ASHESI UNIVERSITY COLLEGE

GUI-BASED FLEET PLANNING VISUALISATION TOOL

By

FRANK ANAMUAH-KOUFIE

Applied Project Report submitted to the Department of Computer Science

Ashesi University College

In partial fulfilment of Bachelor of Science degree in Computer Science

APRIL 2013

Declaration

I hereby declare that this the Applied Project Report is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:……

Candidate's Name:…….

Date:……

I hereby declare that the preparation and presentation of the Applied Project Report were supervised in accordance with the guidelines on supervision of Applied Projects laid down by Ashesi University College.

Supervisor's Signature:……

Supervisor's Name:……

Date:……

Acknowledgement

My sincerest thanks and gratitude go to Dr. Ayorkor Korsah, who has guided me through this project. I would also like to thank my mother, brother and friends – Elysia Amarteifio, Edem Diaba and Daniel Botchway for the myriad help and support they offered me throughout the project.

Abstract

Visualising data and interpreting it is a common activity we do on a daily basis. The human mind is able to identify patterns easily and extract information from visual contents much faster than it does from raw data. For this reason, visualisation tools of any kind serve to facilitate understanding of otherwise complex information or situations. Visualisation serves as a very useful tool for planning routes and scheduling tasks. In this project, visualisation is employed to increase the usefulness of a fleet planning application. The purpose of visualisation here is to provide a platform for easy formulation of problems and also for easy representation of solutions to these problems.

Table of Contents

1 Chapter 1: Introduction

1.1 Background

Fleet planning problems are problems that involve planning the activities of a number of actors that work together at varied collaborative levels. These actors may be a group of paramedics and ambulances at the scene of an emergency or a set of fire trucks fighting a bush fire. The activities performed by the actors may be performed individually or together by any number of actors as required. Several tools exist for defining and solving fleet planning problems. However, most of the available tools are highly customized to specific problems and therefore are not of much use in general situations. Another downside of fleet planning applications is the absence in many of the planners, of good visualisation tools for defining the problem to be solved.

Problem solving is an everyday activity we perform without knowing most of the time. While most problems are extremely easy and are solved unconsciously by the brain some are more advanced and require mathematics or programming to solve them. As problems increase in complexity, it becomes increasingly difficult to solve them using regular programming or mathematical techniques. Depending on the level of complexity, problems have to be defined using complex mathematical models to allow them to be solved with complex algorithms. For example, while 1+3(5-2) can be easily evaluated to 10 even without a calculator, finding the total surface area of a hollow octagonal pyramid cannot be

solved that easily. It requires a formula to be given and a careful methodology followed in order to arrive at the solution. Mathematics helps to solve several problems because most problems can be defined mathematically regardless of the complexity. A number of everyday problems such as the amount of money to invest in order to receive a certain yield at the end of the year or the right quantities of a product to manufacture in order to maximize profit and make efficient use of machinery can be solved by defining mathematical models for the problems. There are however a number of drawbacks of mathematical problem solving which leaves it to become a preserve of experts and enthusiasts only. One such drawback is the complex nature of the models themselves, making it difficult for users to understand the way in which they work and thus prevents users from defining their problems with those models.

1.2 Objectives

The objective of this project is to explore the usefulness of visualisation as a tool for simplifying problem definition and viewing the process of solving problems. It is aimed at developing insight in the field of complex problem visualisation, fleet planning and optimization problems. It is aimed at studying the benefits visualisation brings to complex problem solving and to evaluate how applications can be developed that derive from these benefits to create easy problem solvers for both industry-specific and everyday problems. It is the hope of this project that in the not too distant future, there will be a good number of visual problem-solving tools with significant patronage from ordinary individuals because they abstract

the mathematical models from users and only present an easy to understand interface for defining problems.

Particularly, this project is about creating a graphical user interface for a fleet planning program developed by Dr. Ayorkor Korsah for her PhD thesis. The application runs by taking file inputs and generating outputs to file. The objective of this project is to create visual tools for defining problems and viewing the solution. It is hoped that this will enhance the application's usability as it will open it up to a larger user base.

1.3 Motivation

My drive for undertaking this project stems from my interests in user interface and experience design. I believe that software of any form should be easy to use and have a large visual component. This makes it more interactive and fun to use. However, I do not believe that functionality should be sacrificed for aesthetics nor should aesthetics be sacrificed for functionality. For any good software a good balance of aesthetics and functionality must be present. With this project I hope to achieve exactly this by providing a graphical user interface for the fleet planner without causing any problems to the functionality.

1.4 Mathematical Problem Visualisation

As stated above, mathematics is of immense importance in everyday life because of the powerful tools it presents for solving both very basic and also very complex problems. Mathematical visualisation is a field of mathematics that allows one to understand and explore mathematical problems using graphical models. Originally, these models were plaster casts and paper drawings but owing to advancements in computer

graphics, visualisation is now done using computers. The field takes advantage of the human brain's ability to easily derive meaning from images. The need for visualization arose from the immense volumes of data that were becoming available as information technology advanced.

1.5 Linear Programming and Optimization Problems

Linear programming and optimization is a branch of mathematics that uses mathematical modelling in solving complex problems with varied solutions. The objective is to identify the best fit solution for the problem given all the constraints that are on it. In particular, the core application on which this project is being done is a fleet planning optimization programme. The application is aimed at optimizing the efficiency and usefulness of a group of robotic agents that perform a set of complex tasks. These tasks are of varying natures as some require independent execution, some shared execution and some bound by certain precedence and synchronization constraints. For this application two scenarios will be used – an emergency rescue situation and a farm harvest situation. These will be discussed in greater detail in chapter 3.

1.6 Outline of Dissertation

The chapters that follow outline in detail how the project is to be carried out. Chapter 2 reviews works done in the field of visualisation and provides scope for the project. Chapter 3 discusses the design stages of the project which addresses conceptual frameworks, design considerations and the general considerations that are made before any code is written. Chapter 4 discusses the actual implementation of the application and covers the technology platforms and the programming that was involved

in the process. The remaining chapters, 5 and 6, discuss testing and evaluation, and conclusions and recommendation respectively.

2 Literature Review

2.1 Introduction

Visualisation is a widely used tool in problem solving. It is used by teachers to aid teaching and learning, by scientists and researchers in interpreting data and generally to make concepts clearer. Despite its significance in this regard, it is mainly employed for the purposes of making sense of already solved problems. This chapter analyses a number of papers on visualisation aimed at gaining a deeper understanding of how visualisation as a tool is used, its advantages and disadvantages and why it is used mainly in as a tool for interpretation of results and not for defining problems.

2.2 Uses of Visualisation

"Visualization links the two most powerful information processing systems known—the human mind and the modern computer. A process, it transforms data, information, and knowledge into a visual form exploiting people's natural strengths in rapid visual pattern recognition" (Gershon & Eick, 1997). While it is true that we interact with large amounts of data all the time, it is also true that the raw data itself is of very little importance to us. Before any of this is useful it must be processed and some information extracted from it. Yet, even the refined information itself may be difficult to understand or make good use of because it is generally not easy to identify patterns in mathematical or statistical information.

Visualisation serves as that bridge that enables us to make sense of the vast amounts of data that engulf us at all times. Visualisation is used in a variety of industries including scientific research, finance and investment, healthcare, agriculture and education. In research, visualisation helps scientists to make sense of abstract concepts such as fluid dynamics and activities of volcanoes and earthquakes. It is used in healthcare for medical imaging and diagnostic purposes and in chemistry for investigating the behaviour of chemicals under certain conditions. In mathematics especially, visualisation has been useful from primary school applications such as Venn Diagrams to applications of higher education as in polynomials and complex number problems.

Without visualisation it would be impossible to extract as much information from data as we do (van Wijk, 2005). In all of these areas a general pattern seems to emerge – that visualisation is used after some data or information has been processed and needs to be analysed.

2.3 Scientific visualisation vs. Information Visualisation.

Computer visualisation has a myriad of applications in both scientific and industry research and in real-world applications. It is used in visualising traffic flow, task scheduling, medical imaging and natural disaster simulation among others. In "The Value of Visualisation" by Jarke van Wijk (van Wijk, 2005), he retraces the history of scientific visualisation and makes a case for how the needs of the scientific community have been key determinants in the kinds of visualisation technologies that are developed. He also argues that visualisation as a maturing field has transcended the bounds of scientific applications only, to become a

general tool and thus has led to a drive for user-centred visualisation tools (van Wijk, 2005). According to him, all visualisation is scientific at the core and develops through similar processes.

This is contrary to the notion presented by Eick and Gershon (Gershon & Eick, 1997)who argue that recent advances have led to the field of information visualisation which is unique in its development and approach. While scientific visualisation like all forms of emerging technologies develops in a four-stage process, the same cannot be said of information visualisation. "An emerging discipline progresses through four stages. It starts as a craft, practiced by artisans using heuristics. Later, researchers formulate scientific principles and theories to gain insights about the processes. Eventually, engineers refine these principles and insights into production rules. Finally, the technology becomes widely available. For information visualization, however, these stages are happening in parallel…" (Gershon & Eick, 1997). It is the belief of these writers that the parallel nature of development presents a challenge to the visualisation industry in meeting the needs of this fast paced development. As a consequence, some stages of the process are not thoroughly covered during the implementation of the tool and ends in low user satisfaction.

The need to recognize information visualisation as a unique field from scientific visualisation is of immense importance as it determines how the industry will be able to meet user needs. Without this understanding, it will be impossible to produce visualisation tools that appeal to the user community. This ties back in with van Wijk's assessment of effective visualisation which states that the quality of a visualisation tool lies more

in its usability and user acceptance than in its efficiency in performing its tasks (van Wijk, 2005).

2.4 The Science and Art of Visualisation

Historically, science and art have remained two very distinct fields with very distinct interests and modes of operation. Science is about data, research, facts, methodology and structure. Art on the other hand leaves room for creativity, adaptability and unconstrained experimentation. Regardless of their differences, both fields have major roles to play in developing visualisation tools (Sorensen, 1989). Sorensen mentions that it may not after all be true that arts and science are distinct fields considering that some of the best known scientists have doubled as artists. Leonardo da Vinci, Archimedes and Benjamin Franklin are but a few of these scientist-artists. He believes that the success of these in both fields of endeavour is proof of the importance of the collaboration between arts and science especially in the field of visualisation.

Popper (Popper, 1994) extends this argument by adding a cultural and psychological context to visualisation. He observes that neither the scientific accuracy and efficacy nor the usability and aesthetics of a tool completely determine its acceptability by users. It is important to understand the user and the context of use of an application in order to design an application that meets that need (Popper, 1994).

2.5 Challenges of Complex Visualisation Tools

Visualisation as a tool, like all others, has a number of drawbacks despite its immense proven benefits. Firstly, having developed from a scientific and research background and having initially been a preserve of

specialists, visualisation tools tend to be quite cumbersome and difficult to understand and use. Most of these tools remain as tools for experts and do not draw the attention of the ordinary person due to their complexity. Another challenge with visualisation is concerned with its parallel nature of growth described in (Gershon & Eick, 1997). According to this analysis, it is more difficult to standardise visualisation principles or guidelines because all the sectors involved are constantly modifying their views on how things are to be done thereby reshaping how visualisation is to be perceived. This also means that any visualisation tools developed are only reliable over a short time period as they may be subject to both minor and major changes in design and even engineering. In the absence of any regulations on standard processes to follow, it is difficult to provide any visualisation solution that will satisfy user needs. Another problem with visualisation is the cost involved. Graphical processes are CPU-intensive systems and as such use significantly high percentages of the computer's memory and processing power. This makes the visualisation tool slow and also slows down the entire system, preventing other system tasks from being performed (van Wijk, 2005).

2.6 Summary

From the works discussed above it is evident that visualisation is a young and very diverse field. It is impossible to define it within any particular field of study as its use covers several fields. Scientific visualisation is the parent of all visualisation but it must be recognised that other quite distinct forms of visualisation, especially information visualisation, have emerged from this field and are quite unique from each other. Also visualisation must be understood as embodying both arts and science and

must reflect in how works relating to this field are done. Another observation made is the strain that visualisation tools have on a system's memory and processing power.

To develop an effective visualisation tool the above considerations have to be made. Disregard for any of the issues discussed here can cause the tool to become difficult to use or even render the tool entirely useless.

3 Design

This chapter addresses the major design considerations made in the development of the application. It discusses the technologies and methods used in development and the reason for those choices. It also discusses the target users, platforms and limitations to the design process.

3.1 Understanding the application

Before designing the application and throughout the implementation, the expert knowledge and opinions of Dr. Korsah, the developer of the fleet planning component were sought. The purpose of these consultations was to gain as much insight as possible into the problem the application is designed to solve. It was to understand fleet planning and optimization in the given context of the project and to ensure that the developed solution meets the identified needs.

3.2 Programming Language

The first design consideration made was the choice of programming language. The criteria for selecting an appropriate language include robustness, speed, portability, development time, support, aesthetics and performance. I began by shortlisting 5 languages that are popular choices for visualisation and graphical interface programming – Adobe Flash/Action Script[1], HTML5[2], C++, Java and Python. These languages are compared below based on their performance with the above criteria.

[1] Adobe Flash and Action Script (the scripting language for Flash) are used interchangeably in this paper.

1. **Robustness:**

Robustness as used here refers to the ability of programmes written in a language to withstand excessive pressure and perform even with low system resources such as CPU time and memory. HTML5 and Flash do not perform well on robustness because they are not used for standalone applications and depend solely on the performance of the browser or program in which they run to perform. C++ and Java on the other hand have a long history in being very robust and adaptable in performance. Python's robustness is good but does not compare to Java or C++.

2. **Speed:**

Speed measures the rate at which instructions are converted from the high level language to low level languages for the computer's use. C++ is the fastest because of its closeness to machine language and is the best option for programming where speedy execution is required. Java is slower than C++ because it has a virtual machine layer that first converts the code to platform-dependent bytecode before being translated to machine language. HTML5 (Javascript), Python and Action Script are much slower because they are scripting languages and have to be compiled by the underlying language platform before execution.

3. **Portability:**

Portability measures the ease of developing applications for multiple platforms and environments with the selected language. Java is the most portable language because it runs in the Java Virtual Machine model which makes it possible to run the same code on different platforms without any

[2] HTML5 as used in this paper refers to the group of web technologies – HTML5, CSS3 and JavaScript, commonly referred to as HTML5. Where reference to a particular feature is being made, it is placed in a parenthesis next to HTML5.

modifications. This portability is extended even to web and mobile platforms with Java Play and Java ME respectively. HTML5 is also highly portable but is limited to web and mobile platforms except for Windows 8, on which native HTML5 applications can run. Python is also platform independent and can run on web and mobile devices as well. However, developing graphical standalone Python applications is quite cumbersome. Flash/Action Script can run in several environments including desktop and web as well as high-end mobile platforms. However, because it is a legacy application it requires the installation of plugins on the host system in order to function. This is a major drawback on the portability of the language.

4. **Development Time:**

This measures the amount of time that is spent on developing applications in the chosen language. Python, Action Script and HTML5 have much faster development times because most of the functionality implemented is abstracted and the programmer only interacts with very high level functions. Java is slower because functionality is not implemented on the levels of the above mentioned languages. However, because of its platform independence Java programming is significantly faster than C++ programming where code has to be written uniquely for every platform that will run the application.

5. **Support**

This measures how much information, help and support is readily available for development in the chosen language. Java and C++ have enormous resources available for developers using those languages. There are unlimited web communities and forums with a rich knowledge base of

information regarding the languages. Java also has a well-defined and up-to-date API documentation which makes development in the language very simple. Python is a much younger language and although it has a well-documented API, it has not yet gathered the following that Java and C++ enjoy. Action Script is difficult to use and requires in-depth knowledge of Flash animations. It is also proprietary software and does not have as much freely available resources as do other languages. HTML5 is a new technology which is still in development. Coupled with the fact that it is a mix of different technologies, it is difficult to obtain standard ways of developing or obtain useful resources for development on the platform.

6. Aesthetics:

Aesthetics measures the possibilities of developing aesthetically pleasing graphical interfaces as well as the ease with which this can be done. Flash scores high on aesthetics because of the high quality and beautiful interfaces and applications that it can develop. It also provides the developer with the most customizability and puts total control of the interface in the programmer's reach. HTML5 (CSS3) is also capable of producing very beautiful graphics applications although it is incapable of developing standalone applications. Java (javax.swing, java.awt, and JavaFX) presents a strong library for graphics programming although the possibilities are not as fancy as those provided by Flash and HTML5. However, the addition of third-party libraries provides means of developing beautiful interfaces. Python and C++ are not excellent choices for graphics programming because they lack easy to use graphics libraries.

7. **Performance:**

This measures how well applications written in these languages make use of system resources and how well they perform generally. C++ is highly efficient because the programmer is forced to keep a close eye on memory usage and procedure calls. Generally programming in C++ assumes that system resources are scarce and must be managed efficiently. Java also does well with performance as it has functionality to check for resource usage. The Java Virtual Machine for instance has methods that optimize the code before it is compiled and executed. It also has a garbage collector which monitors and ensures that memory is being used effectively. Action Script on the other hand makes excessive use of memory and easily slows the system down. It creates strain on the processor when executing Flash animations and can affect the performance of other applications using the system's resources. HTML5 is a lightweight technology and makes very minimal use of system resources. However, the true resource usage depends on the underlying platform. Python also makes very resourceful use of the system and does not overwhelm the system.

Judging from the above criteria, I chose to use Java because it is generally robust, fast, portable, has an adequate development time and a large support system. It is also easy to use for graphical programming and performs excellently. Also, Java integrates easily with C++, which is the language used to develop the fleet planner component based on which this program is being developed.

3.3 Operating System Platform

The purpose of this application is to make it easier for people to use and by so doing achieve a wide user base. To achieve this, the application is designed to run on all major operating systems. This includes most known distributions of Linux[3], Windows and Apple's OSX. The fleet planner component of the application was developed for Linux and therefore the general application will ideally be deployed in Linux. However, considering the numbers of computer users who use Windows, the graphical component of the application being developed is designed to run perfectly in the Windows environment. Since the fleet planner component is currently designed just for Linux, the application will make use of Java's Remote Method Invocation (RMI) and Common Object Request Broker Architecture (CORBA) architecture to run in a distributed architecture. Ideally, the graphical component will be deployed on Windows and will run remotely with the fleet planner component deployed on Linux.

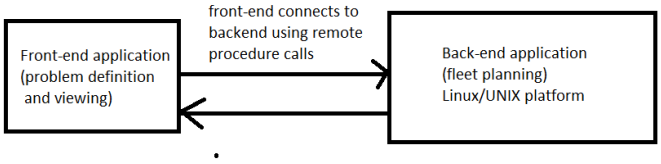

Figure 3.1 Application running on separate platforms connects with RPC

[3] Linux as used in this document refers Redhat and Debian distributions of the Linux kernel. Specifically Ubuntu, Fedora and CentOS are the particular builds of Linux being referenced.

3.4 User Considerations

This application places high value on the end user and as such adopts a User-Centred Design approach to developing the application. This means that user needs will be paramount in making any design decisions regarding the application. The application is designed for two groups of users. The first group is made up of scientists, programmers and mathematicians who already understand fleet planning and optimization to a large extent. This group of users are not traditionally concerned with the look and feel of applications and will be primarily concerned with the value the GUI-based programme offers over the console-based applications they are used to. The application is also designed for non-specialists who are interested in fleet planning for varied reasons. This group of users are concerned with the total visual appeal of the application but may not be particularly concerned about the performance of the application in generating the problem files. The success of this application lies in being able to meet the core needs of both user groups.

3.5 Performance vs. Aesthetics

While the goal of this project is to design an application that is highly graphical in its dealings with the user, it is also important to recognize the high demands graphics has on processing power. This means that as the application becomes more intuitive to use and has a lot of visuals it puts greater strain on the processor and lowers the applications performance. This will make the application slower and make it less useful than the non-graphical equivalent. This defeats the purpose of the project and as such

it is important to ensure that there is a reasonable balance between the application's graphical appeal and overall system performance.

3.6 Software Requirements

The following section details the various requirements that the application must meet. It details the core requirements as well as non-functional and general requirements that it aims at satisfying.

3.6.1 Functional Requirements

- The application will read a predefined problem file as raw text and generate a visual representation of the problem.
- The application will provide a graphical tool for the user to visually define a problem which will then be converted to a problem file saved in raw text format.
- The application will read a solution file as raw text and generate a visual representation of the solution.
- The application will visualise in a timeline view, the planning process developed by the fleet planning component.
- The application will animate the solution process as developed by the fleet planning component.
- The graphical tool will allow the user to add agents to a problem and also to edit the properties of these agents or delete them entirely.
- The graphical tool will allow the user to add tasks to the problem and to edit the properties of the tasks.
- The application will show the precedence and synchronization constraints that exists among the agents in the system.

- The system must save problems to file before exiting.

3.6.2 Non-Functional Requirements

- The application must be able to maintain the state of the user's defined problem and be able to automatically restore this session after an unexpected application close.
- The application will have a configurable icon set which can be set by defining a configuration file.

3.6.3 General Requirements

- The application will run in Linux but must be able to run on all other common platforms (Windows and OSX).
- The application must make minimal use of system resources and should be able to run alongside other applications without significant performance reductions.
- The application must have appropriate firewall permissions to use the system's network resources.

3.7 Context of Use

The application is a general fleet planning application and can be used to solve a wide array of fleet planning and optimization problems. These include carpooling problems, courier services and other delivery routing problems and assembly line organisation problems. However, it is designed for two particular contexts – emergency rescue operations and agricultural harvests. Under emergency rescue, the application is developed to plan the manner in which an emergency rescue operation is to be carried out in order to maximize the efficiency of the rescue team and thereby save lives and resources. In the second scenario, the planner

develops an optimal plan for a farm harvest operation involving a number of combine harvesters, farm trucks and silos. The planner generates a plan that maximises the output of the trucks and harvesters and generally makes the agricultural harvest both efficient and faster.

Figure 3.1 A group of combine harvesters on a farm

Figure 3.2 Paramedics attending to a an accident victim

3.8 Application Classes

This section outlines the structure of the application. It describes the Object-Oriented model of the application giving general details of the classes and high level methods and properties of those classes. The application has ten main objects – problems, worlds, agents, tasks, subtasks, attributes, locations, constraints, solutions and solution graphs.

3.8.1 Problem

A problem is an object that embodies all the actors in the problem definition. It is comprised of a problem world and lists of agents, attributes, tasks, subtasks, attributes, locations and several constraints. An example problem may be an emergency rescue which will involve ambulances, clinics, paramedics and injured persons.

3.8.2 Problem World

This class defines the boundaries within which the problem exists. It defines the four spatial limits of the problem – minimum x, maximum x, minimum y and maximum y coordinates of the area within which all the problem actors are contained. It also defines a plan horizon. An example world may be the site of an accident or a farm.

3.8.3 Agent

The Agent class defines the moving actors in the planning problem. It describes the entities performing the actions that are defined in the problem. For example, an agent may be a paramedic, an ambulance, a truck or combine harvester. Agents have a list of attributes that describe their functionality, start and end locations of their actions, start and end times of their actions as well as the speed at which they act. An agent also has maximum route duration and length, capacity and cost of waiting.

3.8.4 Task

The task class defines the high level actions that are to be performed by the agents in the problem. Tasks are complex activities that are composed of simpler tasks and include such activities as transporting an injured person to a clinic, administering first aid to an injured person or transferring farm produce to a silo. Tasks also have a list of attributes that define their nature and how they must be carried out. It also has a list of subtasks that comprise it, a reward for completing the task and the maximum time within which to complete the task.

3.8.5 Subtask

This class defines the low level individual activities that make up tasks. Subtasks are the actual activities that are performed by the agents in the system. For example, lifting an injured person onto a stretcher and driving to a clinic are two subtasks that will comprise the task of delivering an injured person to a clinic. Each subtask has a service time which is the time it takes to perform it, an earliest start time and a latest start time which defines the window within which the activity must be performed. It also maintains a list of locations where it may be performed and capacity requirement which must be satisfied by an agent that will carry out this task.

3.8.6 Attribute

This class defines a property that must be possessed by an agent or a task in the system. The attribute defines an agent's ability to perform a given task and the attributes a task requires from an agent before it can be performed that agent. As an example Medical, Transport or Harvest may be defined as attributes for an agent administering first aid, the task of delivering a sick person to a clinic or the task of harvesting farm produce respectively.

3.8.7 Location

The location class defines locations in the system where activities may be performed or where agents may temporarily lodge before they perform certain tasks. Locations have x and y coordinates which are points in a Cartesian coordinate system. These points are transformed to maintain

integer coordinate pairs in order for them to be plotted on a canvas. The coordinate system of Java is discussed in greater detail in chapter 4.

3.8.8 Constraint

Constraints are a broad group of constraints that exist among the actors in the problem. The purpose of the constraints is to limit the solutions that are generated by the planner. Five constraint types are relevant in this application. These are outlined below:

1. Precedence Constraint:

   This type of constraint puts a limitation on the order in which subtasks may be performed. It is defined between two subtasks and limits the planner to ensure that one of two subtasks is always performed before the other. For example, in a rescue situation, an injured person must receive first aid treatment before they are transported to a clinic for further treatment.

2. Synchronization Constraint:

   This constraint limits the planner to ensure that two subtasks are performed at the same time. It is defined among any number of subtasks and forces all solutions generated to ensure that these tasks happen within the same time window. As an example, in a harvest situation a combined harvester emptying contents into a truck and the truck loading the contents must happen concurrently.

3. No Overlap Constraint:

This constraint is the opposite of the synchronization constraint. It ensures that two activities are not performed together under any circumstance. This is useful if two activities must not happen within the same time window. In an emergency situation for example, the ambulance must not be in motion at the same time an injured person is being placed in it.

4. Location Capacity Constraint:

This type of constraint ensures that activities being performed in a given location are within the capacity limits of the task being performed. As an example, it must be ensured that the number of patients delivered to a particular clinic does not exceed the number of patients the clinic is able to attend to.

5. Proximity Constraint:

Proximity constraints are defined to check that some group of agents or tasks are within a certain distance of each other. It is also used to ensure that some actors are not within a certain distance of each other. This is useful for example when it is necessary to ensure that two paramedics are not working in the same region when there is a location that does not have any paramedics attending to injured people.

3.8.9 Solution

The class defines a single subtask to be performed by an agent and the conditions under which this task must be undertaken. It defines an agent to perform a task, the subtask to be performed, the location of the activity

and the amount of time to wait before carrying out the task as well as the time the task must be executed. An example solution would be that a paramedic must travel to an accident location, wait for an ambulance and put the patient on the ambulance in five minutes.

3.8.10 Solution Graph

The Solution graph defines the solution output by the fleet planner. It is the set of all solutions present which means it is a set of all agents and all the subtasks they must carry out. For the purpose of simplicity, all the solutions within a graph that belong to a particular agent are grouped in another object, the *Solution Set.* The solution set is only present for simplification purposes and does not affect the application in any way.

4 Implementation

This chapter discusses the implementation process of the application. It details the technologies used and the methods used in developing the product. It also outlines how the application functions.

4.1 Technology and Development Tools

The fleet planning component of the application which was already developed as mentioned earlier was developed in C++ and used the Qt library for the initial graphical modules. The graphical user application which is the subject of this project however, is built purely in Java developed with the Java SE 7 toolkit. The reason for using Java is discussed in chapter 2 above. The application makes extensive use of the Java Swing library, Java 2D Graphics Library, Collections and multithreading. Some interfaces were designed using Netbeans Platforms developed by Oracle Corporation and JFormDesigner developed by FormDev Software GmbH.

It is built using Netbeans IDE version 7.2 for Windows but is tested on other target platforms as discussed in chapter 5. The choice of Netbeans stems from the extensive integration of Netbeans and Java owing to the fact that they are developed by the same organisation. Netbeans has an up-to-date integrated Java API documentation and has plugins to generate Javadoc on the fly. It also has an extensive plugin library to facilitate development, testing and debugging. It also integrates neatly

with JFormDesigner which is a Java Form design plugin used in this application.

4.2 How It Works

The application works in a three step process – Problem definition, Problem Solving and Solution display.

4.2.1 Problem Definition

When the program begins execution, the first window gives the user a choice to either start a new instance of the program or to load a pre-existing problem from file.

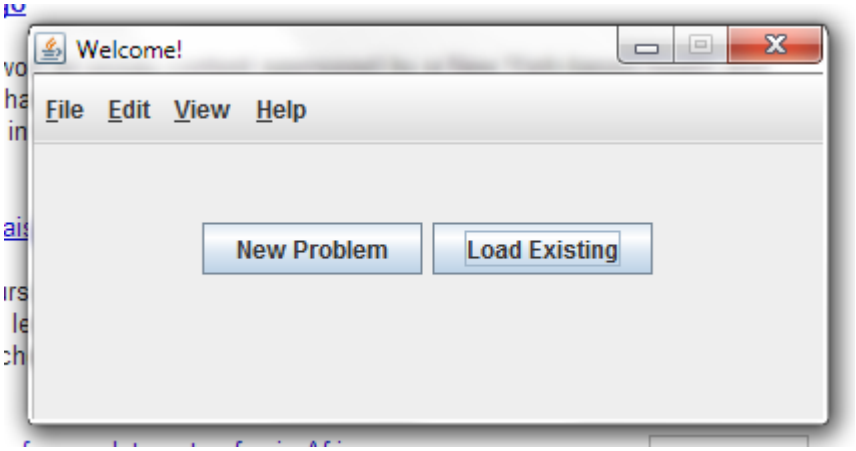

Figure 4.1 Application start window

1. **Loading a pre-existing file:** The application opens a file chooser window allowing the user to navigate to the problem file of interest. Problem files are saved in a simple human-readable txt file format. A file handler process runs in the background which checks the validity of the file. It then reads the contents of the file and generates a problem object from it. This problem object is used to

initialize the application's main window which displays a visual representation of the problem on a canvas.

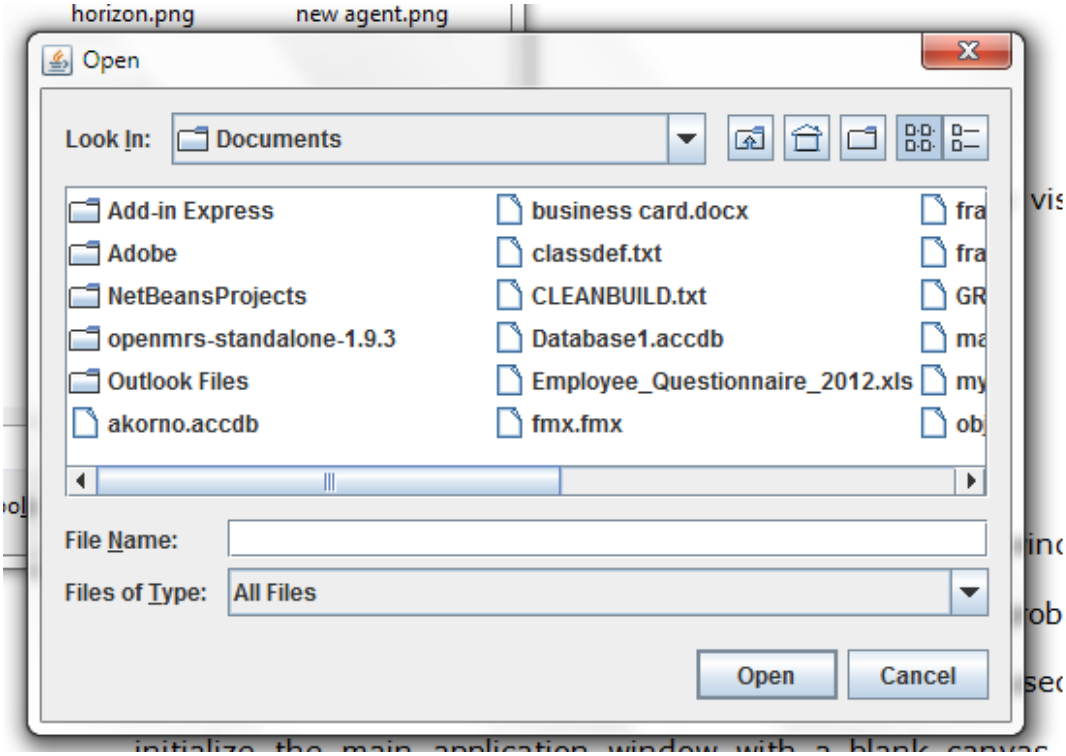

Figure 4.2 Dialog to open existing application.

2. **Starting a new problem:** The application opens a popup window that requests the user to define the coordinates of the problem world. A world object is created from these values and is used to initialize the main application window with a blank canvas. An empty problem object is defined for the new instance.

On the canvas, the user may define agents and tasks by clicking on the canvas. The window presents the user with a number of menus that allow the user to define attributes, tasks, subtasks, agents, locations and constraints of the current problem. The problem can be saved at any point

in time by clicking the save problem button. This generates a simple text file in txt format form the problem object defined.

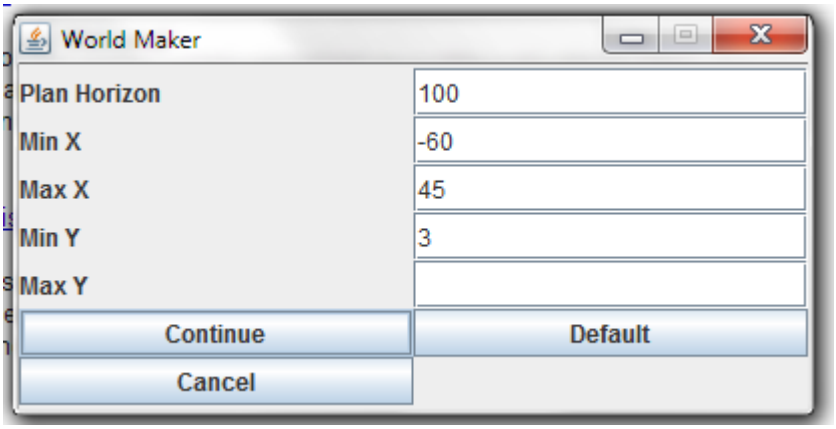

Figure 4.3 Window for defining dimensions of problem world.

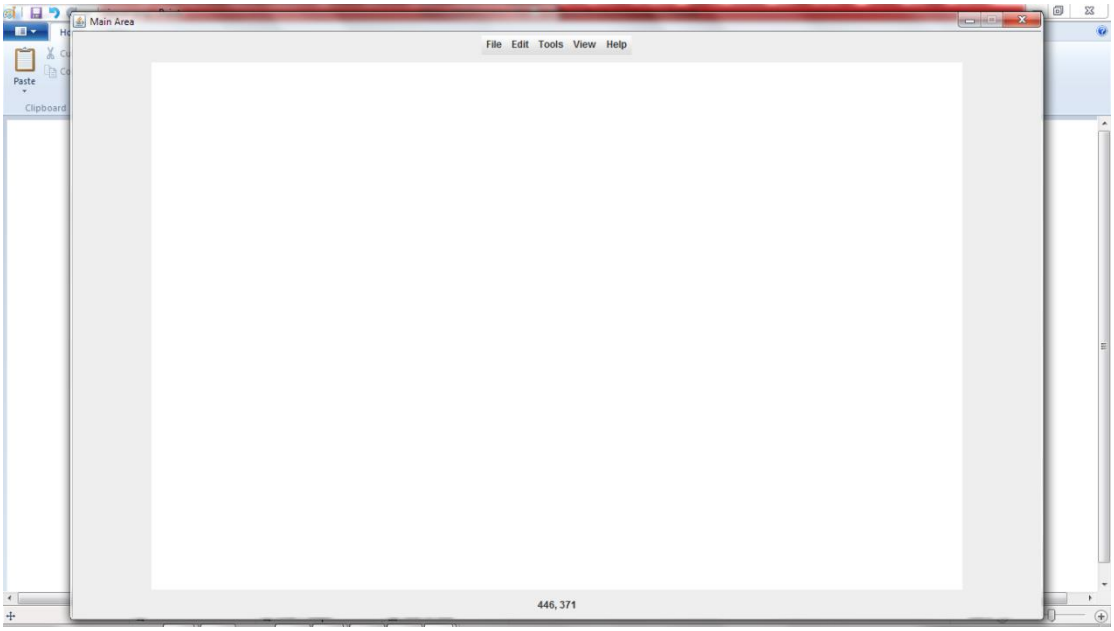

Figure 4.4 Blank Application window (for new problems.

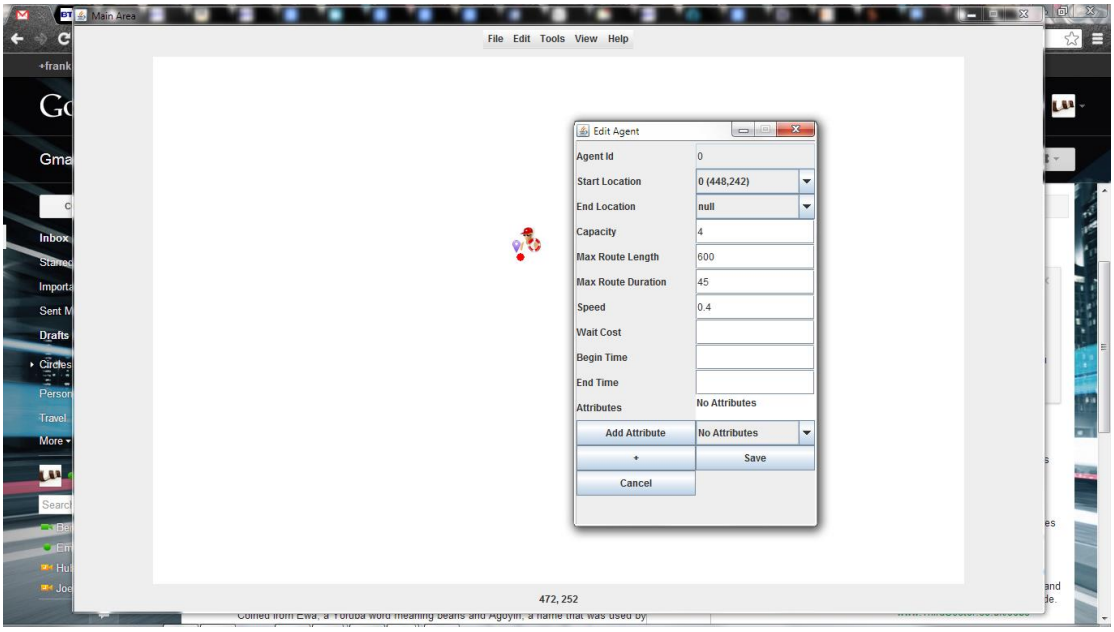

Figure 4.5 Adding an agent to the problem world.

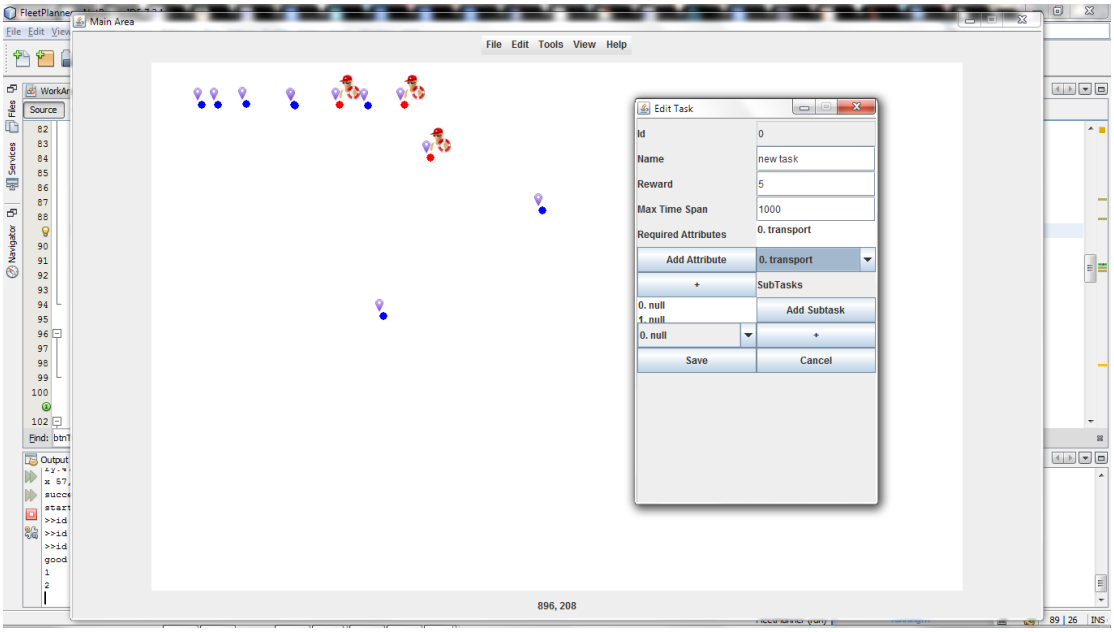

Figure 4.6 Adding a task to the problem world.

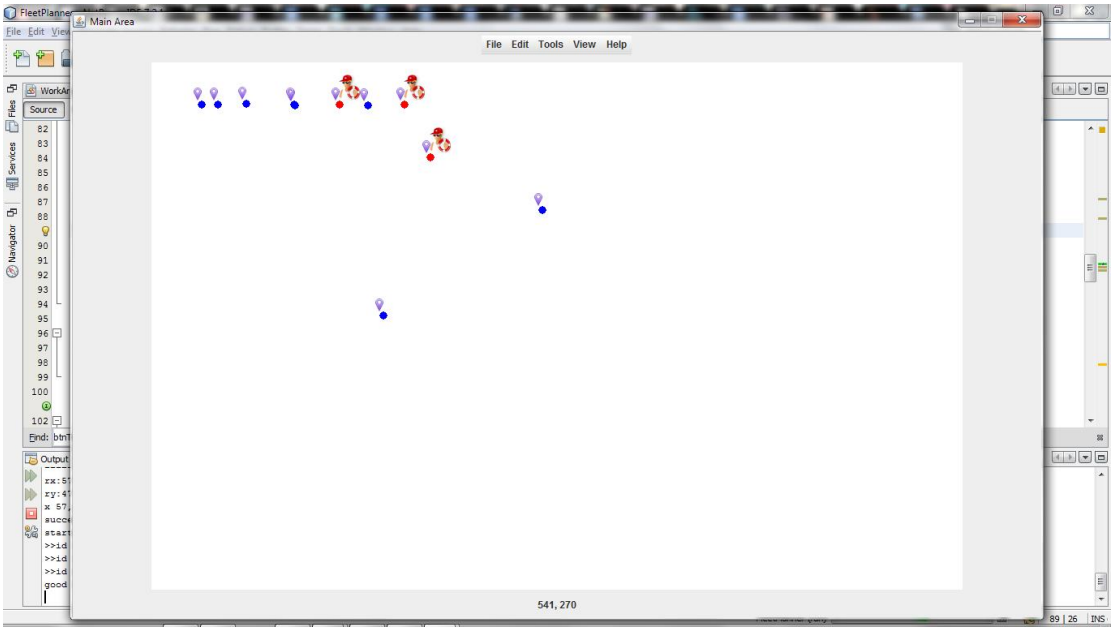

Figure 4.7 Problem world with agents and task locations

Figure 4.8 Edit agent popup menu

4.2.2 Solving the problem

After a valid problem file has been created or loaded, the user can progress to solving the problem. This is done by making a system call to the fleet planner component with the problem file as an argument. As discussed earlier, this may be a local call in a Linux system or a remote

procedure call to the Linux system housing the fleet planning component in the case of a Windows system.

The fleet planner may run in the background and only return a set of solution files on completion or be set to return values as the application is running. The data which are the branch and bound values from the fleet planner can be visualised as a graph as the application runs. This gives the user a feel of the progress of the optimization process. Additionally, given this information, the user may choose to end the optimization when the branch and bound parameters satisfies the user's requirements. A valid solution can be generated at this point. This saves time since it is not always possible or even necessary to obtain the most optimal solution to a problem.

4.2.3 Visualising the solution

After the planner determines the optimal plan for carrying out the tasks defined it generates a series of output files. These files are plain text txt files. The output files are used again by the graphical application for visualising the solution. This is done in two ways – visualizing the solution as a timeline of events and visualising the solution as an animated display of the overall solution process.

1. TIMELINE: The solution timeline shows agents and the activities they carry out starting from time t=0 to the length of the sum of the non-overlapping activity durations in the problem.
2. ANIMATED SOLUTION: This feature shows an animation of the solution in a real-time format. The solution is displayed in the same window as the defined problem and animates the process of agents carrying out their activities in the shared world. Each agent is

animated in a different thread of execution using Timers to track the duration of each agent animation.

4.2.4 Customizability

The application has features to allow the user configure the application's look and feel and behaviour to meet certain needs. For instance, the user is able to select whether the application should maintain the Java look and feel or switch to a native look and feel which is more comfortable for certain user groups. Also, the user is able to select whether an agent should be represented in the world with an ambulance icon or a doctor or a tractor depending on the kind of problem they are dealing with or their personal preferences.

This is possible by defining a configuration file (for advanced users) which will be read by the application once it runs and quickly set up the interface accordingly. For non-advanced users, there is a menu option to customize the application which gives the user an opportunity to do this. If no customizations are made the application runs with a basic configuration defined in a "default.config" file.

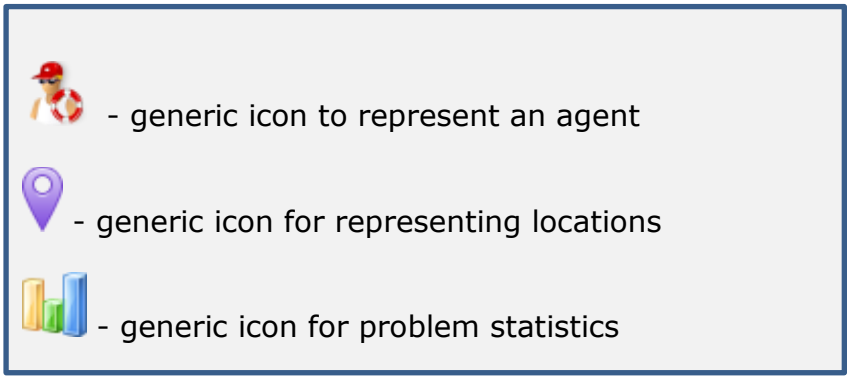

Figure 4.9 Sample icons used by application.

4.2.5 Mapping Points

The application has a number of algorithms it employs in scaling the user-defined world into a viewable world on the application work area and the solution area. In pixel-based mapping two things are relevant. First, it is important to be able to convert any integer value on the graph to a floating point values which are the actual values used by the planning component. Secondly, the application must define an appropriate scale given a problem and rightly transform each point in a problem world into a viewable point on the canvas. For this purpose a *Point* type was defined which is capable of maintaining both the displayable integer x and y coordinates of the point as well as the actual floating point x and y values used by the fleet planning component.

The problem of plotting points is extended by the unconventional nature of Java's canvas coordinate system. Unlike the Cartesian plane that moves from left to right from negative to positive x values and from bottom to top from negative to positive y values, the Java system has no negative values. The coordinate begins from 0 and increases in x value from left to right and starts from 0 and increases y values from top to bottom. In addressing this problem, the *Point* object was modified to include a *type* property that defines which region of the Cartesian plane the point would lie in. This necessitated the creation of an *Origin* type which is defined for each *World* to help orient the points to be plotted. The Origin inherits from the Point class and extends it to add an *AxisType* property which defines the kind of axes the Origin is defined on. An *AxisType* determines which of the four sections of the Cartesian plane the sets of points in a problem world exists.

With these objects it is possible to define appropriate methods to transform points from floating point values used by the fleet planner to integer values used by the graphical display.

5 Tests and Results

This chapter discusses the testing and evaluation the application was subjected to and also discusses the results of research carried out about the topic.

5.1 Application Testing

5.1.1 Unit Testing

Unit testing of the application was done on an Intel dual core Pentium 64bit computer running on Windows 7 Ultimate. This is the same computer on which the application was developed. These tests were carried out during the development stage in an iterative process. The purpose of the unit tests were to check that all intended functionality are implemented correctly. It was also intended to ensure that all identified bugs are fixed before the application is deployed.

5.1.2 User Testing

User tests were also carried out to ensure that the application satisfies the basic needs of target users and also to ensure that unforeseen needs of users are addressed. Three user groups were tasked with testing the application at several stages of development. The first group, a group of five computer science students were tasked to test the robustness of the application. This group run the program evaluating the logic and checking the program's response to invalid logical applications. From these tests,

the logic and flow of the application was redesigned to make it stronger and withstand certain logical errors.

The second group, a random group of 5 students were tasked to evaluate the usability of the application. This group was briefed extensively on the functionality of the program and tasked to point out any challenges faced in using the program. This group was also supposed to test the program's response to invalid inputs. From these tests the program's error tolerance was rechecked and the logic and flow of the application rechecked. The new design had greater error tolerance and had easier flow which users could understand.

The final group of testers was group of five students composed of computer science students and students who admitted to having low computer literacy. This group was simply tasked to crash the program by any means possible. This group was presented with very little information on how to use the application and then were set forth to break the program.

5.2 Focus Group Discussions

Three focus group sessions were held to discuss requirements and design considerations of the application. The first session was held at the beginning of the design process to brainstorm desired functionality, user expectations and limitations. This group included computer science students and design students and it was expected that their knowledge in their respective fields would inform the design choices to be made.

The second session was held halfway through implementation to ascertain whether the design was in line with requirements that had been

previously established. This session was also aimed at checking if any new requirements or ideas had come up that would affect the design and to ensure that these requirements are met. This group had the same composition as the first group.

The final session was held near the completion of the application with a group of users with no interest in programming. This session was to evaluate the acceptance of the application by a user group that had very little knowledge about the application. Results from this session would help refine the application by incorporating the needs of a user group that had not been considered earlier in the design process.

On the whole, the focus group discussions helped to identify a greater number of requirements that would not be easily identified by programmers and scientists. Also, these sessions were aimed at gaining a deeper understanding of various software users and how they view and interact with software applications. The sessions helped to validate and disprove some assertions made about software users and helped to establish new insight beyond what is established in the reviewed literature.

6 Conclusion and Recommendations

6.1 Limitations

The main limitations of this application were the inability to animate the solution process and the unavailability of the fleet planning component of the application. These components describe a bulk of the application's functionality and therefore limit the possibilities for using the visualisation tool in a wide array of situations.

6.2 Further Works

The future of this application is to implement all the functionality that were thought of but not implemented in this version. These include the animated solution viewer, remote method invocation components and the configurable icon sets. Also, work will be done to enhance the performance of the existing application.

6.3 Conclusion

The objective of this project was to develop a visualisation tool that will aid in solving fleet planning problems. This involved both a tool to assist in defining the problem itself as well as a tool to visualise the solution once the planner was done with its work. Although not every component is finished as desired, the application is able to perform some basic roles and is in line with the objectives set for it. In conclusion, it is my belief that this application will serve to reduce the stress involved in defining problems for fleet planning thereby increasing the overall efficiency of the process.

7 Bibliography

Gershon, N., & Eick, S. G. (1997). Information Visualisation. *IEEE Computer Graphics and Applications*, 29-31.

Korsah, G. A. (2011). *Exploring Bounded Optimal Coordination for Heterogeneous Teams with Cross-Schedule Dependencies.* Carnegie Mellon University, Robotics Institute, School of Computer Science. Pittsburgh: Carnegie Mellon University.

Popper, F. (1994). Visualization, Cultural Mediation and Dual Creativity. *Herausforderungen für die Informationstechnik*, pp. 405-415.

Sorensen, V. (1989). *The Contribution of the Artist to Scientific Visualization.* California Institute of the Arts, School of Film and Video. California: San Diego Supercomputer Center.

van Wijk, J. J. (2005). *The Value of Visualisation.* Eindhoven: Technische Universiteit Eindhoven.

Appendix A: Terminology

- **Agent:** An actor in the problem capable of performing an action.
- **Attribute:** A property that describes what an agent is capable of and what a task requires of an agent that will perform.
- **Capacity:** A property of an agent or location that determines how much of an activity or agents it can hold or perform at a time.
- **Constraint:** Any limitation on how an activity is to be performed.
- **Location:** A point on a Cartesian plane which represents the areas where an agent or a task are executed.
- **Plan horizon**
- **Problem:** An object representing a problem.
- **Reward:**
- **Solution:** A line in the output file of the fleet planning.
- **Subtask:** A unit activity that can be performed by an agent.
- **Task:** A complex activity comprised of smaller straight-forward activities.
- **Wait Cost:** The cost incurred by an agent while it waits to perform a task.
- **World (Problem World):** A bounded region within which a problem is defined.

Appendix B: Sample Problem Definition File

```
GLOBAL
x_min -100
y_min -100
x_max 100
y_max 100
plan_horizon 1000

n_attributes 2
n_locations 11
n_agents 3
n_subtasks 24
n_tasks 15
n_prec_constraints 6
n_synch_constraints 0
n_no_overlap_constraints 0
n_mutex_constraints 0
n_loc_capacity_constraints 0
n_proximity_constraints 0


ATTRIBUTES
id 0 name transport
id 1 name medical


LOCATIONS
id 0 x -15.44 y 14.34
id 1 x -22.37 y 14.22
id 2 x 67.75 y 34.34
id 3 x -61.46 y -14.34
id 4 x 45.30 y 14.34
id 5 x 95.39 y 54.34
id 6 x 34.37 y -14.76
id 7 x -56.26 y 94.34
id 8 x 34.24 y -14.34
id 9 x -52.32 y 14.78
id 10 x 11.34 y -14.34


AGENTS
id 0 start_loc 2 end_loc -1 capacity 3 max_rte_len 12
max_rte_dur 1000 speed 0.35 wait_cost 0 begin_t 0 end_t 1000
n_attribs 1 attrib 0
id 1 start_loc 3 end_loc -1 capacity 2 max_rte_len 11
max_rte_dur 300 speed 0.1 wait_cost 1 begin_t 4 end_t 400
n_attribs 2 attrib 0 attrib 1
id 2 start_loc 4 end_loc -1 capacity 0 max_rte_len 14
max_rte_dur 400 speed 0.4 wait_cost 0 begin_t 0 end_t 200
n_attribs 1 attrib 1


SUBTASKS
```

```
id 0 task 0 earliest_start 0 latest_start 995 service_t 5
req_capacity 1 n_loc_choices 1 loc 5
id 1 task 0 earliest_start 5 latest_start 1000 service_t 5
req_capacity -1 n_loc_choices 1 loc 0
id 2 task 1 earliest_start 0 latest_start 995 service_t 5
req_capacity 1 n_loc_choices 1 loc 6
id 3 task 1 earliest_start 5 latest_start 1000 service_t 5
req_capacity -1 n_loc_choices 1 loc 1
id 4 task 2 earliest_start 0 latest_start 995 service_t 5
req_capacity 1 n_loc_choices 1 loc 7
id 5 task 2 earliest_start 5 latest_start 1000 service_t 5
req_capacity -1 n_loc_choices 1 loc 0
id 6 task 3 earliest_start 0 latest_start 995 service_t 5
req_capacity 1 n_loc_choices 1 loc 8
id 7 task 3 earliest_start 5 latest_start 1000 service_t 5
req_capacity -1 n_loc_choices 1 loc 0
id 8 task 4 earliest_start 0 latest_start 995 service_t 5
req_capacity 1 n_loc_choices 1 loc 9
id 9 task 4 earliest_start 5 latest_start 1000 service_t 5
req_capacity -1 n_loc_choices 1 loc 0
id 10 task 5 earliest_start 0 latest_start 995 service_t 5
req_capacity 1 n_loc_choices 1 loc 10
id 11 task 5 earliest_start 5 latest_start 1000 service_t 5
req_capacity -1 n_loc_choices 1 loc 0
id 12 task 6 earliest_start 0 latest_start 1000 service_t 15
req_capacity 0 n_loc_choices 1 loc 5
id 13 task 7 earliest_start 0 latest_start 1000 service_t 15
req_capacity 0 n_loc_choices 1 loc 6
id 14 task 8 earliest_start 0 latest_start 1000 service_t 15
req_capacity 0 n_loc_choices 1 loc 7
id 15 task 9 earliest_start 0 latest_start 1000 service_t 15
req_capacity 0 n_loc_choices 1 loc 8
id 16 task 10 earliest_start 0 latest_start 1000 service_t 15
req_capacity 0 n_loc_choices 1 loc 9
id 17 task 11 earliest_start 0 latest_start 1000 service_t 15
req_capacity 0 n_loc_choices 1 loc 10
id 18 task 12 earliest_start 0 latest_start 1000 service_t 0
req_capacity 0 n_loc_choices 1 loc 2
id 19 task 12 earliest_start 0 latest_start 1000 service_t 0
req_capacity 0 n_loc_choices 1 loc -1
id 20 task 13 earliest_start 0 latest_start 1000 service_t 0
req_capacity 0 n_loc_choices 1 loc 3
id 21 task 13 earliest_start 0 latest_start 1000 service_t 0
req_capacity 0 n_loc_choices 1 loc -1
id 22 task 14 earliest_start 0 latest_start 1000 service_t 0
req_capacity 0 n_loc_choices 1 loc 4
id 23 task 14 earliest_start 0 latest_start 1000 service_t 0
req_capacity 0 n_loc_choices 1 loc -1


TASKS
id 0 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 0 subtask 1
id 1 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 2 subtask 3
```

```
id 2 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 4 subtask 5
id 3 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 6 subtask 7
id 4 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 8 subtask 9
id 5 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 10 subtask 11
id 6 reward 0 max_time_span 0 n_req_attribs 1 attrib 1
n_subtasks 1 subtask 12
id 7 reward 0 max_time_span 0 n_req_attribs 1 attrib 1
n_subtasks 1 subtask 13
id 8 reward 0 max_time_span 0 n_req_attribs 1 attrib 1
n_subtasks 1 subtask 14
id 9 reward 0 max_time_span 0 n_req_attribs 1 attrib 1
n_subtasks 1 subtask 15
id 10 reward 0 max_time_span 0 n_req_attribs 1 attrib 1
n_subtasks 1 subtask 16
id 11 reward 0 max_time_span 0 n_req_attribs 1 attrib 1
n_subtasks 1 subtask 17
id 12 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 18 subtask 19
id 13 reward 0 max_time_span 1000 n_req_attribs 1 attrib 0
n_subtasks 2 subtask 20 subtask 21
id 14 reward 0 max_time_span 1000 n_req_attribs 1 attrib 1
n_subtasks 2 subtask 22 subtask 23


PREC_CONSTRAINTS
id 0 subtask1 12 subtask2 0
id 1 subtask1 13 subtask2 2
id 2 subtask1 14 subtask2 4
id 3 subtask1 15 subtask2 6
id 4 subtask1 16 subtask2 8
id 5 subtask1 17 subtask2 10

SYNCH_CONSTRAINTS

NO_OVERLAP_CONSTRAINTS

MUTEX_CONSTRAINTS

LOC_CAPACITY_CONSTRAINTS

PROXIMITY_CONSTRAINTS
```

APPENDIX C: Sample Solution File

```
% agent subtask x y wait_t start_t
      0 2 -4.57327 -8.60688 181.59 199.62
      0 3 -8.22409 -7.57042 0 214.1
      1 8 6.85044 -4.49372 141.62 154.27
      1 0 9.81207 -4.08563 0 166.74
      1 4 8.99279 -2.3565 0 176.52
      1 1 4.01953 6.19353 0 206.24
      1 5 4.01953 6.19353 0 211.24
      1 9 4.01953 6.19353 0 216.24
      1 10 1.93253 1.12841 0 234.93
      1 6 -3.5458 9.76731 0 265.5
      1 7 4.01953 6.19353 0 291.41
      1 11 4.01953 6.19353 0 296.41
      2 15 -3.5458 9.76731 0 21.76
      2 17 1.93253 1.12841 0 62.33
      2 14 8.99279 -2.3565 0 97.01
      2 12 9.81207 -4.08563 0 116.79
      2 16 6.85044 -4.49372 0 139.26
      2 13 -4.57327 -8.60688 0 184.61
```