME, and OBJECT. Since they are currently not enforced by Jess, we will not use these.

```
(deftemplate policeman
    "A policeman."
    (slot x)
    (slot y)
    (slot status (default unknown)))
```

Figure 38 Example of a deftemplate for a policeman

The example in Figure 38 would allow us to define facts like this:

```
(assert (policeman (x 100) (y 200)  (status idle)))
(assert (policeman (y 200) (x 100))
```

Figure 39 Examples of facts, defined by the deftemplate of Figure 38

Note that the status of the policeman is unknown by default. If we don't supply a default value for a slot, and then don't supply a value when a fact is asserted, the special value `nil` is used.

Because we want to get all information about which icons we can use from the XML files, we need to make deftemplates for every icon in the list. Of course we have to read in the data from the XML file first and then form Jess commands of it. This will all happen in the initialisation phase of the server. When the server is up and running, it will get information from the clients, in the form of the placed icons. These need to be transformed into Jess facts, and asserted into the knowledge base. How this is done will be discussed in section 4.3.1: From XML to Jess.

Since we will have rules that apply to all icons, rather than just an icon of a specific type it would be a good idea to let every type of icon extend an overarching *icon* fact. Figure 40 shows how this can be done.

```
(deftemplate icon
    (slot name)
    (slot x)
    (slot y)
    (multislot supporters))

(deftemplate policeman extends icon
    (slot status))

(assert (policeman (name policeman) (x 100) (y 100) (status busy)
     (supporters Paul)))
```

Figure 40 Extending from other deftemplates

There seems to be a little bit of overhead, because when we assert a new policeman now, we specifically add a slot *name* to the icon, where the *name* should just be ‘policeman’. This is the easiest way however, to see what the type of the icon is. The server can now add facts according to the Jess code in Figure 40, when the client provides them with the icons that need to be added.

4.2.2 Deleting Facts from Jess

When looking back to Figure 24, we see that the clients can also tell the server to delete some icons. Rather than doing this without any questions asked, we would prefer to add a fact that tells the server to delete a certain icon. This approach resembles a blackboard where everyone can put sticky notes on with a request. The owner of the blackboard, the server in this case, can then take off the notes and decide what to do with them. The facts that will be asserted in order to let the server know what icons to delete is shown in Figure 41. When facts like these are asserted, the rule to delete icons will be triggered. A rule like this is shown in Figure 42.

```
(deftemplate icon_to_delete (slot name)(slot x)(slot y)(slot deleted))

(assert (icon_to_delete (name policeman) (x 100) (y 100) (deleted 0)))
```

Figure 41 Asserting icon_to_delete

Figure 42 Rule for deleting icons

4.2.3 Modifying Jess Facts

Facts can also be modified. The client will send a separate list of icons that need to be modified by the server. There is a built-in function to modify facts, Figure 43 shows the modify construct and an example of how to modify an icon. To modify an icon you need the fact-id of the fact that needs modification. In the example we get this by asserting a fact first, in the real system we will have to search for the right id, using queries.

Figure 43 Modifying an icon

4.2.4 Rules about Double Placed Icons

Often it will happen that the server receives double icons. If one client reports an icon and another client hasn't updated his world yet, this client could place the same icon and the server will receive the same information twice. Jess should filter these double occurrences out.

When a double event is reported by only one client there is a big chance that there are actually two distinct occurrences. In this case the Jess engine should not filter one of them out.

However, when two clients report a same event in approximately the same place these occurrences are probably the same, and one should be filtered. This is the reason we keep track of which reporter reported which icon. The supporters of an icon is a list with all the reporters that reported this icon. The reporters have a name and a location, which are defined at the start up of the client. The filtering can happen in several ways:

1) Both the occurrences get deleted, and the system adds one occurrence at the average location. That is:

$$\text{newX} = \frac{(x_1+..+x_n)}{n}$$

$$\text{newY} = \quad (y\ +..+y\ )$$

2) The system calculates which client is closest to its reported event, and deletes all the other occurrences. This might be a reasonable solution because we think the closer a client is to the event, the more accurate his report will be.

3) A combined version of 1. and 2. Although we think that the closer client might provide the more accurate report, we don't think that the other client just reports nonsense. That's why we could add weights to the reports of the clients. The closer clients report will get a higher weight than the other reports.

$$\text{newX} = \frac{(w_1x_1 + .. + w_nx_n)}{n(w_1 + .. + w_n)}$$

$$\text{newY} = \quad (w\ y\ \ + .. + w\ y\ )$$

Now the question remains how to distribute the weights. In practice, when this rule is fired there will be one icon already in the knowledge base, which might be reported by several reporters, and one new icon that is just placed. This means there are 2 groups, which results in the following formula:

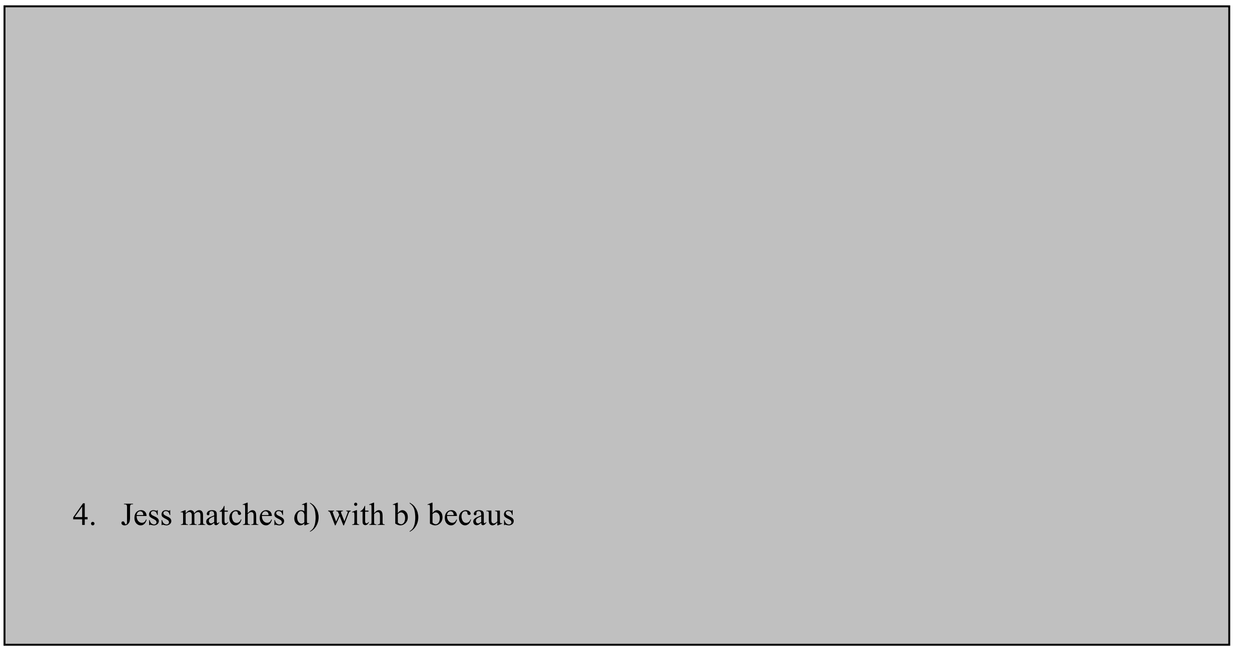
$$\text{newX} = \frac{w_1(x_{11} + .. + x_{1n}) * \text{groupsize1} + w_2(x_{21} + .. + x_{2n}) * \text{groupsize2}}{w_1 * \text{groupsize1} + w_2 * \text{groupsize2}}$$

$$\text{newY} =$$

Where the weights w1 and w2 have either value 1 or 2. The group that is closest to the event they report gets weight 2, the other group gets weight 1.We have chosen to implement this last, most sophisticated design.

Besides the filtering we have to check if the events are close enough to each other to even consider counting them as double reported events. So we will only filter when the events are within a certain range of each other. We have set that range at a maximum of 10 pixels. Note that this range is dependent on the resolution of the map.

For this rule to work correctly we need to keep track of a list of supporters of each reported icon. If we don't do this some icons could be deleted unwanted.



4. Jess matches d) with b) becaus

Figure 44 Double reported icons, example 1

In example 1 the approach of giving each reported icon 1 owner seems to work. In steps 3 and 5 Jess adds a new icon, with the 'system' as owner. The next example, however, shows how this approach will go wrong. If we take a look at example 2 in Figure 45 we have a problem, because in the end we have only one policeman left, even though both clients reported 2 of them.

We will solve this problem by not using one single owner per event, but instead keep track of which clients 'supported' the event. When each icon has a list of supporters this problem can be avoided. In this case Jess action 4 will not occur because e) is supported by the same client that reports a new policeman. Instead the outcome of the example will look like Figure 46.

Example 2:
We will assume again that the clients are at the same distance from their reported event.
a) Client 1 reports a police man at location 200,210
b) Client 2 reports a police man at location 215,208
c) Client 1 reports a police man at location 210,210
d) Client 2 reports a police man at location 210,205

This will be filtered by Jess as follows:
1. Jess adds a)
2. Jess matches b) with a) and deletes both to add a new police man at 207.5,209 (e)
3. Jess matches c) with e) because a) and b) are deleted already.
4. Jess deletes c) and e) and adds a new police man at 208.75,209.5 (f)
5. Jess matches d) with f) because c) and e) are deleted already.
6. Jess deletes d) and f) and adds a new police man at 209.375,207.25 (g)

Figure 45 Double placed icons, example 2

1. Jess adds a)
2. Jess matches b) with a) and deletes both to add a new police man at 207.5,209 (e)
3. Jess adds c) because it cant be matched with e). The reason for this is that the owner of c) also supports e)
4. Jess matches d) with c) and not with e) because the owner of d) also supports e)
5. Jess deletes d) and c) and adds a new police man at 210,207.5 (f)

Figure 46 Outcome of the example while using multiple owners

Now that we still have our two police men, we need to develop an actual rule that can be used in Jess. Because this rule does not only apply to policemen but to all icons, the rule should apply to *icon*, rather than each individual type of icon. We do however have to take into consideration the actual type of icon we are dealing with. We don't want to filter out a policeman, just because somebody else placed a fire icon at around the same position.

The rule we developed is shown in Figure 47.

```
(defrule filter_double_icons
        ?icon1 <- (icon (name ?n) (x ?x) (y ?y) (supporters $?list1))
        ?icon2 <- (icon (name ?n)
            (x ?x1&: (< (- ?x ?x1) 10)& ?x1&: (< (- ?x1 ?x) 10))
            (y ?y1&: (< (- ?y ?y1) 10)& ?y1&: (< (- ?y1 ?y) 10))
            (supporters $?list2))
        (test (neq ?icon1 ?icon2))
        (test (not (has_shared_element ?list1 ?list2)))
      =>
        (combine_icons ?icon1 ?icon2)
)
```

Figure 47 Filter double icons rule

This rule matches all icons with each other and if they have the same name, which means, if they are of the same type, e.g. policeman, it matches the position of the icons. If the difference in position is smaller than 10 in x direction, or 10 in y direction, it will fire. Of course it has to be checked if the icon is not matched against itself, or it will obviously always fire. We do this with the built-in `neq` function, which only returns `true` if the first argument is not the same as the second. Furthermore it tests if the icons have a shared supporter, by testing it with function `has_shared_element`. The rule will only fire when it has no shared supporters.

When all the preconditions have been met, and the rule fires, we will combine the two icons into a new one, according to the formulas we stated before. Besides calculating the new position, the combine_icons function combines the supporters as well. The function can be seen in Figure 48.

```
(deffunction combine_icons (?icon1 ?icon2)
      (bind ?f1 (fact-id ?icon1))
      (bind ?supporters1 (fact-slot-value ?f1 supporters))
      (printout t "supporters1: " ?supporters1 crlf)
      (bind ?f2 (fact-id ?icon2))
      (bind ?supporters2 (fact-slot-value ?f2 supporters))
      (printout t "supporters2: " ?supporters2 crlf)
      (printout t "now we should combine them" crlf)
      (bind ?supporters_to_add (complement$ ?supporters1
                                            ?supporters2))
      (bind ?supporters_total (insert$ ?supporters1 1 ?supporters2))
      (bind ?x1 (fact-slot-value ?f1 x))
      (bind ?y1 (fact-slot-value ?f1 y))
      (bind ?x2 (fact-slot-value ?f2 x))
      (bind ?y2 (fact-slot-value ?f2 y))
      (bind ?number_of_supporters1 (length$ ?supporters1))
      (bind ?number_of_supporters2 (length$ ?supporters2))
      (bind ?closest_x1 9999999)
      (bind ?closest_y1 9999999)
      (bind ?closest_x2 9999999)
      (bind ?closest_y2 9999999)
      (bind ?count 0)
      (while (< ?count (length$ ?supporters1))
      ;DETERMINE CLOSEST REPORTER IN GROUP 1
            (bind ?supporter_name (nth$ (+ 1 ?count) (create$
                                                ?supporters1)))
            (bind ?count (+ ?count 1))
            (bind ?rep (run-query search_reporters ?supporter_name))
            (while (?rep hasNext)
                  (bind ?token (call ?rep next))
                  (bind ?fact (call ?token fact 1))
                  (bind ?x (fact-slot-value ?fact x))
                  (bind ?y (fact-slot-value ?fact y))
                  (bind ?n (fact-slot-value ?fact name))
            )
            (bind ?distance_x (abs(- ?x ?x1)))
            (if (<  ?distance_x ?closest_x1)
                  then (bind ?closest_x1 ?distance_x)
            )
            (bind ?distance_y (abs(- ?y ?y1)))
            (if (<  ?distance_y ?closest_y1)
                  then (bind ?closest_y1 ?distance_y)
            )
      )
```
continued on next page…

```
      (bind ?distance_group1 (sqrt (+ (* ?closest_x1 ?closest_x1)
                                      (* ?closest_y1 ?closest_y1))))
      (bind ?count 0)
      (while (< ?count (length$ ?supporters2))
      ;DETERMINE CLOSEST REPORTER IN GROUP 2
            (bind ?supporter_name (nth$ (+ 1 ?count)
                                  (create$ ?supporters2)))
            (bind ?count (+ ?count 1))
            (bind ?rep (run-query search_reporters ?supporter_name))
            (while (?rep hasNext)
                  (bind ?token (call ?rep next))
                  (bind ?fact (call ?token fact 1))
                  (bind ?x (fact-slot-value ?fact x))
                  (bind ?y (fact-slot-value ?fact y))
                  (bind ?n (fact-slot-value ?fact name))
            )
            (bind ?distance_x (abs(- ?x ?x2)))
            (if (<  ?distance_x ?closest_x2)
                  then (bind ?closest_x2 ?distance_x)
            )
            (bind ?distance_y (abs(- ?y ?y2)))
            (if (<  ?distance_y ?closest_y2)
                  then (bind ?closest_y2 ?distance_y)
            )
      )
      (bind ?distance_group2 (sqrt (+ (* ?closest_x2 ?closest_x2)
                                      (* ?closest_y2 ?closest_y2))))
      (bind ?weight_group1 1)
      (bind ?weight_group2 1)
      (if (< ?distance_group1 ?distance_group2)
       then (bind ?weight_group1 2)
      )
      (if (< ?distance_group2 ?distance_group1)
       then (bind ?weight_group2 2)
      )
      ;the 'winning group' now has weight 2
      (bind ?new_x (/(+
            (* ?weight_group1 (* ?x1 ?number_of_supporters1))
            (* ?weight_group2 (* ?x2 ?number_of_supporters2)))
            (+ (* ?weight_group1 ?number_of_supporters1)
               (* ?weight_group2 ?number_of_supporters2))))
      (bind ?new_y (/(+
            (* ?weight_group1 (* ?y1 ?number_of_supporters1))
            (* ?weight_group2 (* ?y2 ?number_of_supporters2)))
            (+ (* ?weight_group1 ?number_of_supporters1)
               (* ?weight_group2 ?number_of_supporters2))))

      ;now combine the icons by modifying one and deleting the other
      (modify ?f1 (x ?new_x))
      (modify ?f1 (y ?new_y))
      (modify ?f1 (supporters ?supporters_total))
      (retract ?f2)
)
```

Figure 48 Jess code to combine 2 icons

The code in Figure 48 may seem a bit overwhelming, especially for those who are not familiar with Jess. What the function does is:

- Combine the supporters
- Determine the closest supporter of ‘team 1’
- Determine the closest supporter of ‘team 2’
- Give the ‘winning team’ a double weight
- Calculate the new position
- Modify one icon to have the combined supporters and new location
- Delete the other icon

Note that this function does not average the attributes. When a busy and an idle policeman are reported, the last reported attribute will be chosen.

4.2.5 Determining the Current Scenario

At the server side there will be some thoughts about scenario’s. The system will read in different scenario’s from the XML files, and by looking at which icons are in the world model, it will determine in what scenario we are. The contents of each scenario, and the values the icons get awarded in the XML files is abstracted from news articles which can be found in Appendix A: Scenarios. The scenario information will be send back to the reporters as general feedback.

The procedure of determining the scenario is:

- We make a variable for each scenario, and set it to 0.
- Every time an icon gets added, it will award points to the corresponding scenarios.
- When a certain threshold is reached the scenario is believed to be true and set to be the current scenario.
- When another scenario gets a higher score than the current scenario, the current scenario gets exchanged.

To keep track of the likely scenario we will make a variable for each defined scenario. The code will award points towards each of these variables, if a newly placed icon would be appropriate for the scenario. The easiest way to do this is simply add a rule for each placed icon, as can be seen in Figure 49.

If we do this for each added icon we would get a score for each scenario. However, there is a problem if we delete icons again. Then we would have to subtract all the awarded points for each scenario. This would mean that in addition of having a rule for every type of icon that gets added, we would also have to make a rule for every type of icon that gets deleted.

```
(bind ?scenario_riot 0)
(bind ?scenario_carcrash 0)
(bind ?scenario_fire 0)
(bind ?scenario_bombscare 0)
(bind ?scenario_shooting 0)

(defrule new_policeman
      ?fact <- (policeman (name ?policeman) (x ?x) (y ?y) (status ?s)
                              (supporters ?sup))
=>
      (bind ?scenario_riot (+ ?scenario_riot 4))
      (bind ?scenario_carcrash (+ ?scenario_carcrash 1))
      (bind ?scenario_fire (+ ?scenario_fire 1))
      (bind ?scenario_bombscare (+ ?scenario_bombscare 2))
      (bind ?scenario_shooting (+ ?scenario_shooting 3))
)
```

Figure 49 Awarding points to scenarios, with a rule for every icon

This means a lot of extra bookkeeping especially in the case of double placed icons. In this case there would be two icons added, one icon deleted and one icon modified. This means we have to modify the scenario variables 4 times, each time an icon is double placed. It also means that if a reporter adds 5 icons at once, the scenario values get modified 5 times. It would be better, and more efficient if we would just calculate which scenario is most likely, every time we want to know. A good time to calculate this would be when the client sends his icons. The server will then calculate a new world model, possible scenario and suggested icons. When the server has calculated these things, it can be send back to the client.
To calculate which scenario is going on, Java will fire a Jess function and get back the value of the scenarios. At start up of the application we will add all the scenario_chances, according to the XML file. See Figure 50.

```
(deftemplate scenario_chance
      (slot scenario_name)
      (slot icon_name)
      (slot value))

(deftemplate scenario_suggestion
      (slot scenario_name)
      (slot value))
```

This will results in facts like:

```
(assert (scenario_chance (scenario_name riot) (icon_name flames)
                              (value 2)))
(assert (scenario_chance (scenario_name riot) (icon_name policeman)
                              (value 3)))
(assert (scenario_chance (scenario_name carcrash) (icon_name car)
                              (value 5)))
```

and at start up we will assert the following facts:

```
(assert (scenario_suggestion (scenario_name riot) (value 0)))
(assert (scenario_suggestion (scenario_name fire) (value 0)))
(assert (scenario_suggestion (scenario_name carcrash) (value 0)))
```

Figure 50 Examples of facts that get asserted at start up of the application

With asserted facts like these we can define a function that calculates the chances for each scenario. We will define 2 variables first, to keep track of what types of icon are already placed, and what types of icons exist, according to the XML files.

```
(bind $?*placed* (create$ policeman soldier bomb))
(bind $?*all* (create$ policeman soldier bomb flames victim …))
```

Figure 51 The creation of 2 useful variables

The final function that awards the points to the scenarios is shown in Figure 52. It will award points by looking at the icons that are already placed, and the values these icons give to the different scenarios.

```
(deffunction check_suggested_scenarios ()
      (foreach ?x $?*placed*
            (bind ?y (run-query is_in_scenario ?x))
            (while (?y hasNext)

                  (bind ?token (call ?y next))
                  (bind ?fact (call ?token fact 1))
                  (bind ?name (fact-slot-value ?fact scenario_name))
                  (bind ?value (fact-slot-value ?fact value))
                  (bind ?suggestion (run-query
                              search_scenario_suggestion ?name))
                  (bind ?sug_token (call ?suggestion next))
                  (bind ?sug_fact (call ?sug_token fact 1))
                  (bind ?old_value (fact-slot-value ?sug_fact value))
                  (modify ?sug_fact (value (+ ?value ?old_value)))
            )
      )
)
```

Figure 52 Function for awarding points to the scenarios

After executing this function the scenario_suggestion facts are updated, and the suggestion with the highest value can be picked as suggestion.

4.2.6 The Next Icon Predictor

Besides giving the reporter feedback with the newly calculated icons and the most likely scenario, the server will also give some suggestions for icons that might be placed next. This is useful if the reporter forgot to report an icon. According to the placed icons in the current world model, the server will suggest icons to the client that it thinks are missing, or would be a good choice as a next icon. For example, if flames were reported, the server may suggest a smoke icon as well, as long as there is no smoke icon already.

In order to make a reasonable prediction, the server can make use of 3 sources of information:

1) Every icon has a list of probable next icons
2) Every icon has a list of probable previous icons
3) Every scenario has a list of icons

In the XML file for the icons, besides it's name, image, attributes, etc, every icon has a list of probable next and previous icons. These icons also have a number attached to them, telling how probable the relations are. These numbers are on a scale from 1 to 5, where 1 would mean possibly and 5 would mean definitely. For example 'fire' would have as probable next icon 'smoke' with probability 5. It would also have 'smoke' as probable previous icon with probability 5, because often smoke is reported first, when the fire cannot be seen yet. The next and previous relations are therefore causal relations.

The server is already keeping track of scenario's. Every scenario has a list of icons that makes up the scenario. When enough of these icons are in the world model, the scenario is probably happening. With this, we have a great source to predict new icons, namely those icons that should be in the scenario that's going on, but are not on the map yet.

To implement a way to suggest new icons, we need to keep track of how many points each predicted icon has been given. Just as in the calculation of the scenarios we will only calculate this when we need to know the results, in order to prevent us from too much overhead in the bookkeeping of the variables. Icons will only be suggested if they are not on the map yet. This will help us in performance, since we only have to calculate the values for icons that are not on the map yet.

We want to calculate suggestions every time the clients sends its icons and wants to receive an update of the world model. In order to do this, we obviously need a function that will calculate the chances of every icon to be placed next. Since we only need to check for icons that are not on the map yet, we will be using the variable `$?*placed*` again, together with variable `$?*all*` to calculate which icons have not been placed yet.
The function we designed is shown in Figure 53.

```
      (bind ?z (run-query previous_of ?x))"+
      (while (?z hasNext)
            (bind ?toke  (cal  ?z n xt))
            (bind ?fact (call ?token fact 1))
            (bind ?name (fact-slot-value ?fact name))
            (bind $?icon (create$ ?name))"+
            (bind $?intersection (intersection$ $?*placed* $?icon))
            (if (> (length$ $?intersection) 0) then
                  (bind ?value (fact-slot-value ?fact value))
                  (bind ?suggestion (run-query search_suggestion ?x))
                  (bind ?sug_token (call ?suggestion next))
                  (bind ?sug_fact (call ?sug_token fact 1))
                  (bind ?old_value (fact-slot-value ?sug_fact value))
                  (modify ?sug_fact (value (+ ?value ?old_value)))
            )
      )
      (bind ?q (run-query is_in_scenario ?x))
      (while (?q hasNext)
            (bind ?token (call ?q next))
            (bind ?fact (call ?token fact 1))
            (bind ?name (fact-slot-value ?fact scenario_name))
            (bind ?value (fact-slot-value ?fact value))
            (bind ?s (run-query search_scenario_suggestion ?name))
            (bind ?token_s (call ?s next))
            (bind ?fact_s (call ?token_s fact 1))
            (bind ?factor (/ (fact-slot-value ?fact_s value) 10))
            (bind ?suggestion (run-query search_suggestion ?x))
            (bind ?sug_token (call ?suggestion next))
            (bind ?sug_fact (call ?sug_token fact 1))
            (bind ?old_value (fact-slot-value ?sug_fact value))
            (modify ?sug_fact(value (+ (* ?value ?factor) ?old_value)))
      )
  )
)
```

Figure 53 Jess code to calculate the icon suggestions

This function looks for all the icons that exist but have not been placed yet. Then for every icon in this group it starts looking for other icons that can award points to it. Of course the icons that may award points have to be placed themselves. First this is done by looking if any placed icon has this icon as a next relation, if so the points are awarded to the suggestion. After this we do the same for previous relations, again points get awarded to the suggestion. Finally we look if the suggested icon is present in any scenario. The points awarded for this are of course dependent on the chance that the scenario is actually happening. When this is done for all the icons that are not placed yet, we have a list of suggestions and their score. The highest scoring suggestions will be send to the reporter as feedback.

4.3 Algorithms and Functions

In this section we will discuss some of the functions and algorithms that are used in the system. First we will explain in detail how we can create Jess code from the XML files, and let it run in Java. After that we will discuss the algorithms we designed for zooming on the map.

4.3.1 From XML to Jess

This section will discuss how get from the information that is stored in the XML files, to the generation of Jess code to get intelligent behaviour of the server. First we need to construct an instance of the Rete class in the Jess package. Then we can add literal Jess code to it with the `executeCommand()` function. This is very convenient, because we designed all the Jess code first, and when it was finished we integrated it with Java.Figure 54 shows an example of how we can add the code to the Rete instance in Java.

```
Rete r = new Rete();
r.executeCommand("(deftemplate icon (slot name)(slot image)
                  (slot x)(slot y)(multislot supporters))");
r.executeCommand("(deftemplate next_relation (slot name)
                  (slot next_icon)(slot value))");
r.executeCommand("(deftemplate previous_relation (slot name)
                  (slot previous_icon)(slot value))");
r.executeCommand("(deftemplate suggestion (slot name)(slot value))");
```

Figure 54 Integrated Jess code in Java

To read in the information of the XML files we make use of the class MyXMLReader, and create Strings from the information we get out of it. After that we insert them into the Rete instance. Figure 55 shows how the deftemplates and previous and next relations are created dynamically from the XML files.

```
MyXMLReader iconsXML = new MyXMLReader("icons/iconlist.xml");

int numberOfGroups = iconsXML.getGroups();
for(int i=0; i<numberOfGroups; i++) //for every group:
{
        String[] iconsInGroup = iconsXML.getIconsInGroup(i);
        for(int j=0; j<iconsInGroup.length; j+=2)
        //for every icon in the group
        {
          //adding deftemlates from the XML files for all icons
          String iconname = iconsInGroup[j];
          String jessCode = "(deftemplate "+iconsInGroup[j]+
                                " extends icon";
          //get information about slots, next and previous relations
          int [] numbers = iconsXML.getNumbers(iconsInGroup[j]);
          int slots = numbers[0];
          int next = numbers[1];
          int prev = numbers[2];
```

Continued on next page ...

```
          for(int k=0; k<slots;k++)
          //get the slots
          {
            Vector slot = iconsXML.getSlot(iconname, k);
            jessCode = jessCode + " (slot "+slot.get(0)+")";
          }
          jessCode = jessCode + ")";
          r.executeCommand(jessCode);
          r.executeCommand("(assert (suggestion (name "+iconname+
                              ")(value 0)))");

          //now add the next relations from XML file
          for(int k=slots; k<next+slots;k++)
          //get the next icons
          {
            jessCode = "(assert (next_relation (name "+iconname+") ";
            Vector slot = iconsXML.getSlot(iconname, k);
            jessCode = jessCode+ "(next_icon "+slot.get(1)+") ";
            jessCode = jessCode+ "(value "+slot.get(2)+")))";
            r.executeCommand(jessCode);
          }

          //now add the previous relations from XML file
          for(int k=slots+next; k<slots+prev+next;k++)
          //get the previous icons
          {
            jessCode= "(assert (previous_relation (name "+iconname+") ";
            Vector slot = iconsXML.getSlot(iconname, k);
            jessCode = jessCode+ "(previous_icon "+slot.get(1)+") ";
            jessCode = jessCode+ "(value "+slot.get(2)+")))";
            r.executeCommand(jessCode);
          }
}
```

Figure 55 Dynamically adding the deftemplates and icon relations to the Rete instance

To read in the XML information about scenarios we follow a similar approach. Besides the Jess code that gets generated in this dynamic way, a lot of rules, templates, queries and functions cannot be described in the form of XML in an easy way. Rules such as the double icon filter are thus hard coded.

4.3.2 Zooming

Zooming on the map will make it easier for the users to place icons on the map. Since more detail is shown, it is easier to select the correct location for the icon. Of course we can't just zoom the icons together with the background, as this could cause icons to take up half of the map when zoomed in, or become almost invisibly tiny when zoomed out. That's why we don't zoom the icons, but instead keep their original 32x32 size and translate their position on the new background.

When the client is zooming there are three things happening.

1) The background is changing to represent the new area that needs to be shown
2) The already placed icons need to be translated to the right location
3) The newly placed icons need to be given global coordinates.

To zoom on the map (background) we offer 3 possibilities.

1) zoom in, while preserving the middle of the map
2) zoom out, while preserving the middle of the map
3) zooming in by dragging an area on the map.

In all cases we are using some prefixed zooming levels, each defined to show a fixed size region. They are defined in the 6 by 2 matrix in Figure. Zoom level 1 shows 500 by 400 pixels of the original map, which is all of it, while zoom level 6 only shows 50 by 40 pixels, stretched over the size of the map, which is 500 by 400.

Zoom level	X	Y
1	500	400
2	375	300
3	250	200
4	125	100
5	100	80
6	50	40

Figure 56 Zoom levels

4.3.2.1 Normal Zooming

Zooming while preserving the middle of the map is rather straight forward. When the zoom in button is pressed we look at the global coordinates of the middle of the map (250,200) and look at which zoom level should be shown. For the java function getSubImage to work we need to present the top left and size as arguments. The image that is returned will then be stretched over the entire 500 by 400 map. Figure 57 shows how the background gets zoomed on.

```
left = global middle x - (new x size)/2
top = global middle y - (new y size)/2

image = image.getSubimage(left,top,
zoom[zoomlevel-1][0],zoom[zoomlevel-1][1]);
zoomedImage = image.getScaledInstance(500,400,0);
```

Figure 57 Formulas for zooming while preserving the middle of the map

In case we go from 500x400 to 375x300 we get the situation in figure 58. We then use the formulas in Figure 57, with `getSubImage(62.5, 50, 375, 300).` This image is then scaled to 500x400. Note that for zooming out the same formulas hold.

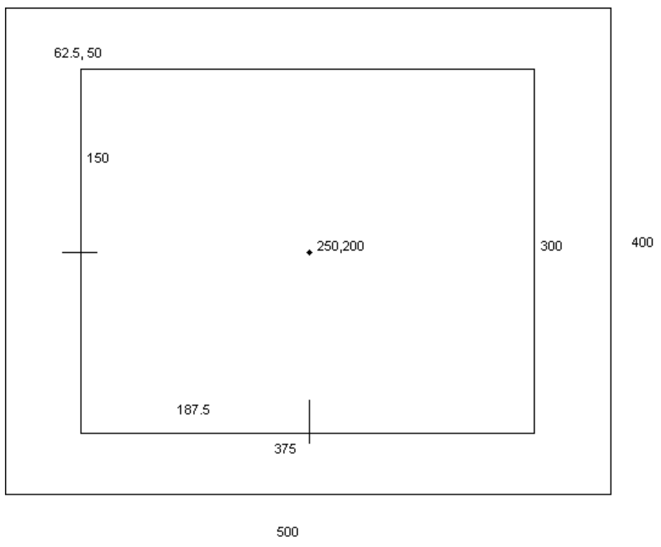


Figure 58 Zooming from level 1 to level 2

Translating global to local coordinates

When there are icons that are already placed on the map we need to translate their global coordinates (as they are stored on the map) to local coordinates (as they are shown on the selected area). The relevant variables are the centre of the map and the zoom factor. The zoom factor is the local size divided by the global size. When we hold on to the example in Figure 58 we see that the zoom factor in this case is 500/375 = 400/300 = 1 1/3. Since we have the sizes of the zooming levels fixed and proportional, the zoom factor for the x coordinates is the same as that for the y coordinates.

The formula for the new x and y coordinates for a placed icon can be defined as:

```
xLocal = (xGlobal - middleX)*zoomFactor + middleX;
yLocal = (yGlobal - middleY)*zoomFactor + middleY;
```

Figure 59 Formula to go from global to local coordinates

When we look ahead a bit, we see that in the drag zoom function the left and top coordinates will be given directly. Therefore it could be a good idea to rewrite the formula to:

```
xLocal = (xGlobal - left)*zoomFactor;
yLocal = (yGlobal - top)*zoomFactor;
```

Figure 60 Alternative formula to go from global to local coordinates

E.g. when an icon has global coordinates (300,300) the icon gets translated to

```
xLocal = (300-62.5)*1 1/3 = 316 2/3
yLocal = (300-50)*1 1/3   = 333 1/3
```

An icon with coordinates (20,20) would be translated to (–56.67, -40) which is outside the visible map. An icon with coordinates (62.5, 50) would be translated to (0,0) which is obviously correct since this would be exactly at the edge of the new visible map.

Translating local to global coordinates

When the map is zoomed in while icons get placed, we need to calculate their global coordinates, before we can store them on the map. We can use an inverse formula of the previous one:

```
xGlobal = xLocal * zoomFactor + left;
yGlobal = yLocal * zoomFactor + top;
```

Figure 61 Formula to go from local to global coordinates

When we reverse the example in the last section, we place an icon when we are zoomed in at 375x300. The coordinates of the icon then need to be translated to global coordinates. The zoom factor in this case is 375/500 = 300/400 = 0.75. When we place an icon at (300,300) this time and input them into the formula we get:

```
xGlobal = 300*0.75 + 62.5 = 287.5
yGlobal = 300*0.75 + 50   = 275
```

4.3.2.2 Drag Zooming

The other way to zoom the background is by dragging an area. The selected area will be adjusted to fit the proper zoom level. This is to prevent dragging only 1 pixel, which would result in the entire 500x400 dimension to become one colour. If only 1 pixel is dragged, the smallest resolution is shown, 50x40. When we drag, the top left coordinates and the bottom right coordinates are given by the user. The size of the dragged area gets mapped on the best fitting predefined resolution. The top left will remain unchanged, while the bottom right may be adjusted a bit to fit one of the zoom levels. The code remains the same as in Figure 57.

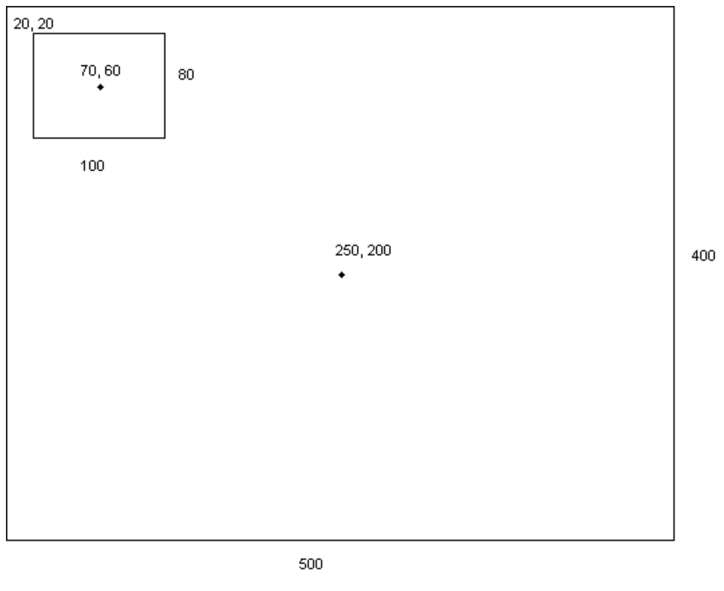


Figure 62 Zooming in, using drag zoom

Translating global to local coordinates

When we zoom in as shown in Figure 62 most of the placed icons will probably not be shown in the zoomed area. The ones that will be shown will be translated according to the formula in Figure 60, where:

```
zoomFactor = 500/100 = 400/80 = 5
```

Some examples of translations from global to local coordinates are given in Figure 63:

```
(20,20) will be translated to
xLocal = (20 - 20)*5 = 0
yLocal = (20 - 20)*5 = 0

(120,100) will be translated to
xLocal = (120 - 20)*5=500
yLocal = (100 - 20)*5=400

(70,60) will be translated to
xLocal = (70 - 20)*5 = 250
yLocal = (60 - 20)*5 = 200
```

Figure 63 Example translations from global to local coordinates

Translating local to global coordinates

When icons are placed when the map is zoomed in we need to translate the local coordinates to global ones before we can store them on the map. To do this we use an inverse formula. When we continue the example from Figure 62, we see that the zoomFactor is now 1/5. The inverse formula is the same as in Figure 61. In Figure 64 we show some examples of local to global translations.

```
(0,0) will be translated to
xGlobal = 0*1/5 + 20 = 20
yGlobal = 0*1/5 + 20 = 20

(250,200) will be translated to
xGlobal = 250*1/5 +20 = 70
yGlobal = 200*1/5 +20 = 60

(500,400) will be translated to
xGlobal = 500*1/5 +20 = 120
yGlobal = 400*1/5 +20 = 100
```

Figure 64 Example translation from local to global coordinates

Chapter 5: User Test

It is important to test the system, in order to detect errors and unexpected or unwanted behaviour, it can also give some conclusions of the ease of use and the complexity of the system. The goal of our test is to get a first impression of the users. The number of test persons and test material is very limited. We have designed a test that will allow us to test the *usability*, the *correctness*, and the *completeness* of the system at the same time.

Usability
One of the goals of our system was that it is easy to use and understand. Questions that arise are:

- How easy is it for the users to select an icon from the hierarchical order and to place them on the map?
- How do the users experience the zooming functionality?
- How easy is it to make corrections?
- How helpful are the systems suggestions for next icons and scenarios?
- Can we just introduce the system, or do the users need some training first?

Correctness
The system was designed to have intelligent behaviour by fusing data from different reports, updating the data in an automated way, and to give suggestions for icons and scenarios. Questions for this aspect are:

- Do the reports get fused in a correct way?
- Does the system come up with the right conclusions regarding the scenario?
- Are the suggestions for next icons correct?
- Is the intelligent behaviour of the system transparent for the users?

Completeness
The system has a limited range of different icons. They are extracted form the news articles in Appendix A. Questions are:

- Do the users understand all the icons in the system?
- Do the icons have enough attributes to meet the complexity of the situation?
- Is the set of icons sufficient to report about a crisis?

5.1 Design of the Test

There are two users, the respondents, that will test the system simultaneously, using a scenario presented to them in the form of photographs. The respondents are both not experienced in reporting events, let alone reporting about emergency situations. We have taken a new scenario for this test, as opposed to using one we used to determine the Jess rules. We have done this to make the test less trivial, after all it wouldn't be much of a challenge to predict which scenario is going on, if the test case is exactly what the rules are derived from.

Unfortunately, the system can't be tested in the real world, because 1) there is no fire or other disaster going on when we want to test it, and 2) the system is running on normal PCs at this time. That is why, instead of going outside the respondents are presented photographs of the scenario. A disadvantage of this is that it is very hard to determine your position if you can't look around freely to orientate on the situation. That's why the map of the environment explicitly tells the respondents where they are, and what direction they are looking. We have done this by putting letters of the corresponding photograph on the map, accompanied by a directional arrow. Because in a real situation the reporters will not both be reporting from the same location, they will be given different pictures.

During the test the respondents are asked to think aloud, telling what they are thinking and what they are trying to accomplish. This will allow us to determine if specific tasks need to be made more intuitive, or need more functionality. In terms of the overview in Figure 1 the photographs represent the Real World, the explanation of what the respondent sees and tries to accomplish is the Mental World Model, and the Structured World Model is what the respondent actually reports using the system.

5.2 Test Results

Respondent 1 was shown the pictures in Figure 65. She started with picture A and decided to report a fire truck and a fireman. She found the corresponding icons without any trouble and placed them on the map. She then decided that she could not place them at the correct location very well because she had forgotten to zoom in first. She deleted the icons, then zoomed in a few times to the desired resolution and placed the icons again. She decided that the woman on the picture was not important to report about because she didn't seem to be a wounded victim. She didn't notice any other events on the picture and moved on to picture B. She started with a fireman icon and selected with the attributes there are 3-5 fireman. After that she reported flames, small in size, with medium intensity. As a next step she decided to send the information to the server. The server made some suggestions for next icons, and the respondent decided that the smoke suggestion was actually an event she had overlooked. After placing it on the map she sent the report again, and was satisfied with her current report. The final results are shown in Figure 66.

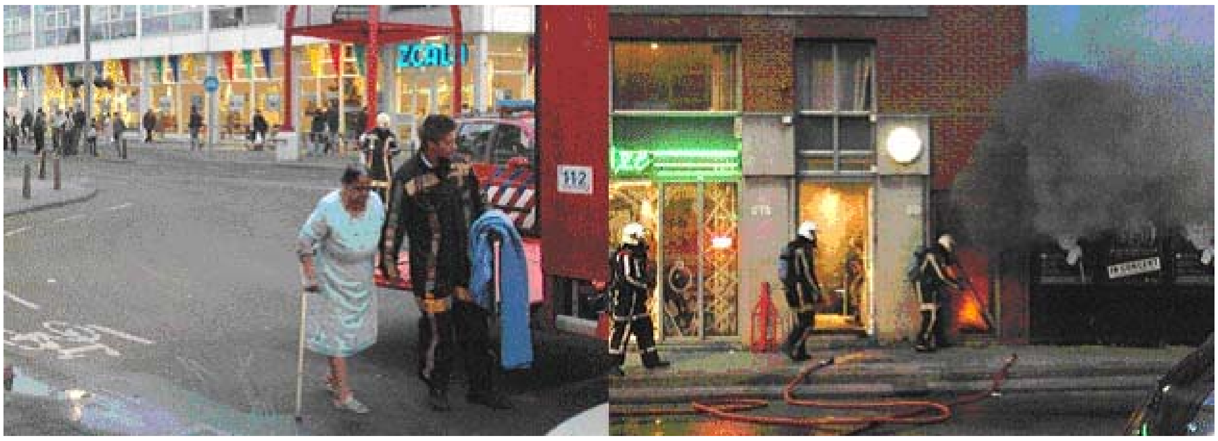


Figure 65 Photographs A and B presented to Respondent 1. Photographs by Jos van Leeuwen.

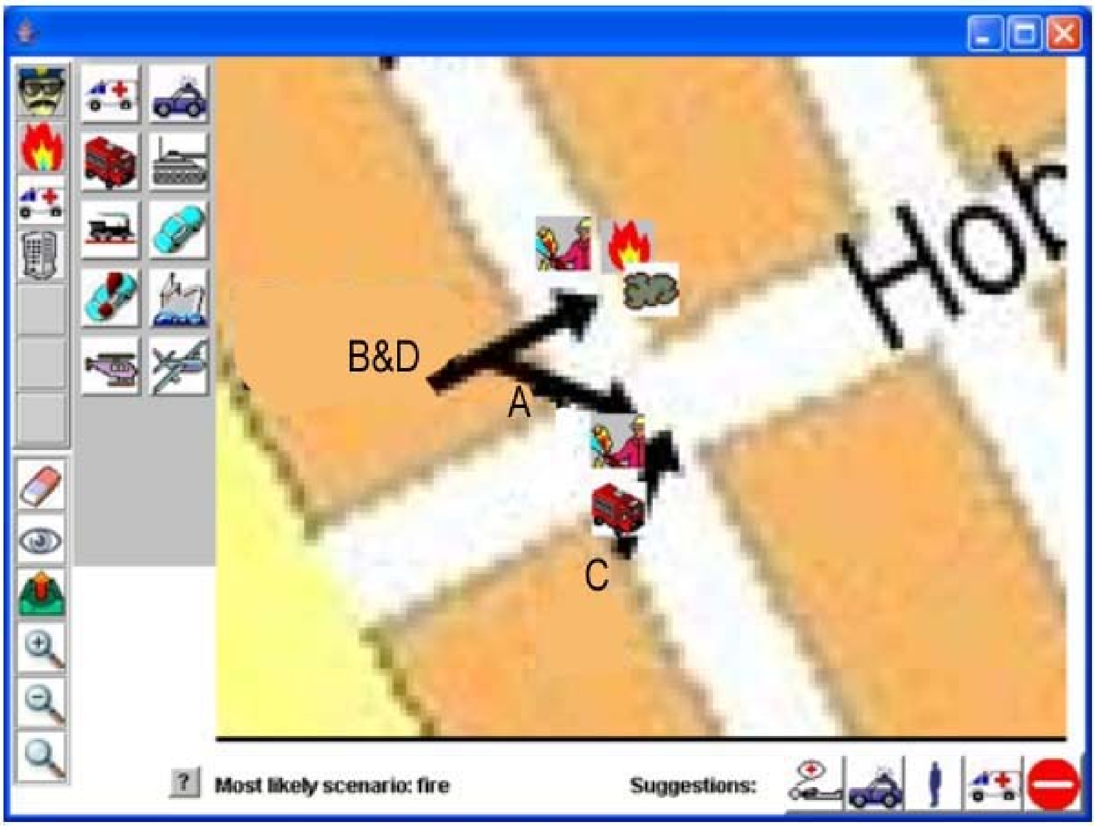


Figure 66 Report of respondent 1

Respondent 2 was presented photographs C and D, in Figure 67. He started with picture A and decided to zoom in on the location first. As opposed to respondent 1 he used the drag zoom tool, which allowed him to see the picture at the highest resolution at once. He then placed a fire truck and 3-5 fireman. He wasn't sure if the light in the window were flames or just plain illumination. He decided to not report fire there and wished he could have seen the building from a shorter distance. Respondent 2 continued with picture D and reported 3-5 firemen, a building, smoke and fire. The result of the report is shown in Figure 68.

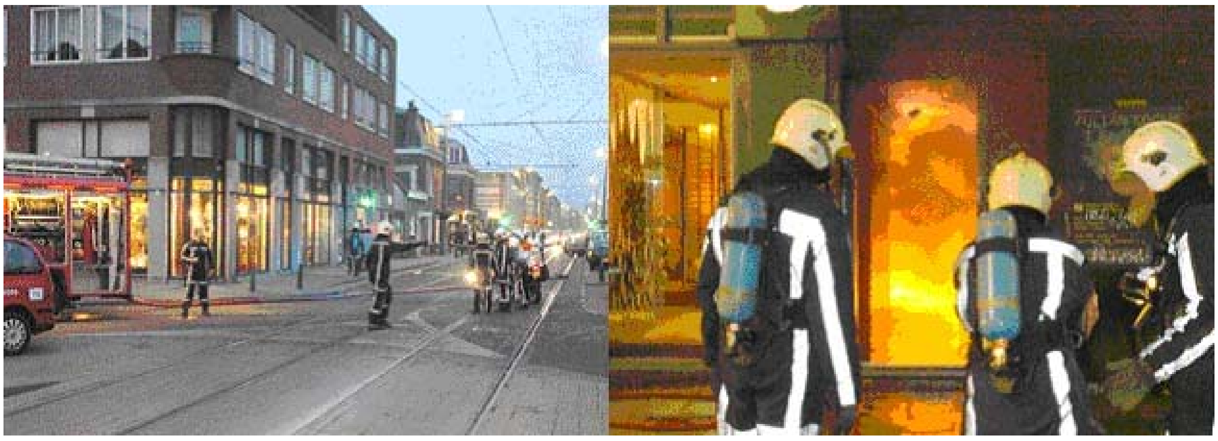


Figure 67 Photographs C and D presented to Respondent 2. Photographs by Jos van Leeuwen.

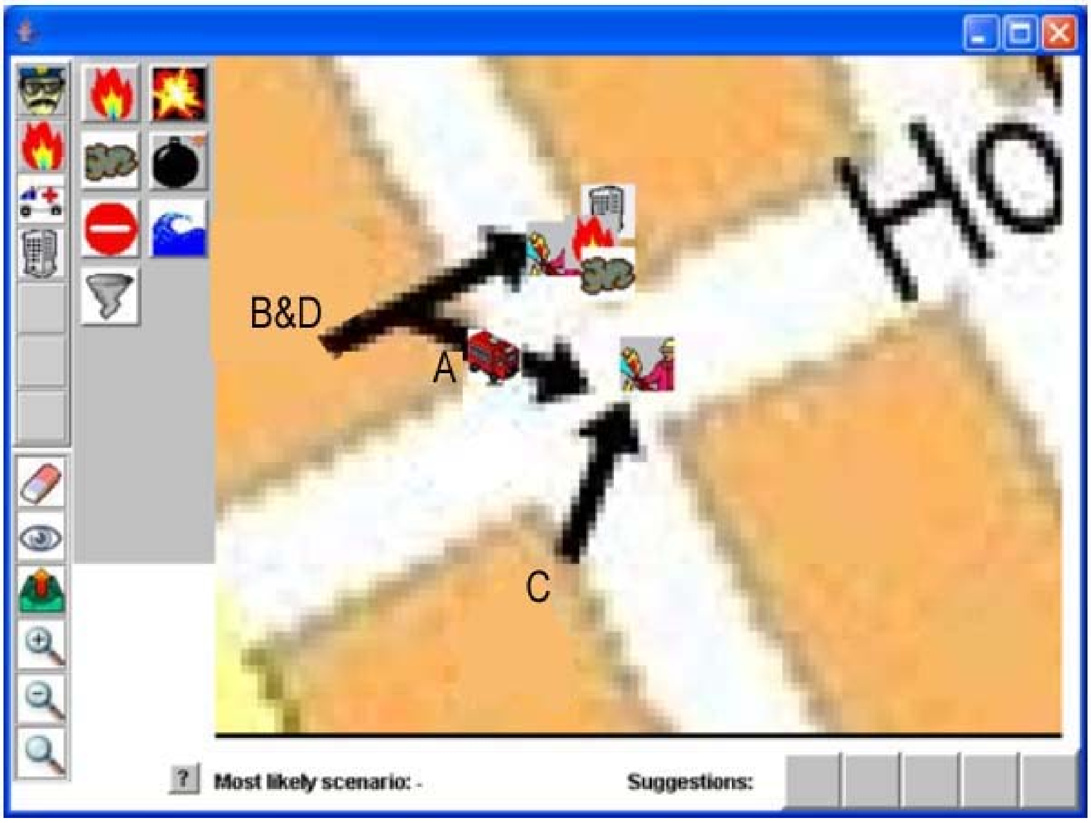


Figure 68 Report by respondent 2

Respondent 2 pressed the send button and the two world models were fused by the server. Both respondents got a new map from the server with the new world model, which can be seen in Figure 69.

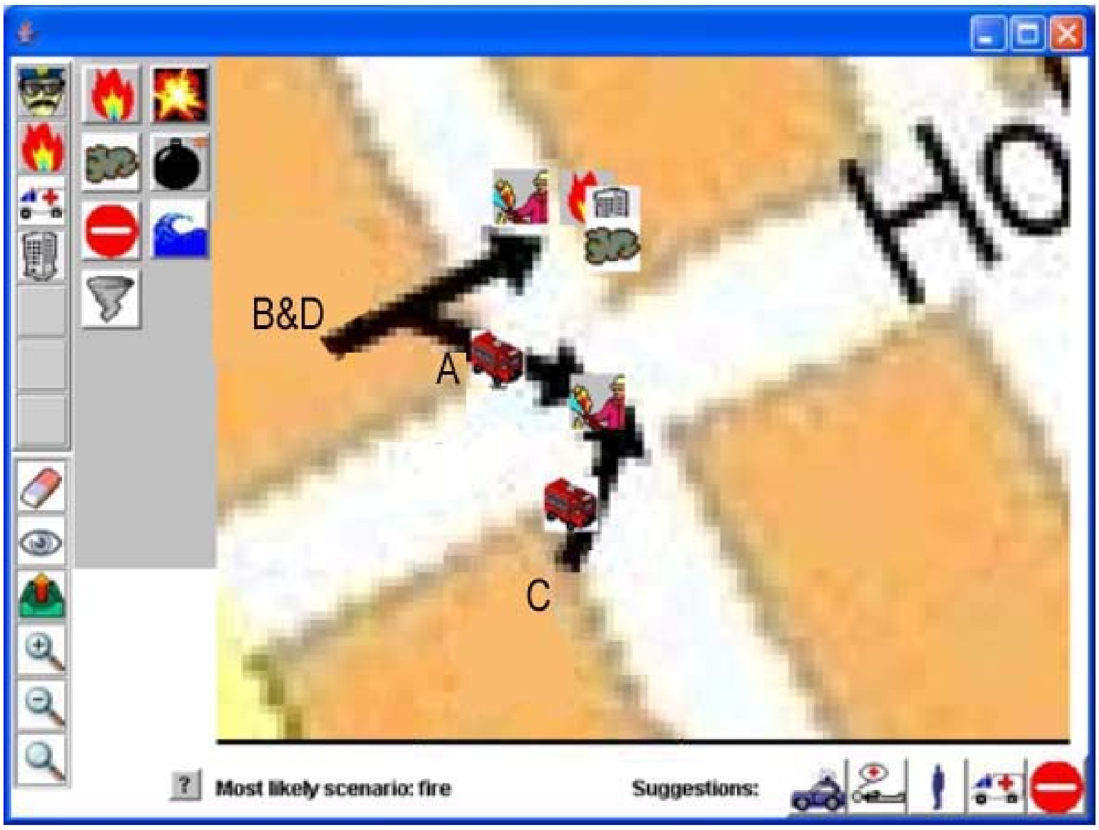


Figure 69 new world model

Both respondent 1 and 2 were not surprised by the new world model. They already expected that some icons would have been moved a bit to take a weighted average of the two reports. Respondent 1 noticed that she had forgotten to put a building icon on the map at first.

5.3 Test Conclusions

The conclusions that can be made from the test will be divided in the three aspects we tried to test. First of all the *usability* of the system. The overall opinions of both tester is that the system can be used quite easily.

- The needed icons were easily found in the hierarchical order and were placed on the map without trouble.
- The zooming in and out of the map was used by both testers and was very helpful.
- Respondent 1 made some corrections in her placed icons, by deleting them and then placing them in the right location. It might have been easy if the icons could be moved instead.
- The suggestions that were coming from the server where helpful. Respondent 1 saw she had forgotten to place smoke, and decided to agree on the systems suggestion to place it.
- There were some problems with what to report. This will be due to the fact that neither of the respondents are experienced in making reports. Respondent 1 decided that the person that was on photograph C was not relevant enough to the situation to be reported. From this can be concluded that even though we are structuring the mental world model of the reporter, there is still some room for an own opinion. To get all reporters to report what they should, some training should be given, and agreements on what to report should be made.

Secondly, the *correctness* of the system seemed to be ok.

- Of course the respondents didn't know exactly how the two reported world models should have been fused, but they did not notice any strange modifications of their own model. The fusion seemed to be correct.
- The suggestions that the system made were understandable, although most of the suggested events were not seen on the photographs. This is because most of what could be seen on the pictures was reported about already.
- The scenario suggestion, a fire, was obviously correct.
- Both respondents had read the user manual in Appendix C, and knew in advance that their reports were going to be fused by the server. They were not surprised when some icons were added on their screen and some were moved a bit. It was transparent enough.

Finally, the *completeness* of the system seemed to be sufficient.

- The users understood all the icons that they needed.
- The attributes of the icons were enough to meet our respondents need. Because they were both laymen they had no desire for more complexity in the attributes. Maybe when the system gets tested by real firemen additional attributes may be needed.
- Neither of the respondents wanted to report anything that was not represented by an icon, and the icons they needed were found very quickly. Respondent 2 was not sure if he saw a fire on picture C. It would be nice if he could report about this, by adding an uncertainty factor to his report.

Chapter 6: Conclusions and Recommendations

In this chapter the results of the project are discussed. After that we will evaluate to what extend the projects goals are reached. Finally we will discuss some possibilities for further research and development of the system.

6.1 Results

This section presents the results of the project. We will present the results made on the interface and the intelligence, and the dynamical aspects of both.

6.1.1 The Interface

The interface of the system was one of the first clear goals of the project. Before we could make a working system we needed some way to let the user feed input to it. Many prototypes were developed. The first prototypes were made in Macromedia Director [DIR], in which we could very quickly make a good looking interface. We did not add any functionality to this prototype, since the used tool was not really suited for it. These prototypes were basically some interactive screenshots, in which the icons could be selected by opening the icon groups.

After reading a lot of papers about the use and development of icons we made up a list of possible icons. Since we could not get our hands on things like a 'policeman handbook'. We had to reason about what concepts should be represented by icons ourselves, at the hand of some news articles presented in Appendix A. After making a list of these concepts we collected, developed and adjusted the icons that could be used.

The next prototypes were developed in Java, in a manner that allowed us to build up the system incrementally. After some basic functionality was added we could build on top of this, delete the changes if we didn't like them and replace them with improvements. This iterative process resulted in some different early stage versions. Like we showed in Chapter 2.1.2 we experimented with creating combinations of icons. We made the icons transparent, stack on top of each other and alternate each other. The results were not exactly what we were looking for and we searched for a new solution. We introduced zooming on the map. This, in combination with an attribute window to provide additional information, made combining items and representing them as combined icons unnecessary. After this prototype was designed and implemented, we began with the development of the server. The server didn't need a graphical interface.

When we arrived to the point in which we defined XML files to contain all the information about the icons, we needed a more dynamical interface. We reserved some space for additional icon groups and icons, and started working on ways to import the icons dynamically. When this was properly working we had a highly dynamic interface. By just editing some information in the XML files the interface is able to present more icon groups, more icons, other icons, etc. The advantage of this is that the Java code does not need to be revised, so non programmers can adjust the system as well, without having to recompile the code.

The dynamic version with zooming and an attribute window was nearly what we needed. The only thing missing was a way to give feedback to the user. We developed the bar with icon suggestions and the scenario information window for this. The result of this is the final version of the client. This final version was tested as discussed in Chapter 5.

6.1.2 The Intelligence

When we had a first working version of the client we needed to implement a server as well. The server needed to be able to connect with several clients at the same time, so we designed it to start up a new thread for each client, which could then be handled simultaneously. The first version of the server could receive input from clients and send it back. It did not care about correct world models at that time, it just sent back its most recently received world map, and disposed the older versions.

The next version of the server did not receive entire maps from the client, but only the placed icons. Now the server sent back all the icons it received, not exactly what we wanted, but clients could see what other clients had sent as well. Going on with this version, we added the intelligence to the server. From now on it would have a Jess component to handle the world model. This was the largest increase of the system, but we added it as one increment. It was so large because we needed to extract the rules and information about icons from the XML files and add this to the Jess component. Also all the icons that were send to the server needed to be added to the knowledge base. To keep an overview of this increment we designed the Jess code by itself at first. We used a command line interface to test the code, before embedding it in Java.

In designing the Jess code we first made it possible to add new icons to the knowledge base. When that was working we also designed code to delete and modify the icons. After that we developed a way to filter out double icons. This is the only intelligence that actually deletes icons by itself, without asking the users for feedback. We did think about deleting icons that were not supported by other users, but found that deleting them would be to dangerous. Since the system is designed for life threatening situations we don’t want the system to act completely autonomous. We decided that the intelligence for that part should remain at the human side.

Behaviour that could be added in a matter that relies both on human and machine intelligence is to let the server make suggestions for possible next icons. To just add the icons would be too dangerous, and the location of the to be placed icon is too hard to predict. The compromise we designed is to let the server calculate which icons it expects, and ask the human user to decide if they should really be placed, and where.

The server also makes some assumptions about what scenario is going on. Since this is just an assumption that does not have a big impact on the working of the system and its world model, we decided to keep this intelligence completely at the server side, without human intervention. The adding of these rules resulted in the final version of the server. A server that has a consistent world model at all times, and makes suggestions to the user.

6.2 Conclusions

In this section we will evaluate to what extend the project goals are achieved. We will do so by using the split up we made in the Design chapter:

Design and implement a demonstrator for a system that is suited for iconic communication in a crisis situation, using a map of the surroundings (1), which is expressive enough to handle complex and unexpected situations (2), yet intuitive enough to use without making (a lot of) errors (3). The system should be intelligent enough to assemble and maintain a correct and up to date world model (4). It should detect possible errors in the form of missing, double and wrongly placed icons (5). Furthermore the system should be dynamic in the sense that new concepts and rules can easily be added (6).

The conclusions for each requirement will be discussed below:

1) This part of the problem is solved. We made an interface which allows to use icons, and place them on a map.
2) The system is expressive in a sense that there are a lot of concepts that can be reported about. Furthermore the icons that are used to represent the concepts can be given attributes to add more information. The complexity of the system can be further increased by extending the XML files. Unexpected situations would be situations in which new icons are needed that were not implemented yet. This can be done by adjusting the XML files to add the concept and its relation to other concepts. There is a problem with this however, because the server and the clients have to be restarted in order for the changes to take effect. When this is done, the already reported icons will be lost.
3) It is easy to select the right icon because the icons are distributed over logical icon groups, which can be altered if needed by adjusting the XML files. To provide extra information, an attribute window will pop up where the values of the attributes can be given. The values can be selected out of a small list, this decreases the chance of making a wrong selection, and eliminates the chance to make an illegal selection. When icons are placed, the user is able to delete them again, or to inspect or alter its attributes. To prevent placing icons on the wrong location, the user can easily zoom in and out of the map, to be able to place the icon exactly where it should be. According to the respondents of the user test, it is an easy to use system.
4) In order to assemble and maintain a world model we collect all information at one server. The Jess component in the server is responsible for keeping the world model up-to-date. Because the server only has 1 world model that gets adjusted over time, the clients will all be send the same information, which is always the newest. The correctness of this model is dependent on the information the users send. When they report nonsense, the systems world model is worthless. During the user test, both users were sending correct information, and the server fused their world models in a correct new one.
5) To detect missing icons, the system looks at the already placed icons and the possible scenario. From this it gives suggestions to the user, rather than adding icons autonomously. The user can then decide if the suggested icon should be placed or not. Double and wrongly placed icons are detected by the systems double icon filter. When two or more of the same icons are placed very closely to each other, the system combines them, as long as they were reported by different clients. When multiple reports of the same icon are made, the system will take a weighted average of the icons location, and thus incorrectly placed icons will be placed on a better position. If

a client reports an icon that is too far from its correct location, and out of the filters range, it cannot be detected.

6) New concepts can easily be added or adjusted by altering the XML files. The relations between the icons can also be adjusted in this way. Adjusting these will result in the system to give other icon suggestions or scenario overviews. There are however still some hard coded functions that are not dynamical, such as the double icon filter.

Concluding we can say that all the goals, as stated in the problem description are met. However, there are still many things we would like to see done in a different or more elaborate way. We will discuss these in the next section.

6.3 Recommendations

The development of the ISME system has been a single student effort with a time span of approximately a year. A similar system, like the C2000 project was done by hundreds of people, costing about 700 million Euro, and its development is lasting many years already. From this it should be clear that our constraints on resources have made it impossible to develop and implement every aspect we wanted. In this chapter we will discuss some of the ideas we were unable to work out and implement.

Extending the system by non human observers

At this time we only get input from human observers. To extend the system we could add some non human observers as well. This could be done by sensors, which could for example report about smoke development. We could add smart cameras to the system, which can report about various things like unexpected crowds of people, smoke, or traffic jams. Systems like these could place their own icons on the map and send them. In an ideal case our system could get input from all sorts of security systems. If a fire alarm goes off in a building it could send a report to our system as well, we know the location of the building and that there is probably a fire. Information like this is exactly what we need for ISME. The same goes for burglar alarms, in banks or even houses.

Improving the network

With the current version of ISME we make use of a central server. This is a very vulnerable solution, especially in the case of terrorist attacks. If the location of the server is known, it is an easy target to eliminate the entire system. What would be better is to have a number of servers that are physically not located near each other. These servers would have to be kept up to date by synchronizing them with the other servers. At all times all the servers should have the same information.

Besides distributed servers other measurements may be taken. Just like the C2000 system we could make an option for the clients to communicate directly with each other. Especially in situations where no connection to a server is possible for all clients this is important (Figure 70). This was the case in the 9/11 terrorist attacks, for example. The entire communication infrastructure was down. In such a peer to peer network there has to be some sort of distributed blackboard where all the information is stored and reasoned about. This should be done by letting one of the clients simulate a central server as we use at the moment [Kla05]. Problems will arise if this clients connection is getting worse or even vanish. In this case another client has to take over the role of server. To prevent data from getting lost too often in such scenarios we need algorithms that store data at multiple places and know when another client has to take over the role of server. Factors in such algorithms would be the connectivity of each client and the strength of the connections. This proposed network has to be an ad-hoc network, meaning that the connections between the clients may change with time and clients may exit and enter the network at any time. Currently there is no platform that supports functionality like this, but JADE seems to provide some possibilities for it. There are still unsolved problems, however, when clients lose their connection, or when new clients want to enter the network.

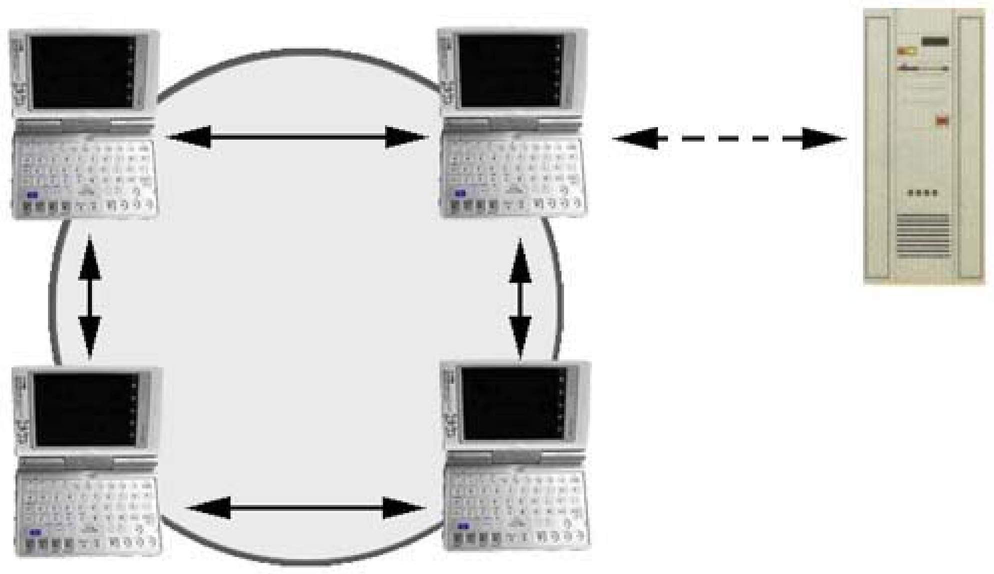


Figure 70 Proposed network architecture

Another improvement in the network is the use of GPS in it. Global Positioning System provides ways to exactly define your location. Since the clients have to define their positions at start up now, this could be replaced by a GPS that does it for them. On start up the system could place the client on the map and even show his viewing direction. There is some security issue involved however. GPS can be intercepted, and positions of the clients can be revealed in this way. Unless there is a good way to prevent this from happening we might need other ways for the clients to locate each other. Some research has been done already about localization without GPS. Algorithms that are able to determine locations are then often based on the time it takes for different agents to contact each other and the angle of arrival [Bul02], [Vel05].

Expanding and improving the intelligence

The current intelligence of the system filters out double icons and gives scenario information and suggestions for icons that could be placed next. Room for improvement lies in the information we are using. When there is a lot of information available about scenarios we could improve the information in the XML files to provide for more realistic calculations of the scenarios and icon suggestions [Cha05].

Besides improving the intelligence by using more reliable and accurate information we can also expand the intelligence. The following aspects could be investigated:

- suggestions for deleting icons
- intelligence over time
- giving the clients roles

Suggestions for deleting icons would occur if some icons are not supported by the rest of the icons. A simple example is when flames are reported, but smoke is not. The current system would suggest to place a smoke icon, but if this is ignored it could also suggest to delete the flames. The system should not delete it on its own for the same reason it should not place suggestions on its own. It is too dangerous to let a system decide such things in unexpected emergency situations. Feedback would be required from the users. An idea to implement this feedback would be to add the icon that should be deleted to the suggestion bar, but with a red cross through it. When it is clicked the icon in question should draw attention to itself, perhaps by blinking. After that a confirmation to really delete it should be asked.

Intelligence over time can be achieved by keeping track of a history. In this way there can be reasoned about icons that may have disappeared already. Think about an explosion for instance, this will be only there for a very short amount of time, but does have an impact on what can be expected next. We have already taken this into consideration in the development. We don't actually delete the facts from Jess, but mark them as deleted. The time of arrival of the icons can also be important in certain scenarios.

Things like cars and persons will move over time. Anticipation of moving object over time can be a next improvement. If we can get information about what direction the icons will be moving we can reason about their location better. If we have an ambulance on the map that is moving north east, we can expect an ambulance there soon. If that ambulance is reported by somebody we can suggest to remove the old icon.

An additional type of intelligence can be to give the clients different roles. We could let certain roles constrain to only a limited set of icons. For example someone who defuses bombs could provide more information about the status of a bomb then an ambulance driver. This could lead to a situation where experts can provide attributes of an icon, where others can not.

From causal relations to probabilistic relations

The current reasoning is based on causal relations. If there is smoke there is fire. In some cases causal relations are not easy to see through, that's when it could be convenient to use probabilistic relations. A well known example of a probabilistic model is the use of Bayesian belief networks, which have proven their use in a many applications. In Bayesian belief networks one can indicate the effect a certain event has on another event. They can be visualized as directed graphs. Given some events and the probabilities that they happened, it can be calculated what events are likely to happen next. In contrast of causal relations we can make use of indirect relations as well now.
When events are reported in this case, the reporter could provide a probability of the event himself, or the server could do this, based on the credibility of the reporter. A nurse could for example have a higher probability of reporting correctly about the status of a victim then other reporters. This is another example of giving reporters roles, as written above.

Security issues

When sending information over a network there is always a risk that it will be intercepted by eavesdroppers. This is in particular a problem with wireless communication. Hackers can quite easily figure out what kind of messages are send to the server, and can then either get an overview of what is going on, or worse, send messages to the server on their own. False reports and an incorrect world model will be the result.

To prevent messages from being intercepted and read by unauthorized people, we can use cryptography. Cryptography is the science that focuses on ciphering messages to prevent them from being read. [Ove00] A suggestion is to give every client and the server a public key and a private (secret) key. The messages can then be encoded using a asymmetric cipher algorithm. The client code their messages with the public key of the server, after which only the server can decipher the message, using its own private key. Vice versa, the server will have to encode the messages with the public key of the client it wants to send to.

We can add a digital signature to the message by encoding it with the clients secret key. The server will then decode the message using the public key of that particular client. Because a correct message could only have been made by using the clients secret key, the message is proofed to be coming from the client.

To combine both the prevention of unauthorized reading of messages and unauthorized senders, we can first encode the message with the public key of the server, and then code the result with the clients secret key. The server then has to use the clients public key first, and knows the message is really coming from the client. After that it decodes the message using its own secret key, to actually read the original message.

Bibliography

[Bea94]	*Iconic Communication* , C. Beardon, 1994. In: *Intelligent Tutoring Media*, 5(2), pp.58-62. ISSN 0957-9133
[Bru00]	*Object-oriented Software Engineering, conquering complex and changing systems*, B. Bruegge, A.H. Dutoit, 2000, Prentice-Hall, Inc.
[Bul02]	*Scalable, Ad Hoc Deployable RF-based Localization*, Nirupama Bulusu, Vladimir Bychkovskiy, Deborah Estrin and John Heidemann, October 2002, University of California at Los Angeles
[C2000]	The C2000 system, designed in order of the Dutch government, see www.c2000.nl
[Cha02]	*Semiotics the Basics*, Daniel Chandler, 2002. Or for a similar online version see http://www.aber.ac.uk/media/Documents/S4B/
[Cha05]	MSc Thesis of Jan Chau, still under construction at this time. Delft University of Technology
[CLIPS]	Expert System tool, see http://www.ghg.net/clips/CLIPS.html
[Comb]	Combined Systems group, see www.decis.nl
[DIR]	Macromedia Director, a multimedia tool designed to create interactive CDs, but also useable in fast prototyping activities. See www.macromedia.com
[Dor94]	*Self-Explaining Icons*, Claire Dormann, 1994, In: *Intelligent Tutoring Media*. Vol 5, No 2. 1994. pp. 81-85
[JADE]	Java Agent DEvelopment Framework, see http://jade.tilab.com/
[JESS]	Java Expert System Shell, see http://herzberg.ca.sandia.gov/jess/
[Jon96]	*DynamIcons as Dynamic Graphic Interfaces: Interpreting the Meaning of a Visual Representation,* D.H. Jonassen, R. Goldman-Segal, H. Maurer In: *Intelligent Tutoring Media* , vol. 6 (3/4) (1996), 149-158
[Kla05]	MSc Thesis of Paul Klapwijk, still under construction at this time. Delft University of Technology
[Mea91]	A Computer-based Iconic Language , S. Mealing & M. Yazdani, 1991 In: *Intelligent Tutoring Media*, 1(3):133-136, 1992
[Mea94]	*A Computer Hinterface,* S. Mealing, 1994, see http://www.intellectbooks.com/iconic/hint/hint.htm
[NRC03]	*Met 61 woorden de wereld rond,* about the theory of Anna Wierzbicka, NRC Handelsblad 20 september 2003.

[Ove00] *Informatiebeveiliging onder controle*, Paul Overbeek, Edo Roos Lindgreen, Marcel Spruit, 2000, Pearson Education

[Ric94] *The Use of Metaphors in Iconic Interface Design*, Stephen Richards, Philip Barker, Ashok Banerji, Charles Lamont and Karim Manji. In: *Intelligent Tutoring Media,* Vol 5, No 2, 73-80, 1994

[Shn98] *Designing the User Interface, Strategies for Effective Human-Computer Interaction,* Ben Shneiderman, 1998, Addison Wesley Professional

[Tat03] *Iconic Communication*, Iulia Tatomir, December 2003, Bachelor Thesis, Delft University of Technology

[Vel05] MSc Thesis of Marcel van Velden, still under construction at this time. Delft University of Technology

Appendix A: Scenarios

We have considered 5 different scenarios from which we extracted concepts that can be represented by icons. These scenarios are based on news articles. The XML file for scenarios is constructed by looking at the icons per scenario. The other XML file, for icons and their relations to each other, is also based on these articles. We extracted the next and previous relations between the icons from them. The next sections explain the scenario and which icons are extracted from them. Sometimes we extracted concepts that are not literary in the article, but could have occurred with a big chance nonetheless.

Scenario 1: Riot

A riot is happening when a lot of people are on the streets and start to make trouble. Usually there are a few people that start breaking things like windows and bus stops. Violent behaviour like this often leads to escalations as more and more people join in with the violence. In the scenario of a riot we can distinguish several concepts which can be represented by an icon. The following list of icons is extracted from the article below.

Table 1 Icons extracted from a Riot

Icon name	Icon	Description	Next icons	Previous icons
Armed Person	🔫	Rioter, armed with knifes, stones, Molotov cocktails.	Policeman Crashed Car Flames Victim Dead person	
Person	🧍	Spectator, not fighting but only watching.		
Policeman	👮	Trying to control the crowd of people.		Policecar
Policecar	🚓	To transport policemen to the scene.	Policeman	
Crashed Car	🚗	Set on fire by the rioters.	Flames	
Flames	🔥	Small fires, caused by Molotov cocktails.	Smoke Victim	
Smoke	☁	Caused by fire or by the police (tear-gas).	Flames Victim	Flames
Victim	🛌	Got hurt by either the rioters or the police.	Ambulance	
Dead Person	✝	Someone that died during the riot.	Ambulance	
Helicopter	🚁	Helicopter, used by the police.		
Road Block	⛔	Caused by the police or by rioters.		Victim Policeman
Ambulance	🚑	To take care of possible victims.		Victim

Riots in Den Bosch

by our Internet desk, 18 December 2000

The southern Dutch town of Den Bosch was plunged into serious rioting over the weekend following the death of a local football supporter early on Saturday. 29 people were arrested after two nights of violent disturbances, in which an estimated 300 rioters went on the rampage, fighting pitched battles with riot police. The authorities are bracing for more trouble this week.

Cars on fire, bus shelters and telephone booths vandalised, youths hurling bricks and Molotov cocktails at riot police leaving a trail of destruction. The otherwise sleepy town of Den Bosch turned into a battleground this past weekend following the death of a 31-year-old supporter of the local football team. He was shot dead by a police officer who intervened during a row between neighbours. The policeman reportedly acted in self-defence when the man threatened him with a large knife.

"Liquidation"
Later on Saturday, around 300 youths and supporters of FC Den Bosch assembled in a local bar, responding to calls on the Internet. They prepared for a silent march to the city centre to demonstrate their anger at what they called "the liquidation" of their fellow fan, a prominent member of the so-called hard core of FC Den Bosch supporters. But the silent march soon turned violent. Shop windows were smashed, police were pelted with stones, there were arson attacks on a bar, a school and a local blood bank, journalists were intimidated and a satellite television van was overturned.

Apparently, the supporters' fury was triggered by the Mayor's decision to cancel Saturday evening's fixture between FC Den Bosch and VVV Venlo, which they had wanted to use to mark their colleague's death. The Mayor later said he had called off the match amid clear indications that it would lead to trouble.

More Disturbances
But trouble came, even without the match. An army of riot police was deployed, assisted by a helicopter. They used tear gas and carried out charges to disperse the crowd, but failed to quell the rioting. Police then sealed off the district to prevent the violence from spreading to the city centre. After several hours, a tense calm returned. It lasted until Sunday evening, when the trouble started again. The authorities and local residents say they expect more disturbances during the week.

Source: http://www.rnw.nl/hotspots/html/denbosch001218.html

Figure 71 News article Riot

Scenario 2: Car Crash

A car crash can happen in many ways, usually it involves driving at high speed or under influence. Sometimes it involves just one car, in other cases it can involve many cars. In the scenario of a car crash we can distinguish several concepts which can be represented by an icon. The following list of icons is extracted from the article below.

Table 2 Icons extracted from a Car Crash

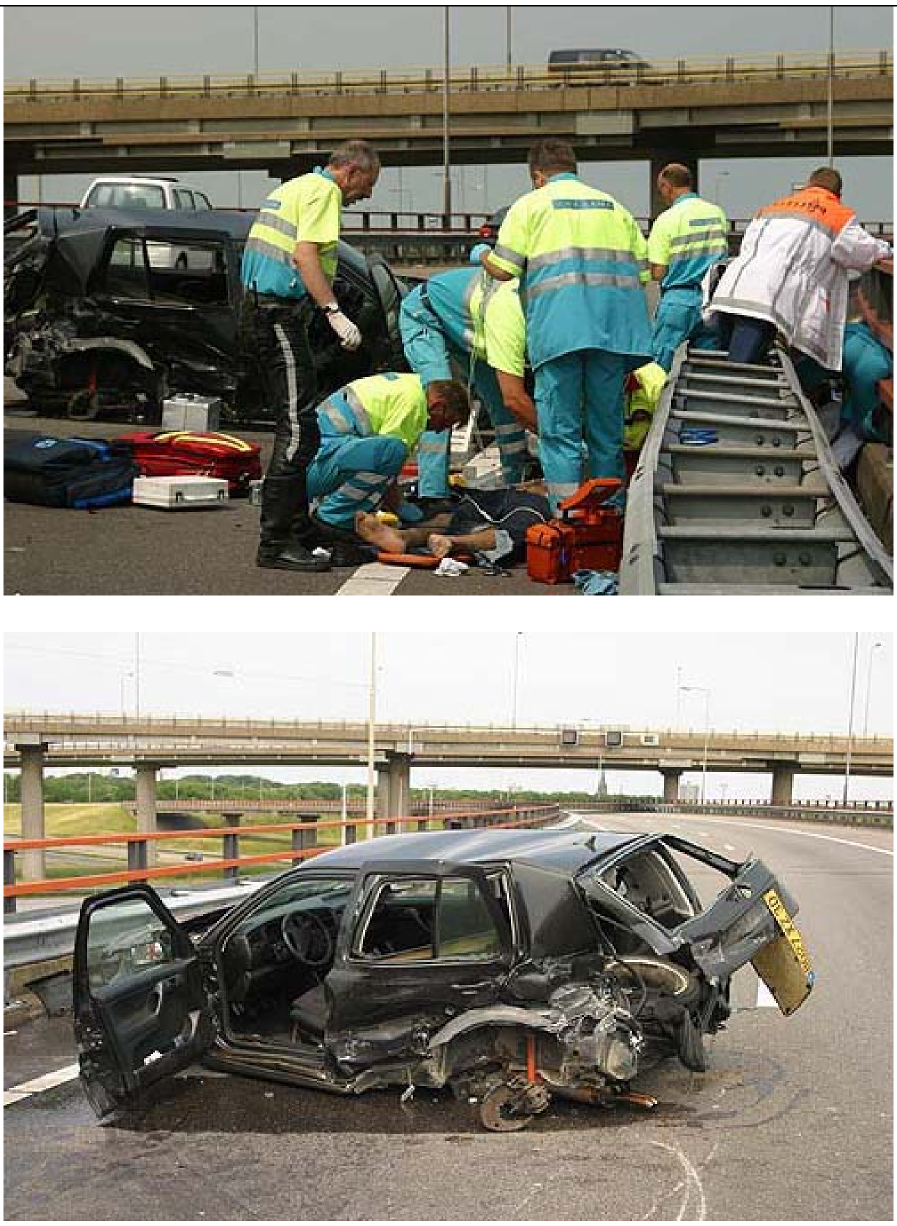
Icon name	Icon	Description	Next icons	Previous icons
Policeman	👮	Trying to control the traffic.		Policecar
Roadblock	⛔	Caused by the police to prevent traffic from coming near.	Car	Policeman Victim
Crashed Car	🚗	The car that has been crashed.	Fireman Policeman	
Car	🚗	Cars that are not involved in the accident, but are in the traffic jam caused by it.		
Victim	✚	Injured person that was involved in the accident.	Ambulance Nurse	
Helicopter	🚁	Used to transport the victims.		Victim
Ambulance	🚑	Used to transport the victims.	Nurse	Victim
Nurse	👩‍⚕️	To give first aid on the scene.		Ambulance Victim
Fire truck	🚒	To transport firemen to the scene.	Fireman	
Fireman	🧑‍🚒	Sometimes firemen are needed to free the victims from the car.		Firetruck
Police car	🚓	To transport policemen to the scene.	Policeman	
Person	🧍	Spectators that came to look at the situation.		
Dead person	✝	The victims involved in the accident could have died.	Ambulance Nurse	

Persbericht Haaglanden District8.net
Bericht nummer : 14
Plaats : Fly-over, A12
Datum : 14 juni 2003, 12.00uur
Revisie : 14-06-2003/23:55uur

Zwaargewonden bij auto-ongeluk Fly-over A12

DEN HAAG - Bij een ongeval op de fly-over richting de Utrechtsebaan zijn twee mannen zwaargewond geraakt. Vermoedelijk door veel te hard rijden verloren ze tijdens een inhaalmanoeuvre de macht over het stuur en klapte daarbij op de vangrail.
Door de impact van de klap werden beide slachtoffers uit het voertuig geslingerd. Een van de slachtoffers kwam hierbij tussen de vangrail terecht. Doordat er sprake was van zeer ernstig letsel kwam ook een MMT met de traumahelikopter ter plaatse om assistentie te verlenen. Beide slachtoffers zijn per ambulance onder poltie begeleiding met een spoed transport overgebracht naar ziekenhuizen in de regio. De technische recherche stelt een nader onderzoek in naar de precieze toedracht.De fly-over richting Den Haag is geruime tijd in zijn geheel afgesloten geweest voor het verkeer.

Edited by: Richard Hijdra Internetdiensten
Tekst en foto's: © Remco Suiker



Source: www.district8.net

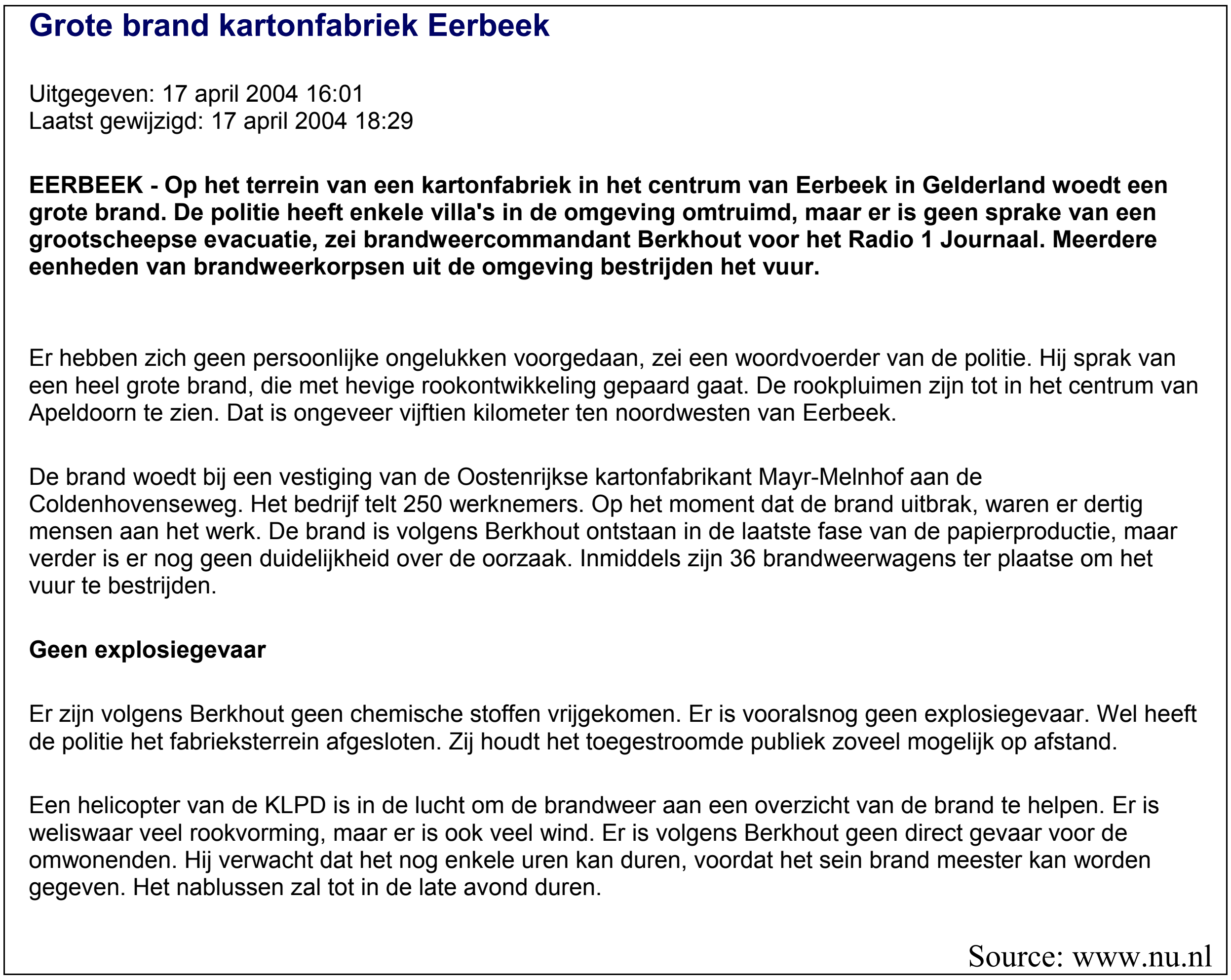
Figure 72 News article Car Crash

Scenario 3: Fire

A fire can occur in many places and in many varieties, but usually the fire is inside a building. In the scenario of a fire we can distinguish several concepts which can be represented by an icon. The following list of icons is extracted from the articles below.

Table 3 Icons extracted from a Fire

Icon name	Icon	Description	Next icons	Previous icons
Flames	🔥	The actual fire.	Smoke Firetruck Fireman Victim Person Explosion Policecar Policeman	Smoke Explosion
Smoke	💨	Smoke caused by the flames	Flames Firetruck Fireman Policecar Policeman Person Victim	Flames
Fire truck	🚒	To transport firemen to the scene and to provide equipment to extinguish the fire.	Fireman Person	
Fireman	🧑‍🚒	To extinguish the fire.		Firetruck
Person	🧍	Spectator, looking at the situation.		
Policeman	👮	To keep spectators and traffic at a distance.		Policecar
Police car	🚓	To transport policemen to the scene.	Policeman	
Roadblock	⛔	Caused by the police or firemen to keep the spectators at a distance		Policeman
Helicopter	🚁	Used to get an overview of the situation.		
Explosion	💥	In case of gas leaks.	Flames Policecar Person Victim	Flames
Victim	🛌	Person that got injured because of the fire.	Ambulance Nurse	
Dead Person	✝	Person that died because of the fire.	Ambulance Nurse	
Ambulance	🚑	Used to transport medical personnel to the scene.	Nurse	Victim
Nurse	🧑‍⚕️	To give first aid on the scene.		Victim Ambulance



Grote brand kartonfabriek Eerbeek

Uitgegeven: 17 april 2004 16:01
Laatst gewijzigd: 17 april 2004 18:29

EERBEEK - Op het terrein van een kartonfabriek in het centrum van Eerbeek in Gelderland woedt een grote brand. De politie heeft enkele villa's in de omgeving omtruimd, maar er is geen sprake van een grootscheepse evacuatie, zei brandweercommandant Berkhout voor het Radio 1 Journaal. Meerdere eenheden van brandweerkorpsen uit de omgeving bestrijden het vuur.

Er hebben zich geen persoonlijke ongelukken voorgedaan, zei een woordvoerder van de politie. Hij sprak van een heel grote brand, die met hevige rookontwikkeling gepaard gaat. De rookpluimen zijn tot in het centrum van Apeldoorn te zien. Dat is ongeveer vijftien kilometer ten noordwesten van Eerbeek.

De brand woedt bij een vestiging van de Oostenrijkse kartonfabrikant Mayr-Melnhof aan de Coldenhovenseweg. Het bedrijf telt 250 werknemers. Op het moment dat de brand uitbrak, waren er dertig mensen aan het werk. De brand is volgens Berkhout ontstaan in de laatste fase van de papierproductie, maar verder is er nog geen duidelijkheid over de oorzaak. Inmiddels zijn 36 brandweerwagens ter plaatse om het vuur te bestrijden.

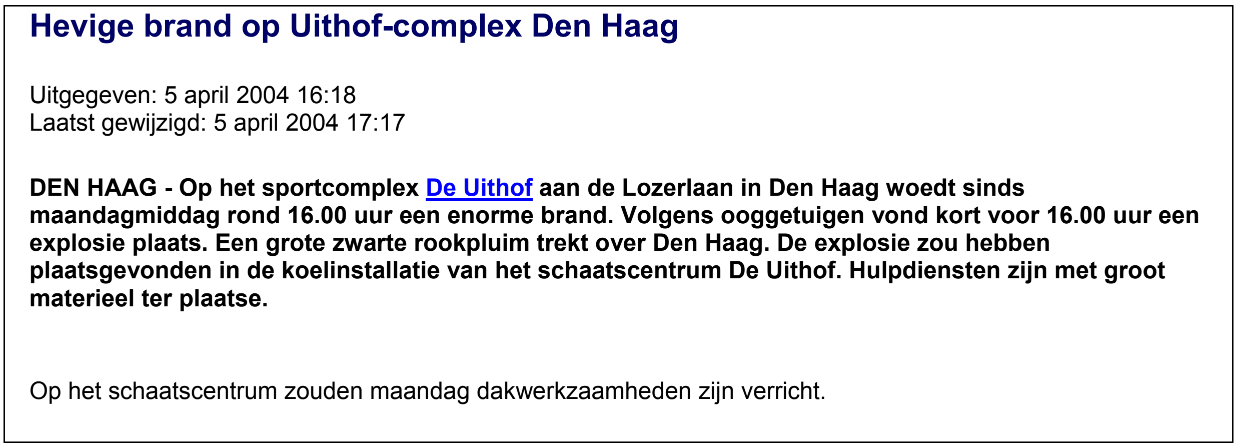
Geen explosiegevaar

Er zijn volgens Berkhout geen chemische stoffen vrijgekomen. Er is vooralsnog geen explosiegevaar. Wel heeft de politie het fabrieksterrein afgesloten. Zij houdt het toegestroomde publiek zoveel mogelijk op afstand.

Een helicopter van de KLPD is in de lucht om de brandweer aan een overzicht van de brand te helpen. Er is weliswaar veel rookvorming, maar er is ook veel wind. Er is volgens Berkhout geen direct gevaar voor de omwonenden. Hij verwacht dat het nog enkele uren kan duren, voordat het sein brand meester kan worden gegeven. Het nablussen zal tot in de late avond duren.

Source: www.nu.nl

Figure 73 News article Fire 1



Hevige brand op Uithof-complex Den Haag

Uitgegeven: 5 april 2004 16:18
Laatst gewijzigd: 5 april 2004 17:17

DEN HAAG - Op het sportcomplex De Uithof aan de Lozerlaan in Den Haag woedt sinds maandagmiddag rond 16.00 uur een enorme brand. Volgens ooggetuigen vond kort voor 16.00 uur een explosie plaats. Een grote zwarte rookpluim trekt over Den Haag. De explosie zou hebben plaatsgevonden in de koelinstallatie van het schaatscentrum De Uithof. Hulpdiensten zijn met groot materieel ter plaatse.

Op het schaatscentrum zouden maandag dakwerkzaamheden zijn verricht.

Foto: Sander Hak | Enorme rookwolk boven Den Haag

Rond 16.30 uur vonden meerdere explosies plaats. Het is vooralsnog onbekend of er mensen gewond zijn geraakt.

Foto: M. Agsteribbe |

Het sportcomplex en de directe omgeving zijn door de politie hermetisch afgesloten. Flatbewoners aan de overzijde van De Uithof zijn geëvacueerd. Het is nog onduidelijk of er schadelijke stoffen zijn vrijgekomen. De brandweer is bezig met een meetonderzoek.

Foto: Ferry Monsma | Brand gezien vanaf de Beresteinlaan

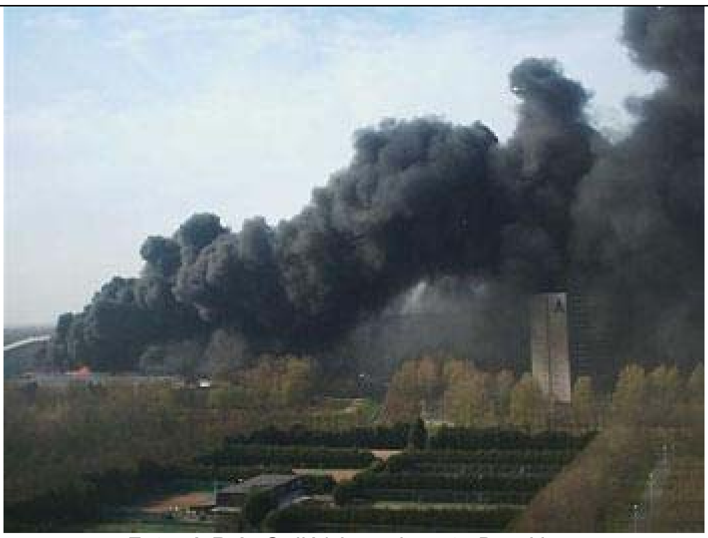
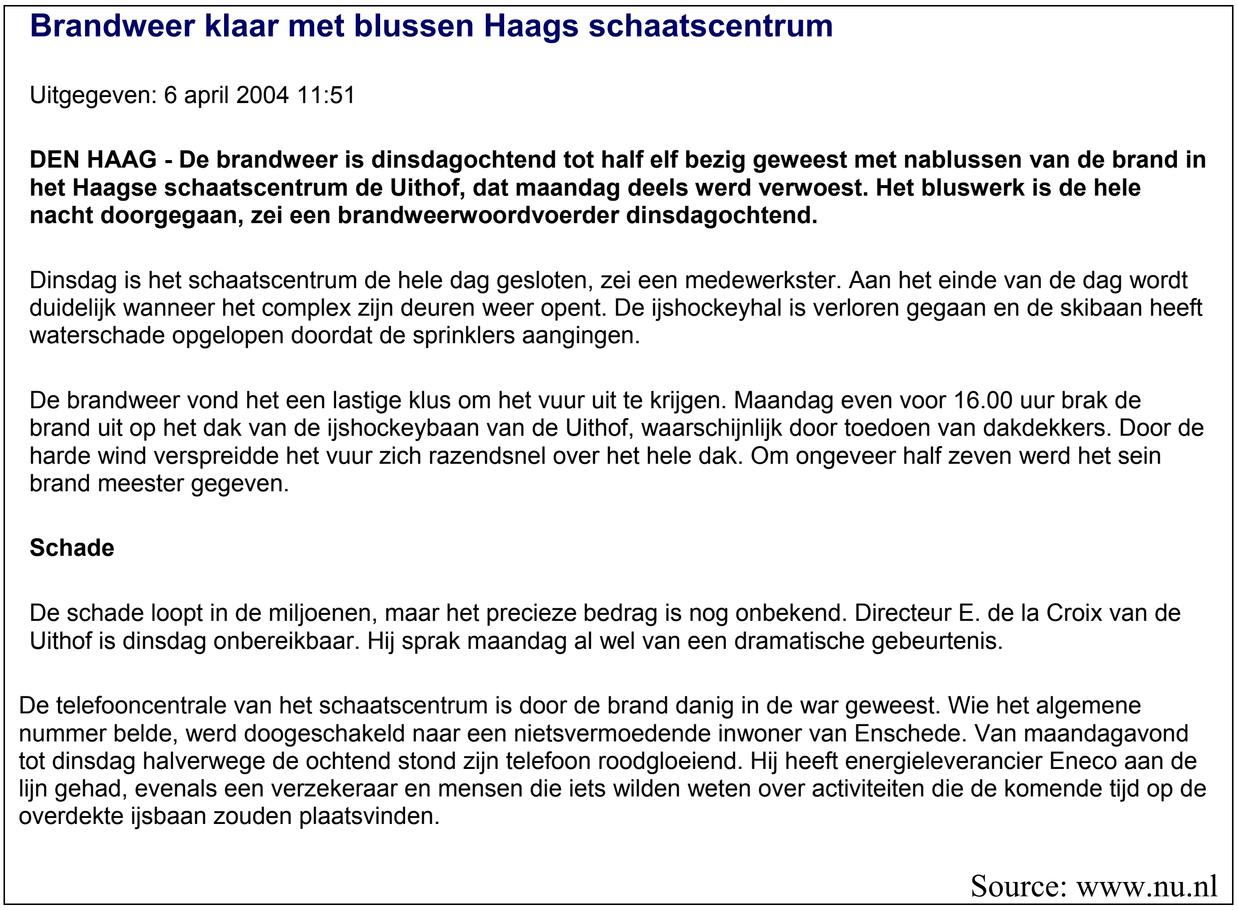


Foto: A.P.A. Galjé | Lozerlaan te Den Haag

Source: www.nu.nl

Figure 74 News article Fire 2



Brandweer klaar met blussen Haags schaatscentrum

Uitgegeven: 6 april 2004 11:51

DEN HAAG - De brandweer is dinsdagochtend tot half elf bezig geweest met nablussen van de brand in het Haagse schaatscentrum de Uithof, dat maandag deels werd verwoest. Het bluswerk is de hele nacht doorgegaan, zei een brandweerwoordvoerder dinsdagochtend.

Dinsdag is het schaatscentrum de hele dag gesloten, zei een medewerkster. Aan het einde van de dag wordt duidelijk wanneer het complex zijn deuren weer opent. De ijshockeyhal is verloren gegaan en de skibaan heeft waterschade opgelopen doordat de sprinklers aangingen.

De brandweer vond het een lastige klus om het vuur uit te krijgen. Maandag even voor 16.00 uur brak de brand uit op het dak van de ijshockeybaan van de Uithof, waarschijnlijk door toedoen van dakdekkers. Door de harde wind verspreidde het vuur zich razendsnel over het hele dak. Om ongeveer half zeven werd het sein brand meester gegeven.

Schade

De schade loopt in de miljoenen, maar het precieze bedrag is nog onbekend. Directeur E. de la Croix van de Uithof is dinsdag onbereikbaar. Hij sprak maandag al wel van een dramatische gebeurtenis.

De telefooncentrale van het schaatscentrum is door de brand danig in de war geweest. Wie het algemene nummer belde, werd doogeschakeld naar een nietsvermoedende inwoner van Enschede. Van maandagavond tot dinsdag halverwege de ochtend stond zijn telefoon roodgloeiend. Hij heeft energieleverancier Eneco aan de lijn gehad, evenals een verzekeraar en mensen die iets wilden weten over activiteiten die de komende tijd op de overdekte ijsbaan zouden plaatsvinden.

Source: www.nu.nl

Figure 75 News article Fire 3

Scenario 4: Bomb Scare

A bomb scare usually happens when someone makes a (anonymous) phone call to warn the police. In other cases it occurs when a suspected package is reported. In the scenario of a bomb scare we can distinguish several concepts which can be represented by an icon. The following list of icons is extracted from the articles below.

Table 4 Icons extracted from a bomb square

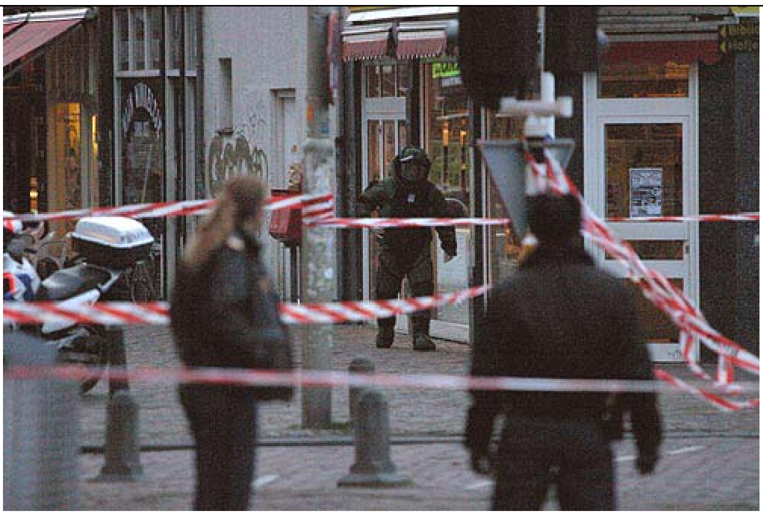
Icon name	Icon	Description	Next icons	Previous icons
Bomb	💣	A bomb or suspected package.	Explosion Policecar Firetruck Person Bombsquad Roadblock	
Building	🏢	A building where the bomb is placed in.		
Bomb squad	💣	A person who defuses bombs.	Explosion	Bomb
Ambulance	🚑	To be prepared for a victims.		
Fire truck	🚒	To be prepared for fire.		
Fireman	🧑‍🚒	To be prepared for fire.		
Police car	🚓	To transport policemen to the scene.	Policeman	
Policeman	👮	To keep spectators at a distance.		Policecar
Roadblock	⛔	To keep spectators at a distance.		Policeman
Explosion	💥	When the bomb explodes.		Bomb
Car	🚗	Bombs are often hidden inside cars.		

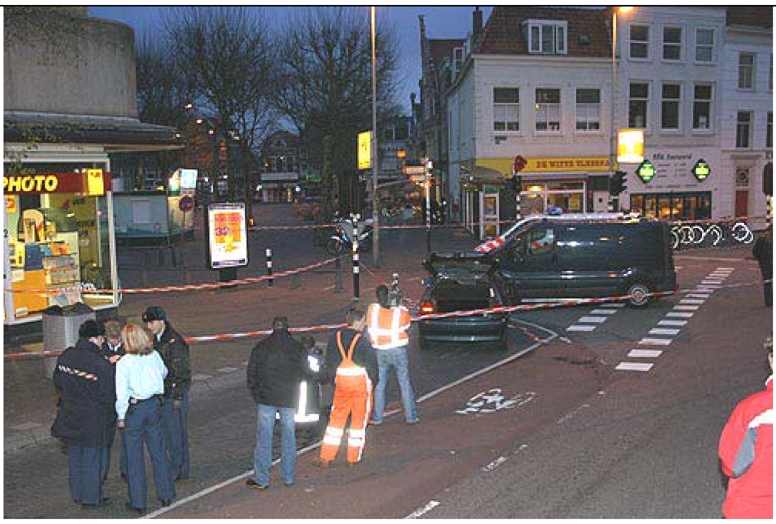
Bommelding

Dinsdagavond even na 19.00 uur wordt de omgeving van de Botermarkt in Haarlem afgezet i.v.m. een verdacht pakketje. Het doosje ligt tegen een woning aan en is met tape dichtgeplakt. De omgeving wordt ruimschoots afgezet voor het publiek en omwonende moesten hun woningen verlaten. Ook de restaurants in de omgeving van de Botermarkt werden ontruimt.
De TS746, OVD-midden en de HOVD werden ter plaatse verzocht evenals een ambulance. Wanneer de EOC omstreeks 20.00 uur ter plaatse is wordt het pakketje doorgelicht en blijkt er alleen rommel in te zitten. De omgeving werd weer rond 20.45 uur vrijgegeven.

746
BRANDWEER

WITTE VLEESHAL
Bibliotheek
Hofje van
Route
Stads

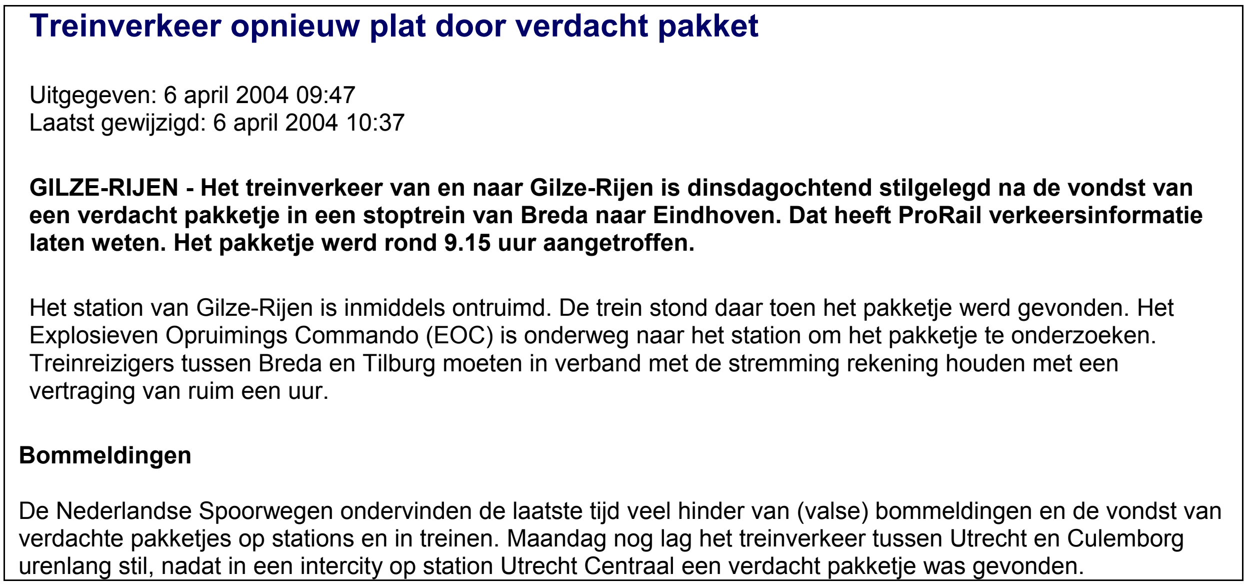




Fotos: Arno de Kock

Source: http://www.brandweerhaarlem.nl

Figure 76 News article Bomb Scare 1



Treinverkeer opnieuw plat door verdacht pakket

Uitgegeven: 6 april 2004 09:47
Laatst gewijzigd: 6 april 2004 10:37

GILZE-RIJEN - Het treinverkeer van en naar Gilze-Rijen is dinsdagochtend stilgelegd na de vondst van een verdacht pakketje in een stoptrein van Breda naar Eindhoven. Dat heeft ProRail verkeersinformatie laten weten. Het pakketje werd rond 9.15 uur aangetroffen.

Het station van Gilze-Rijen is inmiddels ontruimd. De trein stond daar toen het pakketje werd gevonden. Het Explosieven Opruimings Commando (EOC) is onderweg naar het station om het pakketje te onderzoeken. Treinreizigers tussen Breda en Tilburg moeten in verband met de stremming rekening houden met een vertraging van ruim een uur.

Bommeldingen

De Nederlandse Spoorwegen ondervinden de laatste tijd veel hinder van (valse) bommeldingen en de vondst van verdachte pakketjes op stations en in treinen. Maandag nog lag het treinverkeer tussen Utrecht en Culemborg urenlang stil, nadat in een intercity op station Utrecht Centraal een verdacht pakketje was gevonden.

Figure 77 News article Bomb Scare 2

Scenario 5: Shooting

A shooting occurs when one or more person start using or threatening to use their guns. In the scenario of shooting we can distinguish several concepts which can be represented by an icon. The following list of icons is extracted from the articles below.

Table 5 Icons extracted from a shooting

Icon name	Icon	Description	Next icons	Previous icons
Dead person	✝	A person that died because of the shooting.	Ambulance Nurse	
Armed person	🔫	A person that has a gun and is shooting with it.	Victim Dead person Policeman	
Victim	➕	A person who got injured because of the shooting.	Ambulance Nurse	
Police car	🚓	Used to transport policemen	Policeman	
Policeman	👮	To render the shooter harmless or keep the spectators at a distance.		Policecar
Person	🧍	Spectator that is looking at the scene.		
Ambulance	🚑	To transport victims from the scene.	Nurse	Victim
Nurse	🧑‍⚕️	To give first aid at the scene.		Ambulance
Roadblock	⛔	To prevent spectators from coming close.		Policeman Victim Dead person

Doden bij schietpartij in Tiel

Uitgegeven: 15 april 2004 20:25
Laatst gewijzigd: 16 april 2004 07:27

TIEL - Bij een schietpartij in Tiel zijn volgens een woordvoerster van de politie donderdagavond rond half acht drie mensen om het leven gekomen. Een vierde persoon ligt zwaargewond in kritieke toestand in een ziekenhuis, meldde de politie.

Volgens getuigen heeft de schietpartij plaatsgehad in een Aldi-supermarkt aan de Kwelkade in de Gelderse plaats. Of er sprake is geweest van een overval is onduidelijk. De zegsvrouw van de politie wil daar nog niets van zeggen. De advocate van de Aldi-directie laat weten dat de schietpartij niet in de Aldi-winkel heeft plaatsgevonden, maar in het winkelcentrum, waar de Aldi is gevestigd.

Ooggetuigen

Ooggetuigen zeggen dat er een mannelijk slachtoffer in de hal bij de supermarkt lag. Een eindje verder zou een tweede neergeschoten man hebben gelegen. De andere slachtoffers zouden in winkels zijn getroffen. Of de slachtoffers mannen of vrouwen zijn, is nog niet bekend. Een ooggetuige zei dat ze vier schoten had gehoord.

Een buurvrouw van de Aldi vertelde donderdagavond dat zij een politiewagen met hoge snelheid zag naderen. Uit de wagen sprongen agenten in kogelvrije vesten, die de winkel binnenrenden. De buurt heeft niets gehoord van schoten of ander lawaai. Ongeveer een uur na het incident zagen omwonenden personeelsleden van de Aldi naar buiten komen. De politie wil niet zeggen of er mogelijk een personeelslid onder de slachtoffers is.

Plastic

De schietpartij vond plaats in een klein winkelcentrum aan de rand van een Tielse woonwijk. Er kwamen zeer veel belangstellenden op af, maar de politie maakte voor zover mogelijk elk zicht onmogelijk door plastic te spannen. Rond half elf verdwenen de meeste belangstellenden. Het zwaargewonde slachtoffer was toen al via een achteruitgang van het winkelcentrum afgevoerd.

In en rond het winkelcentrum is de politie bezig met sporenonderzoek, dat nog geruime tijd zal gaan duren. De politie is uiterst terughoudend met mededelingen, omdat het om een zeer ernstige zaak gaat, aldus de woordvoerster. Of het winkelcentrum vrijdagmorgen weer open gaat, was donderdagavond nog niet bekend.

www.nu.nl

Figure 78 News article Shooting

Appendix B: XML files

The XML files as used in the latest version are listed here.

iconlist.xml

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
  <!DOCTYPE icon (View Source for full doctype...)>
- <iconlist>
  - <group>
    - <icon>
        <icon_name>policeman</icon_name>
        <icon_image>./icons/people/policeman.jpg</icon_image>
      - <slot>
          <slot_name>number</slot_name>
          <slot_value>1</slot_value>
          <slot_value>2</slot_value>
          <slot_value>3-5</slot_value>
          <slot_value>5-10</slot_value>
          <slot_value>10+</slot_value>
        </slot>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>busy</slot_value>
          <slot_value>idle</slot_value>
          <slot_value>wounded</slot_value>
          <slot_value>dead</slot_value>
        </slot>
      - <previous_icon>
          <icon_name>policecar</icon_name>
          <chance>2</chance>
          <timespan>1</timespan>
        </previous_icon>
      </icon>
    - <icon>
        <icon_name>soldier</icon_name>
        <icon_image>./icons/people/soldier.jpg</icon_image>
      - <slot>
          <slot_name>number</slot_name>
          <slot_value>1</slot_value>
          <slot_value>2</slot_value>
          <slot_value>3-5</slot_value>
          <slot_value>5-10</slot_value>
          <slot_value>10+</slot_value>
        </slot>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>busy</slot_value>
          <slot_value>idle</slot_value>
          <slot_value>wounded</slot_value>
          <slot_value>dead</slot_value>
        </slot>
      </icon>
    - <icon>
        <icon_name>fireman</icon_name>
        <icon_image>./icons/people/fireman.jpg</icon_image>
      - <slot>
          <slot_name>number</slot_name>
          <slot_value>1</slot_value>
          <slot_value>2</slot_value>
          <slot_value>3-5</slot_value>
          <slot_value>5-10</slot_value>
          <slot_value>10+</slot_value>
      </slot>
    - <slot>
        <slot_name>status</slot_name>
        <slot_value>busy</slot_value>
        <slot_value>idle</slot_value>
        <slot_value>wounded</slot_value>
        <slot_value>dead</slot_value>
      </slot>
    - <previous_icon>
        <icon_name>firetruck</icon_name>
        <chance>4</chance>
        <timespan>1</timespan>
      </previous_icon>
    </icon>
  - <icon>
      <icon_name>nurse</icon_name>
      <icon_image>./icons/people/nurse.jpg</icon_image>
    - <slot>
        <slot_name>number</slot_name>
        <slot_value>1</slot_value>
        <slot_value>2</slot_value>
        <slot_value>3-5</slot_value>
        <slot_value>5-10</slot_value>
        <slot_value>10+</slot_value>
      </slot>
    - <slot>
        <slot_name>status</slot_name>
        <slot_value>busy</slot_value>
        <slot_value>idle</slot_value>
        <slot_value>wounded</slot_value>
        <slot_value>dead</slot_value>
      </slot>
    - <previous_icon>
        <icon_name>ambulance</icon_name>
        <chance>4</chance>
        <timespan>1</timespan>
      </previous_icon>
    - <previous_icon>
        <icon_name>victim</icon_name>
        <chance>2</chance>
        <timespan>3</timespan>
      </previous_icon>
    </icon>
  - <icon>
      <icon_name>bombsquad</icon_name>
      <icon_image>./icons/people/bomb_squad.jpg</icon_image>
    - <slot>
        <slot_name>number</slot_name>
        <slot_value>1</slot_value>
        <slot_value>2</slot_value>
        <slot_value>3-5</slot_value>
        <slot_value>5-10</slot_value>
        <slot_value>10+</slot_value>
      </slot>
    - <slot>
        <slot_name>status</slot_name>
        <slot_value>busy</slot_value>
        <slot_value>idle</slot_value>
        <slot_value>wounded</slot_value>
        <slot_value>dead</slot_value>
      </slot>
    - <next_icon>
        <icon_name>explosion</icon_name>
        <chance>2</chance>
        <timespan>3</timespan>
      </next_icon>
    - <previous_icon>
        <icon_name>bomb</icon_name>
        <chance>4</chance>
        <timespan>1</timespan>
      </previous_icon>
    </icon>
  - <icon>
      <icon_name>person</icon_name>
      <icon_image>./icons/people/person.jpg</icon_image>
    - <slot>
        <slot_name>number</slot_name>
        <slot_value>1</slot_value>
        <slot_value>2</slot_value>
        <slot_value>3-5</slot_value>
        <slot_value>5-10</slot_value>
        <slot_value>10+</slot_value>
      </slot>
    - <slot>
        <slot_name>status</slot_name>
        <slot_value>spectator</slot_value>
        <slot_value>evacuated</slot_value>
      </slot>
    </icon>
  - <icon>
      <icon_name>armedperson</icon_name>
      <icon_image>./icons/people/armedperson.jpg</icon_image>
    - <slot>
        <slot_name>number</slot_name>
        <slot_value>1</slot_value>
        <slot_value>2</slot_value>
        <slot_value>3-5</slot_value>
        <slot_value>5-10</slot_value>
        <slot_value>10+</slot_value>
      </slot>
    - <slot>
        <slot_name>weapon</slot_name>
        <slot_value>unknown</slot_value>
        <slot_value>gun</slot_value>
        <slot_value>knife</slot_value>
      </slot>
    - <next_icon>
        <icon_name>policeman</icon_name>
        <chance>4</chance>
        <timespan>3</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>crashedcar</icon_name>
        <chance>1</chance>
        <timespan>3</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>flames</icon_name>
        <chance>1</chance>
        <timespan>3</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>victim</icon_name>
        <chance>4</chance>
        <timespan>3</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>deadperson</icon_name>
        <chance>3</chance>
        <timespan>3</timespan>
      </next_icon>
    </icon>
  - <icon>
      <icon_name>victim</icon_name>
      <icon_image>./icons/people/victim.jpg</icon_image>
    - <slot>
        <slot_name>number</slot_name>
        <slot_value>1</slot_value>
```

```
            <slot_value>2</slot_value>
            <slot_value>3-5</slot_value>
            <slot_value>5-10</slot_value>
            <slot_value>10+</slot_value>
          </slot>
        - <slot>
            <slot_name>status</slot_name>
            <slot_value>injured</slot_value>
            <slot_value>ill</slot_value>
          </slot>
        - <slot>
            <slot_name>priority</slot_name>
            <slot_value>low</slot_value>
            <slot_value>medium</slot_value>
            <slot_value>high</slot_value>
          </slot>
        - <next_icon>
            <icon_name>ambulance</icon_name>
            <chance>4</chance>
            <timespan>3</timespan>
          </next_icon>
        - <next_icon>
            <icon_name>nurse</icon_name>
            <chance>4</chance>
            <timespan>3</timespan>
          </next_icon>
        </icon>
      - <icon>
          <icon_name>deadperson</icon_name>
          <icon_image>./icons/people/deadperson.jpg</icon_image>
        - <slot>
            <slot_name>number</slot_name>
            <slot_value>1</slot_value>
            <slot_value>2</slot_value>
            <slot_value>3-5</slot_value>
            <slot_value>5-10</slot_value>
            <slot_value>10+</slot_value>
          </slot>
        - <next_icon>
            <icon_name>ambulance</icon_name>
            <chance>3</chance>
            <timespan>4</timespan>
          </next_icon>
        - <next_icon>
            <icon_name>nurse</icon_name>
            <chance>3</chance>
            <timespan>4</timespan>
          </next_icon>
        </icon>
      </group>
    - <group>
      - <icon>
          <icon_name>flames</icon_name>
          <icon_image>./icons/events/flames.jpg</icon_image>
        - <slot>
            <slot_name>status</slot_name>
            <slot_value>under_control</slot_value>
            <slot_value>expanding</slot_value>
            <slot_value>decreasing</slot_value>
          </slot>
        - <slot>
            <slot_name>size</slot_name>
            <slot_value>small</slot_value>
            <slot_value>medium</slot_value>
            <slot_value>big</slot_value>
            <slot_value>huge</slot_value>
          </slot>
        - <slot>
            <slot_name>intensity</slot_name>
      <slot_value>small</slot_value>
      <slot_value>medium</slot_value>
      <slot_value>big</slot_value>
      <slot_value>huge</slot_value>
    </slot>
  - <next_icon>
      <icon_name>smoke</icon_name>
      <chance>5</chance>
      <timespan>1</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>firetruck</icon_name>
      <chance>4</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>fireman</icon_name>
      <chance>4</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>victim</icon_name>
      <chance>2</chance>
      <timespan>3</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>person</icon_name>
      <chance>3</chance>
      <timespan>3</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>explosion</icon_name>
      <chance>1</chance>
      <timespan>3</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>ambulance</icon_name>
      <chance>2</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>policecar</icon_name>
      <chance>2</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>policeman</icon_name>
      <chance>2</chance>
      <timespan>4</timespan>
    </next_icon>
  - <previous_icon>
      <icon_name>flames</icon_name>
      <chance>1</chance>
      <timespan>5</timespan>
    </previous_icon>
  - <previous_icon>
      <icon_name>explosion</icon_name>
      <chance>1</chance>
      <timespan>5</timespan>
    </previous_icon>
  - <previous_icon>
      <icon_name>smoke</icon_name>
      <chance>5</chance>
      <timespan>5</timespan>
    </previous_icon>
  </icon>
- <icon>
    <icon_name>explosion</icon_name>
    <icon_image>./icons/events/explosion.jpg</icon_image>
  - <slot>
      <slot_name>intensity</slot_name>
      <slot_value>small</slot_value>
      <slot_value>medium</slot_value>
      <slot_value>big</slot_value>
      <slot_value>huge</slot_value>
    </slot>
  - <next_icon>
      <icon_name>flames</icon_name>
      <chance>4</chance>
      <timespan>2</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>policecar</icon_name>
      <chance>3</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>firetruck</icon_name>
      <chance>4</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>person</icon_name>
      <chance>3</chance>
      <timespan>3</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>victim</icon_name>
      <chance>3</chance>
      <timespan>2</timespan>
    </next_icon>
  - <previous_icon>
      <icon_name>flames</icon_name>
      <chance>1</chance>
      <timespan>100</timespan>
    </previous_icon>
  - <previous_icon>
      <icon_name>bomb</icon_name>
      <chance>1</chance>
      <timespan>100</timespan>
    </previous_icon>
  </icon>
- <icon>
    <icon_name>smoke</icon_name>
    <icon_image>./icons/events/smoke.jpg</icon_image>
  - <slot>
      <slot_name>intensity</slot_name>
      <slot_value>small</slot_value>
      <slot_value>medium</slot_value>
      <slot_value>big</slot_value>
      <slot_value>huge</slot_value>
    </slot>
  - <slot>
      <slot_name>size</slot_name>
      <slot_value>small</slot_value>
      <slot_value>medium</slot_value>
      <slot_value>big</slot_value>
      <slot_value>huge</slot_value>
    </slot>
  - <next_icon>
      <icon_name>flames</icon_name>
      <chance>4</chance>
      <timespan>2</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>policecar</icon_name>
      <chance>3</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>policeman</icon_name>
      <chance>2</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>firetruck</icon_name>
      <chance>3</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>fireman</icon_name>
      <chance>2</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>person</icon_name>
      <chance>3</chance>
      <timespan>3</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>victim</icon_name>
      <chance>2</chance>
      <timespan>2</timespan>
    </next_icon>
  - <previous_icon>
      <icon_name>flames</icon_name>
      <chance>1</chance>
      <timespan>100</timespan>
    </previous_icon>
  </icon>
- <icon>
    <icon_name>bomb</icon_name>
    <icon_image>./icons/events/bomb.jpg</icon_image>
  - <slot>
      <slot_name>size</slot_name>
      <slot_value>unknown</slot_value>
      <slot_value>small</slot_value>
      <slot_value>medium</slot_value>
      <slot_value>big</slot_value>
    </slot>
  - <next_icon>
      <icon_name>explosion</icon_name>
      <chance>4</chance>
      <timespan>2</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>roadblock</icon_name>
      <chance>3</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>policecar</icon_name>
      <chance>3</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>firetruck</icon_name>
      <chance>3</chance>
      <timespan>4</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>person</icon_name>
      <chance>3</chance>
      <timespan>3</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>bombsquad</icon_name>
```

```
        <chance>5</chance>
        <timespan>4</timespan>
      </next_icon>
    </icon>
  - <icon>
      <icon_name>roadblock</icon_name>
      <icon_image>./icons/events/roadblock.jpg</icon_image>
    - <slot>
        <slot_name>size</slot_name>
        <slot_value>small</slot_value>
        <slot_value>medium</slot_value>
        <slot_value>big</slot_value>
      </slot>
    - <next_icon>
        <icon_name>car</icon_name>
        <chance>2</chance>
        <timespan>2</timespan>
      </next_icon>
    - <previous_icon>
        <icon_name>victim</icon_name>
        <chance>3</chance>
        <timespan>5</timespan>
      </previous_icon>
    - <previous_icon>
        <icon_name>policeman</icon_name>
        <chance>1</chance>
        <timespan>5</timespan>
      </previous_icon>
    </icon>
  - <icon>
      <icon_name>flood</icon_name>
      <icon_image>./icons/events/flood.jpg</icon_image>
    - <slot>
        <slot_name>size</slot_name>
        <slot_value>small</slot_value>
        <slot_value>medium</slot_value>
        <slot_value>big</slot_value>
      </slot>
    - <next_icon>
        <icon_name>firetruck</icon_name>
        <chance>2</chance>
        <timespan>2</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>person</icon_name>
        <chance>3</chance>
        <timespan>3</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>victim</icon_name>
        <chance>2</chance>
        <timespan>5</timespan>
      </next_icon>
    </icon>
  - <icon>
      <icon_name>tornado</icon_name>
      <icon_image>./icons/events/tornado.jpg</icon_image>
    - <slot>
        <slot_name>size</slot_name>
        <slot_value>small</slot_value>
        <slot_value>medium</slot_value>
        <slot_value>big</slot_value>
      </slot>
    - <next_icon>
        <icon_name>flood</icon_name>
        <chance>2</chance>
        <timespan>2</timespan>
      </next_icon>
    - <next_icon>
```

```
        <icon_name>person</icon_name>
        <chance>3</chance>
        <timespan>3</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>victim</icon_name>
        <chance>2</chance>
        <timespan>5</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>car</icon_name>
        <chance>3</chance>
        <timespan>3</timespan>
      </next_icon>
    </icon>
  </group>
- <group>
  - <icon>
      <icon_name>ambulance</icon_name>
      <icon_image>./icons/transport/ambulance.jpg</icon_image>
    - <slot>
        <slot_name>status</slot_name>
        <slot_value>driving</slot_value>
        <slot_value>crashed</slot_value>
        <slot_value>idle</slot_value>
      </slot>
    - <next_icon>
        <icon_name>nurse</icon_name>
        <chance>4</chance>
        <timespan>1</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>person</icon_name>
        <chance>2</chance>
        <timespan>5</timespan>
      </next_icon>
    - <previous_icon>
        <icon_name>victim</icon_name>
        <chance>3</chance>
        <timespan>20</timespan>
      </previous_icon>
    </icon>
  - <icon>
      <icon_name>policecar</icon_name>
      <icon_image>./icons/transport/policecar.jpg</icon_image>
    - <slot>
        <slot_name>status</slot_name>
        <slot_value>driving</slot_value>
        <slot_value>crashed</slot_value>
        <slot_value>idle</slot_value>
      </slot>
    - <next_icon>
        <icon_name>policeman</icon_name>
        <chance>4</chance>
        <timespan>1</timespan>
      </next_icon>
    - <next_icon>
        <icon_name>person</icon_name>
        <chance>2</chance>
        <timespan>5</timespan>
      </next_icon>
    </icon>
  - <icon>
      <icon_name>firetruck</icon_name>
      <icon_image>./icons/transport/firetruck.jpg</icon_image>
    - <slot>
        <slot_name>status</slot_name>
        <slot_value>driving</slot_value>
        <slot_value>crashed</slot_value>
      <slot_value>idle</slot_value>
      <slot_value>deployed</slot_value>
    </slot>
  - <next_icon>
      <icon_name>fireman</icon_name>
      <chance>4</chance>
      <timespan>1</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>person</icon_name>
      <chance>2</chance>
      <timespan>5</timespan>
    </next_icon>
  </icon>
- <icon>
    <icon_name>tank</icon_name>
    <icon_image>./icons/transport/tank.jpg</icon_image>
  - <slot>
      <slot_name>status</slot_name>
      <slot_value>driving</slot_value>
      <slot_value>crashed</slot_value>
      <slot_value>idle</slot_value>
    </slot>
  - <next_icon>
      <icon_name>soldier</icon_name>
      <chance>4</chance>
      <timespan>1</timespan>
    </next_icon>
  - <next_icon>
      <icon_name>person</icon_name>
      <chance>2</chance>
      <timespan>5</timespan>
    </next_icon>
  </icon>
- <icon>
    <icon_name>train</icon_name>
    <icon_image>./icons/transport/train.jpg</icon_image>
  - <slot>
      <slot_name>status</slot_name>
      <slot_value>driving</slot_value>
      <slot_value>crashed</slot_value>
      <slot_value>idle</slot_value>
    </slot>
  - <slot>
      <slot_name>type</slot_name>
      <slot_value>passenger</slot_value>
      <slot_value>cargo</slot_value>
    </slot>
  </icon>
- <icon>
    <icon_name>car</icon_name>
    <icon_image>./icons/transport/car.jpg</icon_image>
  - <slot>
      <slot_name>status</slot_name>
      <slot_value>driving</slot_value>
      <slot_value>parked</slot_value>
      <slot_value>waiting</slot_value>
    </slot>
  </icon>
- <icon>
    <icon_name>crashedcar</icon_name>
    <icon_image>./icons/transport/crashedcar.jpg</icon_image>
  - <slot>
      <slot_name>damage</slot_name>
      <slot_value>small</slot_value>
      <slot_value>medium</slot_value>
      <slot_value>total_loss</slot_value>
    </slot>
  - <next_icon>
```

```
          <icon_name>flames</icon_name>
          <chance>1</chance>
          <timespan>1</timespan>
        </next_icon>
      - <next_icon>
          <icon_name>policeman</icon_name>
          <chance>3</chance>
          <timespan>1</timespan>
        </next_icon>
      - <next_icon>
          <icon_name>fireman</icon_name>
          <chance>2</chance>
          <timespan>1</timespan>
        </next_icon>
      </icon>
    - <icon>
        <icon_name>boat</icon_name>
        <icon_image>./icons/transport/boat.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>sailing</slot_value>
          <slot_value>anchored</slot_value>
          <slot_value>sinking</slot_value>
        </slot>
      </icon>
    - <icon>
        <icon_name>helicopter</icon_name>
        <icon_image>./icons/transport/helicopter.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>landed</slot_value>
          <slot_value>moving</slot_value>
          <slot_value>pivoting</slot_value>
          <slot_value>crashed</slot_value>
        </slot>
      - <previous_icon>
          <icon_name>victim</icon_name>
          <chance>2</chance>
          <timespan>2</timespan>
        </previous_icon>
      </icon>
    - <icon>
        <icon_name>airplane</icon_name>
        <icon_image>./icons/transport/airplane.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>landed</slot_value>
          <slot_value>moving</slot_value>
          <slot_value>crashed</slot_value>
        </slot>
      </icon>
    </group>
  - <group>
    - <icon>
        <icon_name>building</icon_name>
        <icon_image>./icons/buildings/building.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>empty</slot_value>
          <slot_value>evacuating</slot_value>
          <slot_value>occupied</slot_value>
        </slot>
      </icon>
    </group>
</iconlist>
```

scenariolist.xml

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
  <!DOCTYPE icon (View Source for full doctype...)>
- <scenariolist>
  - <scenario>
      <scenario_name>riot</scenario_name>
    - <icon>
        <icon_name>policeman</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>person</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>armedperson</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>crashedcar</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>flames</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>smoke</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>victim</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>helicopter</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>roadblock</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>policecar</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>ambulance</icon_name>
        <chance>2</chance>
      </icon>
    </scenario>
  - <scenario>
      <scenario_name>carcrash</scenario_name>
    - <icon>
        <icon_name>policeman</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>crashedcar</icon_name>
        <chance>5</chance>
      </icon>
    - <icon>
        <icon_name>roadblock</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>victim</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>helicopter</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>policecar</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>ambulance</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>nurse</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>firetruck</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>fireman</icon_name>
        <chance>1</chance>
      </icon>
    </scenario>
  - <scenario>
      <scenario_name>fire</scenario_name>
    - <icon>
        <icon_name>policeman</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>roadblock</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>victim</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>helicopter</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>policecar</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>ambulance</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>nurse</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>firetruck</icon_name>
        <chance>5</chance>
      </icon>
    - <icon>
        <icon_name>fireman</icon_name>
        <chance>5</chance>
      </icon>
    - <icon>
        <icon_name>flames</icon_name>
        <chance>5</chance>
      </icon>
    - <icon>
        <icon_name>smoke</icon_name>
        <chance>5</chance>
      </icon>
    - <icon>
        <icon_name>explosion</icon_name>
        <chance>2</chance>
      </icon>
    </scenario>
  - <scenario>
      <scenario_name>bombscare</scenario_name>
    - <icon>
        <icon_name>policeman</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>roadblock</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>victim</icon_name>
        <chance>1</chance>
      </icon>
    - <icon>
        <icon_name>policecar</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>ambulance</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>firetruck</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>fireman</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>explosion</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>building</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>bombsquad</icon_name>
        <chance>8</chance>
      </icon>
    - <icon>
        <icon_name>bomb</icon_name>
        <chance>8</chance>
      </icon>
    </scenario>
  - <scenario>
      <scenario_name>shooting</scenario_name>
    - <icon>
        <icon_name>policeman</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>roadblock</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
```

```
        <icon_name>victim</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>policecar</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>ambulance</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>nurse</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>armedperson</icon_name>
        <chance>5</chance>
      </icon>
  </scenario>
</scenariolist>
```

Appendix C: User Manual ISME

This is the user manual for the ISME system. The functionality and usage of the client is explained here in a non technical approach.

What is the Application for?

This application is a demonstration of how emergency services can communicate with each other, by the use of icons and a map. It is a simulation for the application on a handheld, for example the Zaurus SL-C750.

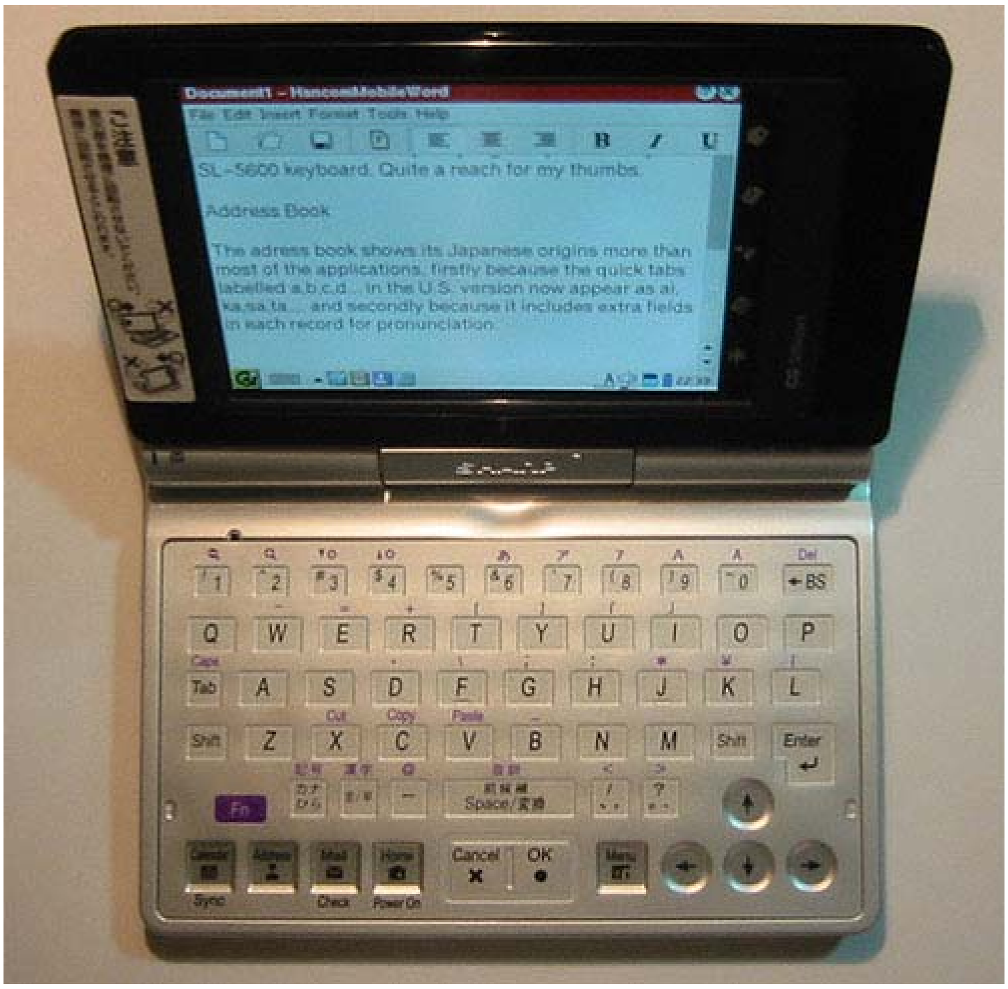


Figure 79 The Sharp Zaurus SL-C750

The application is a proof of concept, as the emergency services are very scarce with providing information about emergency situations. Therefore the icons and AI are based on rules that were extracted from news articles.

How does it Work?

The functions of ISME will be explained by the use of the graphical user interface (GUI).

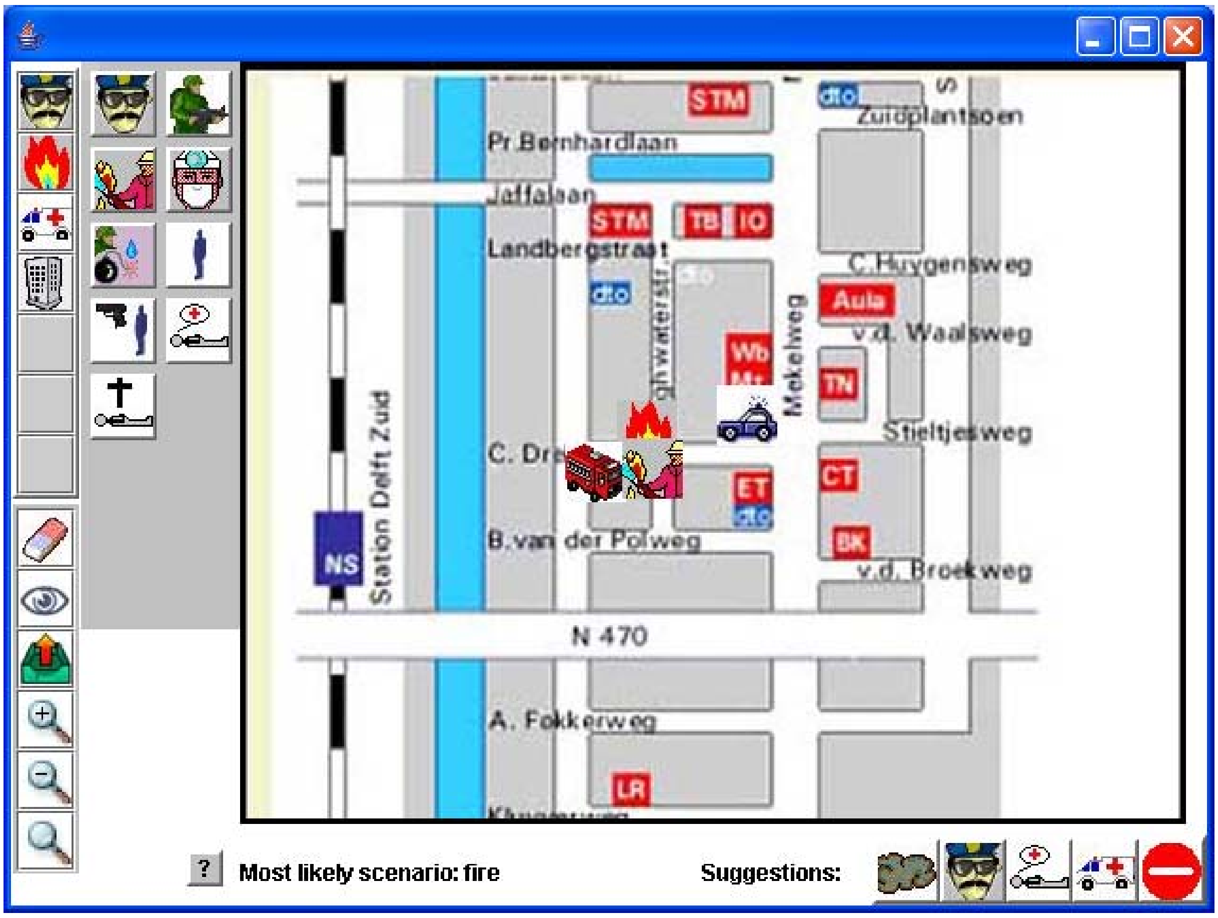
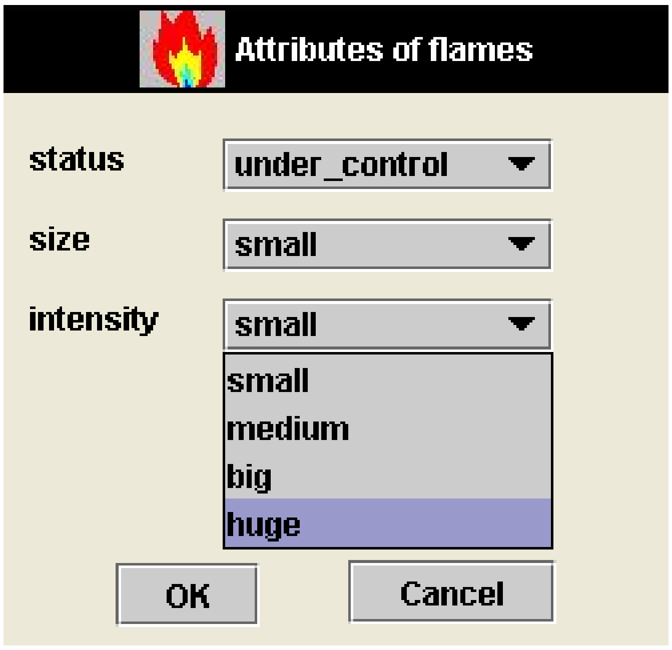


Figure 80 GUI of the report tool

On the far top left the different icon groups are represented by their first icon in the group. In this example there are groups for *people, events, transportation,* and *buildings*. There is also space for 3 more groups. The groups can be easily expanded and changed, as will be explained later.

Currently the group *events* is selected, and there are just 2 icons in it for now. Each group can be expanded to contain 14 different icons. The 2 icons in the grey area can be selected by clicking on them. The icon for *flames* is selected now, which can be seen by the red border surrounding the icon.

Now that an icon is selected, it can be placed by clicking on the map. Click on the location were the icon needs to be placed, and an attribute window will be shown, to provide additional information about the icon.


Attributes of flames

status: under_control

size: small

intensity: small

Figure 81 Attribute window for Flames

Select the appropriate attribute values and click ok to place the icon, or click cancel to not place the icon after all.

When all the icons are placed, click the send and receive button on the bottom left toolbar. This will send the icons you placed to the (central) server which will update its world model, and send it back to the client. This means clicking on the send/receive buttons will cause new icons to appear on the map, namely those icons that other clients may have placed.

When an icon needs to be deleted, select the eraser tool, the first icon on the toolbar, and click on the icon that needs to be erased. After deletion, click on the send/receive button again, so the server can update its world model and inform the other clients. Only delete an icon when you are absolutely sure it is necessary.

The next icon on the toolbar is an eye. Clicking this will cause the application to go in inspect mode. Click on any icon on the map to see and alter its attributes. This means you can also change the attributes of icons that were placed by another client.

The toolbar contains 3 more icons, which all have to do with zooming. The *zoom in* and *zoom out* buttons will zoom in and out of the map, keeping the centre of the map in the middle. The other zoom icon will allow users to drag a specific area of the map, and zoom in on that part.

After some icons have been send, the server will suggest new icons to be placed to the clients. Those icons will be shown on the bottom right bar. These icons work exactly as the icons in the icon groups; they can be selected and placed on the map immediately. Again a red border shows which icon is selected.

Finally the server will also send its world model in the form of a most likely scenario. Each icon will award points to certain scenarios, and the most likely will be shown on the bottom label in the application. Click on the question mark icon to see how many points each scenario was awarded.

Adding or Editing the Icons

It is possible to easily edit and change the icons that can be used in the application. All the icon information is stored in 2 XML files. The application will read the icons and their rules out of these files. The scenario XML file is called scenariolist.xml and can be found in the map *icons*. Do not move it to another map, or the application will not find it. The file can be opened by any browser to see its content in an organized way. For example in MS Internet Explorer it looks like this:

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
  <!DOCTYPE icon (View Source for full doctype...)>
- <scenariolist>
  + <scenario>
  + <scenario>
  + <scenario>
  + <scenario>
  - <scenario>
      <scenario_name>shooting</scenario_name>
    - <icon>
        <icon_name>policeman</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>roadblock</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>victim</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>policecar</icon_name>
        <chance>4</chance>
      </icon>
    - <icon>
        <icon_name>ambulance</icon_name>
        <chance>3</chance>
      </icon>
    - <icon>
        <icon_name>nurse</icon_name>
        <chance>2</chance>
      </icon>
    - <icon>
        <icon_name>armedperson</icon_name>
        <chance>4</chance>
      </icon>
    </scenario>
  </scenariolist>
```

Click on the + and – to explore the information in the different scenarios. To edit the information open the file in any text editor. Make sure that the structure of the file remains correct. To see how it is defined open the scenariolist.dtd, the Document Type Definition.

The AI of the server can be changed by editing these files as well. When the values of the *chance* fields are changed, the server may give different suggestions and award the points for the scenarios in another way.

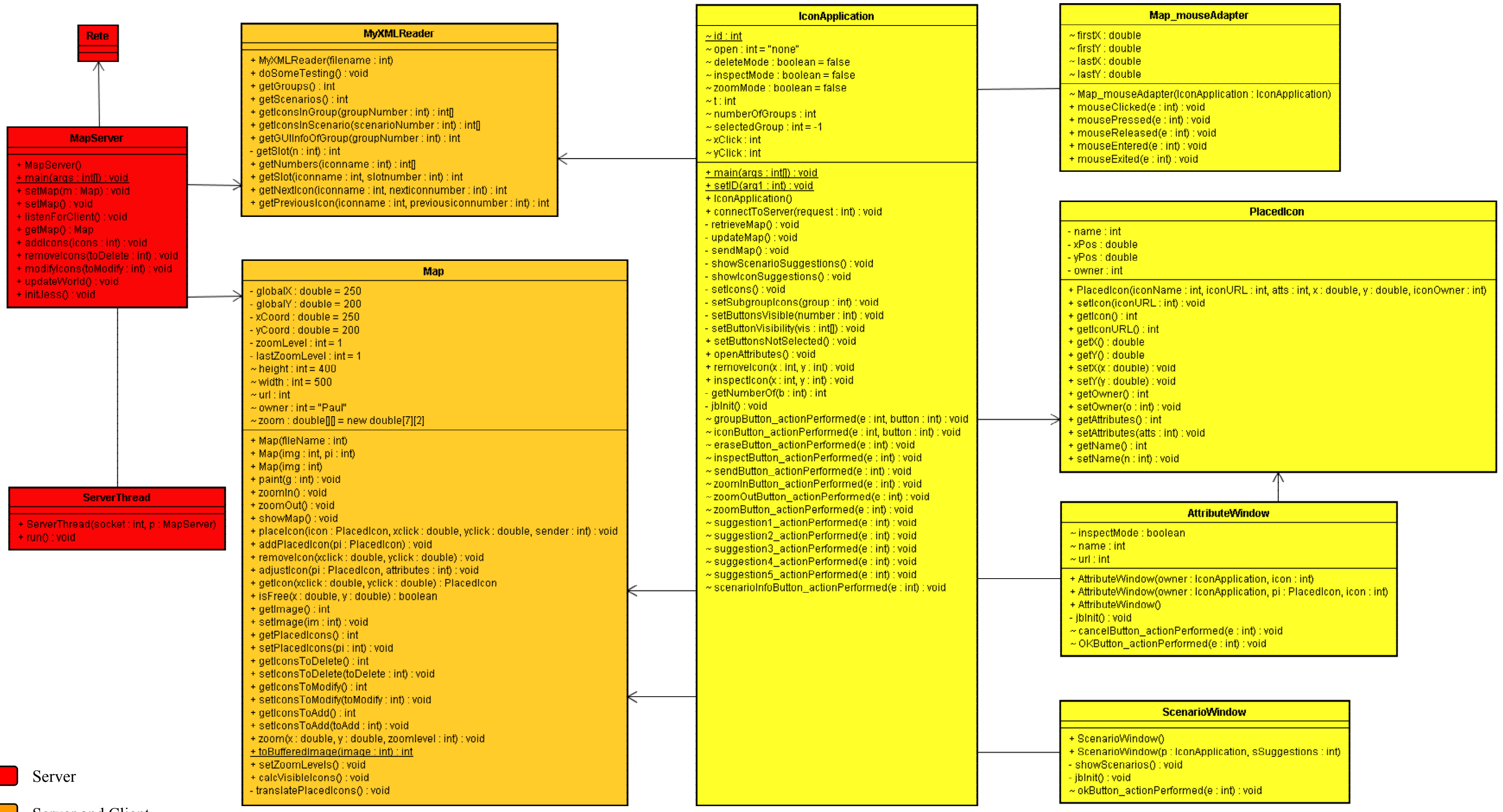
The same goes for the iconlist.xml file, which is defined by the iconlist.dtd definition file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
 <!DOCTYPE icon (View Source for full doctype...)>
- <iconlist>
  - <group>
    - <icon>
        <icon_name>policeman</icon_name>
        <icon_image>./icons/people/policeman.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>busy</slot_value>
          <slot_value>idle</slot_value>
          <slot_value>wounded</slot_value>
          <slot_value>dead</slot_value>
        </slot>
      - <previous_icon>
          <icon_name>policecar</icon_name>
          <chance>2</chance>
          <timespan>1</timespan>
        </previous_icon>
      </icon>
    - <icon>
        <icon_name>soldier</icon_name>
        <icon_image>./icons/people/soldier.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>busy</slot_value>
          <slot_value>idle</slot_value>
          <slot_value>wounded</slot_value>
          <slot_value>dead</slot_value>
        </slot>
      - <previous_icon>
          <icon_name>policecar</icon_name>
          <chance>2</chance>
          <timespan>1</timespan>
        </previous_icon>
      </icon>
    - <icon>
        <icon_name>fireman</icon_name>
        <icon_image>./icons/people/fireman.jpg</icon_image>
      - <slot>
          <slot_name>status</slot_name>
          <slot_value>busy</slot_value>
          <slot_value>idle</slot_value>
          <slot_value>wounded</slot_value>
          <slot_value>dead</slot_value>
        </slot>
      - <previous_icon>
          <icon_name>firetruck</icon_name>
          <chance>4</chance>
          <timespan>1</timespan>
        </previous_icon>
      </icon>
    </group>
  + <group>
  + <group>
  + <group>
  </iconlist>
```

This is where the applications gets its icons from. Icons can be added to the groups, as long as the *icon_image* url is a link to an existing image. Also entire groups can be added or deleted in this file. Keep in mind that the GUI cannot support more then 7 groups of 14 icons each while adding icons, which should be enough since there is room for 98 different icons).

The server also gets AI information from this file. Causal links between icons can be defined in this file by adding *previous_icon*s and *next_icon*s to the basic icons in the groups. Editing these, and their values, will have an effect on which icons the server will suggest.

Appendix D: Detailed Class Diagram


Rete
MapServer
+ MapServer()
+ main(args : int[]) : void
+ setMap(m : Map) : void
+ setMap() : void
+ listenForClient() : void
+ getMap() : Map
+ addIcons(icons : int) : void
+ removeIcons(toDelete : int) : void
+ modifyIcons(toModify : int) : void
+ updateWorld() : void
+ initJess() : void
ServerThread
+ ServerThread(socket : int, p : MapServer)
+ run() : void
MyXMLReader
+ MyXMLReader(filename : int)
+ doSomeTesting() : void
+ getGroups() : int
+ getScenarios() : int
+ getIconsInGroup(groupNumber : int) : int[]
+ getIconsInScenario(scenarioNumber : int) : int[]
+ getGUIInfoOfGroup(groupNumber : int) : int
- getSlot(n : int) : int
+ getNumbers(iconname : int) : int[]
+ getSlot(iconname : int, slotnumber : int) : int
+ getNextIcon(iconname : int, nexticonnumber : int) : int
+ getPreviousIcon(iconname : int, previousiconnumber : int) : int
Map
- globalX : double = 250
- globalY : double = 200
- xCoord : double = 250
- yCoord : double = 200
- zoomLevel : int = 1
- lastZoomLevel : int = 1
~ height : int = 400
~ width : int = 500
~ url : int
~ owner : int = "Paul"
~ zoom : double[][] = new double[7][2]
+ Map(fileName : int)
+ Map(img : int, pi : int)
+ Map(img : int)
+ paint(g : int) : void
+ zoomIn() : void
+ zoomOut() : void
+ showMap() : void
+ placeIcon(icon : PlacedIcon, xclick : double, yclick : double, sender : int) : void
+ addPlacedIcon(pi : PlacedIcon) : void
+ removeIcon(xclick : double, yclick : double) : void
+ adjustIcon(pi : PlacedIcon, attributes : int) : void
+ getIcon(xclick : double, yclick : double) : PlacedIcon
+ isFree(x : double, y : double) : boolean
+ getImage() : int
+ setImage(im : int) : void
+ getPlacedIcons() : int
+ setPlacedIcons(pi : int) : void
+ getIconsToDelete() : int
+ setIconsToDelete(toDelete : int) : void
+ getIconsToModify() : int
+ setIconsToModify(toModify : int) : void
+ getIconsToAdd() : int
+ setIconsToAdd(toAdd : int) : void
+ zoom(x : double, y : double, zoomlevel : int) : void
+ toBufferedImage(image : int) : int
+ setZoomLevels() : void
+ calcVisibleIcons() : void
- translatePlacedIcons() : void
IconApplication
~ id : int
~ open : int = "none"
~ deleteMode : boolean = false
~ inspectMode : boolean = false
~ zoomMode : boolean = false
~ t : int
~ numberOfGroups : int
~ selectedGroup : int = -1
~ xClick : int
~ yClick : int
+ main(args : int[]) : void
+ setID(arg1 : int) : void
+ IconApplication()
+ connectToServer(request : int) : void
- retrieveMap() : void
- updateMap() : void
- sendMap() : void
- showScenarioSuggestions() : void
- showIconSuggestions() : void
- setIcons() : void
- setSubgroupIcons(group : int) : void
- setButtonsVisible(number : int) : void
- setButtonVisibility(vis : int[]) : void
+ setButtonsNotSelected() : void
+ openAttributes() : void
+ removeIcon(x : int, y : int) : void
+ inspectIcon(x : int, y : int) : void
- getNumberOf(b : int) : int
- jbInit() : void
~ groupButton_actionPerformed(e : int, button : int) : void
~ iconButton_actionPerformed(e : int, button : int) : void
~ eraseButton_actionPerformed(e : int) : void
~ inspectButton_actionPerformed(e : int) : void
~ sendButton_actionPerformed(e : int) : void
~ zoomInButton_actionPerformed(e : int) : void
~ zoomOutButton_actionPerformed(e : int) : void
~ zoomButton_actionPerformed(e : int) : void
~ suggestion1_actionPerformed(e : int) : void
~ suggestion2_actionPerformed(e : int) : void
~ suggestion3_actionPerformed(e : int) : void
~ suggestion4_actionPerformed(e : int) : void
~ suggestion5_actionPerformed(e : int) : void
~ scenarioInfoButton_actionPerformed(e : int) : void
Map_mouseAdapter
~ firstX : double
~ firstY : double
~ lastX : double
~ lastY : double
~ Map_mouseAdapter(IconApplication : IconApplication)
+ mouseClicked(e : int) : void
+ mousePressed(e : int) : void
+ mouseReleased(e : int) : void
+ mouseEntered(e : int) : void
+ mouseExited(e : int) : void
PlacedIcon
- name : int
- xPos : double
- yPos : double
- owner : int
+ PlacedIcon(iconName : int, iconURL : int, atts : int, x : double, y : double, iconOwner : int)
+ setIcon(iconURL : int) : void
+ getIcon() : int
+ getIconURL() : int
+ getX() : double
+ getY() : double
+ setX(x : double) : void
+ setY(y : double) : void
+ getOwner() : int
+ setOwner(o : int) : void
+ getAttributes() : int
+ setAttributes(atts : int) : void
+ getName() : int
+ setName(n : int) : void
AttributeWindow
~ inspectMode : boolean
~ name : int
~ url : int
+ AttributeWindow(owner : IconApplication, icon : int)
+ AttributeWindow(owner : IconApplication, pi : PlacedIcon, icon : int)
+ AttributeWindow()
- jbInit() : void
~ cancelButton_actionPerformed(e : int) : void
~ OKButton_actionPerformed(e : int) : void
ScenarioWindow
+ ScenarioWindow()
+ ScenarioWindow(p : IconApplication, sSuggestions : int)
- showScenarios() : void
- jbInit() : void
~ okButton_actionPerformed(e : int) : void

🟥 Server

🟧 Server and Client

🟨 Client

Figure 82 Detailed Class Diagram

Appendix E: Paper ISME

Paul Schooneman
Faculty of Information Technology and Systems
Delft University of Technology
Mekelweg 4, 2628 CD Delft
p.schooneman@student.tudelft.nl

Abstract

Since the terrorist attacks on 9/11 the issue about the collaboration of emergency services has become increasingly important. One of the conclusions is that better communication between the different services is needed. This motivated us to try a different modality for communication. This thesis describes how iconic communication can be applied to the field of Emergency Management.

We have designed and implemented a prototype application, ISME, which is suited for emergency services to communicate with each other, using a map and icons. The system is designed as a client server application, where the client is focussed on the GUI and the server concentrates on the intelligence. We use a Jess knowledge and rule base to provide a consistent world model at all times, while we store the concepts in XML files. The interface and network is implemented in Java.

ISME gives the users the possibility to report about what they observe by placing icons on a map. The maps will be send to the server, which fuses the multiple observations and constructs a new world model of them. Besides the world model, the server also sends information about the most likely scenario, and it will suggest icons that are expected in the world model but are not placed yet.

Keywords: icon, communication, map, emergency, crisis, interface, Jess, world model

1 Introduction

Since the terrorist attacks on September 11th 2001 the issue about the collaboration of emergency services has become increasingly important. An important issue that day was the total breakdown of the communication infrastructure. One of the conclusions is that better communication between the different services is needed. Besides the network issues, this could be provided by either more communication, or communication in new ways. The latter solution seemed most appealing for a research project. Because our MMI department is doing research on, among other topics, multi modal interfaces and AI, it seemed promising to develop a wireless, ad-hoc, intelligent system with a new modality for communication. The chosen modality is a graphical one, using icons and a map of the surroundings.

The proposed goal is a system that can be used by the emergency services to keep each other up to date about what's going in a particular area, e.g. a city, by placing icons on the map and sending them to each other. The focus of this paper is on the interface and intelligence. What we would like to achieve is to get a structured World Model from a real life crisis situation. A World Model is composed of objects, characteristic features of the objects, and relations between the objects. Every observer has his own World Model. Police officers, firemen and laymen have different views of the world. An observer will look at the situation that is going on, from this he will form his own ideas of what is happening. The brain will construct its own model about the situation, the mental world model. Thoughts like these are based on what he observes, but also on his background knowledge.

We may assume that observers are positioned differently in time and space. Not all observers will be able to see the same things because they report at another time, or from another place. What we want to do is mould these mental models into a computer system. To do this we need to make the mental model more concrete, so that it can be stored in a structured way. Next we want to fuse the different reports into one shared World Model, see Figure 1.

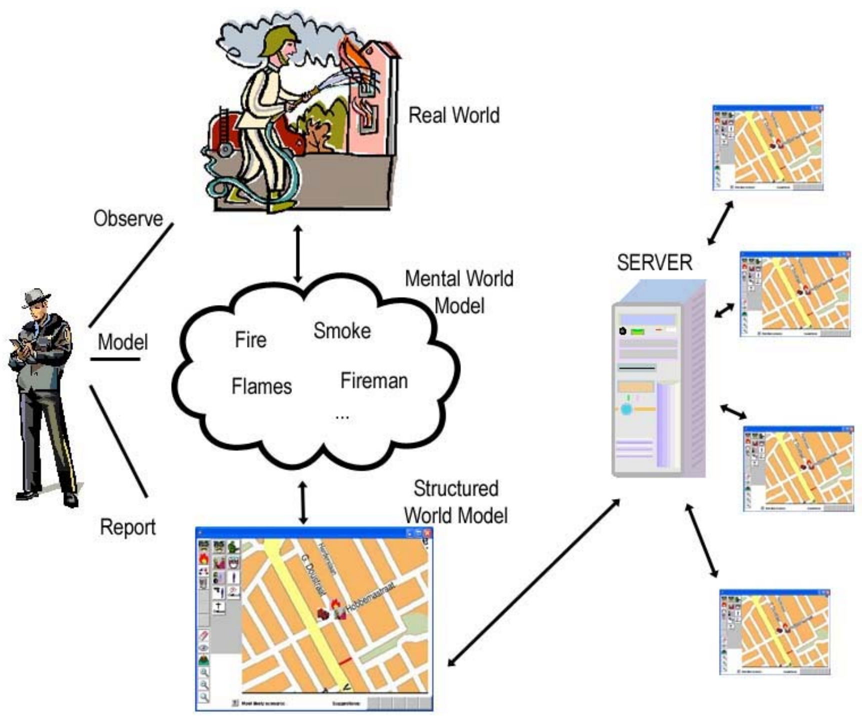


Figure 83 Overview of the proposed system

The agent in the field observes what is going on in the Real World and forms his own Mental World Model of the situation. He then wants to report his thoughts with the report tool of the system. The tool will only be able to handle structured information; concepts represented by icons. Thus the reporter has to concretise his ideas into icons, which he can place on the map. Next the Structured World Model gets send to a central server, which collects reports of all the agents in the field. The server will fuse the ideas and form its own structured world model that gets send back to the agents, along with suggestions that the agent might have forgotten to report. The agent will see these suggestions, forcing him to observe the situation again, to see if he missed anything.

For an intelligent communication system like this we will have to look into several aspects:

- First of all we have to *define a World Model*, different sets of icons, corresponding to different crisis situations, and a priori information about characteristics and relations between the objects. What icons will we need? How are these icons related to each other, and what are the specific characteristics of each?
- Then there needs to be the *interface* of the report tool. The interface should provide a clear structure for the communication. What kind of information should be reported to the system, and what kind of information should be distributed back to the users? How will the information be represented to the user?
- A next aspect is the *intelligence* of the system, in particular in the fusion of the different reports. How does the system handle double or missing information? How should it deal with contradicting information? How does it keep its world model unambiguous and up to date? How does the system handle time and dynamic events?
- Another issue is the *security* of the system. Since it will be based on wireless communication, how can we prevent outsiders to intercept information? Should all information be send to all the users, and if not, how do we define different roles of users? How can we prevent the server from going down? And if it does go down, how can we prevent losing the information?

Since we are only focussing on the interface and the intelligence of the system, the problem description is defined as follows:

Design and implement a demonstrator for a system that is suited for iconic communication in a crisis situation, using a map of the surroundings (1), which is expressive enough to handle complex and unexpected situations (2), yet intuitive enough to use without making (a lot of) errors (3). The system should be intelligent enough to assemble and maintain a correct and up to date world model (4). It should detect possible errors in the form of missing, double and wrongly placed icons (5). Furthermore the system should be dynamic in the sense that new concepts and rules can easily be added (6).

In this paper we will first discuss some related work. After that we will discuss the Model and the Implementation of the system. The next section will present the user test we performed and after that we will discuss our results, conclusions and recommendations for future work.

2 Related work

Prior to designing and implementing our system we have studied some related work, which we will present in this section.

2.1 About Icons

Icons are graphical symbols representing a concept or thing in reality. The term icon has been adapted from the Russian word *ikon*, which is a religious painting or statue. Icons have been around for a very long time, as early as the middle ages complex iconic systems have been used, for example to denote systems of astrological signs. Even the ancient Egyptians were using graphics as a language. [Bea94], [Cha02], [Dor94], [Jon96], [Mea91], [Mea94], [NRC03], [Ric94], [Shn98]

In the 1930s Otto Neurath developed Isotype, a system for communication which uses stylised graphics within a two-dimensional syntax. Neuraths work ranges from a very specific example of how a complex idea can be conveyed graphically, to a proposal for an international set of iconic images. In the 1950s, Charles Bliss developed a set of atomic icons that represent basic objects in the world, and their features. These can be combined to form complex icons that map on to the set of words found in natural languages. Figure 2 shows how we can construct a symbol for telephone using: mouth-ear-language-electricity-telephone

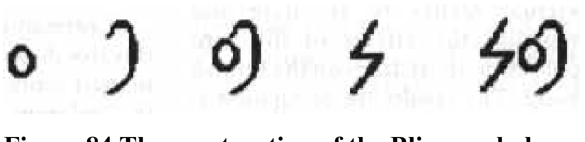


Figure 84 The construction of the Bliss symbol for telephone

The work of Bliss has some resemblance with the work of linguist Anna Wierzbicka, who claims to be able to describe any concept with using only 61 different words. The combination of these atomic words lead to a new concept, just as the atomic pictures of Bliss lead to a new concept. Although Wierzbicka does not use icons, the possibility of mapping her atomic words to atomic icons seems interesting.

Within the computing context the word icon is used to denote a small graphical representation of a program, resource, state, option or window. As such, icons form an important part of the Graphical User Interface (GUI).

An ideal icon language wouldn't need any explanation, the intuition of the user, based upon his life experience, should be enough to immediately understand it. Of course, this is not a very realistic goal. Just as any language, icon language is something that does need some training. Most people already have some training in recognizing icons however, because icons can be found anywhere. In many public places they are used extensively, for example to indicate the toilets, or to show where the emergence exits are. A lot of icons are used in traffic signs, they point out if you are allowed to overtake other cars, if a road is one way only, or is a dead end.

The challenge in designing icons is that they should be as easy as possible to learn, as easy as possible to remember, and as easy as possible to recognize. Therefore icons should be designed with the following criteria in mind:

- Graphically clear
- Semantically unambiguous
- Cultural independent
- Simple

There are three styles of icons that are commonly used, see Figure 3:

1) Silhouette style; this one is very straight forward and clear, the drawback is that it is somewhat limited in the range of things it can represent.
2) Three-quarter top view; this style is very informative, but it requires some visual understanding.
3) Realistic style; this one is easy to recognize, but it is not very generalizing.

Although the use of these different styles makes it possible to select the best one for each icon, it is not recommended to use a mix of different styles, as it can be confusing.

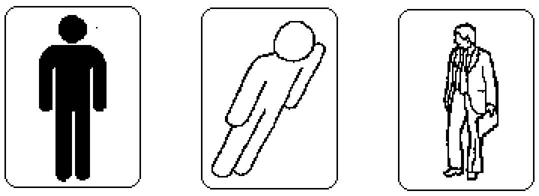


Figure 3 Three styles of icons

In systems that have a lot of different functions, it is not easy to design an icon for each function. To improve the recognition of the different icons, they may be divided in subsets. For example one could use a single group icon for representing surfaces, and have as subset different icons for representing circular and rectangular surfaces. The division of icons in subsets can also improve the overview and layout of applications.

2.2 C2000

In the Netherlands a new digital radio network for the communication of emergency services is being developed [C2000]. It is called C2000, and its goal is to maximally facilitate the communication between the fire-brigades, ambulance services, police-brigades and military police. The mobile communication between these emergency services should be supported and improved. The system should guarantee fast and secure communication, make communication between different emergency services possible, and help improve the safety of the emergency personnel.

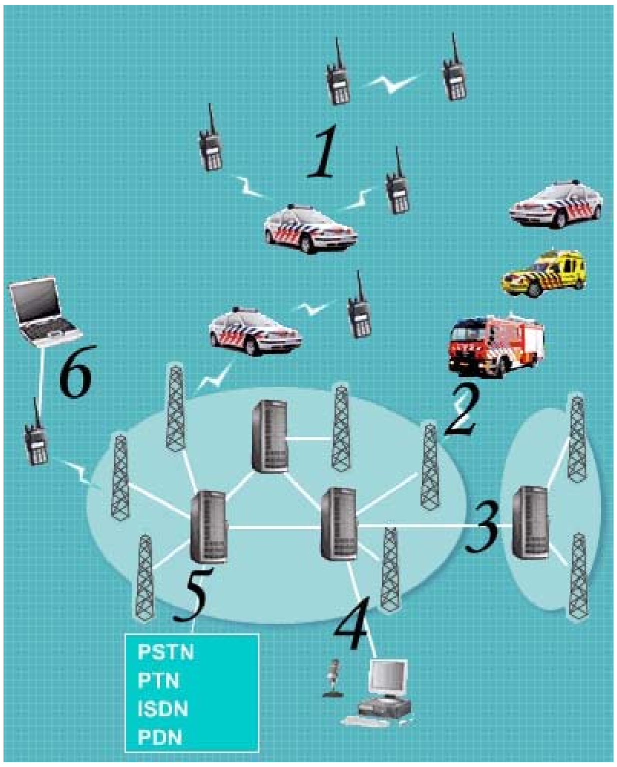


Figure 4 Overview of the C2000 system

The need for a reliable communication system for these services is high. Not only for the day to day activities, but also in case the different services need to cooperate with each other, for example in crisis situations as in Enschede, where a fire occurred in a firework deposit. The emergency services themselves are closely involved in the development if C2000.

In Figure 4 the design of the C2000 network is shown. The numbered components are explained below:

1) Direct Mode Operation, allows for car phones and walkie-talkies to communicate with each other directly, without making use of the network.
2) Air Interface. Communication of a mobile station takes place using electromagnetic waves, with a transmitter mast, or with another mobile station via the Air Interface.
3) Inter System Interface. Multiple TETRA networks can be linked using the Inter System Interface. This is used for international communication.
4) Direct link with the central emergency room.
5) Gateways, make it possible to link the system to other external networks, such as the public telephone network, or the national emergency network.
6) Peripheral Equipment Interface, supports communication between laptops and mobile stations, such as car phones.

C2000 should make the communication between emergency services fast, simple and reliable. This one national system will replace almost 100 local systems that are currently used by the different services. The digital network has big advantages over the old analogue systems:

- C2000 is suitable for multidisciplinary communication, whereas this was impossible with the old systems.
- C2000 is designed in a way that is easy to secure, making it virtually impossible to eavesdrop on it.
- C2000 has a national coverage, whereas the old systems only have regional coverage.
- C2000 has a much better sound quality for speech.
- C2000 is very suitable for data communication.
- All car phones and walkie-talkies are provided with emergency buttons.
- C2000 supports communication with foreign co-workers, improving provided services near the borders.

2.3 Iconic Communication

A closely related project involving icons and emergency situations was done by Iulia Tatomir [Tat03]. The goal of this project was to create an application that allows its users to communicate with each other using an international 'language', icons.

First the graphical user interface was developed, which is designed to be as easy as possible to use. Because it is designed to be a simulation for a PDA application, big constraints on the applications dimension were imposed. The interface is shown in Figure 5. The main part of the research in this project was done on defining the grammar of the icon sentence. Not all combinations of icons form correct sentences. When an icon is selected, the sentence gets parsed and the system decides which icons can follow. This is extended by the interface, by only making the icons that fit correctly in the sentence selectable.

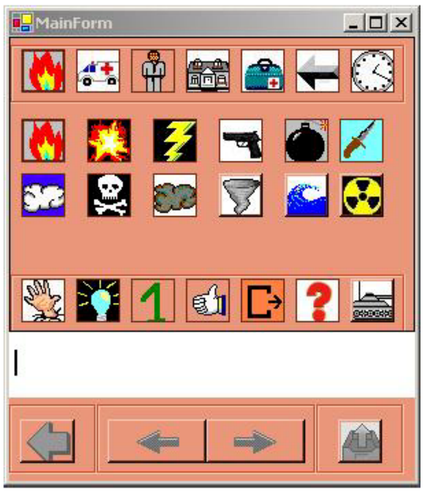


Figure 5 Interface of the application

3 Model and Implementation

The problem description is divided in workable numbered components from which we can extract requirements:

1) To make the system suited for iconic communication in a crisis situation we need to have icons that represent concepts in a crisis, and maps of the surroundings to place the icons on.
2) To make the system expressive enough we will need different categories of icons. Within these categories several icons are needed and we will need a way to add even more information to the icons. The icons will therefore have several attributes. In the case of flames these can be the size of the flames, the intensity and the status.
3) To prevent a lot of errors, the GUI should be intuitive and easy to use. It should be clear which category and icon is selected and they should be easily added to the map. To provide extra information, an attribute window will pop up where the values of the attributes can be given. The values can be selected out of a small list; this decreases the chance of making a wrong selection, and eliminates the chance to make an illegal selection. When icons are placed, the user should be able to delete them again, or to inspect or alter its attributes. To prevent placing icons on the wrong location, the user should be able to easily zoom in and out of the map, to be able to place the icon exactly where it should be.
4) In order to assemble and maintain a world model we will collect all information at one server. To create a world model out of this information we should store it in a database, which will have to be kept consistent at all times. We will choose a client-server implementation for this, where the many clients are the reporters of the crisis, and the server is the part that keeps a consistent world model and distributes all information among its clients.
5) To detect missing, double and wrongly placed icons we need some intelligent agent that constantly works on the information that's being gathered.
6) In order to make the system dynamic, it's useful to store all its information about icons and their rules in a database. The entries of this database should be easy to edit, and the database should be extendible. If we keep this database in separate files that are read by the system on start up, it's possible to adjust rules, and icons in a way that does not require the entire system to be recompiled.

In crisis situations it is not practical to use laptop computers, that's why we want the final system to work with handheld computers, such as the Zaurus, in Figure 6.

Figure 6 Sharp Zaurus SL-C760

The constraints that follow from this are the resolution and the limited speed of the client. That's why we need the server to do all the time intensive calculations, and make the client as light weight as possible. Furthermore we need to use a programming language that can run on the Zaurus. The system will be using wireless communication, that's why we need to keep the data traffic to an absolute minimum.

The main programming language we will use is Java. The storage of the world model of the server and the intelligence of the server are both combined in one component, Jess [JESS]. Jess stands for Java Expert System Shell. In short Jess is an expert system that works with facts, and rules that are automatically triggered when the conditions are met. The last part of the system is the storage of icons and the rules that apply to them. We have chosen XML files for this part.

In Figure 7 an overview of how the different components interact is shown. The XML files contain all information about the icons and scenarios. They also contain parameters that influence the rules in the Server. The server is mainly done in Jess, there is only some Java functionality that lets the server communicate with the clients, and lets it read in the XML files. The Client is done entirely in Java, it gets information about what icons should be in the GUI from the XML files, and then dynamically builds up the interface. A change in the XML files will cause other icons to appear in the client, and other icons and rule parameters will be used in the server as well.

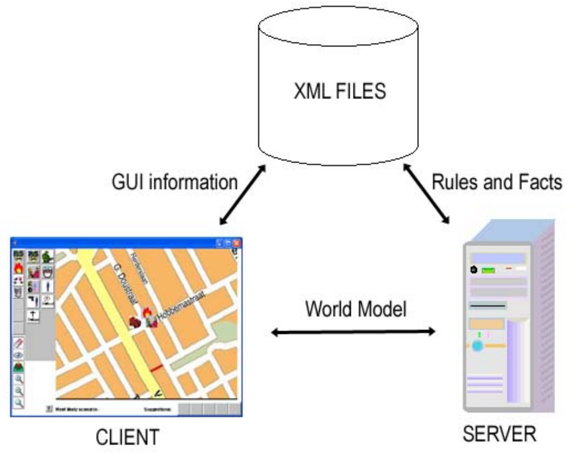


Figure 7 Overview of the components

Besides sending back the world model to the clients, the server will also send some information about scenarios and will suggest some icons that can be placed next.

The scenario information consists of the predefined scenarios and the probability values that the server has added to them. The scenario information is there to give the users of the clients a quick idea of what is happening around them, and to make better predictions of expected icons.

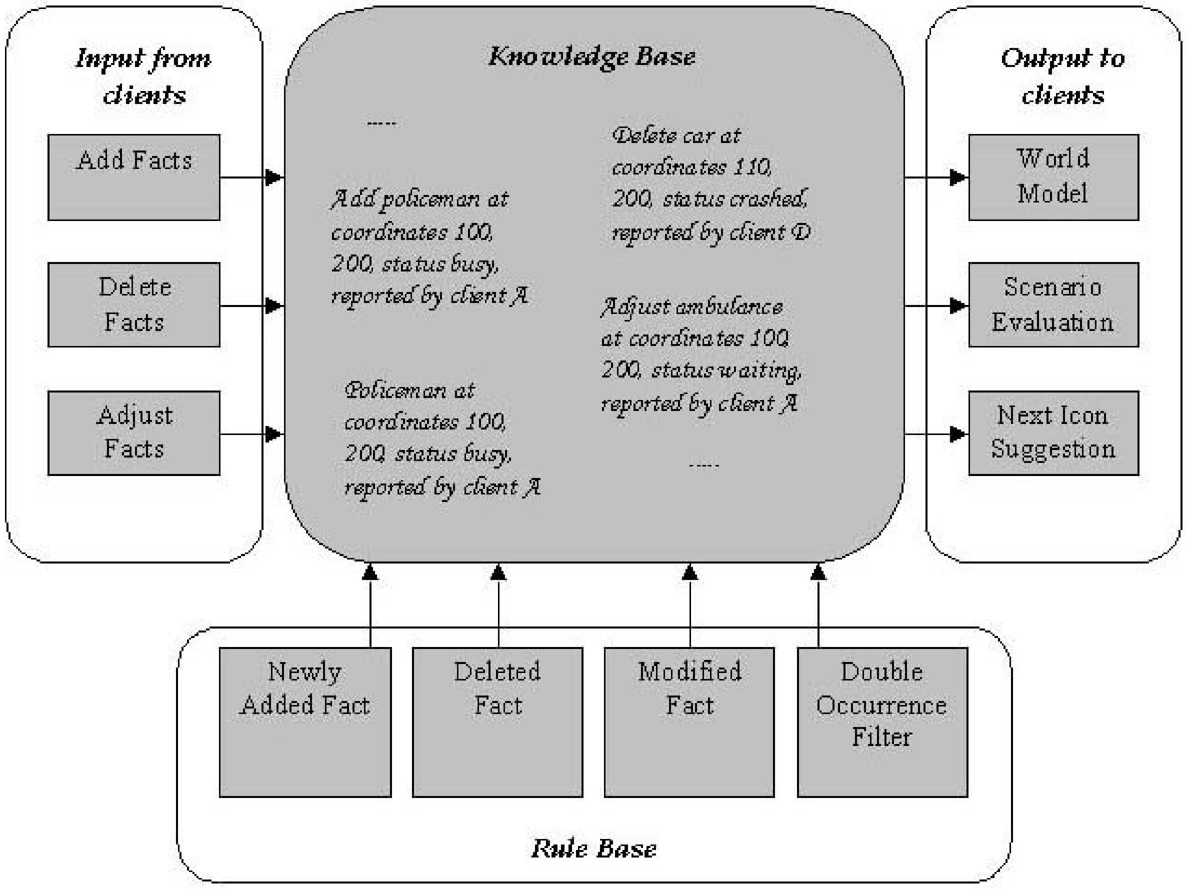
The suggestions for new icons are some feedback for the user. The server has some expectations of what icons it will be receiving next. If an icon is missing, the server will notice this by the information of the XML files. If a fireman is reported, but there is no fire truck, the system will be likely to suggest placing one. The suggestions for placing new icons come from both individual icon relations (icon:flames → icon:smoke) and relations between scenarios and icons (scenario:bomb scare → icon:bomb). Providing suggestions like these, we believe, will add to a more accurate report. Things the reporter might have missed, or did not find important enough to report will be more likely to be reported now. The user will be actively looking for the concept represented by the icon of the suggestion.

The Jess component will act like a blackboard where users can write and read messages on. The knowledge base is the blackboard and the rule base will constantly run functions on it to keep it consistent and correct. See Figure 8.

After the rule base is applied to the knowledge base the resulting world model is send to the clients, together with the scenario information and next icon suggestions. These last two outputs are acquired by performing a function on the knowledge base that calculates the values for the scenarios and suggestions.

The double occurrence filter is implemented to consider the locations of the different reporters. To combine two icons we take a weighted average of the reported locations, based on the distance of the reporter to the event. The closest report from the closest client is probably more accurate, so this report will have a higher weight. When multiple clients have reported the same icon, they are all added to a list of 'supporters' of the icon. This is to prevent deleting double icons that were reported by the same client. After all, if one client reports about two policemen, there probably are really two, so the filter should not apply in this case.

The output to the clients consists of three parts, the World Model is just the collection of placed (and processed) icons and their attributes and locations.


Input from clients
Add Facts
Delete Facts
Adjust Facts
Knowledge Base

Add policeman at coordinates 100, 200, status busy, reported by client A
Policeman at coordinates 100, 200, status busy, reported by client A
Delete car at coordinates 110, 200, status crashed, reported by client D
Adjust ambulance at coordinates 100, 200, status waiting, reported by client A

Output to clients
World Model
Scenario Evaluation
Next Icon Suggestion
Newly Added Fact
Deleted Fact
Modified Fact
Double Occurrence Filter
Rule Base

Figure 8 The Jess Component as Virtual Blackboard

The Scenario Evaluation is a suggestion for the most likely scenario. This gets calculated by letting each placed icon award points to certain scenarios. The scenario that ends up with the highest score is the most likely scenario.

The Next Icon Suggestions are based on both the scenario information and the inter icon relations. The scenarios award points to each icon that could be in the scenario, but is not placed yet. These points are weighted by the likeliness of the scenario. If a scenario is most likely not right, it will not add many points to the missing icons. The inter icon relations are direct relations, such as flames→ smoke. All the placed icons will award points to their directly related icons if they are not placed already.

The awarded points for scenarios and icons are all defined in the XML files. Changing these would result in different a different scenario and icon suggestion.

4 User Test

It is important to test the system, in order to detect errors and unexpected or unwanted behaviour, it can also give some conclusions of the ease of use and the complexity of the system. We have designed a test that will allow us to test the *usability*, the *correctness*, and the *completeness* of the system at the same time. The usability will determine if the system is easy to use, the correctness if the algorithms work and the completeness to determine if we have enough icons to express ourselves.

4.1 Design of the test

There are two users, the respondents, that will test the system simultaneously, using a scenario presented to them in the form of photographs. The respondents are both not experienced in reporting events, let alone reporting about emergency situations. We have taken a new scenario for this test, as opposed to using one we used to determine the Jess rules. We have done this to make the test less trivial, after all it wouldn't be much of a challenge to predict which scenario is going on, if the test case is exactly what the rules are derived from.

Because the system can not be tested in the real world, the respondents are presented photographs of the scenario. A disadvantage of this is that it is very hard to determine your position if you can't look around freely to orientate on the situation. That's why we have added letters of the corresponding photograph on the map, accompanied by an arrow that represents the viewing direction. Because in a real situation the reporters will not both be reporting from the same location, they will be given different pictures.

During the test the respondents are asked to think out loud, telling what they are thinking and what they are trying to accomplish. This will allow us to determine if specific tasks need to be made more intuitive, or need more functionality. In terms of the overview in Figure 1 the photographs represent the Real World, the explanation of what the respondent sees and tries to accomplish is the Mental World Model, and the Structured World Model is what the respondent actually reports using the system.

4.2 Test results

Respondent 1 was shown the pictures in Figure 9. She started with picture A and decided to report a fire truck and a fireman. She found the corresponding icons without any trouble and placed them on the map. She then decided that she could not place them at the correct location very well because she had forgotten to zoom in first. She deleted the icons, then zoomed in a few times to the desired resolution and placed the icons again. She decided that the woman on the picture was not important to report about because she didn't seem to be a wounded victim. She didn't notice any other events on the picture and moved on to picture B. She started with a fireman icon and selected with the attributes there are 3-5 fireman. After that she reported flames, small in size, with medium intensity. As a next step she decided to send the information to the server. The server made some suggestions for next icons, and the respondent decided that the smoke suggestion was actually an event she had overlooked. After placing it on the map she sent the report again, and was satisfied with her current report. The final results are shown in Figure 10.

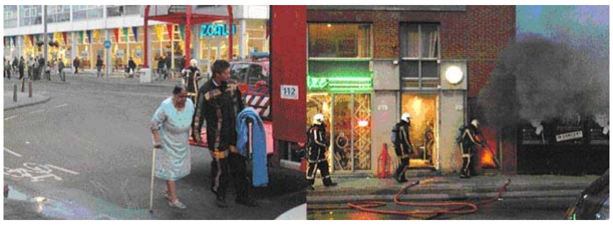


Figure 9 Photographs A and B

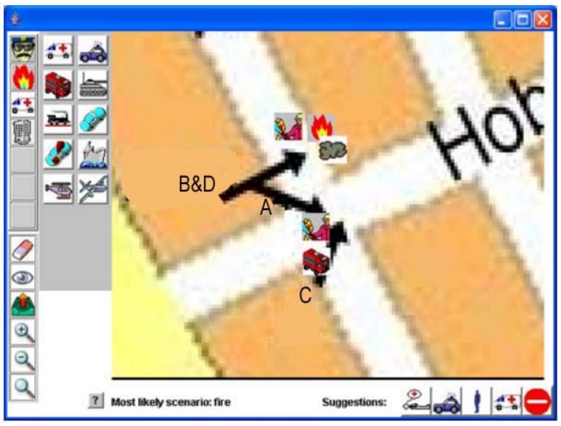


Figure 10 Report by respondent 1

Respondent 2 was presented photographs A and B, in Figure 11. He started with picture C and decided to zoom in on the location first. As opposed to respondent 1 he used the drag zoom tool, which allowed him to see the picture at the highest resolution at once. He then placed a fire truck and 3-5 firemen. He wasn't sure if the light in the window were flames or just plain illumination. He decided to not report fire there and wished he could have seen the building from a shorter distance. Respondent 2 continued with picture D and reported 3-5 firemen, a building, smoke and fire. The result of the report is shown in Figure 12.

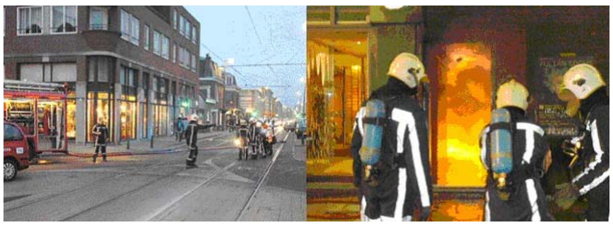


Figure 11 Photographs C and D

Respondent 2 pressed the send button and the two world models were fused by the server. Both respondents got a new map from the server with the new world model, which can be seen in Figure 13.

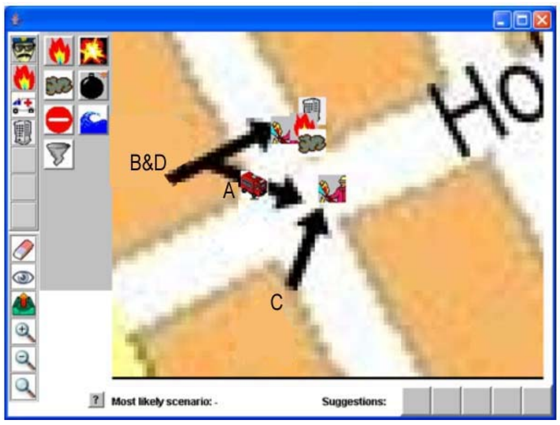


Figure 12 Report by respondent 2

Both respondent 1 and 2 were not surprised by the new world model. They already expected that some icons would have been moved a bit to take a weighted average of the two reports. Respondent 1 noticed that she had forgotten to put a building icon on the map at first.

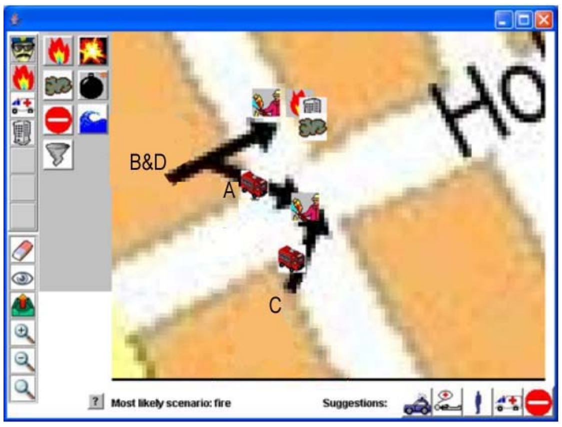


Figure 13 Result of the fusion

4.3 Test Conclusions

The conclusions that can be made from the test will be divided in the three aspects we tried to test. First of all the *usability* of the system. The zooming in and out of the map was used by both testers and was very helpful. The suggestions that were coming from the server where helpful as well. The overall opinions of both tester is that the system can be used quite easily, although some training in what to report would be welcomed. Respondent 1 decided that the person on photograph A was not relevant enough to the situation to be reported. From this can be concluded that even though we are structuring the mental world model of the reporter, there is still some room for an own opinion. To get all reporters to report what they should, some training should be given, and agreements on what to report should be made.

The *correctness* of the system seemed to be ok. Of course the respondents didn't know exactly how the two reported world models should have been fused, but they did not notice any strange modifications of their own model. Some icons were added, but that's exactly what they would expect, since they were not the only one reporting. The suggestions that the system made were understandable, although most of the suggested events were not seen on the photographs. This is because most of what could be seen on the pictures was reported about already. Finally the scenario suggestion, a fire, was obviously correct as well.

The *completeness* of the system looked to be sufficient. Neither of the respondents wanted to report anything that was not represented by an icon, and the icons they needed were found very quickly. Respondent 2 was not sure if he saw a fire on picture C. It would be nice if he could report about this, by adding an uncertainty factor to his report.

5 Conclusions and Recommendations

In this section we will evaluate to what extend the project goals are achieved. We will do so by discussing each point of the problem description.

1) We made an interface which allows to use icons, and place them on a map.
2) The system is expressive in a sense that there are a lot of concepts that can be reported about. Furthermore the icons that are used to represent the concepts can be given attributes to add more information. The complexity of the system can be further increased by extending the XML files. Unexpected situations would be situations in which new icons are needed that were not implemented yet. This can be done by adjusting the XML files to add the concept and its relation to other concepts
3) It is easy to select the right icon because the icons are distributed over logical icon groups, which can be altered if needed by adjusting the XML files. To provide extra information, an attribute window will pop up where the values of the attributes can be given. The values can be selected out of a small list, this decreases the chance of making a wrong selection, and eliminates the chance to make an illegal selection. When icons are placed, the user is able to delete them again, or to inspect or alter its attributes. To prevent placing icons on the wrong location, the user can easily zoom in and out of the map, to be able to place the icon exactly where it should be. According to the

respondents of the user test, it is an easy to use system.

4) In order to assemble and maintain a world model we collect all information at one server. The Jess component in the server is responsible for keeping the world model up-to-date. Because the server only has 1 world model that gets adjusted over time, the clients will all be send the same information, which is always the newest. During the user test, both users were sending information, and the server fused their world models in a correct new one.
5) To detect missing icons, the system looks at the already placed icons and the possible scenario. From this it gives suggestions to the user, rather than adding icons autonomously. The user can then decide if the suggested icon should be placed or not. Double and wrongly placed icons are detected by the systems double icon filter. When two or more of the same icons are placed very closely to each other, the system combines them, as long as they were reported by different clients. When multiple reports of the same icon are made, the system will take a weighted average of the icons location, and thus incorrectly placed icons will be placed on a better position. If a client reports an icon that is too far from its correct location, and out of the filters range, it cannot be detected.
6) New concepts can easily be added or adjusted by altering the XML files. The relations between the icons can also be adjusted in this way. Adjusting these will result in the system to give other icon suggestions or scenario overviews. There are however still some hard coded functions that are not dynamical, such as the double icon filter.

Concluding we can say that all the goals, as stated in the problem description are met. However, there are still many things we would like to see done in a different or more elaborate way:

- Extending the system by non human observers: To extend the system we could add some non human observers, such as smart cameras, burglar and fire alarms.
- Improving the network: The current network is very vulnerable when the server goes down. We could improve this by distributing the information over several servers, or to make an ad-hoc network of only handheld computers. [Kla05]
- Adding GPS or other localisation techniques: Global Positioning System provides ways to exactly define your location. On start up the system could place the client on the map and even show his viewing direction. There is some security issue involved however. GPS can be intercepted, and positions of the clients can be revealed in this way. Unless there is a good way to prevent this from happening we might need other ways for the clients to locate each other. Some research has been done already about localization without GPS. [Bul02], [Vel05]
- Expanding and improving the intelligence: Room for improvement lies in the information we are using. When there is a lot of information available about scenarios we could improve the information in the XML files to provide for more realistic calculations of the scenarios and icon suggestions. [Cha05]
- Intelligence over time: This can be achieved by keeping track of a history, so there can be reasoned about icons that may have disappeared already. Think about an explosion for instance, this will be only there for a very short amount of time, but does have an impact on what can be expected next. The time of arrival of the icons can also be important in certain scenarios.
- Anticipation on moving objects: Things like cars and persons will move over time. If we can get information about what direction they will be moving we can reason about their location better.
- Giving the clients different roles: We could let certain roles constrain to only a limited set of icons. For example someone who defuses bombs could provide more information about the status of a bomb then an ambulance driver. This could lead to a situation where experts can provide attributes of an icon, where others can not.
- From causal relations to probabilistic relations: The current reasoning is based on causal relations. If there is smoke there is fire. In some cases causal relations are not easy to see through, that’s when it could be convenient to use probabilistic relations. With Bayesian Belief Networks it can be calculated what events are likely to happen next, given some events and the probabilities that they happened.
- Security issues: To prevent messages from being intercepted and read by unauthorized people, we can use cryptography [Ove00]. A suggestion is to give every client and the server a public key and a private (secret) key. The messages can then be encoded using a asymmetric cipher algorithm and a digital signature. We can first encode the message with the public key of the server, and then code the result with the clients secret key. The server then has to use the clients public key first, and knows the message is really coming from the client. After that it decodes the message using its own secret key, to actually read the original message.

Bibliography

[Bea94] *Iconic Communication* , C. Beardon, 1994. In: *Intelligent Tutoring Media*, 5(2), pp.58-62. ISSN 0957-9133

[Bul02] *Scalable, Ad Hoc Deployable RF-based Localization*, Nirupama Bulusu, Vladimir Bychkovskiy, Deborah Estrin and John Heidemann, October 2002, University of California at Los Angeles

[C2000] The C2000 system, designed in order of the Dutch government, see www.c2000.nl

[Cha02] *Semiotics the Basics*, Daniel Chandler, 2002. Or for a similar online version see http://www.aber.ac.uk/media/Documents/S4B/

[Cha05] MSc Thesis of Jan Chau, still under construction at this time, Delft University of Technology

[Dor94] *Self-Explaining Icons*, Claire Dormann, 1994, In: *Intelligent Tutoring Media*. Vol 5, No 2. 1994. pp. 81-85

[JADE] Java Agent DEvelopment Framework, see http://jade.tilab.com/

[JESS] Java Expert System Shell, see http://herzberg.ca.sandia.gov/jess/

[Jon96] *DynamIcons as Dynamic Graphic Interfaces: Interpreting the Meaning of a Visual Representation,* D.H. Jonassen, R. Goldman-Segal, H. Maurer In: *Intelligent Tutoring Media* , vol. 6 (3/4) (1996), 149-158

[Kla05] MSc Thesis of Paul Klapwijk, still under construction at this time, Delft University of Technology

[Mea91] A Computer-based Iconic Language, S.Mealing & M. Yazdani, 1991 In: *Intelligent Tutoring Media*, 1(3):133-136, 1992

[Mea94] *A Computer Hinterface,* S. Mealing, 1994, see http://www.intellectbooks.com/iconic/hint/hint.htm

[Nrc03] *Met 61 woorden de wereld rond,* about the theory of Anna Wierzbicka, NRC Handelsblad 20 september 2003.

[Ove00] *Informatiebeveiliging onder controle*, Paul Overbeek, Edo Roos Lindgreen, Marcel Spruit, 2000, , Pearson Education

[Ric94] *The Use of Metaphors in Iconic Interface Design*, Stephen Richards, Philip Barker, Ashok Banerji, Charles Lamont and Karim Manji. In: *Intelligent Tutoring Media,* Vol 5, No 2, 73-80, 1994

[Shn98] *Designing the User Interface, Strategies for Effective Human-Computer Interaction,* Ben Shneiderman, 1998, Addison Wesley Professional.

[Tat03] *Iconic Communication*, Iulia Tatomir, December 2003

[Vel05] MSc Thesis of Marcel van Velden, still under construction at this time, Delft University of Technology