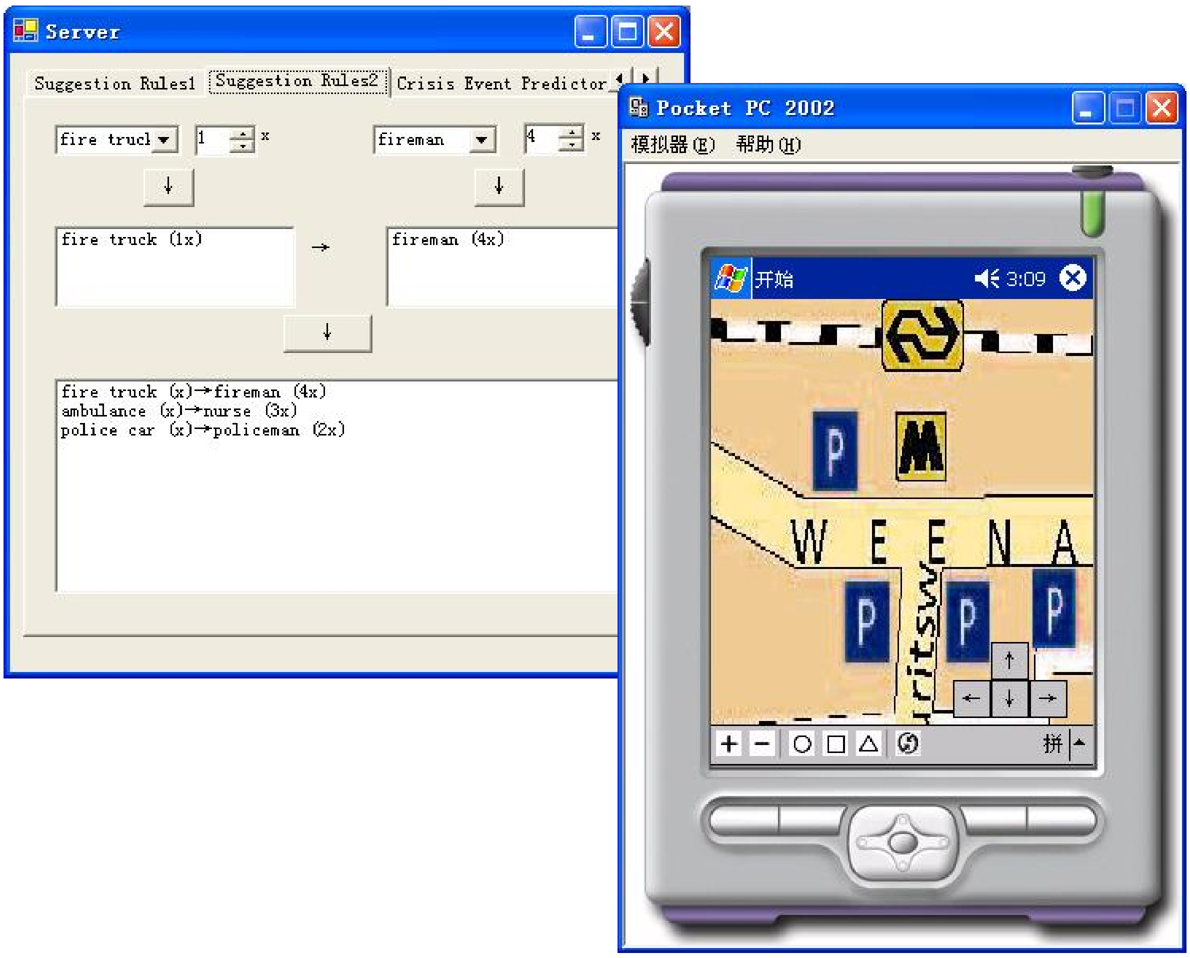
CRAS

Crisis Reporting and Analysis System



```
fire truck (x)→fireman (4x)
ambulance (x)→nurse (3x)
police car (x)→policeman (2x)
```

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

Delft University of Technology

Master Thesis of:
Xiao Ma
Student number: 1190938
Date: August 2005

Abstract

Since the terrorist attacks on September 11th 2001 the issue about the collaboration of emergency services has become increasingly important. In crisis events, such as natural disasters, paroxysmal accidents, and acts of terrorism, a global infrastructure breakdown is inevitable. Therefore, a more mobile and flexible device to do better communication between the crisis scene and emergency services is needed.

Digital handheld devices, such as Personal Digital Assistant (PDA), are becoming more affordable and commonplace in the workplace. As devices become smaller, modes of interaction other than keyboard and stylus are a necessity. In particular, small handheld devices like cell phones and PDAs serve many functions and contain sufficient processing power to handle a variety of tasks. Present and future devices will greatly benefit from the use of multimodal access methods. Some specialities of PDA, such as instant communication and portability, can be used in crisis situation.

The research aims at designing and implementing a prototype for a system which 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 server should be intelligent enough and based on a knowledge base. Furthermore the system should be dynamic in the sense that new concepts and rules can easily be added. The system is implemented in Visual Studio .NET 2003.

Keywords: PDA, wireless, server, interface, map, world model, knowledge base, crisis, Visual Studio .NET

Preface

This project started in March, 2005, and ended in August, 2005. It is conducted at 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 focused on the research of decision support systems, seeking to order information in complex and chaotic situations.

The original motivation of this project is to continue Paul Schooneman's ISME project. After analysis his project, we decided to develop a new crisis reporting and analysis system in which observers can report crisis situation by clicking and dragging in a map at PDA device and server can analyze everything based on knowledge base. Then in the following months, we designed and developed a prototype for the system.

Acknowledgements

First of all I am very grateful to my supervisor Leon Rothkrantz. He helped me a lot from the very beginning to the end of this project. What I learned from his suggestions are not only on how to design, but also on how to think, how to analyze, and how to present. I believe that what I learned from my supervisors will make me perform better in my future.

Next I would like to thank my advisor Charles van der Mast for giving me useful information. Also, I will thank my fellow students for giving me useful input and assistance. They are, in no particular order: Siska Fitrianie, Dragos Datcu, Bogdan Tatomir and Paul Klapwijk. They are always kind when I need help.

Last, I will thank all my friend in China and Netherlands. They support me to fight with every trouble and give me great confidence.

Table of Contents

Chapter 1: Introduction

1.1 Background

Digital handheld devices, such as Personal Digital Assistant (PDA), are becoming more affordable and commonplace in the workplace. As devices become smaller, modes of interaction other than keyboard and stylus are a necessity. In particular, small handheld devices like cell phones and PDAs serve many functions and contain sufficient processing power to handle a variety of tasks. Present and future devices will greatly benefit from the use of multimodal access methods. Some specialities of PDA, such as instant communication and portability, can be used in crisis situation. Since the terrorist attacks on September 11th 2001 the issue about the collaboration of emergency services has become increasingly important. In crisis events, such as natural disasters, paroxysmal accidents, and acts of terrorism, a global infrastructure breakdown is inevitable. Therefore, a more mobile and flexible device to do better communication between the crisis scene and emergency services is needed. Despites its applicability, the user interaction options for a PDA are quite limited. User interaction with a PDA can be realized by a pointing device. Most PDAs only contain a few physical buttons and a few of them have a small-sized keyboard. Most PDAs only contain a few physical buttons and a few of them have a small-sized keyboard. Recent researches have been done in adding speech input on a PDA. However, the current technology makes the speech input less suitable for mobile activities.

When crisis happend, reports from all parties involved in a crisis event, such as rescue teams, witness, victims, families, etc, are essential to have a clear description of the situation for effective problem solving and preventing further damages. To avoid creating language problems, we should select a representation that offers a potential across language barriers. Icons and other concepts from semiotics are selected for this type of representations.

Icons have been used as early as the middle ages complex iconic system such as the heraldic coats of arms, astrological signs, and the communication system of ancient Egyptian. In modern society, icons are familiar from the everyday context of living to the packaging for the last products, for example, door signs, road signs, electronic goods manual, etc. In the computer world, they have role as a small graphical representation of a program, resource, state, option or window. An icon is understood as a representation of a concept, i.e. object, action, or relation. By virtue of a resemblance between an icon and the object or the movement it stands for, the icon functions as communication means. Icons also offer a direct method for conversion to other modalities. The meaning of individual icons can represent a word or a phrase, which is created according to the metaphors appropriate for the context of this type of languages. Once a set of iconic representation is established, increased usages can lead to more stylized and ultimately abstract representation, as has occurred in the evolution of writing systems, e.g. the Chinese fonts. As icons offer a potential across language barriers, the icon language can also offer a potential as an independent human natural language like Esperanto. Thereby, any interaction using this type of languages opens opportunities for the development of different applications in different languages and domains.

A recent research has been done in developing computer-based iconic communication on PDAs applied for communication in crisis situation by Paul Schooneman. An observer can report about a crisis stuation by placing icons on a map where the event occurs. The iconic interface creates a scenario that describes situations on the map and sends it to a collaborative information system for further processing. However, there are some limits when observers use the developed iconic interface to report crisis situation, such as:

1. The provided icon amount is limited. They may not be able to describe a crisis situation (especially for a complex one) exactly and thoroughly.
2. The need of users to emphasis a situation is not provided.
3. The need of users to redirect others on a situation on a certain location is not provided.
4. The need of users to ask for more information is not provided.
5. The need of users to prioritize a situation on a certain location is not provided.

Therefore, we need more features on the iconic interface to fulfill these requirements. Some people are used to describe their observations by sketching on a piece of paper to support their verbal explanation. The interaction between user and computer and the communication of human beings have nearly nothing in common. Human beings e.g. use sketches to communicate with other persons. This special kind of communication is used since the Stone Age to visualize space-relevant features like water supplies near a given location. Today they are still used to highlight spatial relationships. The human being is accustomed to use them. To overcome the depicted problems an extension to traditional interaction metaphors can be used. Especially, a graph- and sketch-based interaction metaphor can be applied. The processing of flexible user inputs is helping to realize a more successful communication between man and computer. More directiveness (less difference between the goals of various users, their expectations, proceedings and the user interface) is supported. Bringing this idea to an iconic interface, we will be able to develop more features on the iconic interface. Because it is not easy to analysis and fuse sketch in server, and sketch can be splited to some geometrical element, now we can use some geometrical icons to replace icon string. Besides placing icons on a map where the event occurs to report about a crisis situation, observers can also input text because geometrical icons are no semantic. For example, the observer can put a triangle icon onto the map and input "terrorist" as its meaning. That means that we can use more combination icons between geometrical icons and text to describe crisis event in detail.

1.2 Problem Description

First of all, we should have a overview about the project. Figure 1 is a basic concept about how the system works in real world. Now we will talk about this one by one.

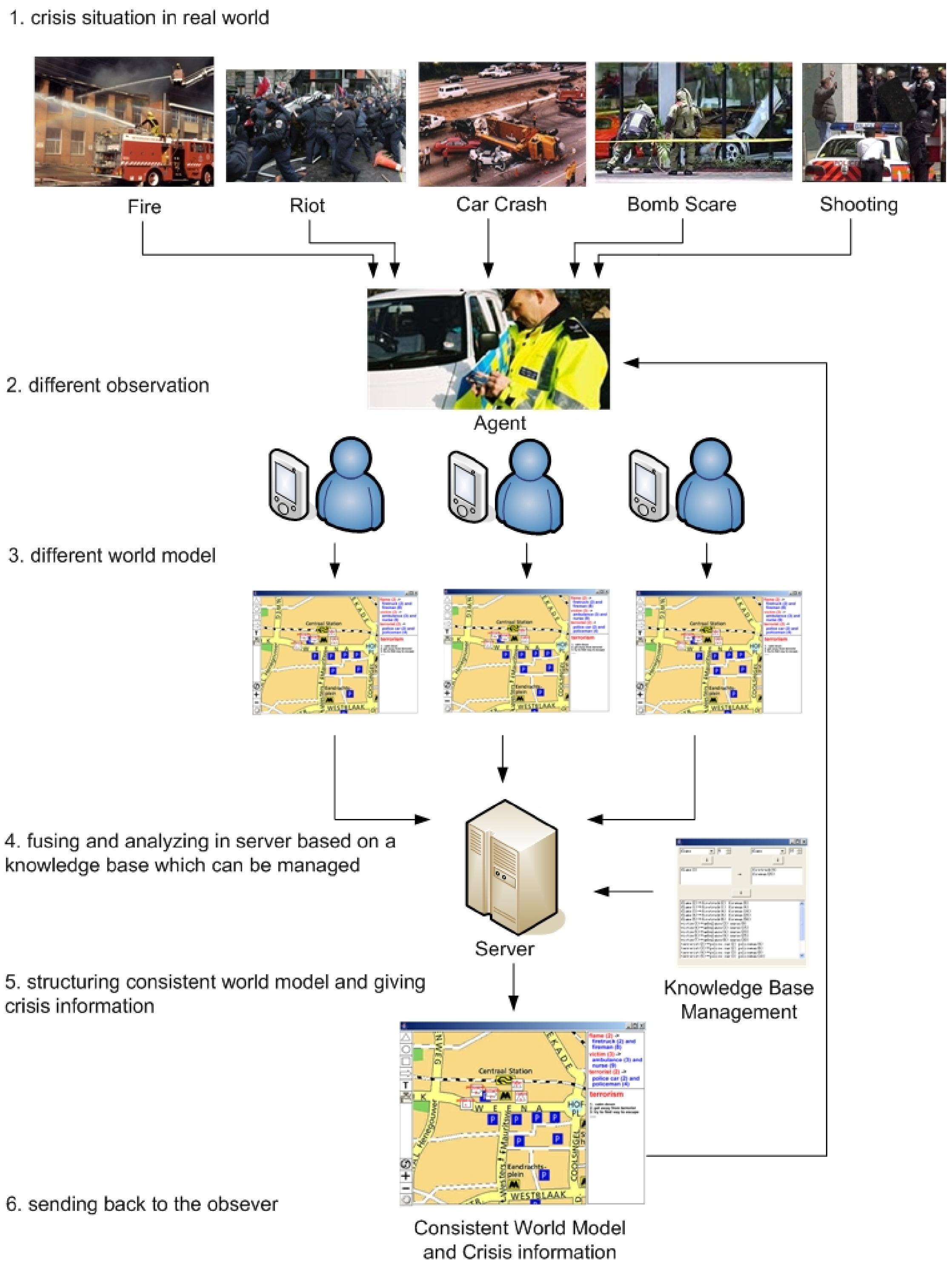

1. crisis situation in real world
Fire
Riot
Car Crash
Bomb Scare
Shooting
2. different observation
Agent
3. different world model
4. fusing and analyzing in server based on a knowledge base which can be managed
Server
Knowledge Base Management
5. structuring consistent world model and giving crisis information
6. sending back to the obsever
Consistent World Model and Crisis information

Figure 1. Project Overview

a. There are always some crisis situations in the world everyday, such as fire, riot, car crash, and bomb scare and shooting. How to improve efficiency of the rescue team? How to save more victims and families in short time? The most important thing is how to get overall situation in short time. If everyone in crisis scene know overall situation, included rescuers, witness and victims, it is good for them to establish rescue plan and escape way. So we need every observer to report their situation in different attribute and position.

b. In crisis scene, there are many different rescue teams (i.e. the fire-fighters, ambulance services, police, the military, and other crisis management organizations). Depending on their different professional attribute, they have different views of the world. 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.

c. Every observer has his own World Model. A World Model is composed of objects, characteristic features of the objects, and relations between the objects. The agent in crisis scene observes what happens in the real world and forms his own Mental World Model of the situation. Then they need to report their thoughts with the report tool of the system. It should be a interface in digital handheld devices, such as PDA. That is one of my thesis projects. The tool will only be able to handle structured information; concepts represented by icons, text and sketch. Thus the reporter has to concretize his ideas in sketch, that he can then place geometrical icons and text attribute on the map to describe his situation. Then the Structured World Model will be send to a central server, which collects reports of all the agents in the field.

d. The server will fuse and analyze all the information from different observers. It is a intelligent agent base on knowledge base which can be managed.

e. By collaborating with all received information, these reports can be filtered to have consistent and complete information. Then the system will form its own structured consistent world model. Server also can give suggestion and crisis information depends on knowledge base.

f. The consistent world model and crisis information will be sent back to the agents, along with suggestions that the agent might have forgotten to report. The agent will depend on these consistent world model and crisis information to make rescue plan.

The research aims at designing and implementing 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. We expect the user can have more natural interaction by these feature. The geometrical sketch function should simulate with windows paint which is familiar to most people. The text input function should be related to the PDA writing software. You can choose the on-screen keyboard to tap in the letters and numbers, or write on screen directly. The server should be intelligent enough to assemble and maintain a correct and up to date world model. Furthermore the system should be dynamic in the sense that new concepts and rules can easily be added.

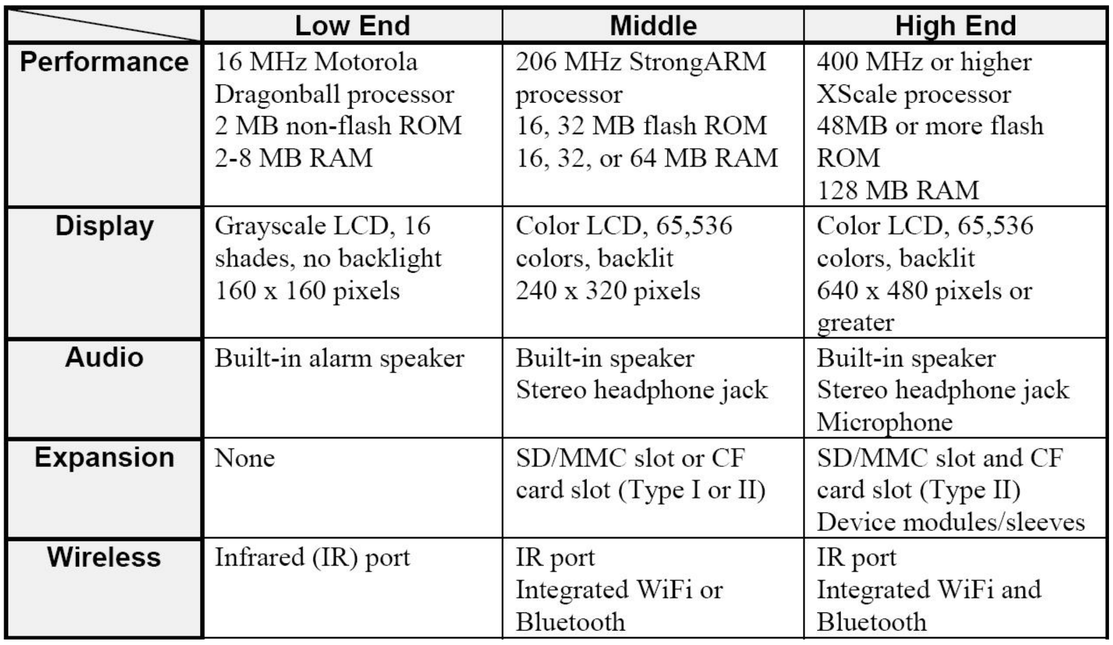
Chapter 2: Literature Survey

2.1 About PDA

Digital handheld devices, such as Personal Digital Assistants (PDAs), are becoming more affordable and commonplace in the workplace. They provide highly mobile data storage in addition to computational and networking capabilities for managing appointments and contact information, reviewing documents, communicating via electronic mail, and performing other tasks. Individuals can store and process personal and sensitive information independently of a desktop or notebook computer, and optionally synchronize the results at some later time. As digital technology evolves, the capabilities of these devices also continues to improve rapidly, taking advantage of new forms of removable media, faster processors that consume less power, touch screens with higher pixel resolution, and other components designed specifically for mobile devices. When handheld devices are involved in a crime or other incident, forensic examiners require tools that allow the proper retrieval and speedy examination of information present on the device. This report gives an overview of current forensic software, designed for acquisition, analysis, reporting of data discovered on PDAs, and an understanding of their capabilities and limitations.

PDA differs in several important ways compared with personal computers (PC). For example, PDA is designed for mobility, hence compact in size and battery powered; they store user data in volatile memory instead of a hard disk; and they hibernate, suspending processes when powered off, to avoid a time-consuming reboot when powered on again. Most types of PDA have comparable features and capabilities. They house a microprocessor; flash read only memory (ROM), random access memory (RAM), a variety of hardware keys and interfaces, and a touch sensitive, liquid crystal display. RAM, which normally contains user data, is kept active by batteries whose failure or exhaustion causes all information to be lost. Compact Flash (CF) and Secure Digital (SD)/MultiMedia slots support memory cards and peripherals, such as wireless communication cards. The latest high-end PDA is equipped with fast processors and considerable memory capacity, giving the user performance comparable to a desktop machine from only a decade ago. Moreover, PDA capabilities are sometimes combined with those of other devices such as cell phones, global positioning systems (GPS), and cameras to form new types of hybrid devices.

The table illustrates the range of hardware components found in present-day pure PDA devices.



	Low End	Middle	High End
Performance	16 MHz Motorola Dragonball processor 2 MB non-flash ROM 2-8 MB RAM	206 MHz StrongARM processor 16, 32 MB flash ROM 16, 32, or 64 MB RAM	400 MHz or higher XScale processor 48MB or more flash ROM 128 MB RAM
Display	Grayscale LCD, 16 shades, no backlight 160 x 160 pixels	Color LCD, 65,536 colors, backlit 240 x 320 pixels	Color LCD, 65,536 colors, backlit 640 x 480 pixels or greater
Audio	Built-in alarm speaker	Built-in speaker Stereo headphone jack	Built-in speaker Stereo headphone jack Microphone
Expansion	None	SD/MMC slot or CF card slot (Type I or II)	SD/MMC slot and CF card slot (Type II) Device modules/sleeves
Wireless	Infrared (IR) port	IR port Integrated WiFi or Bluetooth	IR port Integrated WiFi and Bluetooth

The four dominant operating systems for PDA/SmartPhone:
-Windows Mobile 2003 (Microsoft)
-Palm OS (PalmOne)
-Symbian (Nokia)
-Linux (Other)

There are also some disadvantages about PDA, but they will be absolutely solved in future by technical impetus:
-Low processing performance (Moore's Law)
-Limited storage (network distributed storage)
-Low battery capability (more and more charge point)
-Comfortless input/output (multimodal interaction)

2.2 About Wireless

There is no doubt the world is going wireless faster and more broadly than anyone might have expected. Intel demonstrates this new reality and predicts that billions of people will gain high-speed Internet access wirelessly within the next decade. The premise for this vision is clear: all high-speed wireless technologies (3G, Wi-Fi*, WiMAX and Ultra- Wideband) will coexist, working in tandem to meet service provider and customer needs for truly mobile computing and communications across the globe. No single technology will become dominant or ubiquitous they all meet unique user requirements in a wirelessly connected world. In fact, the most robust wireless solutions will use a combination of technologies to enable increased mobility and eventually seamless roaming. Intel wholeheartedly embraces the march to wireless, and is enabling the revolution through industry leadership, technology development, and new silicon products. From Intel Centrino mobile technology to WiMAX technology to Intel PXA processors, you will find Intel at the forefront of the broadband wireless revolution.

As computing and communications converge on broadband wireless platforms and technologies, demand will soar for true mobility. This is the always best-connected goal, where broadband technologies such as 3G, UWB, Wi-Fi and WiMAX will work synergistically to deliver secure data with anytime, anywhere connectivity. These overlapping wireless networks will offer users choices for the best possible connection. In fact, the mobility enabled by wireless technology necessitates overlap between networks and co-existence among technologies - wired and wireless. For the always best connected goal, we can think about the following scenario:

- All types of wireless networks will be deployed around the globe.
- Wi-Fi hotspots will proliferate in public places, businesses and homes.
- Homes and businesses will add UWB (when available) for the fastest distribution of high-definition content.
- First-generation WiMAX technology will be broadly deployed to provide long distance broadband connectivity for Wi-Fi hotspots, as well as cellular and enterprise backhaul.
- Later, 802.16e WiMAX connectivity will be added in densely populated areas to provide a canopy of wireless broadband data access to mobile laptop users.
- Innovations in 3G technologies will add groundbreaking data capabilities to mobile handset and handheld PC users.

What remains is for the entire industry to embrace the broadband wireless vision-coexisting wireless technologies and standards based modular platforms - delivering all solutions with an eye toward high-speed global connectivity.

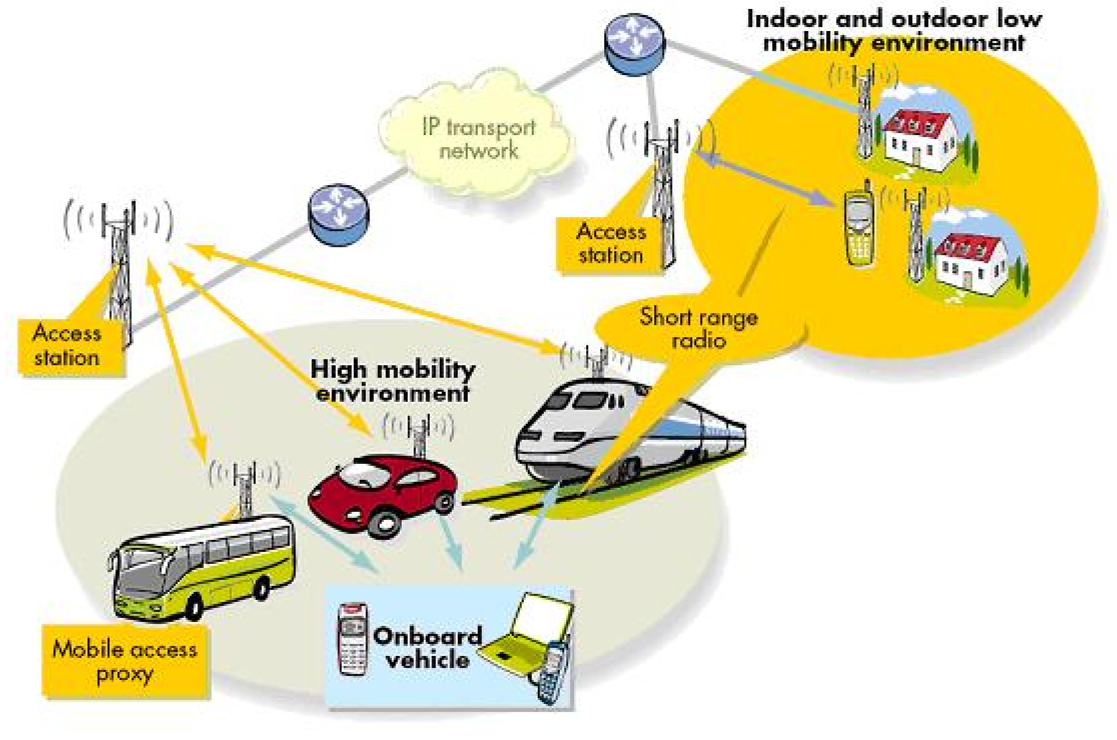


Figure 2. Wireless Solution

2.3 Icon based System for Managing Emergencies

A closely related project involving icons and emergency situations was done by Paul Schooneman. ISME gives the users the possibility to report about what they observe by placing icons on a map. The maps will be sent 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.

They 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. They 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.

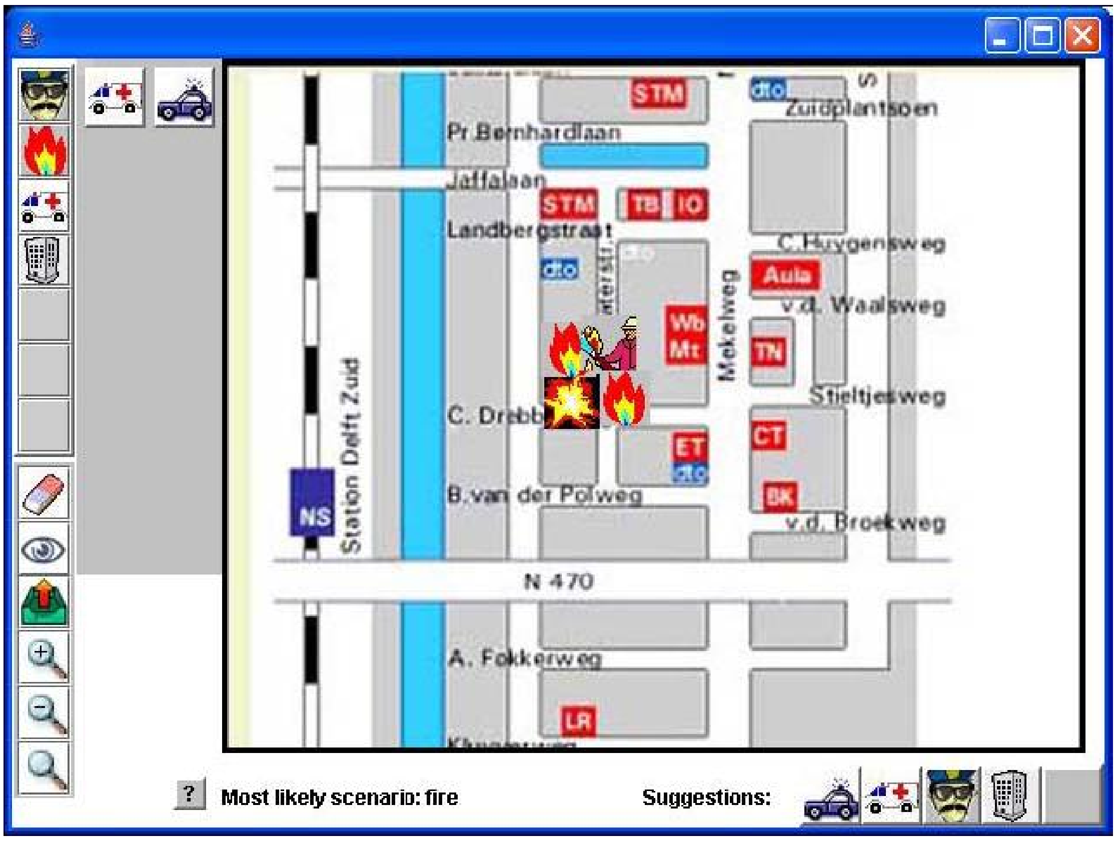


Figure 3. ISME Interface

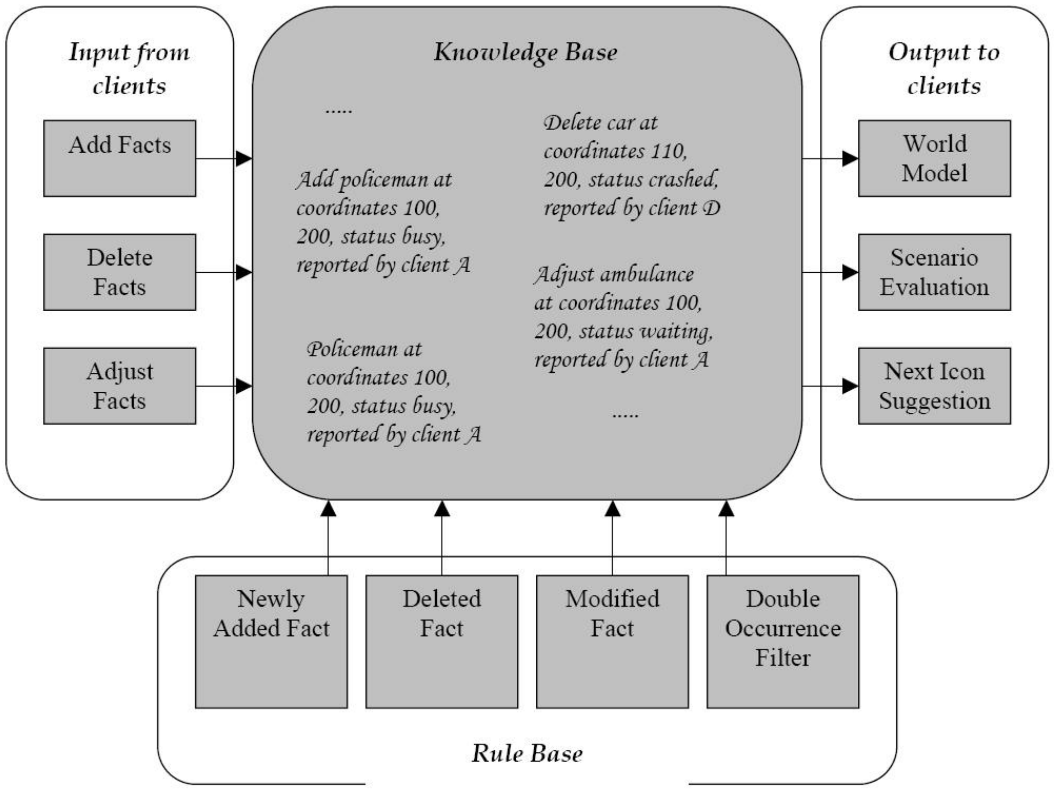

Input from clients
Add Facts
Delete Facts
Adjust Facts
Knowledge Base
.....
Add policeman at coordinates 100, 200, status busy, reported by client A
Policeman at coordinates 100, 200, status busy, reported by client A
Delete car at coordinates 110, 200, status crashed, reported by client D
Adjust ambulance 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 4. ISME Server

2.3.1 Define a World Model

To define a World Model they 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 they encountered during the project is that the emergency services are very conservative with providing information. The information they have used is therefore for a big part originating from emergency scenarios as described in news articles. Although they 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.

2.3.2 Interface

They will be communicating by the means of placing icons on a map. Since they are developing it for crisis situations they 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.

2.3.3 Intelligence

They 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, they 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.

2.3.4 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. They don't want a system with only a central server, because they could lose all information when it goes down. Therefore they 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 are 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.

Chapter 3: Requirement Analysis

3.1 General Requirements

Design and implement a prototype for a system which 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.

Now we will take the problem description to split up the different requirements into workable components.

1. The client is about reporting crisis events and based on a GUI. We use geometrical icons and text to represent concepts in a crisis, and maps of the surroundings to place the icons and text on.
2. To make the system expressive enough we will need different geometrical icons for crisis events, rescue teams and scenarios. Within these geometrical icons are no semantic. They needed more specific information. we will need a way to add information for those geometrical icons. That's why the text is necessary.
3. To prevent a lot of errors, the GUI should be intuitive and easy to use. It should be clear which icon is selected and they should be added to the map with just clicking on the location. When icons are placed, the user should be able to delete them again. Text input is same as the icons.
4. In order to assemble and maintain a world model we will collect all information at one server. 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 different reporting we need a intelligent agent, that constantly works on the information that's being gathered. The agent who depends on different rules from knowledge base will give suggestion and related crisis information to the client.
6. In order to make the system dynamic, it's useful to store all information about crisis event and rules in knowledge base. We need a interface to manage this knowledge base. The entries of this database should be easy to edit, and the database should be extendible.

3.2 Device Constraints

The Emulator is a tool that mimics the behavior of hardware that supports a Microsoft Windows CE .NET–based platform. With the Emulator, we can design and build a Windows CE–based platform and test it using software that mimics hardware rather than testing the platform on hardware.

Architecturally, the Emulator is software that provides the equivalent of a desktop computer in a window on your development workstation. The software imitates the behavior of elements commonly found in a desktop computer, such as a PCI bus, video card, and sound card.

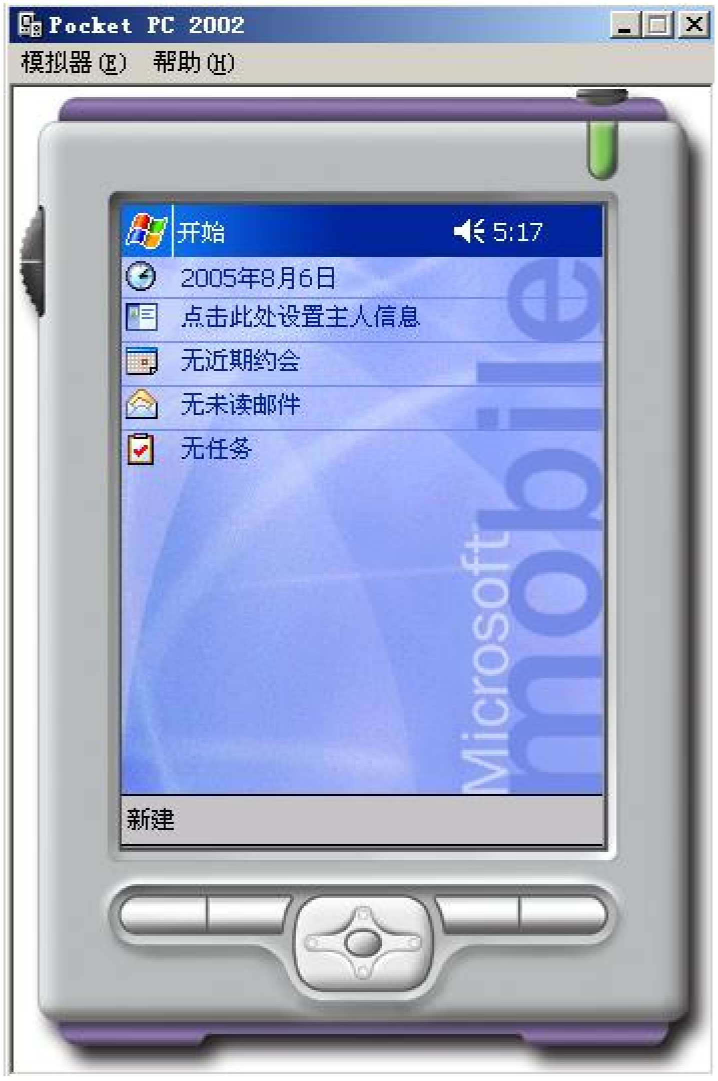


Figure 5. Emulator

The Emulator does not mimic hardware performance. The performance of the Emulator depends on the speed of our CPU and the amount of system memory in our development workstation. We can encounter slower performance with the Emulator than with an actual hardware device running the same application.

3.3 Target User Analysis

Every system belongs to specifical target user. Before we start to implement, it is necessary to define different function and model for different user characteristics first. About this crisis event reporting client, the people in the target group is in following:

1. Witness. Maybe they have no crisis experience before. So they need server to feedback some information and suggestion about crisis event. These informations will help them deal with the crisis.

2. Rescue teams. These people all have professional background, such as, policeman, fireman and ambulanceman. They always can report crisis situation duly and truly.

3.4 Function Requirements

3.4.1 Overview

In Figure 6 an overview of how the different components interact is shown. The whole system included three main parts. They are Client, Server and Knowledge Base Management. We will implement these in Microsoft Visual Studio .net 2003. The Client and the KBM will be done in C#. The Server part will be done in ASP.net. Also, the Server based on Windows Server 2003 with IIS and Microsoft SQL Server 2000. Client will communicate with wireless access point at first, and then message will be send to server via Ethernet. People will manage knowledge base in server administrator terminal.

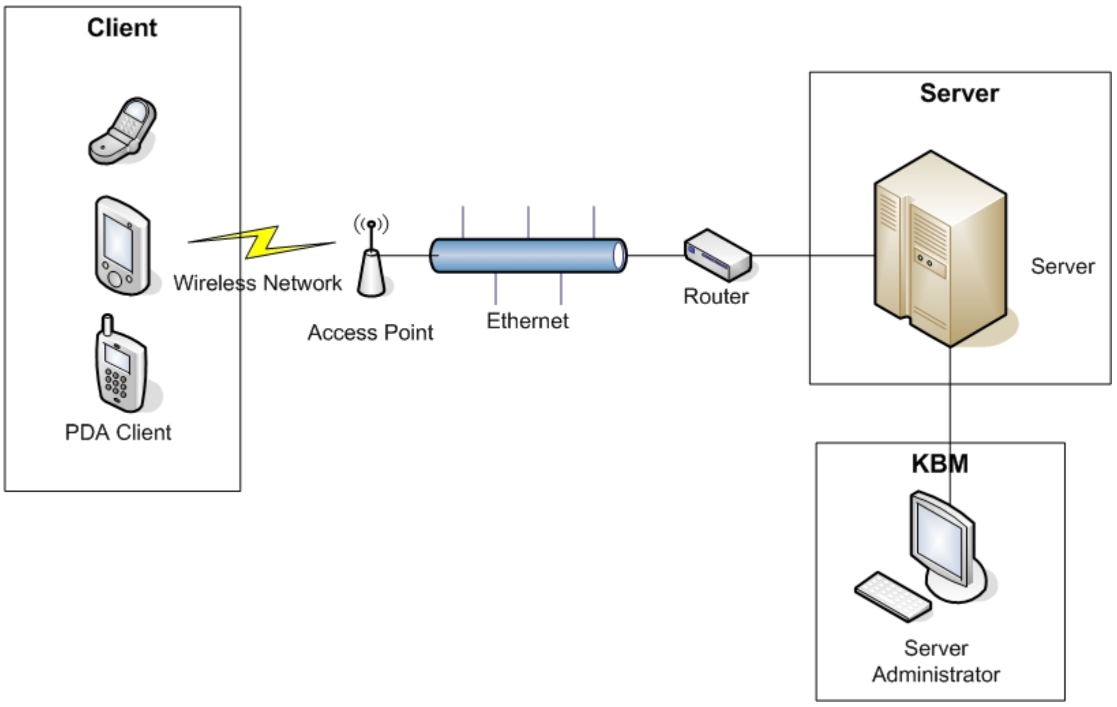


Figure 6. Function Overview

3.4.2 Client Function Requirements

3.4.2.1 icon and text operation

1. add icon

We can choose icon from icons bar and drag (or click) onto the map, then attribute window appear. We need input icon name and amount by using text input. All these informations will be sent to server when we click synchronization button.

2. delete icon

We can select icon on the map firstly, click delete button, then the icon we selected will disappear. The attribute of the icon will also be deleted.

3. move icon

We can select icon on the map firstly, drag (or click) onto other position of the map, then the icon we selected will move to that position. The attribute of the icon will never be changed.

4. modify icon attribute

We can select icon on the map firstly, and then click the text button from icons bar, or we can click the text button firstly and then click icon on the map. They are same operations. After that, the attribute of the icon will appear and then we can modify icon name and/or amount.

3.4.2.2 map operation

1. zoom in

When user click zoom in button, the pointer will change to zoom in picture. We can select an area on the map, then the area will be blown-up and fill with the frame.

2. zoom out

When user click zoom out button, the pointer will change to zoom out picture. We can click map, then the map will be shrinked.

3. move

When user click move button, the pointer will change to move picture. We can move the map by moving mouse or stylus.

3.4.2.3 synchronization operation

When user click synchronization button, client will send world model to the server and server will send new world model, suggestion and crisis information back at same time.

3.4.3 Server Function Requirements

3.4.3.1 intelligent agent based on rules

We need a middleware as an intelligent agent in the server. It will base on rules and have following functions:

1. Fuse and analysis information from multiple clients. Different client will send different world model. Server will fuse and analysis these world models, and then give a consistent world model to every client. The rule is based on time. Server will keep the newest world model which sent by client as consistent world model.
2. Server can give suggestion depend on the relation between crisis event and element. These relations should be some rules and store in knowledge base.

3. Server can give scenario prediction and crisis information after prognosticating the right scenario. The prediction rules also depend on relation between crisis event and element. All these rules and crisis information will store in knowledge base.

3.4.3.2 knowledge base

We also need a database as knowledge base to storage all rules and crisis information.

3.4.4 KBM Function Requirements

3.4.4.1 rules management

We need an interface to add and edit rules easily. The condition and result of rules are flexible to adjust.

3.4.4.2 crisis information management

We also need an interface to add and edit crisis information. It is similar with text editor. We can copy and paste text, and then submit to database.

Chapter 4: Design

4.1 UML

UML stands for Unified Modeling Language and is a system of diagrams that can specify how systems work. System development focuses on three different models of the system:
1) The functional model is represented in UML by Use Case Diagrams, which specifies the systems functionality from a user's 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 behavior 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.

4.2 Requirements Elicitation

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).

The first step in analysis is to figure out what the system will be used for and who will be using it. These are the use cases and actors, respectively. All use cases must begin with an actor, and some will also end with an actor. Actors are people or other systems that are outside of the system we are working on.

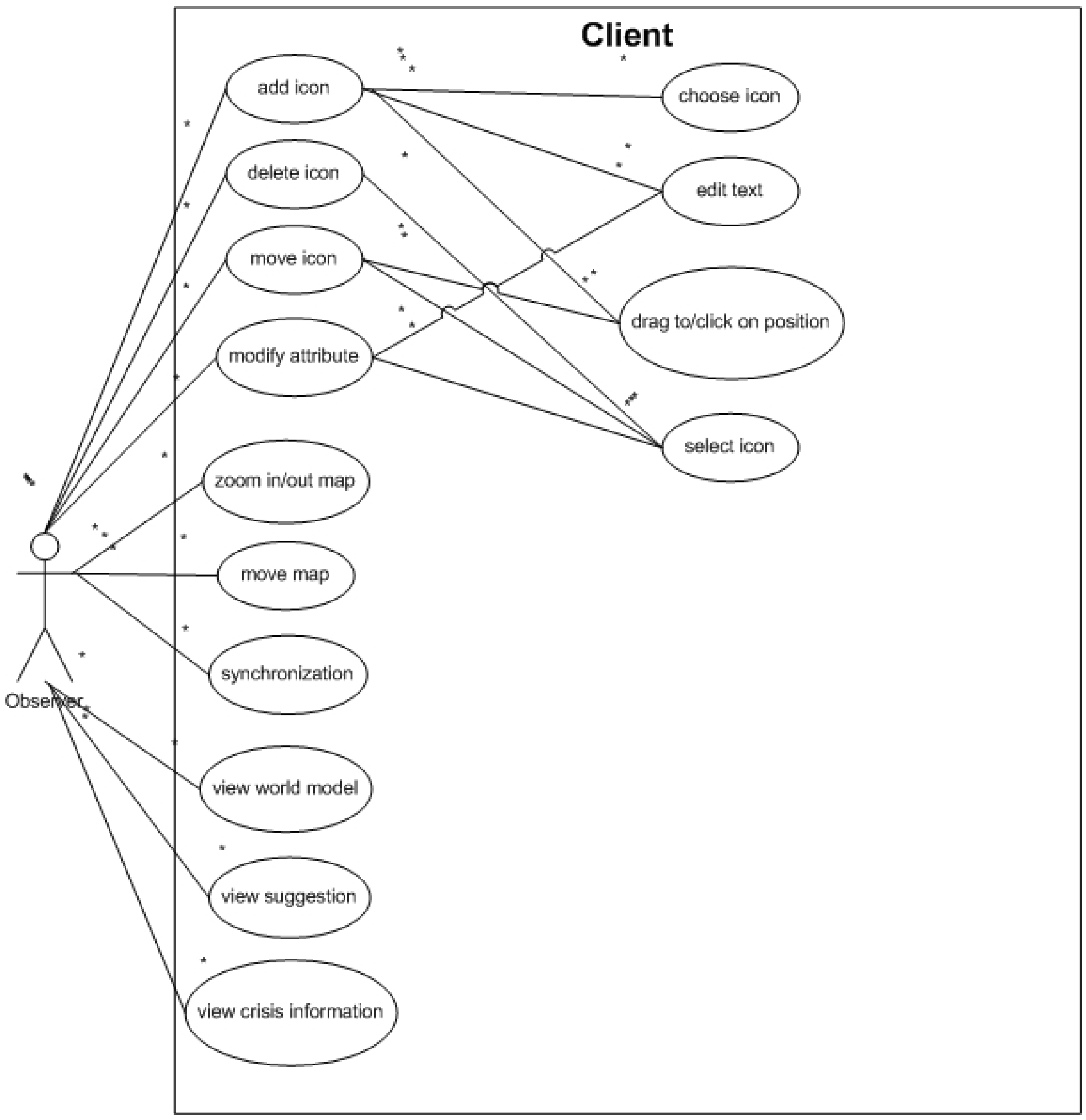


Figure 7. Use Case Diagram of Client

In the Use Case Diagram (Figure 7) the use case of the client is shown. They can add icon, delete icon, move icon, modify icon attribute, view the world model, view the suggestions, view the crisis information, zoom in/out map, move map and synchronization.

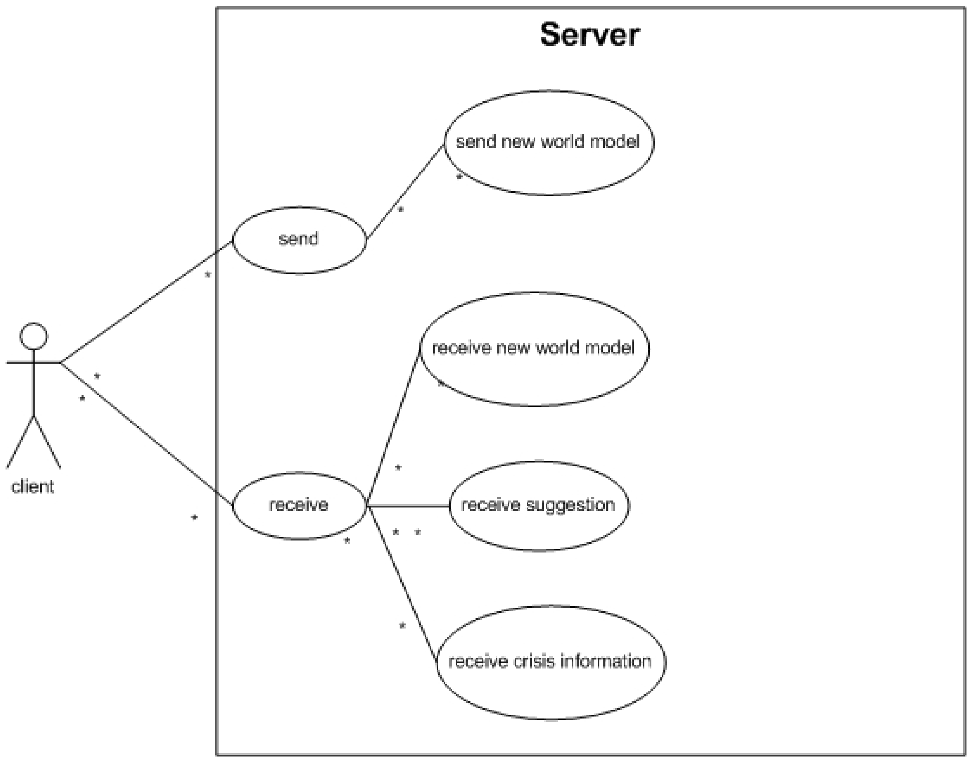


Figure 8. Use Case Diagram of Server

In the next Use Case Diagram (Figure 8) the use case of the server is shown. Just like the observer gets and inputs information to client, the client interacts with the server. The client can either send or receive information from the server. The client can send new world model to server. Receiving from the server is split up in 3 stages: Receive new world model, receive the information about the crisis, and receive the suggestions.

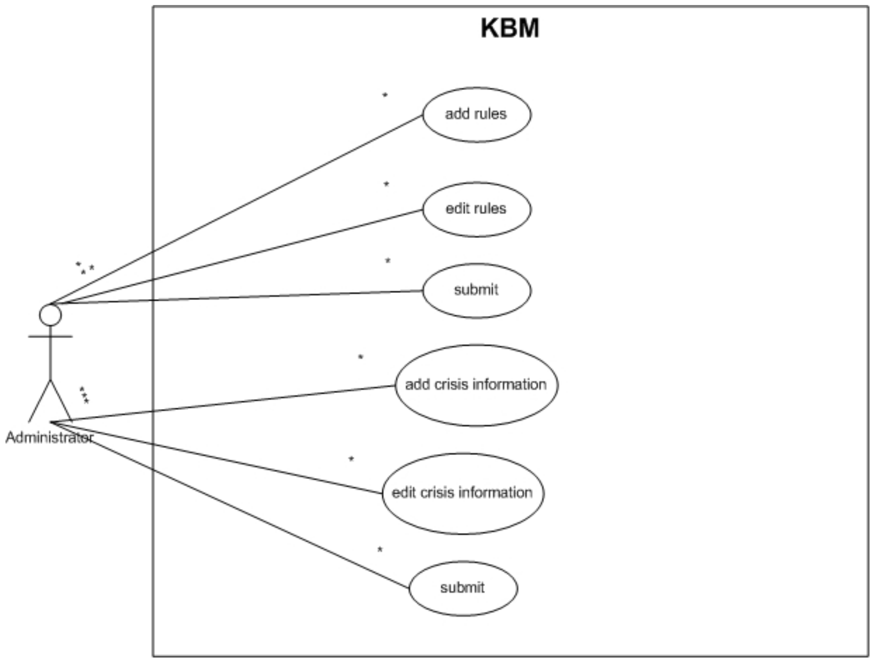


Figure 9. Use Case Diagram of KBM

In the last Use Case Diagram (Figure 9) the use case of the knowledge base management is shown. Administrators can add rules, edit rules and submit to database. Also they can add crisis information, edit and submit to database.

4.3 Architecture Design

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 built up in classes already and therefore it is easy to create a Class Diagram of it.

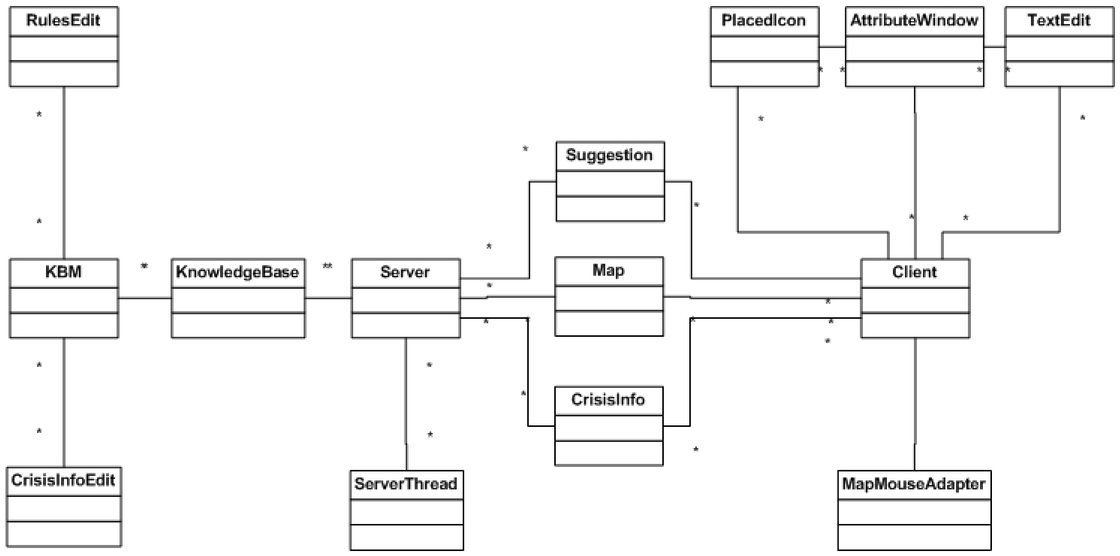


Figure 10. Class Diagram

When we start with the class that has the highest connectivity, Client, we see that this main part of the client has Suggestion is which suggestion will be displayed and CrisisInfo is which the crisis information will be displayed, an AttributeWindow which is used to specify name and amount of icon. TextEdit is a class that we can input and edit text. 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 Client has a MapMouseAdapter, 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 and moving.

The Server is the main class for the intelligent agent, all the reasoning is done via this class. It also has a Map, Suggestion and CrisisInfo. They are not same instances as classes on the Client, but via the network these instances will be kept synchronized. The ServerThread is the class that handles the connections with the Client. Each ServerThread instance will support the connection with one client, but the Server can have many different serverThreads (in fact, as many as the computational power of the server can handle). Finally there is the KnowledgeBase class which communicated with database.

Also we have KBM class to manage knowledge base. RulesEdit is a class that we can add and edit rules and CrisisInfoEdit is similar.

4.4 Detailed Design

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.

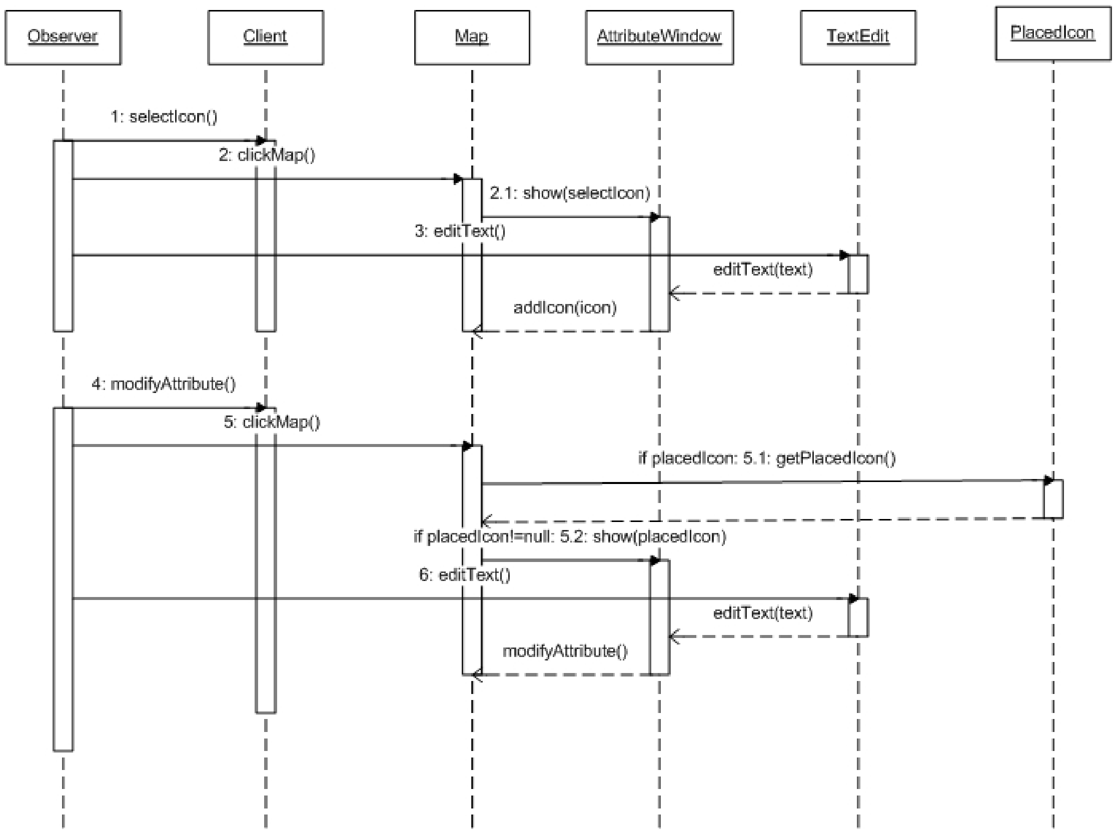


Figure 11. Sequence Diagram of adding and modifying icon

Figure 11 shows how icons can be added and modified on the local map. To add a new icon the observer has to select which icon to place. After that the observer clicks on the map to indicate where the icon needs to be placed. Note that in reality this happens by the use of the MapMouseAdapter, 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 observer then inputs the appropriate text after which the AttributeWindow tells the map to add the icon.

To modify an already placed icon the observer has to activate the modify mode. After clicking on the map (again handled by the MapMouseAdapter) 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 observer can edit it, after which the map gets told to modify the icon.

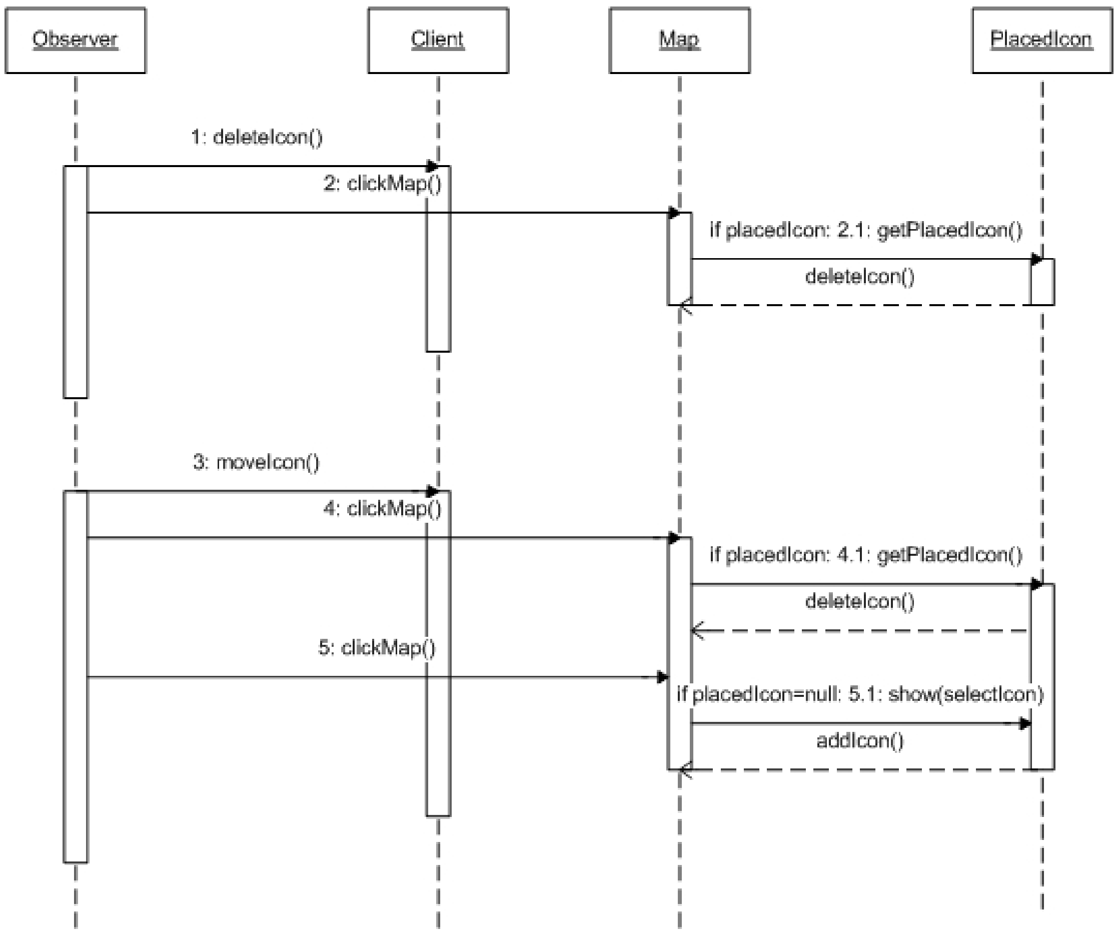


Figure 12. Sequence Diagram of deleting and moving icon

To delete an icon (Figure 12) 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. To move an icon, fist the move 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. Then we click on the position which the icon will be moved to. The icon appear in that position.

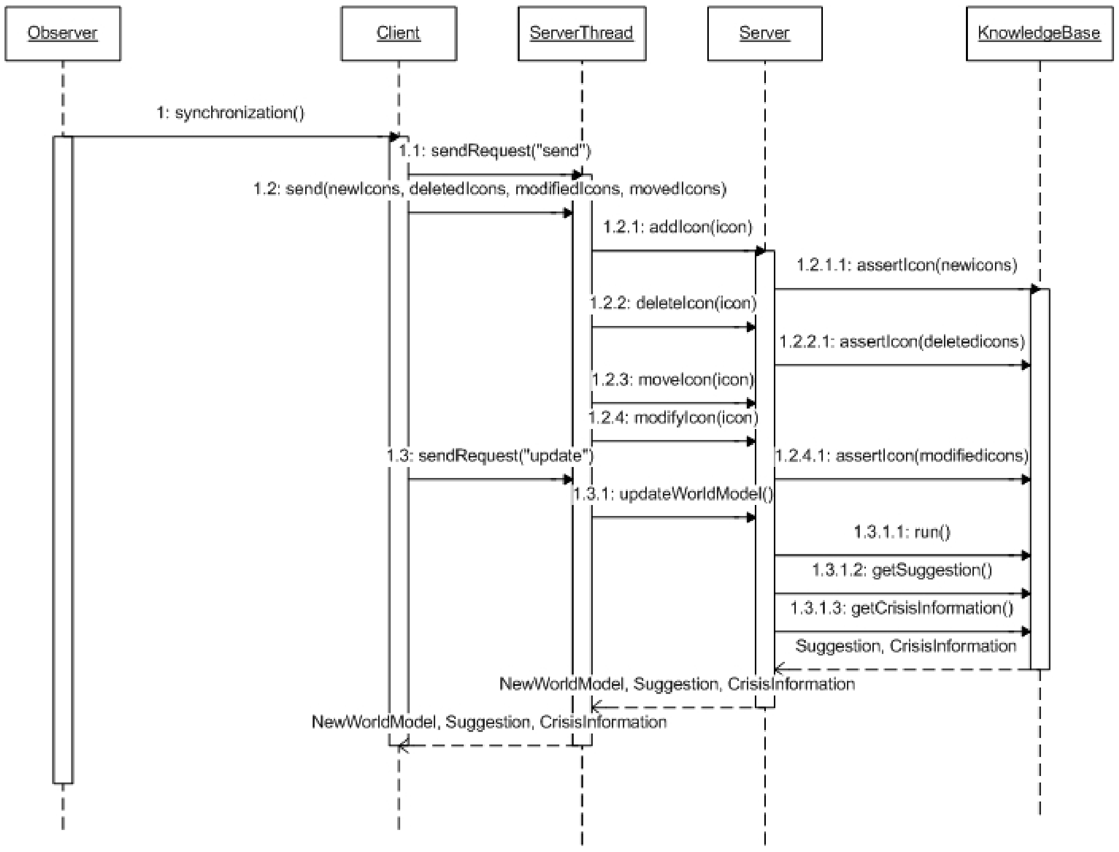

Observer
Client
ServerThread
Server
KnowledgeBase
1: synchronization()
1.1: sendRequest("send")
1.2: send(newIcons, deletedIcons, modifiedIcons, movedIcons)
1.2.1: addIcon(icon)
1.2.1.1: assertIcon(newicons)
1.2.2: deleteIcon(icon)
1.2.2.1: assertIcon(deletedicons)
1.2.3: moveIcon(icon)
1.2.4: modifyIcon(icon)
1.3: sendRequest("update")
1.2.4.1: assertIcon(modifiedicons)
1.3.1: updateWorldModel()
1.3.1.1: run()
1.3.1.2: getSuggestion()
1.3.1.3: getCrisisInformation()
Suggestion, CrisisInformation
NewWorldModel, Suggestion, CrisisInformation
NewWorldModel, Suggestion, CrisisInformation

Figure 13. Sequence Diagram of synchronization

In Figure 13 we can see the flow of events that happen when the observer presses the synchronization button. The Client begins with sending a simple String to the ServerThread. The communication between the Client 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 Server itself directly, is that a Server 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 four Vectors with new icons, deleted icons, moved icons and modified icons. After the Vectors are received the ServerThread will pass the Vectors on to the Server with the request to handle them. The Server will then add the information to the knowledge base, represented by the KnowledgeBase class.

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

4.5 Interface Design

4.5.1 Client Interface Design

The Graphical User Interface (GUI) of the client is made entirely in C#. It provided a good way to report crisis situation to the server. Because it works on PDA and the screen size is limited, we need interface as simple as possible.

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 modify option, the ability to send your information to the server, and some zooming possibilities.

Since the clients will be receiving some information about the suggestion and crisis information there needs to be some room reserved for displaying this information as well. After many prototypes, the final interface of the main screen that we developed can be seen in Figure 14.

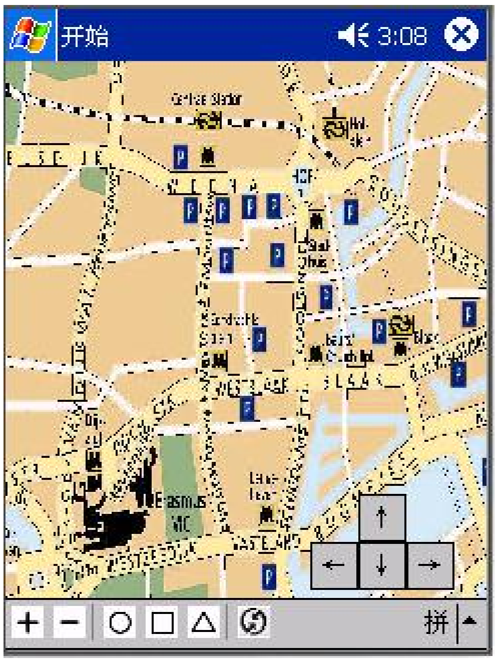
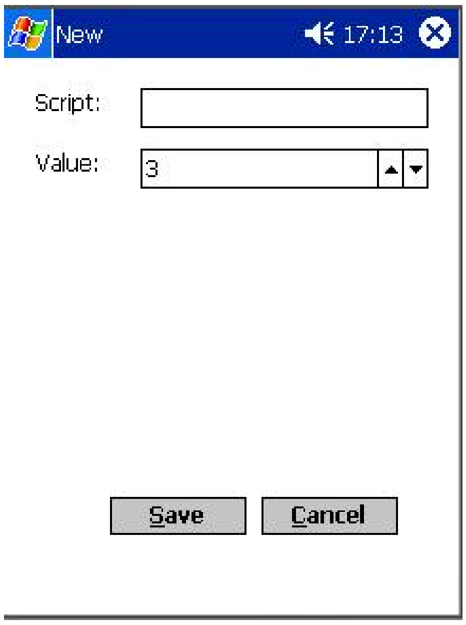


Figure 14. Client GUI

As said before, the application is developed to fit on a handheld computer, and therefore has a resolution of 320X240 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 bottom toolbar is reserved for the icon categories. These can be added to the XML files, and will then automatically be displayed in the GUI. Also the bottom toolbar is included synchronization button that your information can be send to the server and to get an updated world model back. Finally this toolbar contains some options for zooming and movement. The buttons marked with a + and – will zoom in and out. The button marked with hand will move the map. 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.

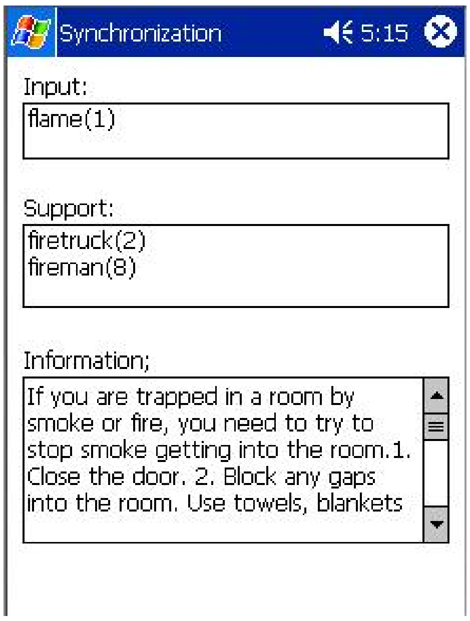


Script:

Value: 3

Figure 15. Attribute Window

The next toolbar is reserved for the control options. Here the options to delete and modify 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.



Input:
flame(1)

Support:
firetruck(2)
fireman(8)

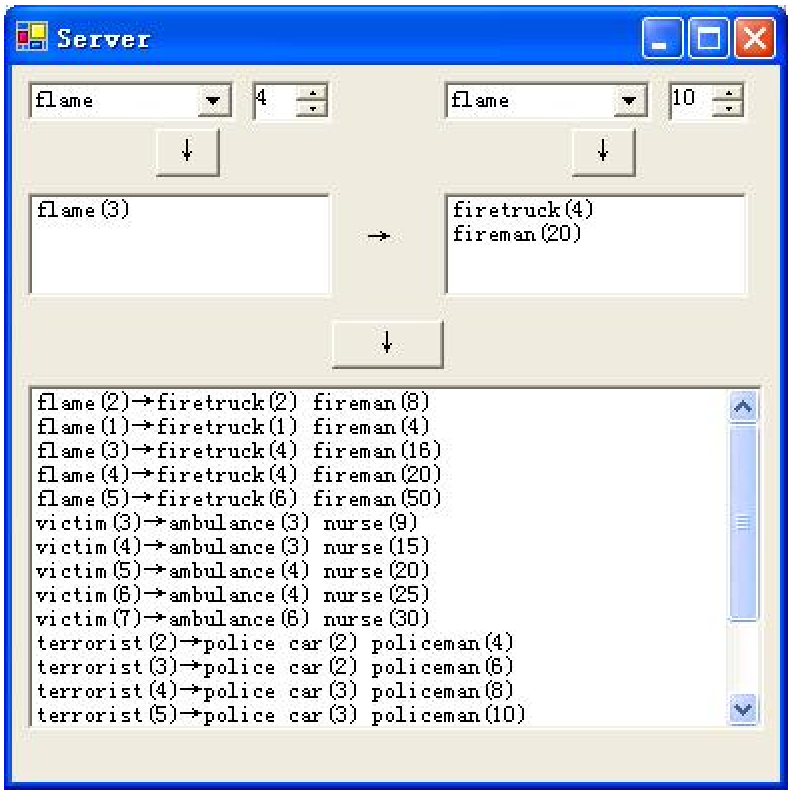
Information;
If you are trapped in a room by smoke or fire, you need to try to stop smoke getting into the room.1. Close the door. 2. Block any gaps into the room. Use towels, blankets

Figure 16. Information Window

After we click synchronization button and update with server, there is a window that shows the suggestion and the crisis information.

4.5.2 Knowledge Base Management Interface Design

The Graphical User Interface (GUI) of the knowledge base management is also made entirely in C#. It provided a good way to manage rules and crisis information of the knowledge base. Because the crisis situation is too much and complex, we have to always add new rules and crisis information for knowledge base. The GUI of this system is very important.



```
flame(3)
```

```
firetruck(4)
fireman(20)
```

```
flame(2)→firetruck(2) fireman(8)
flame(1)→firetruck(1) fireman(4)
flame(3)→firetruck(4) fireman(16)
flame(4)→firetruck(4) fireman(20)
flame(5)→firetruck(6) fireman(50)
victim(3)→ambulance(3) nurse(9)
victim(4)→ambulance(3) nurse(15)
victim(5)→ambulance(4) nurse(20)
victim(6)→ambulance(4) nurse(25)
victim(7)→ambulance(6) nurse(30)
terrorist(2)→police car(2) policeman(4)
terrorist(3)→police car(2) policeman(6)
terrorist(4)→police car(3) policeman(8)
terrorist(5)→police car(3) policeman(10)
```

Figure 17. Knowledge Base Management GUI

The left up side combo box is crisis event. The right side one is rescue teams. Also, we can choose amount easily. The next two text box is rules editor. We can match crisis event and rescue teams in them. The last big text box is rules base.

4.6 Knowledge Base Design

4.6.1 Suggestion Rules

In the table 1, there are suggestion rules. In the left side it is condition which comes from reporter. In the right side it is result which asserted by intelligent agent. Table 2 is about relation of amount between the vehicle and people of rescue teams.

If (x)	**Then (x)**
flame, smoke, fire or explosion	fire truck
victim, cadaver or body	ambulance
terrorist, bomb, thief or robber	police car

If	**Then**
fire truck (x)	fireman (4x)
ambulance (x)	nurse (3x)
police car (x)	policeman (2x)

4.6.2 Crisis Event Predictor Rules

In the table 3 it is relation between the origin and crisis event. In the left side it is keyword of crisis origin and crisis event is in the right side. Table 4 is about crisis event information in detail.

If	**Then**
flame, smoke, fire or explosion and no terrorist	fire (give fire event information)
terrorist	terrorism (give terrorism event information)

Crisis Event	**Information**
fire	If you are trapped in a room by smoke or fire, you need to try to stop smoke getting into the room. 1. Close the door. 2. Block any gaps into the room. Use towels, blankets or spare clothes. 3. If there is a telephone in the room dial 999. 4. If there is no telephone, go to the window and shout for help. 5. Once you know you have been heard and help is on the way, stay near to the floor by the window. Smoke and heat rise so you are safer near to the ground. 6. If your windows are double glazed, use a heavy object and hit the window in a bottom corner. Make any jagged edges safe with a towel or blanket.
terrorism	1. Calm down 2. Don’t stay with terrorist 3. Try to find way to escape

Chapter 5: Implementation

5.1 Implementation Environment

5.1.1 Visual Studio .NET Enterprise Architect 2003

Visual Studio .NET is Microsoft's multi-faceted development tool, targeting both Windows and Web applications. This 2003 edition includes numerous small improvements as well as major new features like the Compact Framework, for applications that run on Pocket PC and other smart devices. It supports multiple languages, with the main ones being Visual Basic, C#, and C++. There is also a Java-like language called J#, although J# applications only work on Windows so this is not a true Java development tool. The Visual Studio .NET environment is truly integrated. It makes extensive use of docking and tabbed windows, and there are plenty of project wizards along with huge amounts of online help. Auto-completion and pop-up help eases the business of editing code.

Microsoft Visual Studio .net
Microsoft Visual Basic .NET
Microsoft Visual C# .NET
Microsoft Visual C++ .NET
Crystal Reports
Microsoft Application Center Test

企业级结构设计版

本产品使用权属于:
马宵
天津

该程序受美国和国际版权法保护，
请参见“帮助”中的“关于”。
© 1987 - 2002 Microsoft Corporation。保留所有权利。

Visual Studio 家族中的已安装产品:

Figure 18. Visual Studio .NET Enterprise Architect 2003

Microsoft's .NET tools are very different from their predecessors. Visual C++ can still compile standard Windows executables, but the other languages all target the .NET Framework, a runtime engine and class library that manages memory and enforces security. Framework applications perform well, since they are compiled to native code at runtime, but there is an overhead in terms of memory usage and the Framework runtime must be installed. In compensation, .NET brings many advantages. All the languages are fully object-oriented, the class library is rich, and XML support is fully integrated. ASP.NET is for web applications, and represents a large advance on the old ASP. Instead of script, ASP.NET supports any of the .NET languages, running on the server and just-in-time compiled to native code. There is a visual web page designer, and carefully designed applications support a wide range of browsers. New in this version of Visual Studio is Mobile Web Forms, which use adaptive rendering to support the browsers in mobile phones and PDAs. The database technology in Visual Studio is called ADO.NET, and uses a disconnected model that is ideal for laptops, smart devices, and wide area networks.

5.1.2 Microsoft .NET Compact Framework

Microsoft developed the .NET Compact Framework with one intention in mind: to build applications. We are talking about applications that display, gather, process and forward information, those applications that give users a reason to carry a device. While they typically will have an interface, they do not require one. The data that they are working with might be local, might be remote, or some combination of both.

The .NET Compact Framework simplifies application development on smart devices. Currently this includes the Pocket PC, Pocket PC 2002, Pocket PC Phone Edition and other devices running Windows CE.NET 4.1 or later.

We will need Visual Studio .NET 2003 to build applications that target the .NET Compact Framework. We can build applications using either Visual C# .NET, Visual Basic .NET, or both. The .NET Compact Framework has two main components: the common language runtime and the .NET Compact Framework class library.

The runtime is the foundation of the .NET Compact Framework. It is responsible for managing code at execution time, providing core services such as memory management and thread management while enforcing code safety and accuracy. Code that targets the runtime is known as managed code; code that does not target the runtime, as is the case with eMbedded Visual C++, is known as unmanaged, or native code.

The .NET Compact Framework class library is a collection of reusable classes that you can use to quickly and easily develop applications. This framework was designed with porting in mind, whether to Microsoft or third-party platforms. What does this mean to you? Simply that the coding techniques and the applications you create today to run on a Pocket PC could run on other platforms, such as a cell phone or another vendor's PDA, if a version of the .NET Compact Framework was created for that platform.

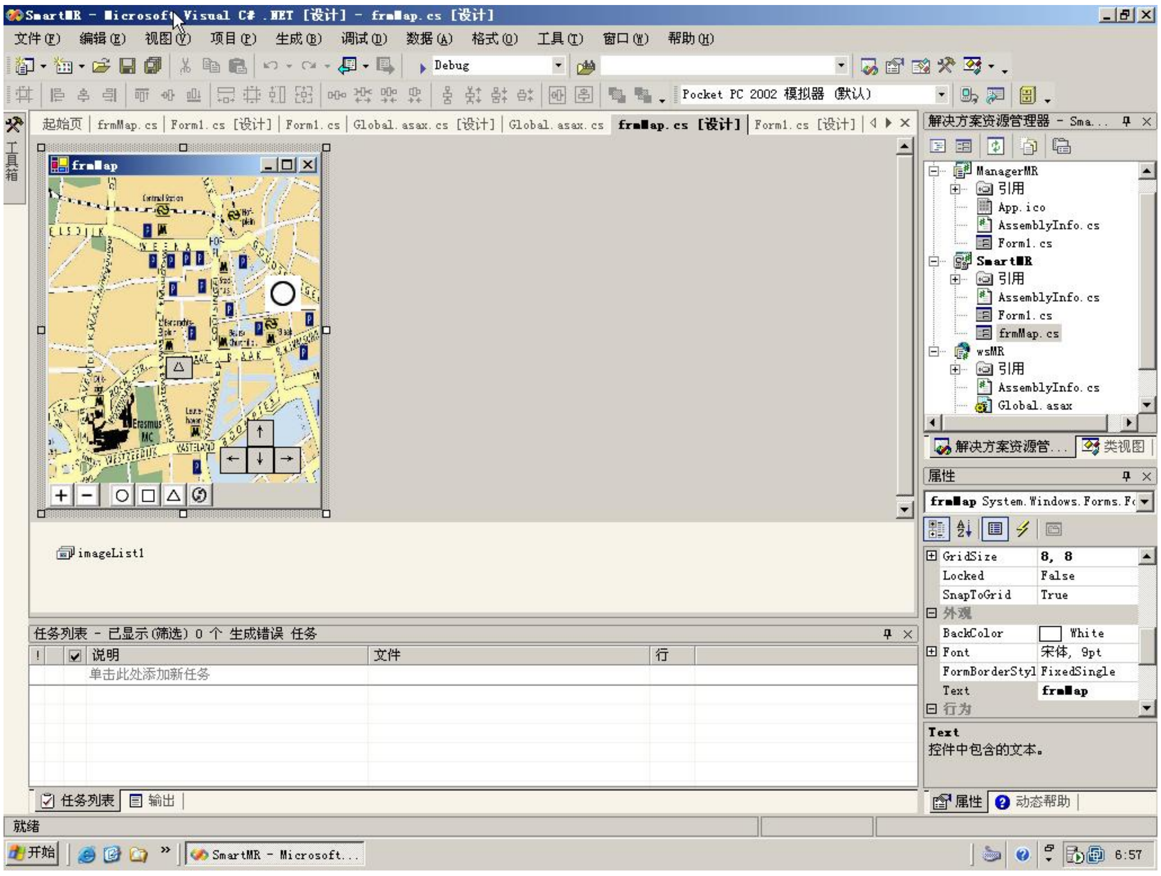
5.1.3 Visual C# .NET 2003

C# (pronounced "C sharp") is a simple, modern, object-oriented, and type-safe programming language. It will immediately be familiar to C and C++ programmers. C# combines the high productivity of Rapid Application Development (RAD) languages and the raw power of C++.

Visual C# .NET is Microsoft's C# development tool. It includes an interactive development environment, visual designers for building Windows and Web applications, a compiler, and a debugger. Visual C# .NET is part of a suite of products, called Visual Studio .NET, that also includes Visual Basic .NET, Visual C++ .NET, and the JScript scripting language. All of these languages provide access to the Microsoft .NET Framework, which includes a common execution engine and a rich class library. The .NET Framework defines a "Common Language Specification" (CLS), a sort of lingua franca that ensures seamless interoperability between CLS-compliant languages and class libraries. For C# developers, this means that even though C# is a new language, it has complete access to the same rich class libraries that are used by seasoned tools such as Visual Basic .NET and Visual C++ .NET. C# itself does not include a class library.

5.2 Client Implementation

Client interface included two parts. One is map view, the other is information view. Also there are three function models in it. There are map option model, iocn and text option model and synchronization operation model. Figure 19 shows the client interface implementation.


SmartMR - Microsoft Visual C# .NET [设计] - frmMap.cs [设计]
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 数据(A) 格式(O) 工具(T) 窗口(W) 帮助(H)
Debug
Pocket PC 2002 模拟器 (默认)
起始页 | frmMap.cs | Form1.cs [设计] | Form1.cs | Global.asax.cs [设计] | Global.asax.cs | frmMap.cs [设计] | Form1.cs [设计]
frmMap
imageList1
解决方案资源管理器 - SmartMR
ManagerMR
引用
App.ico
AssemblyInfo.cs
Form1.cs
SmartMR
引用
AssemblyInfo.cs
Form1.cs
frmMap.cs
wsMR
引用
AssemblyInfo.cs
Global.asax
解决方案资源管... | 类视图
属性
frmMap System.Windows.Forms.Fo
任务列表 - 已显示(筛选) 0 个 生成错误 任务
说明
文件
行
单击此处添加新任务
任务列表 | 输出
属性 | 动态帮助
就绪
开始
SmartMR - Microsoft...
6:57

GridSize	8, 8
Locked	False
SnapToGrid	True
外观	
BackColor	White
Font	宋体, 9pt
FormBorderStyl	FixedSingle
Text	frmMap
行为	

Text
控件中包含的文本。

Figure 19. Client Interface Implementation

After finishing code, we can run it in the emulator which integrated in IDE.

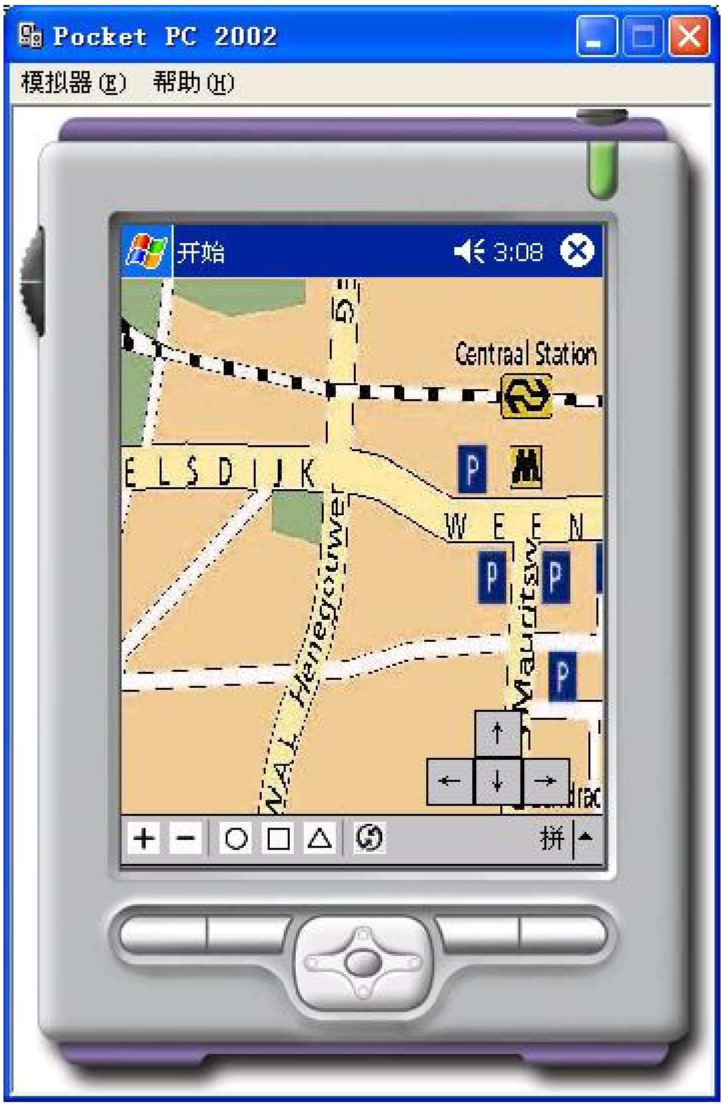
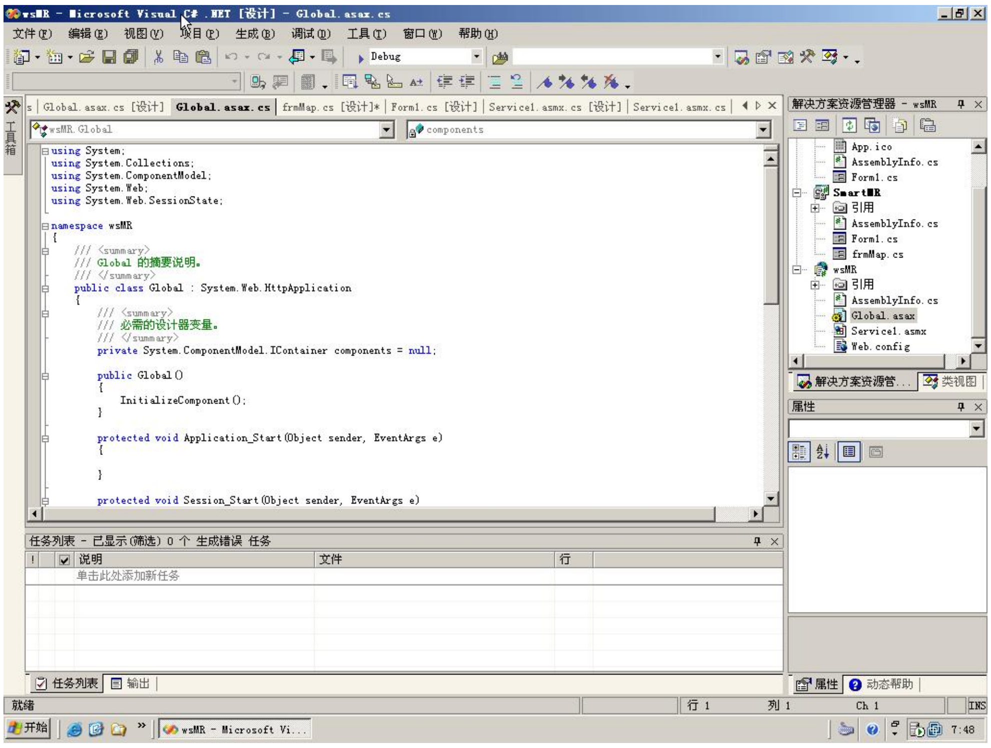


Figure 20. Client Interface

5.3 Server Implementation

We need a middleware as an intelligent agent in the server. It will be integrated in server as a web service. The web service will receive message from client and give responses after analysis in knowledge base.

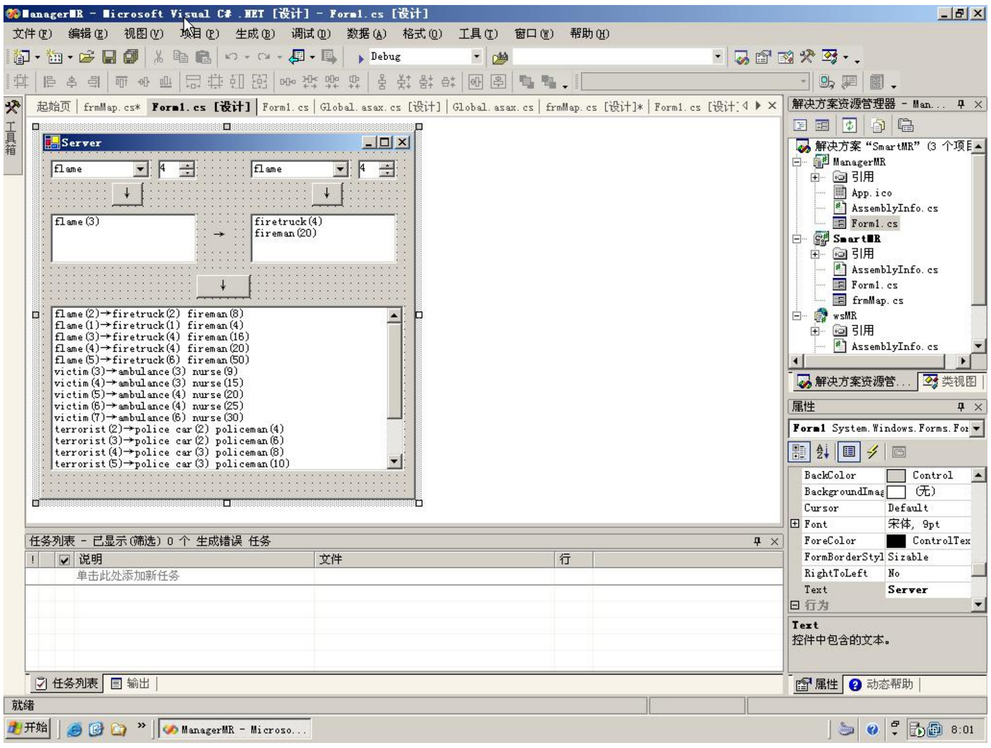


```
using System;
using System.Collections;
using System.ComponentModel;
using System.Web;
using System.Web.SessionState;

namespace wsMR
{
    /// <summary>
    /// Global 的摘要说明。
    /// </summary>
    public class Global : System.Web.HttpApplication
    {
        /// <summary>
        /// 必需的设计器变量。
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        public Global()
        {
            InitializeComponent();
        }

        protected void Application_Start(Object sender, EventArgs e)
        {

        }

        protected void Session_Start(Object sender, EventArgs e)
```

Figure 21. Server Implementation

5.4 Knowledge Base Management Implementation

There are two interfaces in KBM system. One is to manage suggestion rules. Another is to manage crisis information rules. Figure 22 shows suggestion rules implementation.



```
flame(2)→firetruck(2) fireman(8)
flame(1)→firetruck(1) fireman(4)
flame(3)→firetruck(4) fireman(16)
flame(4)→firetruck(4) fireman(20)
flame(5)→firetruck(6) fireman(50)
victim(3)→ambulance(3) nurse(9)
victim(4)→ambulance(3) nurse(15)
victim(5)→ambulance(4) nurse(20)
victim(6)→ambulance(4) nurse(25)
victim(7)→ambulance(6) nurse(30)
terrorist(2)→police car(2) policeman(4)
terrorist(3)→police car(2) policeman(6)
terrorist(4)→police car(3) policeman(8)
terrorist(5)→police car(3) policeman(10)
```

Figure 22. KBM Implementation

Chapter 6: Testing Scenarios and Results

6.1 Client Testing

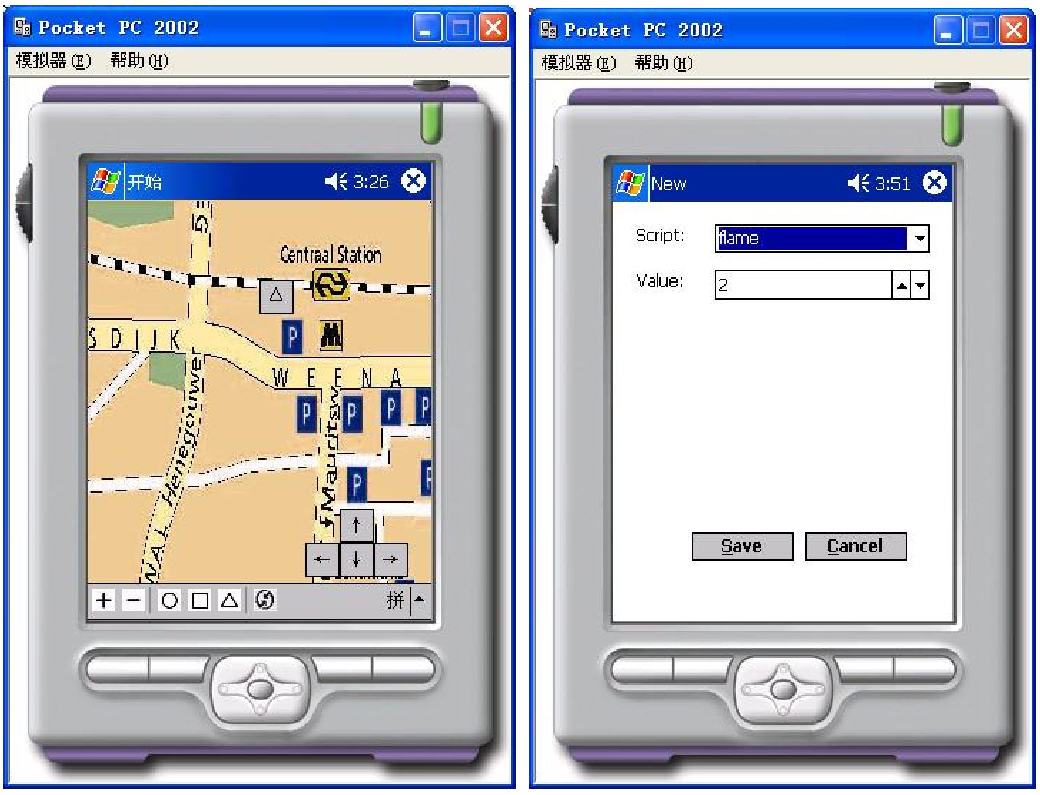
For testing the client, we suppose that there are two crisis scenarios and two observers.

6.1.1 crisis scenario: fire

When fire event happened, there are two observer around here.

Figure 23. Fire Crisis Event

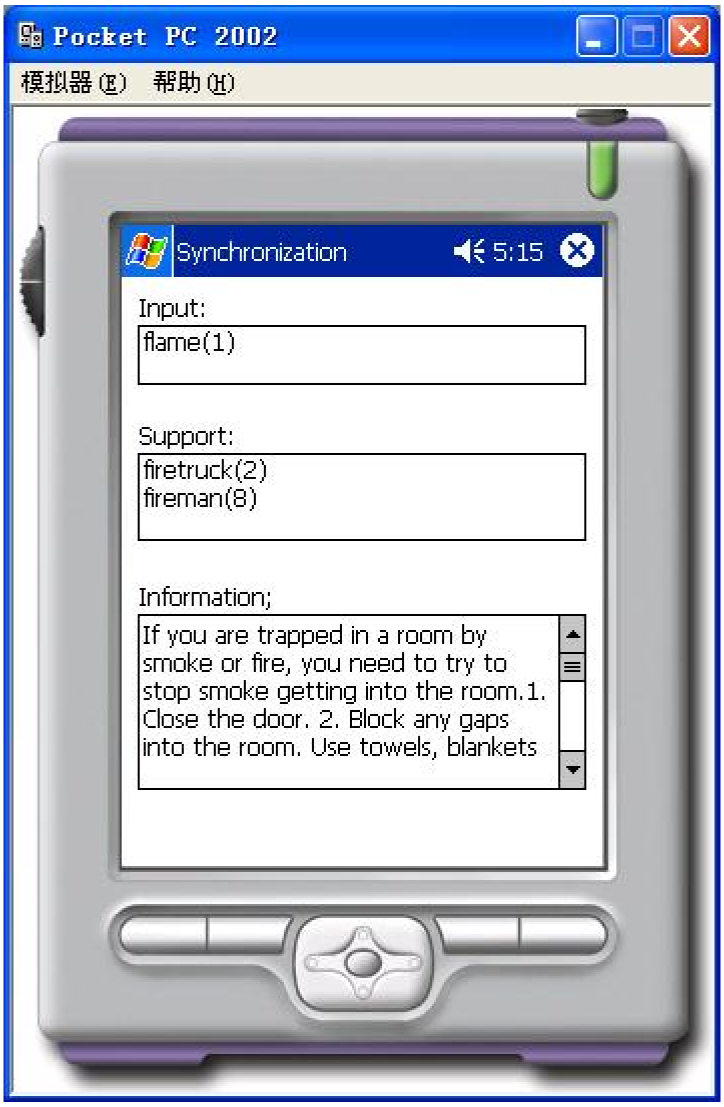
One of them starts his report as follow:



Script: flame
Value: 2

Figure 24. Observer 1 Report

When he click synchronization button, he can read information from server as follow:



Input:

flame(1)

Support:

firetruck(2)
fireman(8)

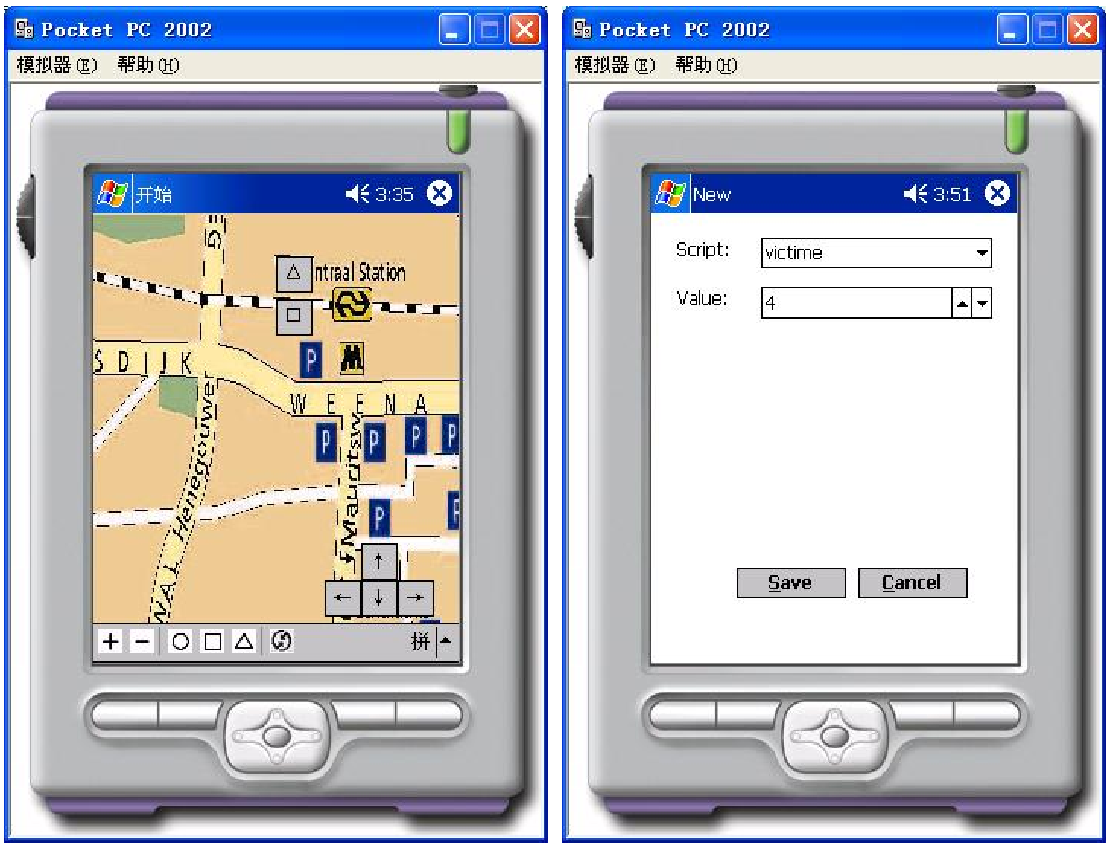
Information;

If you are trapped in a room by smoke or fire, you need to try to stop smoke getting into the room.1. Close the door. 2. Block any gaps into the room. Use towels, blankets

Figure 25. Observer 1 Information

At the same time, server will keep in his world model as the consistent world model.

Another observer is later than the first one. When he begin to report, there are some victims around here. He starts his report as follow:

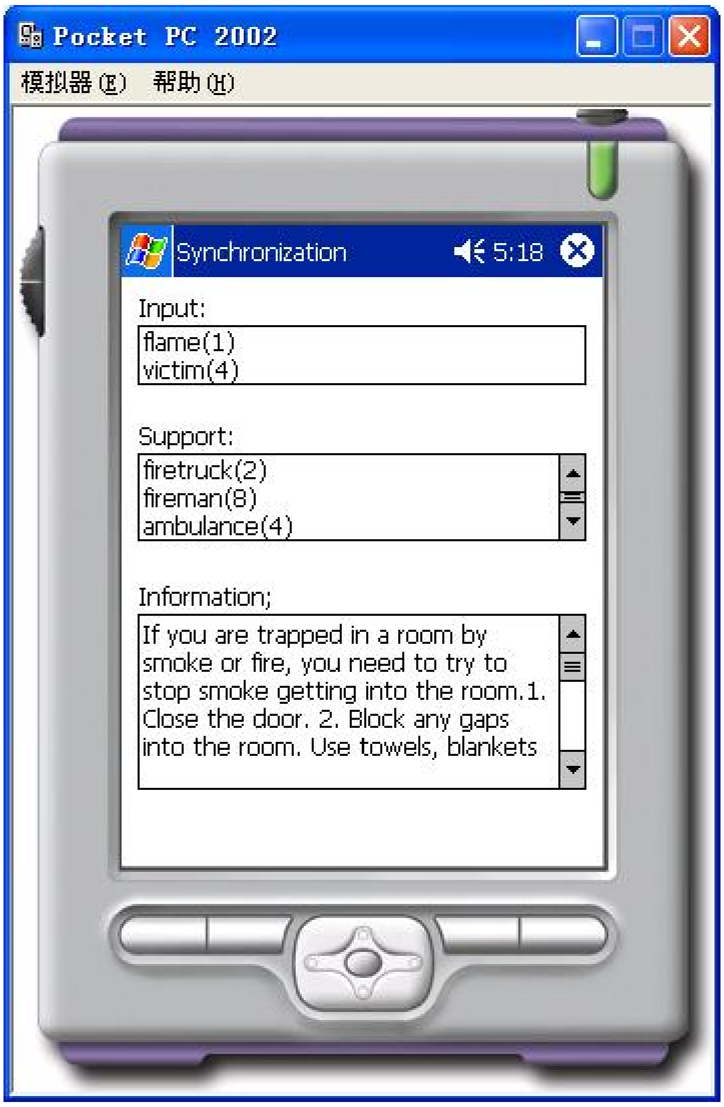


Script: victime

Value: 4

Figure 26. Observer 2 Report

When he click synchronization button, he can read information from server as follow:



Input:

flame(1)
victim(4)

Support:

firetruck(2)
fireman(8)
ambulance(4)

Information;

If you are trapped in a room by smoke or fire, you need to try to stop smoke getting into the room.1. Close the door. 2. Block any gaps into the room. Use towels, blankets

Figure 27. Observer 2 Information

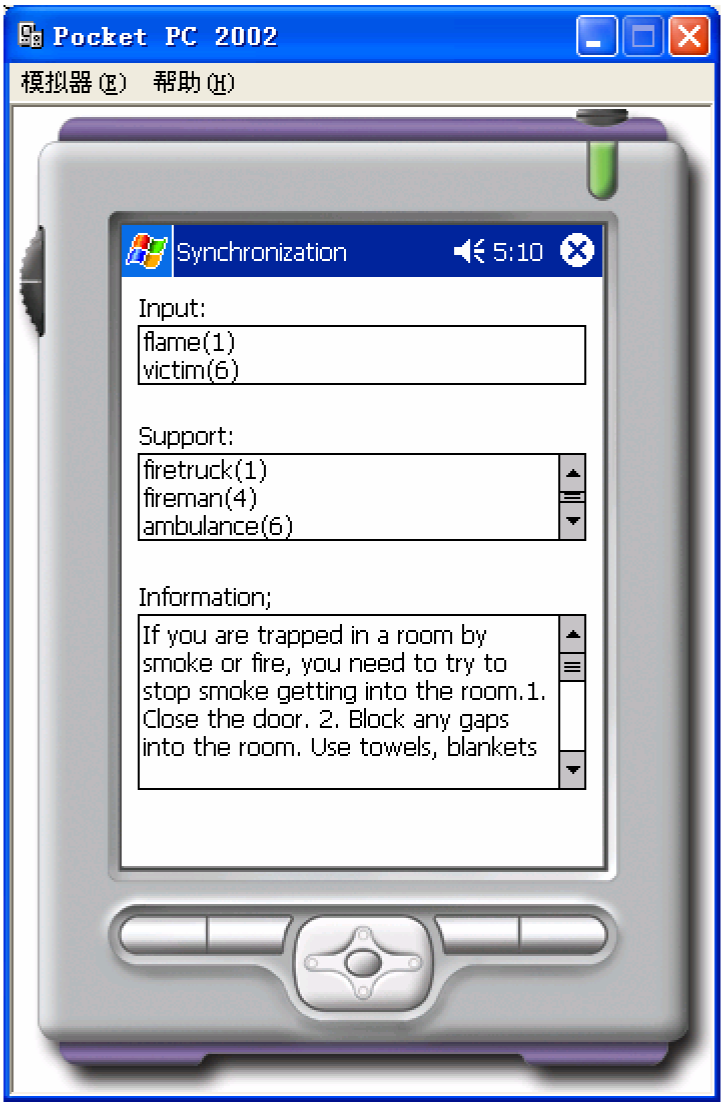
At the same time, server will keep in his world model as the consistent world model because his report is the newest. If the first observer click synchronization button again at this time, he will get the same information with last observer.

6.1.2 crisis scenario: terrorism

London attack, the co-ordinated attacks hit the transport system as the morning rush hour drew to a close. When this happened, there are two observer around here.

Figure 28. Terrorism Crisis Event

The first observer only sees flame and victims. When he sent his world model to server, the crisis predictor gave fire conclusion. When he click synchronization button, he can read information from server as follow:



Input:

flame(1)
victim(6)

Support:

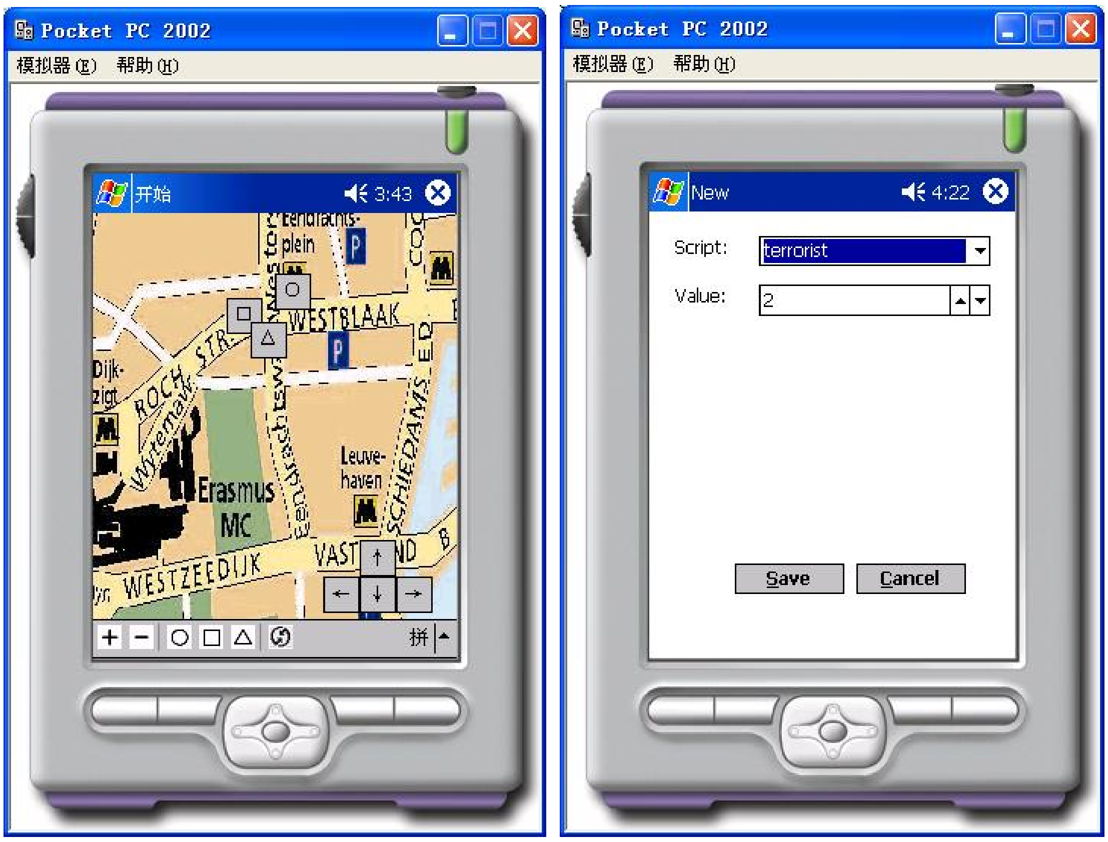
firetruck(1)
fireman(4)
ambulance(6)

Information;

If you are trapped in a room by smoke or fire, you need to try to stop smoke getting into the room.1. Close the door. 2. Block any gaps into the room. Use towels, blankets

Figure 29. Observer 1 Information

The last one also sees the terrorist. He starts his report as follow:

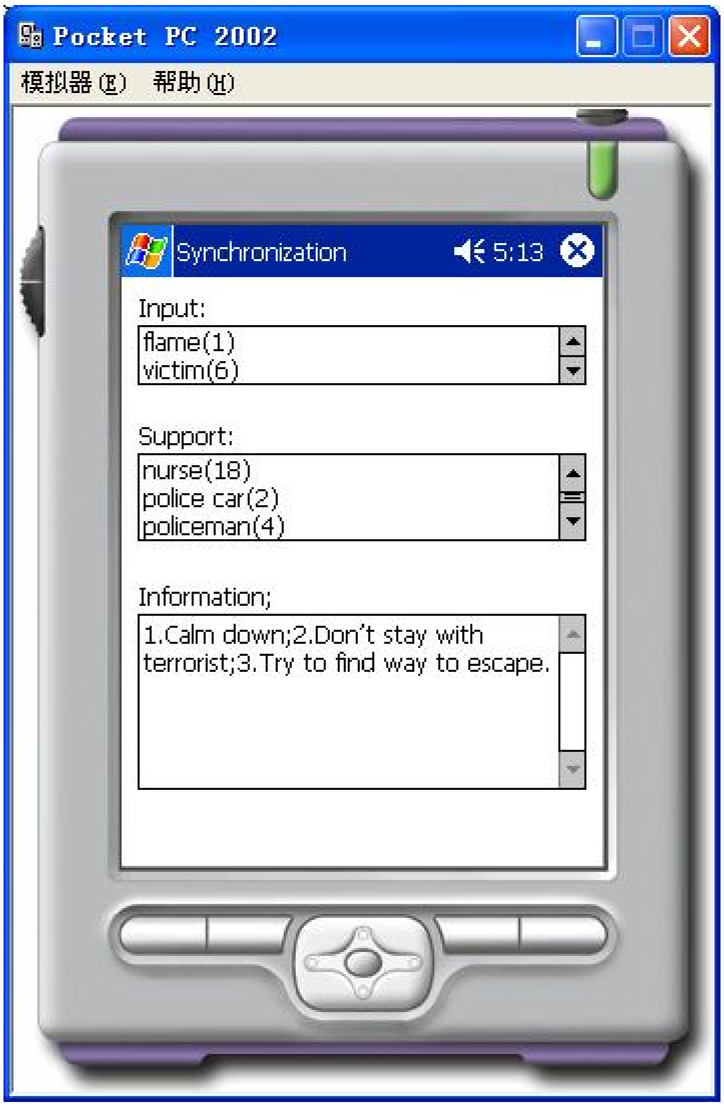


Script: terrorist

Value: 2

Figure 30. Observer 2 Report

When he click synchronization button, server will give terrorism conclusion because of terrorist keyword, and he can read information from server as follow:



Input:

flame(1)
victim(6)

Support:

nurse(18)
police car(2)
policeman(4)

Information;

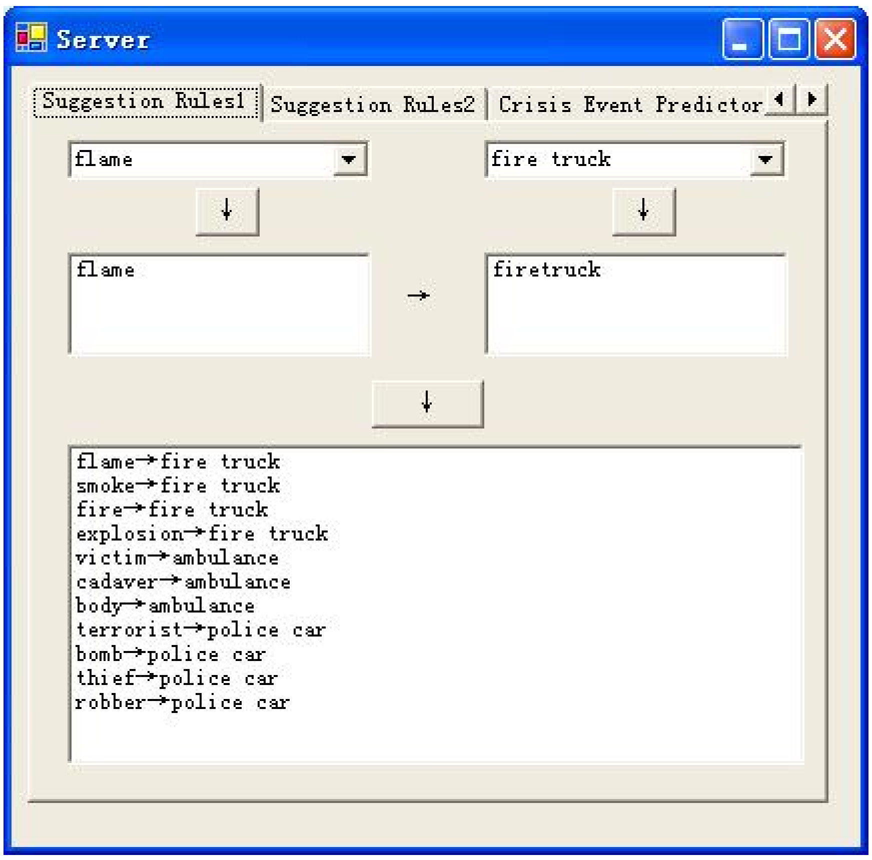
1.Calm down;2.Don't stay with terrorist;3.Try to find way to escape.

Figure 31. Observer 2 Information

6.2 Server Testing

In the server part, we need to test Knowledge Base Management. Because there are four functions about KBM interface, we need test them one by one.

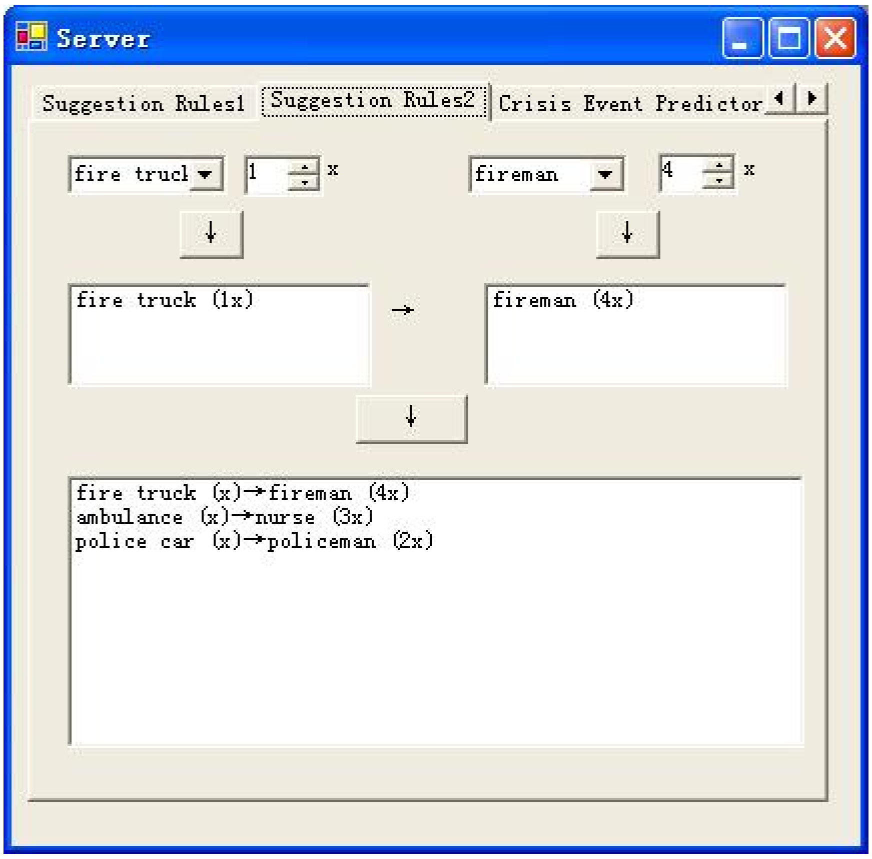
6.2.1 suggestion rules about relation between crisis element and the vehicles



```
flame→fire truck
smoke→fire truck
fire→fire truck
explosion→fire truck
victim→ambulance
cadaver→ambulance
body→ambulance
terrorist→police car
bomb→police car
thief→police car
robber→police car
```

Figure 32. Suggestion Rules 1

6.2.2 suggestion rules about relation of amount between the vehicle and people of rescue teams



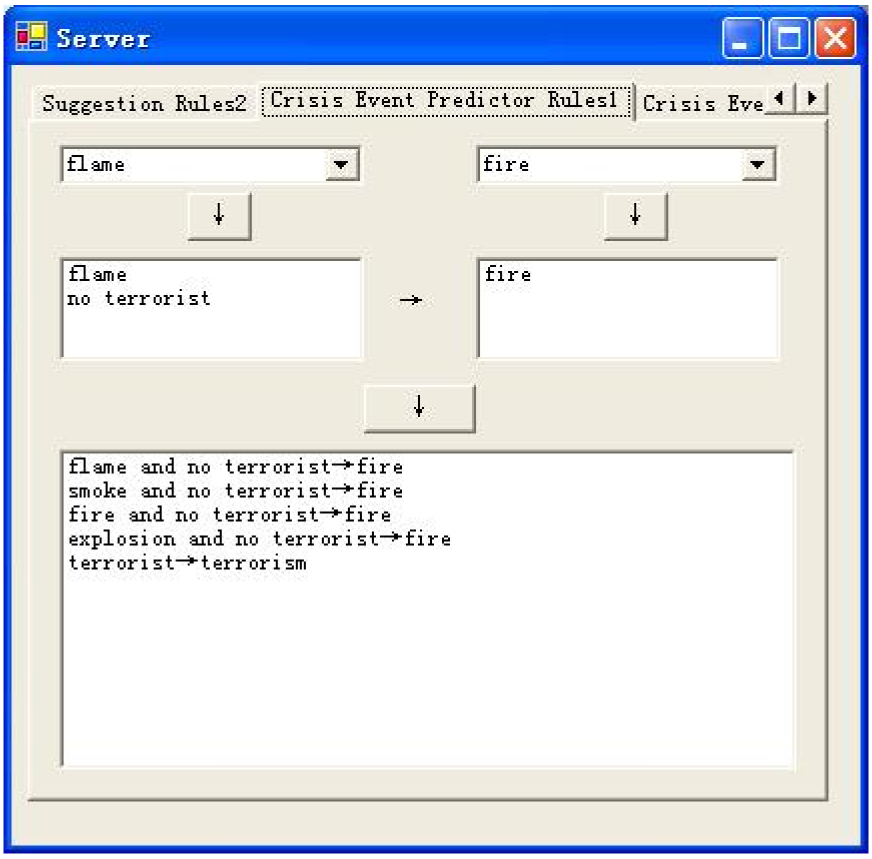
```
fire truck (1x)
```

```
fireman (4x)
```

```
fire truck (x)→fireman (4x)
ambulance (x)→nurse (3x)
police car (x)→policeman (2x)
```

Figure 33. Suggestion Rules 2

6.2.3 crisis event predictor rules about relation between crisis element and crisis event



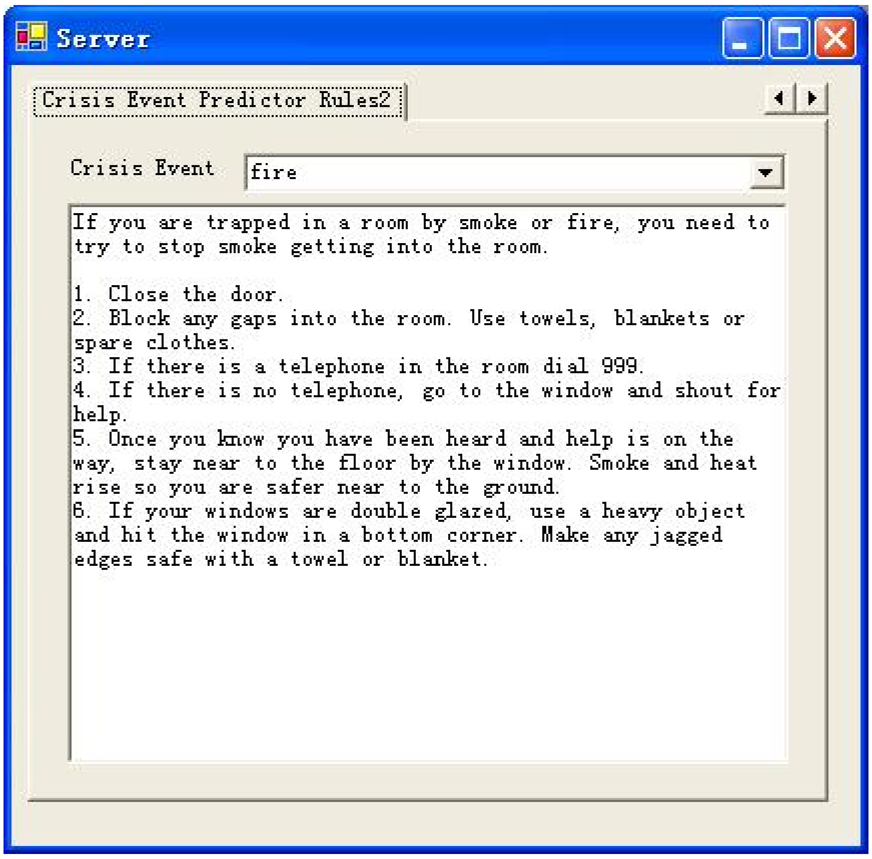
```
flame
no terrorist
```

```
fire
```

```
flame and no terrorist→fire
smoke and no terrorist→fire
fire and no terrorist→fire
explosion and no terrorist→fire
terrorist→terrorism
```

Figure 34.Crisis Event Predictor Rules

6.2.4 crisis event information



```
If you are trapped in a room by smoke or fire, you need to
try to stop smoke getting into the room.

1. Close the door.
2. Block any gaps into the room. Use towels, blankets or
spare clothes.
3. If there is a telephone in the room dial 999.
4. If there is no telephone, go to the window and shout for
help.
5. Once you know you have been heard and help is on the
way, stay near to the floor by the window. Smoke and heat
rise so you are safer near to the ground.
6. If your windows are double glazed, use a heavy object
and hit the window in a bottom corner. Make any jagged
edges safe with a towel or blanket.
```

Figure 35. Cirsis Event information

6.3 Testing Results

After finishing these testing works, we can make a conclusion that all function models of client and server are working. The interface is easy to use and easy to learn. If we have a PDA device, we can test client in wireless environment.

Chapter 7: Conclusion & Recommendation

7.1 Conclusion

The goal of this project was to design and develop a prototype of the crisis situation reporting and analysis system based on the PDA and knowledge base. The system will work with one server and multiple clients. Both of them will communicate with each other via wireless network. 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.

After analysis all requirements, we design that the whole system should include three part: Client, Server and Knowledge Base Management. Client part is based on PDA device. Server and Knowledge Base are running in server. We need an interface to manage knowledge base. After our analysis work, we begin to design system in detail by UML. At the same time, we begin to design 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 PhotoShop, 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. Also, all of icons be done in PhotoShop one by one. After our design work, we begin to implement the whole system in VS.NET.

The next prototypes were developed in C#, 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. 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.

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 intelligent component to handle the world model. The idea is always keeping the last one report from client as the consistent world model.

Behavior that could be added in a matter that relies both on human and machine intelligence is to let the server make suggestions. To just add the icons would be too dangerous, and the location of the 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, how much and where.

The server also makes some information about what crisis event 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 server. A server that has a consistent world model at all times, makes suggestions and gives information to the user.

During the requirement analysis, we have taken the problem description to split up the different requirements into workable components. 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. 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 input by text. When icons are placed, the user is able to delete them again, or to inspect or alter its attributes text. 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 intelligent component in the server is responsible for keeping the world model up-to-date. Because the server only has 1 world model that keep in the newest one, 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 different reporting, 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.
6) New concepts can easily be added or adjusted by altering the knowledge base. The relations between the icons and suggestion can also be adjusted in this way. Adjusting these will result in the system to give other icon suggestions or crisis information.

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

7.2 Recommendation

The development of this system has been a single student effort with a time span of approximately six months. 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.

More web application

In the server side, we need more web application for people who want to know what is going on about crisis event. The traditional way is up to news paper or news conference. In the future, it is possible to put world model from client on the web. Everyone can read that via internet.

More detail map

In this system we only have an overview map to describe crisis situation. We need more detail map to improve precision of the crisis situation. For example, there is fire in one room. If we only have street map, the only thing we can do is reporting the position of the building. But the most important thing is we need to know what situation is in the building. In the future, we can create a map server based on a database. People can download more detail map from the server at first. We can change the map from macrostructure to microstructure.

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.

With satellite technology

One of 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.

Security issues

When we send 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.

We can add a digital signature to the message by encoding it with the client’s 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 client’s 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

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

[2] Object-oriented Software Engineering, conquering complex and changing systems, B. Bruegge, A.H. Dutoit, 2000, Prentice-Hall, Inc.

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

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

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

[6] Expert System tool, see http://www.ghg.net/clips/CLIPS.html

[7] MSDN, see http://msdn.microsoft.com/developercenters/

[8] Combined Systems group, see www.decis.nl

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

[10] 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

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

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

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

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

[15] 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

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

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

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

[19] MSc Thesis of Paul Schooneman, Icon based System for Managing Emergencies, Delft University of Technology

[20] UML Programming Guide, 2001

Appendix A. Client Kernel Code

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace SmartMR
{
        /// <summary>
        /// frmMap 的摘要说明。
        /// </summary>
        public class frmMap : System.Windows.Forms.Form
        {
                private System.Windows.Forms.PictureBox pictureBox1;
                private System.Windows.Forms.Button button7;
                private System.Windows.Forms.ToolBar toolBar1;
                private System.Windows.Forms.ToolBarButton toolBarButton1;
                private System.Windows.Forms.ToolBarButton toolBarButton2;
                private System.Windows.Forms.ToolBarButton toolBarButton3;
                private System.Windows.Forms.ToolBarButton toolBarButton4;
                private System.Windows.Forms.ToolBarButton toolBarButton5;
                private System.Windows.Forms.ToolBarButton toolBarButton6;
                private System.Windows.Forms.ToolBarButton toolBarButton7;
                private System.Windows.Forms.ImageList imageList1;
                private System.Windows.Forms.PictureBox pbMark;
                private System.Windows.Forms.Button button5;
                private System.Windows.Forms.Button button1;
                private System.Windows.Forms.Button button2;
                private System.Windows.Forms.Button button6;
                private int giStatus=0;

                public frmMap()
                {
                        //
                        // Windows 窗体设计器支持所必需的
                        //
                        InitializeComponent();

                        //
                        // TODO: 在 InitializeComponent 调用后添加任何构造函数代码
                        //
```

```
		}

		/// <summary>
		/// 清理所有正在使用的资源。
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			base.Dispose( disposing );
		}

		#region Windows 窗体设计器生成的代码
		/// <summary>
		/// 设计器支持所需的方法 - 不要使用代码编辑器修改
		/// 此方法的内容。
		/// </summary>
		private void InitializeComponent()
		{
			System.Resources.ResourceManager		resources		=		new
System.Resources.ResourceManager(typeof(frmMap));
			this.pictureBox1 = new System.Windows.Forms.PictureBox();
			this.button7 = new System.Windows.Forms.Button();
			this.toolBar1 = new System.Windows.Forms.ToolBar();
			this.toolBarButton1 = new System.Windows.Forms.ToolBarButton();
			this.toolBarButton2 = new System.Windows.Forms.ToolBarButton();
			this.toolBarButton3 = new System.Windows.Forms.ToolBarButton();
			this.toolBarButton4 = new System.Windows.Forms.ToolBarButton();
			this.toolBarButton5 = new System.Windows.Forms.ToolBarButton();
			this.toolBarButton6 = new System.Windows.Forms.ToolBarButton();
			this.toolBarButton7 = new System.Windows.Forms.ToolBarButton();
			this.imageList1 = new System.Windows.Forms.ImageList();
			this.pbMark = new System.Windows.Forms.PictureBox();
			this.button5 = new System.Windows.Forms.Button();
			this.button1 = new System.Windows.Forms.Button();
			this.button2 = new System.Windows.Forms.Button();
			this.button6 = new System.Windows.Forms.Button();
			//
			// pictureBox1
			//
			this.pictureBox1.Image								=
((System.Drawing.Image)(resources.GetObject("pictureBox1.Image")));
			this.pictureBox1.Size = new System.Drawing.Size(240, 272);
			this.pictureBox1.SizeMode							=
System.Windows.Forms.PictureBoxSizeMode.StretchImage;
			//
```

```
            // button7
            //
            this.button7.Location = new System.Drawing.Point(104, 160);
            this.button7.Size = new System.Drawing.Size(24, 20);
            this.button7.Text = "△";
            this.button7.Click += new System.EventHandler(this.button7_Click);
            //
            // toolBar1
            //
            this.toolBar1.Buttons.Add(this.toolBarButton1);
            this.toolBar1.Buttons.Add(this.toolBarButton2);
            this.toolBar1.Buttons.Add(this.toolBarButton3);
            this.toolBar1.Buttons.Add(this.toolBarButton4);
            this.toolBar1.Buttons.Add(this.toolBarButton5);
            this.toolBar1.Buttons.Add(this.toolBarButton6);
            this.toolBar1.Buttons.Add(this.toolBarButton7);
            this.toolBar1.ImageList = this.imageList1;
            this.toolBar1.ButtonClick                    +=                    new
System.Windows.Forms.ToolBarButtonClickEventHandler(this.toolBar1_ButtonClick);
            //
            // toolBarButton1
            //
            this.toolBarButton1.ImageIndex = 8;
            //
            // toolBarButton2
            //
            this.toolBarButton2.ImageIndex = 9;
            //
            // toolBarButton3
            //
            this.toolBarButton3.Style                                         =
System.Windows.Forms.ToolBarButtonStyle.Separator;
            //
            // toolBarButton4
            //
            this.toolBarButton4.ImageIndex = 1;
            //
            // toolBarButton5
            //
            this.toolBarButton5.ImageIndex = 4;
            //
            // toolBarButton6
            //
            this.toolBarButton6.ImageIndex = 6;
```

```
			//
			// toolBarButton7
			//
			this.toolBarButton7.ImageIndex = 7;
			//
			// imageList1
			//

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource1"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource2"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource3"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource4"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource5"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource6"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource7"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource8"))));

	this.imageList1.Images.Add(((System.Drawing.Image)(resources.GetObject("resource9"))));
			this.imageList1.ImageSize = new System.Drawing.Size(16, 16);
			//
			// pbMark
			//
			this.pbMark.Image                                                    =
((System.Drawing.Image)(resources.GetObject("pbMark.Image")));
			this.pbMark.Location = new System.Drawing.Point(192, 88);
			this.pbMark.Size = new System.Drawing.Size(32, 32);
			//
			// button5
			//
			this.button5.Location = new System.Drawing.Point(176, 216);
			this.button5.Size = new System.Drawing.Size(24, 24);
			this.button5.Text = " ↑ ";
			this.button5.Click += new System.EventHandler(this.button5_Click);
			//
			// button1
			//
			this.button1.Location = new System.Drawing.Point(200, 240);
			this.button1.Size = new System.Drawing.Size(24, 24);
			this.button1.Text = "→";
			this.button1.Click += new System.EventHandler(this.button1_Click);
			//
			// button2
			//
			this.button2.Location = new System.Drawing.Point(152, 240);
			this.button2.Size = new System.Drawing.Size(24, 24);
			this.button2.Text = "←";
			this.button2.Click += new System.EventHandler(this.button2_Click);
			//
			// button6
			//
			this.button6.Location = new System.Drawing.Point(176, 240);
			this.button6.Size = new System.Drawing.Size(24, 24);
			this.button6.Text = " ↓ ";
			this.button6.Click += new System.EventHandler(this.button6_Click);
			//
			// frmMap
			//
			this.Controls.Add(this.button5);
			this.Controls.Add(this.button6);
			this.Controls.Add(this.pbMark);
			this.Controls.Add(this.button7);
			this.Controls.Add(this.button2);
			this.Controls.Add(this.button1);
			this.Controls.Add(this.pictureBox1);
			this.Controls.Add(this.toolBar1);
			this.Text = "frmMap";
			this.MouseDown                    +=                    new
System.Windows.Forms.MouseEventHandler(this.frmMap_MouseDown);
			this.KeyPress                     +=                    new
System.Windows.Forms.KeyPressEventHandler(this.frmMap_KeyPress);
			this.Load += new System.EventHandler(this.frmMap_Load);
			this.MouseMove                    +=                    new
System.Windows.Forms.MouseEventHandler(this.frmMap_MouseMove);

		}
		#endregion

		private void button1_Click(object sender, System.EventArgs e)
		{
			pictureBox1.Left-=16;

		}

		private void button2_Click(object sender, System.EventArgs e)
		{
//			pictureBox1.Location.X+=16;
			pictureBox1.Left+=16;
		}

		private void button4_Click(object sender, System.EventArgs e)
		{
			pictureBox1.Width*=2;
			pictureBox1.Height*=2;
		}

		private void button3_Click(object sender, System.EventArgs e)
		{
			pictureBox1.Width/=2;
			pictureBox1.Height/=2;
		}

		private void button6_Click(object sender, System.EventArgs e)
		{
			pictureBox1.Top-=16;
		}

		private void button5_Click(object sender, System.EventArgs e)
		{
			pictureBox1.Top+=16;
		}

		private void button7_Click(object sender, System.EventArgs e)
		{

		}

		private void frmMap_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
		{
//			if (giStatus==1)
//			{
//				pbMark.Left=e.X-pbMark.Width/2;
//				pbMark.Top=e.Y-pbMark.Height/2;
//			}
		}

		private void toolBar1_ButtonClick(object sender, System.Windows.Forms.ToolBarButtonClickEventArgs e)
		{
			switch (e.Button.ImageIndex)
			{
				case 8:
					pictureBox1.Width*=2;
					pictureBox1.Height*=2;

					break;
				case 9:
					pictureBox1.Width/=2;
					pictureBox1.Height/=2;

					break;
				case 1:
					giStatus=1;
					break;
				case 4:
					break;
				case 6:
					break;
				case 7:
					break;

			}
		}

		private void frmMap_Load(object sender, System.EventArgs e)
		{


		}

		private void frmMap_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
		{
			if (giStatus==1)
			{
//				pbMark.Left=e.X-pbMark.Width/2;
//				pbMark.Top=e.Y-pbMark.Height/2;
			}
```

```
        }

        private void frmMap_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
        {
            if (e.KeyChar=='a')
                button7.Left+=16;
            else if (e.KeyChar=='d')
                button7.Left-=16;
        }

        private void menuItem3_Click(object sender, System.EventArgs e)
        {

        }

        private void menuItem1_Click(object sender, System.EventArgs e)
        {

        }
    }
}
```

Appendix B. Server Kernel Code

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Configuration;
using System.IO;

namespace DirectoryMonitorWinForm
{
        /// <summary>
        /// Summary description for DirectoryMonitor.
        /// </summary>
        public class DirectoryMonitorForm : System.Windows.Forms.Form
        {

                private static string Path="";
                private static string Filter="";
                private static bool IncludeSubs=false;
                private System.IO.FileSystemWatcher FileMonitor;
                private System.Windows.Forms.Label Label1;
                private System.Windows.Forms.Label Label2;
                private System.Windows.Forms.Label Label3;
                private System.Windows.Forms.Button ButtonStart;
                private System.Windows.Forms.Button ButtonStop;
                private System.Windows.Forms.TextBox directoryToMonitor;
                private System.Windows.Forms.Label Label4;
                private System.Windows.Forms.ComboBox fileFilterList;
                private System.Windows.Forms.CheckBox subdirectoriesAreIncluded;



                /// <summary>
                /// Required designer variable.
                /// </summary>
                private System.ComponentModel.Container components = null;

                public DirectoryMonitorForm()
                {
                        //
                        // Required for Windows Form Designer support
        //
        InitializeComponent();

        //
        // TODO: Add any constructor code after InitializeComponent call
        //
    }

    /// <summary>
    ///    Clean up any resources being used.
    /// </summary>
    protected override void Dispose( bool disposing )
    {
        if( disposing )
        {
            if (components != null)
            {
                components.Dispose();
            }
        }
        base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.FileMonitor = new System.IO.FileSystemWatcher();
        this.ButtonStart = new System.Windows.Forms.Button();
        this.ButtonStop = new System.Windows.Forms.Button();
        this.Label1 = new System.Windows.Forms.Label();
        this.directoryToMonitor = new System.Windows.Forms.TextBox();
        this.Label2 = new System.Windows.Forms.Label();
        this.Label3 = new System.Windows.Forms.Label();
        this.Label4 = new System.Windows.Forms.Label();
        this.fileFilterList = new System.Windows.Forms.ComboBox();
        this.subdirectoriesAreIncluded = new System.Windows.Forms.CheckBox();
        ((System.ComponentModel.ISupportInitialize)(this.FileMonitor)).BeginInit();
        this.SuspendLayout();
        //
        // FileMonitor
```

```
			//
			this.FileMonitor.EnableRaisingEvents = true;
			this.FileMonitor.NotifyFilter = System.IO.NotifyFilters.FileName;
			this.FileMonitor.SynchronizingObject = this;
			this.FileMonitor.Deleted                    +=                    new
System.IO.FileSystemEventHandler(this.FileMonitor_Changed);
			this.FileMonitor.Renamed                    +=                    new
System.IO.RenamedEventHandler(this.FileMonitor_OnRenamed);
			this.FileMonitor.Changed                    +=                    new
System.IO.FileSystemEventHandler(this.FileMonitor_Changed);
			this.FileMonitor.Created                    +=                    new
System.IO.FileSystemEventHandler(this.FileMonitor_Changed);
			//
			// ButtonStart
			//
			this.ButtonStart.Location = new System.Drawing.Point(144, 176);
			this.ButtonStart.Name = "ButtonStart";
			this.ButtonStart.Size = new System.Drawing.Size(64, 23);
			this.ButtonStart.TabIndex = 0;
			this.ButtonStart.Text = "Start";
			this.ButtonStart.Click += new System.EventHandler(this.ButtonStart_Click);
			//
			// ButtonStop
			//
			this.ButtonStop.Location = new System.Drawing.Point(232, 176);
			this.ButtonStop.Name = "ButtonStop";
			this.ButtonStop.Size = new System.Drawing.Size(64, 23);
			this.ButtonStop.TabIndex = 1;
			this.ButtonStop.Text = "Stop";
			this.ButtonStop.Click += new System.EventHandler(this.ButtonStop_Click);
			//
			// Label1
			//
			this.Label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
			this.Label1.Location = new System.Drawing.Point(16, 8);
			this.Label1.Name = "Label1";
			this.Label1.Size = new System.Drawing.Size(384, 24);
			this.Label1.TabIndex = 2;
			this.Label1.Text = "Directory Monitor";
			this.Label1.TextAlign = System.Drawing.ContentAlignment.TopCenter;
			//
			// directoryToMonitor
			//
```

```
            this.directoryToMonitor.Location = new System.Drawing.Point(96, 80);
            this.directoryToMonitor.Name = "directoryToMonitor";
            this.directoryToMonitor.Size = new System.Drawing.Size(312, 20);
            this.directoryToMonitor.TabIndex = 3;
            this.directoryToMonitor.Text = "";
            //
            // Label2
            //
            this.Label2.Location = new System.Drawing.Point(16, 32);
            this.Label2.Name = "Label2";
            this.Label2.Size = new System.Drawing.Size(384, 32);
            this.Label2.TabIndex = 4;
            this.Label2.Text = "Enter a Directory to Monitor, Select a Filter and then Click
\"Start\" to begin the" +
                " monitor.    Click \"Stop\" to end the monitor.";
            //
            // Label3
            //
            this.Label3.Location = new System.Drawing.Point(8, 80);
            this.Label3.Name = "Label3";
            this.Label3.Size = new System.Drawing.Size(88, 16);
            this.Label3.TabIndex = 5;
            this.Label3.Text = "Directory:";
            this.Label3.TextAlign = System.Drawing.ContentAlignment.TopRight;
            //
            // Label4
            //
            this.Label4.Location = new System.Drawing.Point(16, 112);
            this.Label4.Name = "Label4";
            this.Label4.Size = new System.Drawing.Size(72, 16);
            this.Label4.TabIndex = 6;
            this.Label4.Text = "Filter:";
            this.Label4.TextAlign = System.Drawing.ContentAlignment.TopRight;
            //
            // cboFilter
            //
            this.fileFilterList.Items.AddRange(new object[] {
                                                                ".txt",
                                                                ".doc",
                                                                ".xls"});
            this.fileFilterList.Location = new System.Drawing.Point(96, 112);
            this.fileFilterList.Name = "fileFilterList";
            this.fileFilterList.Size = new System.Drawing.Size(121, 21);
            this.fileFilterList.TabIndex = 7;
```

```
			//
			// checkIncludeSubs
			//
			this.subdirectoriesAreIncluded.CheckAlign                                    =
System.Drawing.ContentAlignment.MiddleRight;
			this.subdirectoriesAreIncluded.Location = new System.Drawing.Point(232, 112);
			this.subdirectoriesAreIncluded.Name = "subdirectoriesAreIncluded";
			this.subdirectoriesAreIncluded.Size = new System.Drawing.Size(144, 24);
			this.subdirectoriesAreIncluded.TabIndex = 9;
			this.subdirectoriesAreIncluded.Text = "Include Sub Directories:";
			//
			// DirMon
			//
			this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
			this.ClientSize = new System.Drawing.Size(416, 224);
			this.Controls.AddRange(new System.Windows.Forms.Control[] {

this.subdirectoriesAreIncluded,

this.fileFilterList,

this.Label4,

this.Label2,

this.directoryToMonitor,

this.Label1,

this.ButtonStop,

this.ButtonStart,

this.Label3});
			this.Name = "DirMon";
			this.Text = "Directory Monitor";
			this.Load += new System.EventHandler(this.DirMon_Load);
			((System.ComponentModel.ISupportInitialize)(this.FileMonitor)).EndInit();
			this.ResumeLayout(false);

		}
		#endregion

		/// <summary>
```

```
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new DirectoryMonitorForm());
        }


        private void FileMonitor_OnRenamed(object source, RenamedEventArgs e)
        {
            // Specify what is done when a file is renamed.
            string originalname = e.OldFullPath;
            string renamed = e.FullPath;
            MessageBox.Show("File: "+originalname+" renamed to "+renamed, e.OldName+"
Renamed");
        }

        private void FileMonitor_Changed(object sender, System.IO.FileSystemEventArgs e)
        {
            string ChangeType = e.ChangeType.ToString();

            //display a message box for the appropriate changetype.
            if (ChangeType=="Created")
            {
                MessageBox.Show("File: " +   e.FullPath + " " + e.ChangeType, e.Name+"
Created");
            }
            else if(ChangeType=="Deleted")
            {
                MessageBox.Show("File: " +   e.FullPath + " " + e.ChangeType, e.Name+"
Deleted");
            }
            else if(ChangeType=="Changed")
            {
                MessageBox.Show("File: " +   e.FullPath + " " + e.ChangeType, e.Name+"
Changed");
            }

        }

        private void ButtonStart_Click(object sender, System.EventArgs e)
        {
```

```
        if (directoryToMonitor.Text == "")
        {
            MessageBox.Show("Please Enter a Directory");
        }
        else
        {
            //get the configuration properties from the form.
            Path = directoryToMonitor.Text;
        }

        if (fileFilterList.SelectedText != "")
        {
            Filter = fileFilterList.SelectedText.ToString();
        }
        else
        {
            Filter = "*.*";
        }

        if (subdirectoriesAreIncluded.Checked == true)
        {
            IncludeSubs = true;
        }
        else
        {
            IncludeSubs = false;
        }

        //Set the properties on the monitor.
        FileMonitor.Path = Path.ToString();
        FileMonitor.Filter = Filter.ToString();
        FileMonitor.IncludeSubdirectories = IncludeSubs;
        FileMonitor.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.LastWrite
            | NotifyFilters.FileName | NotifyFilters.DirectoryName;

        //Begin monitoring.
        FileMonitor.EnableRaisingEvents = true;

    }

    private void ButtonStop_Click(object sender, System.EventArgs e)
    {
        //Begin monitoring.
        FileMonitor.EnableRaisingEvents = false;
```

```
                }

                private void DirMon_Load(object sender, System.EventArgs e)
                {

                }

        }
}
```