The Design and Implementation of a Scheduler and Route Planner for Wheelchair Users

by

Suling Yang

B.S., Simon Fraser University, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

December 2006

Abstract

Currently, Global Positioning Systems and other route-finding systems work well for wheelchair users outdoors. Indoor information is usually not provided. However, it is more important for wheelchair users to know the indoor maps of routes accessible to them. Where the closest elevator is, which door is accessible to them, and how long it takes to get to a destination are major route-finding problems for wheelchair users. Nevertheless, there is no existing system that shows up-to-date and detailed information on route accessibility.

A system that works for the independent user would provide more accurate help for wheelchair users. For example, a strong young man in a wheelchair will have less difficulty getting through moderate ramps and rough roads than the elderly who tend to prefer smooth routes. Therefore, we are motivated to create a system that works according to the client's ability, so that it can figure out which path is best for the client under certain constraints. There are several existing systems that allow the user to easily control their wheelchairs. Some of these systems use automatic or semi-automatic robotic wheelchairs, but they are limited to very small local area movement. Such systems focus on hardware components of an auto-wheelchair. Our system, which provides help in pathfinding, can cooperate with these hardware-equipped wheelchairs. The success of related work on daily activity reminder systems has motivated us to build a simple scheduler that works in conjunction with the route planner.

As well as designing and creating the new system, we have analyzed and implemented a new efficient pathfinding algorithm, which is the major contribution of this thesis. The algorithm deploys the hierarchical structure of real-life maps. Assuming that the distance within an abstract high-level node is significantly shorter than along high-level edges, the algorithm can prune away irrelevant paths. Runtime analysis and experimental results show that this algorithm is efficient in numerous scenarios.

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor, Alan Mackworth. I am grateful to him for his valuable guidance and generous support. I would also like to thank other people who provide me with great help. A special thanks is to Le Chang who helped me throughout the process of this research. Jim Little, Chao Yan, Mingyue Tan, Wei-Lwun Lu and Shuan Wang suggested and discussed algorithm designs and programming methods with me. Zicong Mai, Suwen Yang, Jie Zhao, Wei Li, Xun Sun, Yizheng Cai, Chen Yang, Lin Zhong and Matthew Chan are always generous with help. I am grateful to all the participants involved in the user study. In particular, I would like to thank Dave Symington and Yasaman Mahboubi for their collaboration and suggestions.

I want to thank my father and mother, Huiwen and Miaoqiong, who provide me with continuous support and love. My brother, sister-in-law and their son share and increase my happiness.

SULING YANG

The University of British Columbia
December 2006

Chapter 1

Introduction

1.1 Motivation

The population of wheelchair users is very significant and increasing dramatically [2]. Therefore, finding accessible wheelchair routes is an important problem. In developed countries, most buildings and public transportation are accessible, making the lives of people in wheelchairs easier. For example, these conveniences include elevators to each level of a building, ramps in addition to stairs and buses with accessible services. However, the routes to the closest elevator in a new building, temporary road conditions and bus schedules may be unknown to wheelchair users. Therefore, we were motivated to create route-planning software that can be installed on a small device to give wheelchair users route accessibility information while they are travelling. With the rapid development of related technologies, a number of computer scientists have been carrying out research to devise technology that will assist or replace caregivers for the elderly or for people with disabilities. Currently, even small devices such as laptops and PDAs can provide good pathfinding help for clients. With these advancements in technology, the implementation of route-planning software is now possible.

A scheduler program can synchronize with a route planner to provide more accurate commuting information for clients, such as the remaining time to the next

destination and the means of transportation. A scheduler is a useful tool to remind people of their daily activities. Therefore, our software contains a simple scheduler that can transfer a schedule from and to Microsoft Outlook. After obtaining the destination from the scheduler, the route planner establishes some possible paths to the destination and displays the best one to the client. A possible path means a route that has acceptable obstacles based on the client's ability and shows the travelling time required before the start of the next event in the schedule. The client can make changes to his/her schedule, and the route planner will immediately re-compute a path to the new destination.

The first stage of our project is to implement an algorithm to hierarchically find paths and obtain multiple levels of detail. For example, to find a path between two rooms in two different buildings, we would find a path between the two buildings first, and then refine the path to the two rooms. Since an abstract high-level path can provide a general plan, the client can have an impression of future routing. Instead of being presented with a cumbersome and lengthy low-level path, people, especially the elderly, would prefer a cognitively visible path. In the next stage, the software accommodates our scheduler and real-world maps, including indoor and outdoor applications. At the moment, the software has been tested with the campus map of the University of British Columbia (UBC). Moreover, a shared online database could be constructed. Thus, multiple clients can exchange information or update obsolete maps in real time. For example, if a road is under construction, a client can upload this information after he/she observes the surroundings, and other clients can then download it immediately. Finally, an extension to the software could include voice communication for the optically impaired.

1.2 Challenges and Contributions

Although several existing systems are available to help people with disabilities, none of them show up-to-date and detailed information on route accessibility. As a com-

bination of previous endeavors devoted to implementing robust route-planning and daily activity reminder systems, this thesis addresses the goal of building user-friendly interfaces of the route planner and scheduler system for the elderly and people with disabilities. The system should provide immediate transportation information to the clients and compute the best routes effectively and efficiently. In the process of accomplishing our goal, we were met with several challenges, which we discuss in the following sections.

Cooperation between the Scheduler and the Route Planner. The cooperation between the scheduler and the route planner is especially important. The Aphasia [17] and Nursebot [4] projects have proven successful in reminding users about activities, while the Activity Compass [22] and the CLever [9] project help users find correct paths. However, the researchers who developed these systems did not observe that the scheduler and route planner together can provide more detailed and accurate information. First, the scheduler should synchronize with the real time clock, so it can provide correct reminders. Our scheduler is designed to check the time every minute and remind users of upcoming events. Second, knowing the destination of the next event, the scheduler should communicate with the route planner and transfer the information together with the remaining time to it. Third, the route planner should compute the best path. If the user needs to head to a different destination because of a change to his/her schedule, the route planner should react immediately. On the other hand, the route planner could announce the minimum required transportation time to the scheduler, which could provide an early reminder to the user.

Data Collection and Database Management. People have become aware of the accessible route-finding problem for wheelchair users, yet routing information to many places is not available. We decided to test our system on the UBC campus first, since static outdoor accessibility routing maps are already provided, making

data collection simpler [6]. However, the provided information involves only outdoor data, and it is not specific to the necessary level of detail. Since indoor structures are more complicated, the required data is more difficult to collect and store. Our route planner showing both indoor and outdoor accessible routes for wheelchair users at UBC contains data of several buildings on campus that were collected by an administrator. Moreover, architectures are organized hierarchically. For example, each room is in a building, which in turn is in a city. Maps in our experiments are constructed in such a hierarchical way in XML nested tag format. However, such an XML file for a large map is difficult to create and maintain. Therefore, we built a Java class that reads a map in XML form and creates lists of architectural elements, each of which has a list of neighbours and a list of the corresponding distances. Other Java classes in this project can access these lists and perform actions on each architectural element. In addition, another class was built to modify changes to the map.

Hierarchical Pathfinding Algorithms. Pathfinding on large-sized maps is time-consuming. Classical search algorithms such as A* and IDA* with accurate heuristics may solve difficult problems in polynomial time [10, 13]. However, in real-world pathfinding examples where the search space increases dramatically, A* and IDA* may not be appropriate. Hierarchical pathfinding algorithms that provide abstract plans of future routing, such as HPA* and PRA*, have been explored by previous researchers based on classical ones [8, 28]. Although the two hierarchical algorithms show improvement in efficiency, they only obtain near optimal solutions. In this thesis, we introduce the Hierarchical Shortest Path algorithm (HSP) and a hybrid of the HSP and A* (HSPA*) algorithms, which find optimal solutions in logarithmic time for numerous examples. Our empirical study shows that HSP and HSPA* are superior to the classical algorithms in realistic examples, and our experimental results illustrate the efficiency of the two algorithms.

Java GUI Programming After the prototyping is done on paper, we implemented the system in Java. The interface design must be user-friendly and easy-to-use, since the target users are the elderly and wheelchair users. We chose the Java Foundation Classes (JFC) Swing packages to help us get the necessary software, because not only do the JFC Swing components include all the required classes for our system, but the containment hierarchy of Swing components simplifies implementation of the system interfaces. However, there are some difficulties with Java GUI programming. First, layout managers are preferred, even if exact pixel placement is also available, since the latter can lead to less portable code. However, there are only six types of layout managers, namely, BorderLayout, FlowLayout, CardLayout, GridLayout, GridBagLayout and BoxLayout. Second, the event-handling mechanism is based on the AWT event-handling model. In other words, an event listener is registered on an object whose corresponding component generates such events. Therefore, knowledge acquisition of Java GUI programming is necessary. The current version of our system accomplishes the basic functions of the prototypes created during the system design period. Further interface development may be required.

1.3 Thesis Outline

Following and complementing this introductory chapter, Chapter 2 presents overviews of related projects and indicates our motivation in relation to their usefulness. Chapter 3 shows the designs of several main functions of the system, and these functions are explained in Chapter 4. Detailed implementations are illustrated by UML diagrams in Chapter 5. In Chapter 6, experiments are run based on a given scenario and a user study is conducted to evaluate the system. Finally, conclusions and future work are presented in Chapter 7.

1.4 A Note on Technology

We used a tablet PC for this project, because it is easy to carry and the screen is not too small for people with vision problems. We chose a Fujitsu LifeBook Tablet PC running on an Intel Pentium M, since it is highly recommended by other researchers in our department. With the help of a Garmin GPS 10 system, which is small, the route planner works more effectively.

Chapter 2

Related Work

With the increased need for caregivers for patients and the elderly, a number of *cognitive assistive technologies (CATs)* have been developed or are being developed. The Assisted Cognition project [12], the Nursebot project [4], the Aphasia project [17], the CLever project [1] and the Aware Home project [3] are some interesting developments. These projects have shown great success in helping the elderly and people with mental problems. Systems used in these projects are generally categorized into two types, reminder and routing helper systems. Also, a number of systems have been developed to improve wheelchair physical behaviours. Tin Man, Wheelesley and NavChair are robotic wheelchairs that can take commands from users and smooth the movement in wheelchairs.

2.1 Reminder Systems

Our project is based on a reminder system, because with an accurate schedule, the destination (or sub-destination, e.g. a bus stop on the way to a doctor) can be easily obtained. Microsoft Outlook is a widely-used scheduler, while Autominder [16], ESI Planner II [20], and Adaptive Prompter [12] are reminder systems created for special populations.

2.1.1 Microsoft Outlook

There are numerous existing schedulers available on the web or with some enterprise systems. These schedulers include WebCalendar, Google/Yahoo Calendar, Communications Express from SUN Java Enterprise System, Microsoft Calendar from Microsoft Outlook, etc. Microsoft Outlook is widely used, since it comes with the Windows systems. Moreover, the ease of use of Microsoft Calendar can complement some simple and crude functions of our scheduler. One can easily record an appointment by clicking the "new" button, which leads to a window for inputting the appointment information and multi-attendee scheduling. One can also export some or all appointment information into a text file that can easily be read by other systems. Because of the above favorable aspects of Microsoft Outlook, we selected it as the augmentative scheduler for our system.

2.1.2 Daily Activity Planners for Special Populations

The Autominder, a reminder system, of the Nursebot project is installed in a robot that provides reminders of daily activities to the clients [16]. The Autominder gets a list of tasks that the client needs to do, and provides an optimal schedule. A similar system, the ESI Planner II [20] from the Aphasia project, also provides functionalities such as a daily planner and a reminder system. The ESI Planner II is for people with aphasia, which influences one's communication ability, and is a software program installed on a PDA. The Adaptive Prompter [12] of the Assisted Cognition project and the POMDP formulation of *activities of daily living (ADLs)* [7] provide reminders of the next step in the process of finishing an indoor task for people with Alzheimer's disease. The functionalities of estimating goals and providing reminders to forgetful people or people with mental problems have great potential to be broadly applied, since this population is rapidly increasing. However, these projects focus on populations with certain diseases, and none of them provides instant information about accessibility routing for people in wheelchairs when they

are travelling.

2.2 Route Helper Systems

The main purpose of our project is to aid wheelchair users to find suitable routes. Several existing systems, such as Lifeline and Activity Compass, provide pathfinding help to clients when they are travelling.

2.2.1 A Monitoring System to Provide Pathfinding Help

The Lifeline system from the CLever project is a tool for caregivers to monitor and support people with cognitive disabilities using wireless prompting devices, for example, cellular phones [9]. Users can complete tasks by following pre-planned actions. With this remote support, users can obtain a certain freedom and travel by themselves. More important, a single caregiver can monitor multiple travellers, which greatly reduces the workload and the need of caregivers. The caregiver plays the role of shared information retriever, so a self-learning and self-configuring system with a database shared with other clients may be able to take the place of a caregiver. In addition, the assumption that tasks are pre-designed can be eliminated if clients are allowed to update the shared information. Thus, instantaneous planning is possible by obtaining the latest information from the shared database.

2.2.2 Activity Compass for People with Alzheimer's Disease

The Assisted Cognition project [5] is of note here, in particular, the Activity Compass system. It learns from past daily transportation routines that have been performed by the user and estimates the destination of the current event. When an abnormal behavior occurs, for example, the user fails to get off the bus at the usual stop, the Activity Compass will alert the user [15]. However, the goal of the new event cannot be estimated by the Activity Compass until a partial event has been

completed, and it needs to learn from a large amount of past experience. One extension of the Activity Compass, as stated in [12], is to incorporate information about the time of day and the day of the week into the learning model in order to enhance predictive power. This extension can be further expanded into a route planner with provided user activity schedules. With a known destination goal, the system can focus more on routing planning, such as the time needed to travel one path and the road conditions of a path that are suitable for the client. Moreover, the Activity Compass is created for people with Alzheimer's disease, which affects people's memory. Hence, a route planner based on a reminder system would provide greater help for this population of people.

2.3 Robotic System for Wheelchair Users

The robotic wheelchair is designed mainly to help disabled people in navigation. At present, quite a few robotic wheelchair projects are underway, and some have already been applied to real life. In these projects, the navigation system and the simulator both play an essential role in the development of an intelligent wheelchair. Ojala *et al.* propose an approach to finding fundamental errors in the early design stage by using computer graphics animation and simulation [21]. They also refine the final product by letting the user take part in the design process. In [18], two low-cost wheelchair prototypes, *TinMan I and II*, are presented. These two prototype systems help the handicapped avoid obstacles, reach pre-determined destinations, and maneuver through doorways as well as narrow or crowded areas. Another automated wheelchair, described in [29], makes use of a nonlinear signal processing circuit and pulse steering drive method to smoothly operate. Furthermore, functions for stopping at the desired destination and for preventing collision are imported into the design of the wheelchair navigation system. The paper by Yanco *et al.* introduces a robotic wheelchair named Wheelesley [30]. It includes indoor navigation and provides a user interface that can be easily adapted to a user's capabilities. In

another project, the NavChair assistive wheelchair navigation system is proposed to reduce the cognitive and physical requirements of managing a powered wheelchair [25]. The NavChair is an ordinary wheelchair system equipped with a DOS-based operating system, ultrasonic sensors, and an interaction module. Three operational modes are employed in this system, namely, general obstacle avoidance, door passage and automatic wall following. The NavChair is further developed in [26] to embed a mechanism for automatically selecting the most proper operation mode by considering both the NavChair's current situation and its global location.

The robotic wheelchair projects mentioned above have built strong wheelchair simulation or actual wheelchair systems with high-technical hardware. However, they are limited to use in a localized area. A route planner incorporated into such robotic wheelchair systems would work more intelligently for users.

Chapter 3

The System Design

We used prototyping, one of the Human Computer Interaction (HCI) related techniques, in the system design phase. Through the prototyping process, we can not only define the interfaces of our final product but also test attributes of it before it is complete. We employed paper mockup in the low-fidelity prototype period, because it is a cheap way of providing prototypes for testing and design improvements. In this chapter, we discuss the paper prototype in relation to the improvements and the medium-fidelity prototypes based on it. High-fidelity prototypes are shown and discussed in the last section of this chapter.

3.1 Participants and Design Specifications

Our target users are wheelchair users, so we were encouraged to have wheelchair users as our participants. David, who has been using wheelchairs for more than *thirty* years, came to us from Disability Resource Center (DRC) at UBC and participated thoroughly in our project. Through discussions between David and the researcher, the problems of using a wheelchair and finding paths were identified. Finding the locations of elevators in an unfamiliar building is the core problem for people in wheelchairs. The time required to do so and the difficulty of accessing a path are secondary problems. Design specifications were also identified by David and the

researcher. The major design specifications include the following:

- The route to the next destination should be shown clearly on the screen.
- The route shown should be at an acceptable level of difficulty for the client.
- A schedule following three to five events should be included.
- Adding, deleting and modifying an event's information should be easily achieved from the main screen.

Since people who do not use wheelchairs are considered as possible users of our system, we involved four non-handicapped people, Wei, Suwen, Arthur and Queenie, in our prototyping tests and system evaluation phase. Wei and Suwen are both computer scientists, so they use computers everyday. On the other hand, Arthur and Queenie rarely use computers. Although the frequent computer users outperformed both infrequent computer users and David, they all found the Scheduler and Route Planner system useful for their daily activities.

Table 3.1 summarizes the five participants' relevant information. The education field is the number of years of education completed by the participant.

Table 3.1: Summary of participant information

Participants	Use Wheelchair	Age	Gender	Education	Computer Use
Dave	Yes	50	Male	18	Often
Wei	No	27	Male	18	Often
Suwen	No	26	Female	19	Often
Arthur	No	22	Male	16	Seldom
Queenie	No	25	Female	14	Seldom

3.2 Paper Prototype (Low-Fidelity Prototype)

Once design specifications were defined, the author began by sketching the first draft of the main screen interface. The participants evaluated the sketch and giving

possible feedback and suggestions regarding improper designs. Gradually, the author collected all design ideas and optimized simplicity versus required functionality. With help from the participants, we finished the final paper prototype of the system's main screen, as shown in Figure 3.1. Although it is simple, it contains the two most important parts, the route planner and the scheduler.

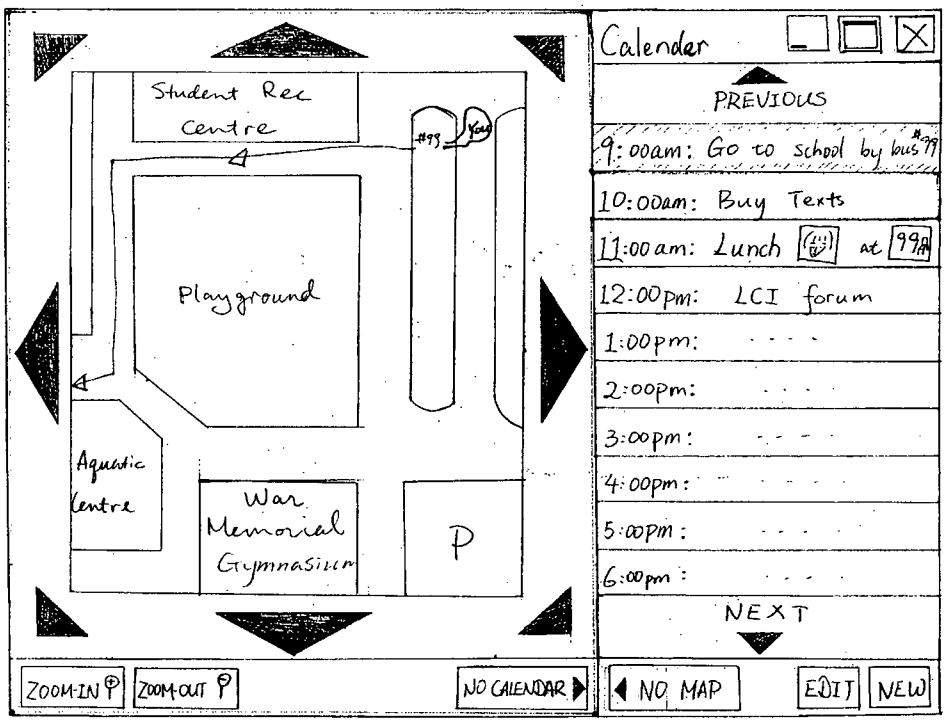

Figure 3.1: The paper prototype of the system's main screen

The participants were involved again in evaluating the final paper prototype. Figure 3.2 shows the gradings for different aspects of the prototype. Interface satisfaction refers to the user's satisfaction regarding the locations of buttons, the number of clicks to access a certain function, etc. Functionality satisfaction indicates whether the user feels satisfied with the provided functions. Ease of use and willingness for future use are as stated. Three different color bars denote the grading from the three different participants. From this evaluation, the participants appear

interested in our system and are willing to use it in the future.

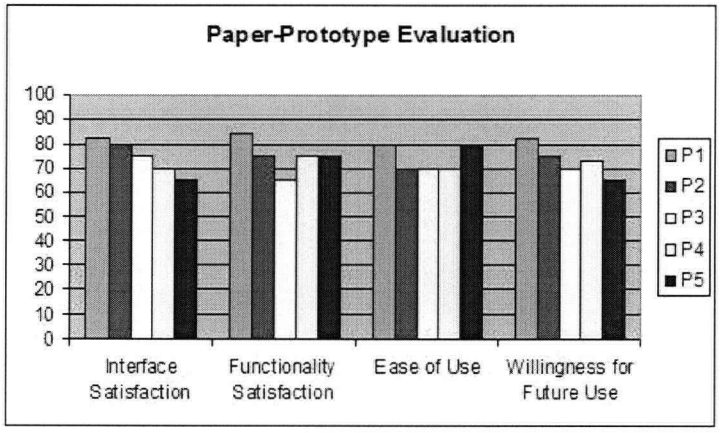

Figure 3.2: Evaluation of the low-fidelity system prototype

3.3 Medium-Fidelity Prototype

After design requirements were finalized and the paper prototypes evaluated, the researcher proceeded with medium-fidelity prototyping. Design adjustments were made based on final design requirements and implementation constraints. In this design phase, we used a vertical prototyping methodology that demonstrates the exact functionality for a small section of the entire product. Adding a new event and showing found paths are two main functions, and the interfaces for them are shown later in this section. Figure 3.8 shows the participant evaluation of this prototype.

3.3.1 The Main Screen

Figure 3.3 shows the medium-fidelity main screen prototype. Note that we made several major changes based on the paper prototype. First, due to implementation difficulty, the "zooming in" and "zooming out" functions were eliminated. Instead, two levels of paths can be displayed, which will be discussed later. Second, a "Detail"

button is added. By clicking this button, the client can view the entire information of each event in the space provided below the schedule, with the detailed path shown on the right side of the screen. Some minor changes includes scroll bars and the date display.

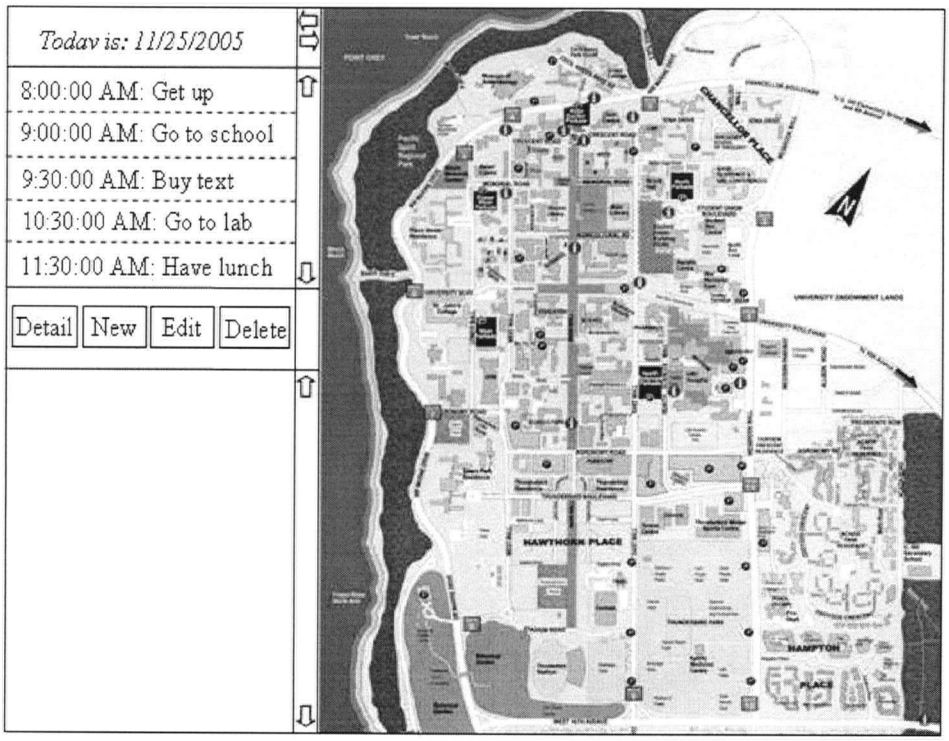

Figure 3.3: The medium-fidelity main screen prototype

3.3.2 Scheduler Interface Design

As specified by the participants and the researchers, users should be able to create or delete event information easily. Therefore, these functions are accessible by one click from the main screen.

By clicking the "New" button on the left-hand side of Figure 3.3, a new window is created, as shown in Figure 3.4 (a). The client then fill in the blanks, as has been done in Figure 3.4 (b). Subject, Start Date and Start Time fields are

required, while other fields are optional. However, Location is necessary if the client wants to know the path from the place specified in the From field or from the last destination. To simplify implementation work, the Start Time and Reminder fields are both required to be "hh:mm:ss AM/PM" form. For this example, if the client has a class at 12:30 PM, he/she should type in "12:30:00 PM" in the Start Time field. If he/she wants a reminder 10 minutes before the Start Time, the client should type in "12:20:00 PM" in the reminder field. After entering the information, the client clicks the "OK" button and an information message will be triggered, as shown in Figure 3.5 (a). The Start Time and Subject of the entered event are displayed in the event list. The client can click the "Detail" button to view the full information, which is shown in the space below the list.

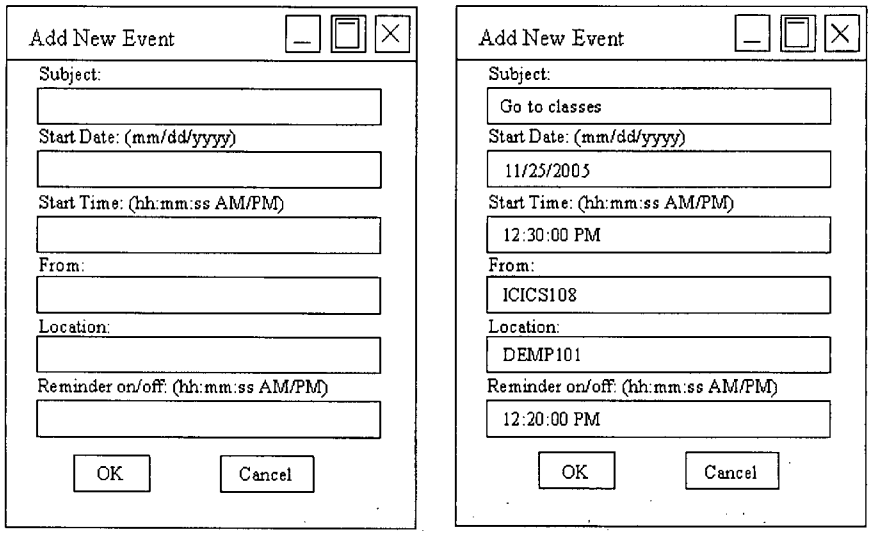

Figure 3.4: The prototype of the frame for adding a new event

The editing event information function works in a similar way to adding a new event. Selecting an event and then clicking the "Edit" button triggers a window similar to the one in Figure 3.4 (b), where information is filled in. Editing one or more fields and then clicking the "OK" button updates the information for that

event. An information message follows to indicate that the change has taken place. To delete an event from the current schedule, the client just needs to select the desired event and press the "Delete" button. A confirmation message, shown in Figure 3.5 (b), is displayed to prevent accidental action.

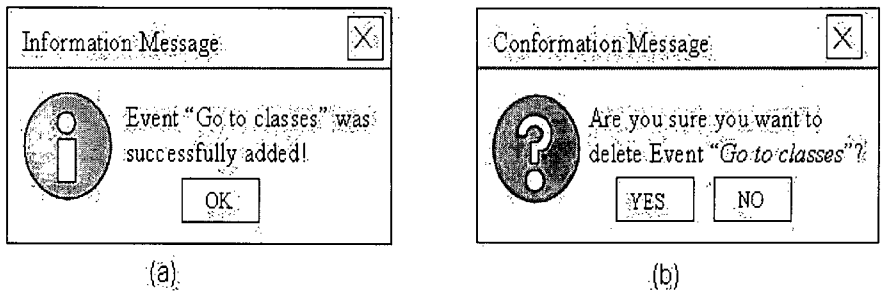

Figure 3.5: The prototype of an information message and a confirmation message

3.3.3 Display Paths

The main goal of the route planner is to display a suitable path for people in wheelchairs. The evaluation by the participants indicated that they appreciated that both an abstracted and a detailed paths are shown. First, the abstracted path can simplify the conception of the whole path for the client. Second, the detailed path shows the exact steps that the client needs to take. In the following sections, we discuss the medium-fidelity prototypes for displaying a path.

Displaying High-Level Paths

To display a high-level (detailed) path to the destination of an event, the client can select the event by clicking it from the event list. The event name is highlighted and a high-level path is shown on the right-hand pane if a path is found. This step is shown in Figure 3.6.

The sub-destinations are shown as buttons, while the directions are shown as arrows. When the user clicks on one of the buttons, some information about the

corresponding (sub)destination is displayed in the text area on the left-hand pane. The current position of the client is denoted using a unique icon. The client can change his/her position by clicking twice on the button representing that place.

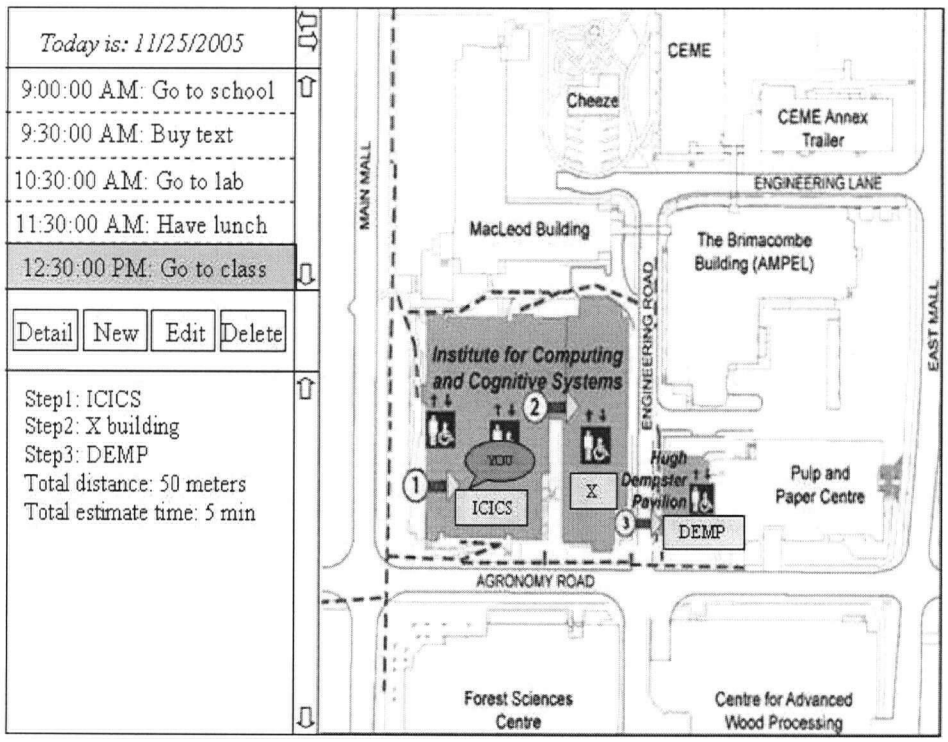

Figure 3.6: The prototype for displaying a high-level path

Displaying Low-Level Paths

Note that the high-level path does not show the steps down to room level. The user may not know the exact procedure for arriving at the "To" destination, but he/she has already acquire an overview of future routing. To view the fully detailed path, the user can click the "Detail" button. Not only the path but also the estimated total distance and time are shown. Here, we need the input of the client's speed. The administrator can set the normal speed of the client and the speeds of elevators and buses. We use the same interface icons to display both high-level paths and

low-level paths.

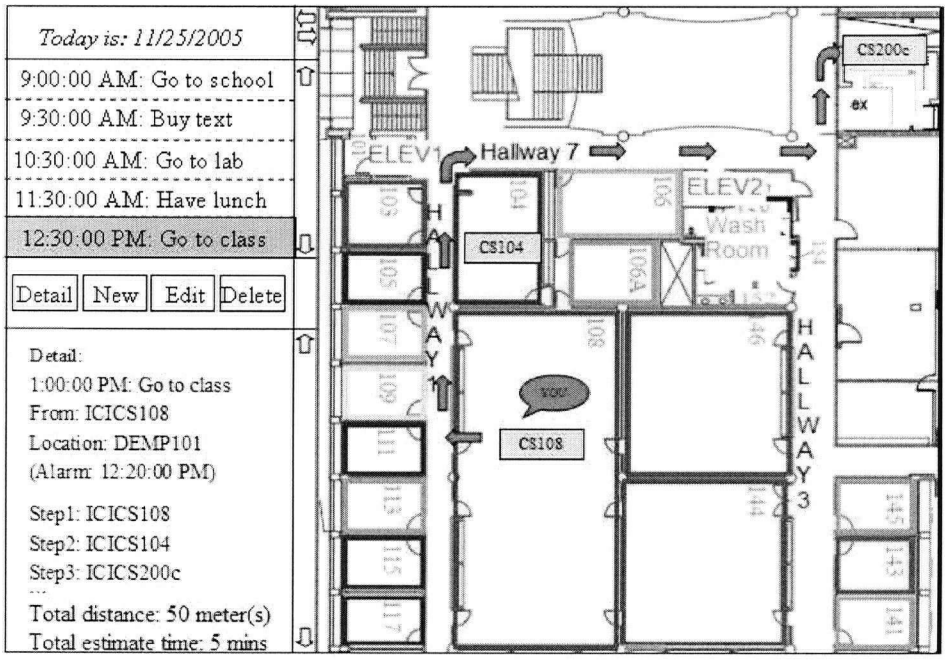

Figure 3.7: The prototype for displaying a low-level path

3.4 High-Fidelity Prototype

In this section, we present several screen shots of the high-fidelity interfaces that are used as final system interfaces. They are done in Java using its Swing package. All of the design requirements and suggestions by the evaluators are met, although some aspects are not perfect, due to implementation challenges. For example, only one arrow is shown between two places. More arrows can be shown by storing more information. However, this will not only require further implementation work, but it will also slow down the system. Since the path is presented hierarchically, the distance between two same level architectural elements that are shown within the same picture is small. Therefore, one arrow is enough to allow users to have a basic idea of routing.

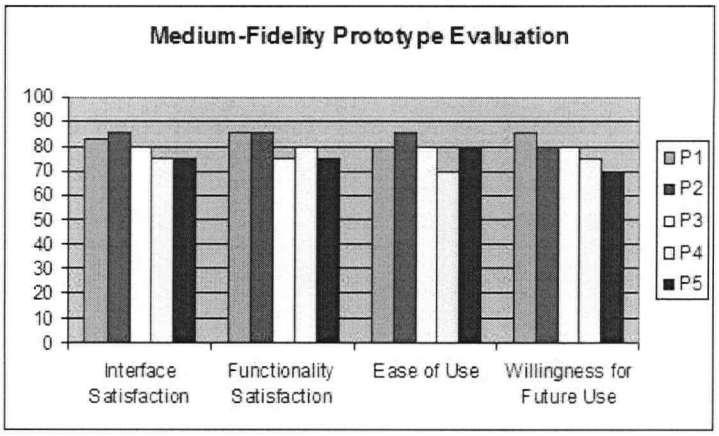

Figure 3.8: Evaluation of the medium-fidelity system prototype

Figures 3.9 to 3.20 present the sequence of screen shots when accessing two main functions. Figure 3.9 is the startup screen. Three buttons are added besides the buttons for managing the scheduler. The administrator can use the "Map" and "User Info" button to view and change the map and user information, respectively. The user can use the "Reject Location" button to report his/her difficulty in accessing the current location. The related functions will be discussed in Chapter 4. Clicking the "New" button triggers another window for adding a new event. Figures 3.10 and 3.11 are respectively two screen shots of the new frame when the information is not entered and when it is. Pressing the "Add Event" button in Figure 3.11 will make the action take place and display an information message, as shown in Figure 3.12. The main screen now will be as shown in Figure 3.15, with the new event listing at the end of the event list. Figures 3.12 to 3.14 are prompting messages to inform or confirm for the user that a certain action took or may take place. Figure 3.13 is triggered if the user selects an event from the event list and clicks the "Delete" button on the main screen. If the user really wants to delete that event, he/she can press the "Yes" button, and then Figure 3.14 is shown. If the user doesn't want to delete that event, he/she can press the "No" button. Nothing changes in this case.

After an event has just been entered, it is highlighted. If a path is found for that event, then the high-level path is shown. Note that the buttons that represent the corresponding (sub)destinations are located where the (sub)destinations are. Since nodes are not able to appear on the same picture, buttons for each picture should be added so that users can view any portion or the whole path. These buttons are placed on the left bottom corner, as shown in Figures 3.15 to 3.20. To view specific information about a place, the user can click the button representing that location. For example, if the user wants to view information regarding the X building, he/she can click the "2" button and screen Figure 3.16 will show. To view the detailed path, the user can click the "Detail" button, and then Figure 3.17 will show. Though the arrows are not aligned perfectly on the hallway, the user can easily find out the correct location. Some (sub)destinations are not shown in this picture, but users can view the next picture by clicking the "Picture 2" button, resulting in Figure 3.18. Clicking on one of the buttons that represents (sub)destinations will display useful information regarding the desired (sub)destination in the left-hand pane.

The result of an evaluation of the above configuration are shown in Figure 3.21.

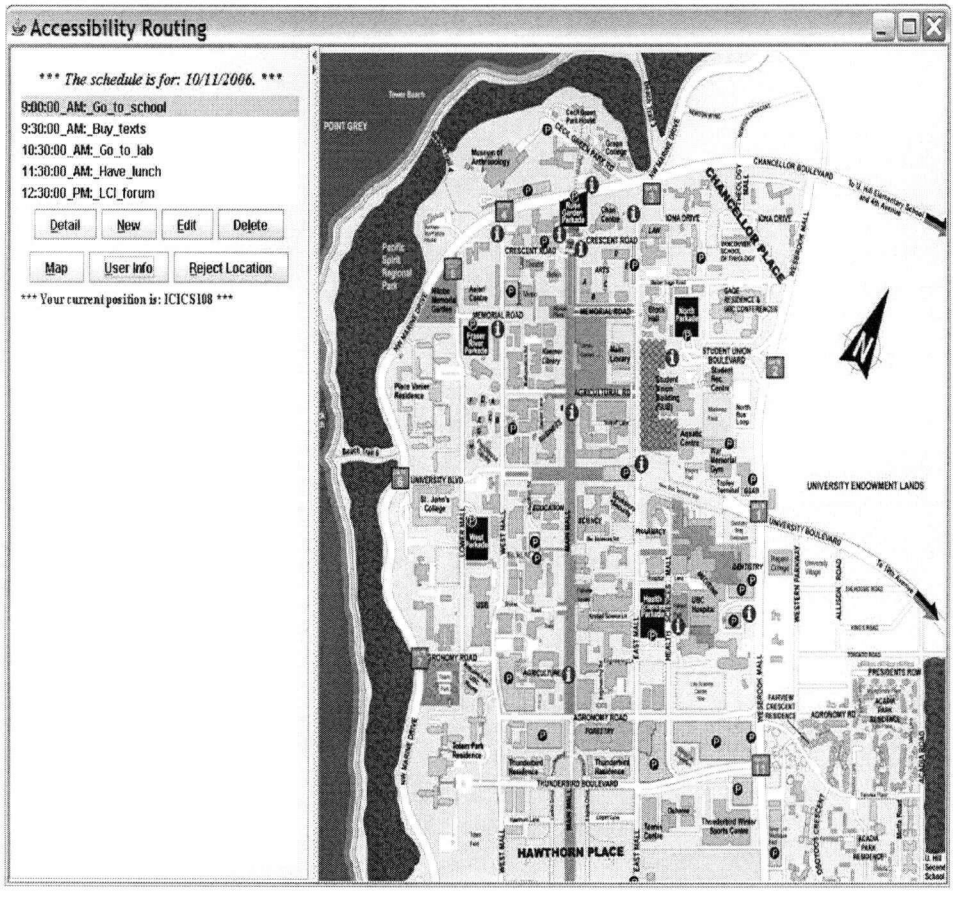

Figure 3.9: A screen shot of the Route Planner when it starts

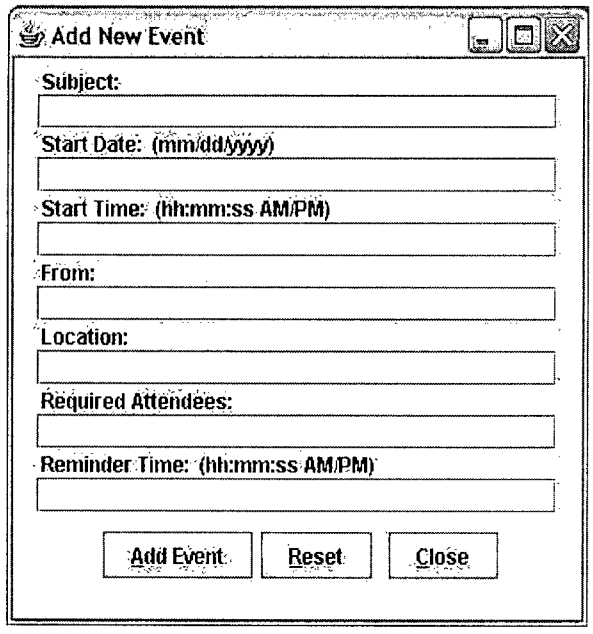

Figure 3.10: A screen shot of the frame for adding a new event

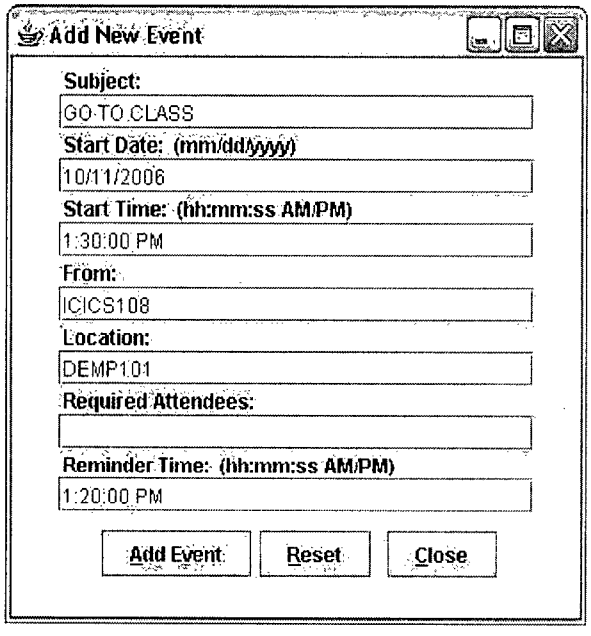

Figure 3.11: A screen shot of the frame for adding a new event, with information entered

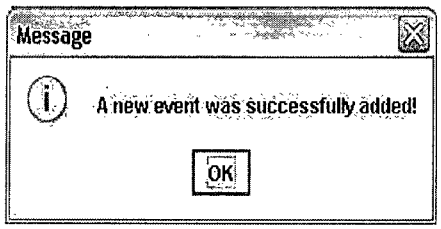

Figure 3.12: A screen shot of an information message

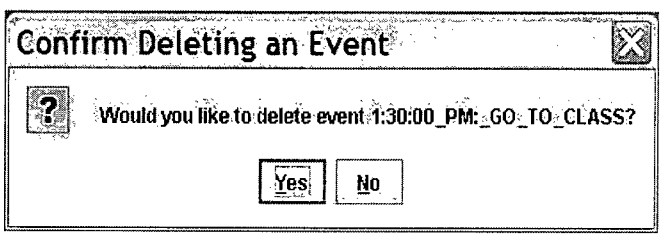

Figure 3.13: A screen shot of a confirmation message

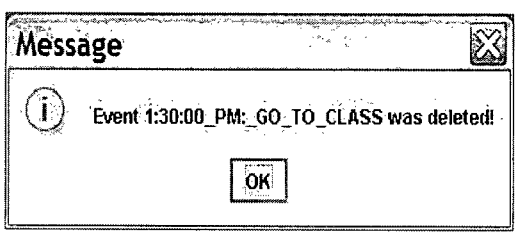

Figure 3.14: A screen shot of a confirmation message

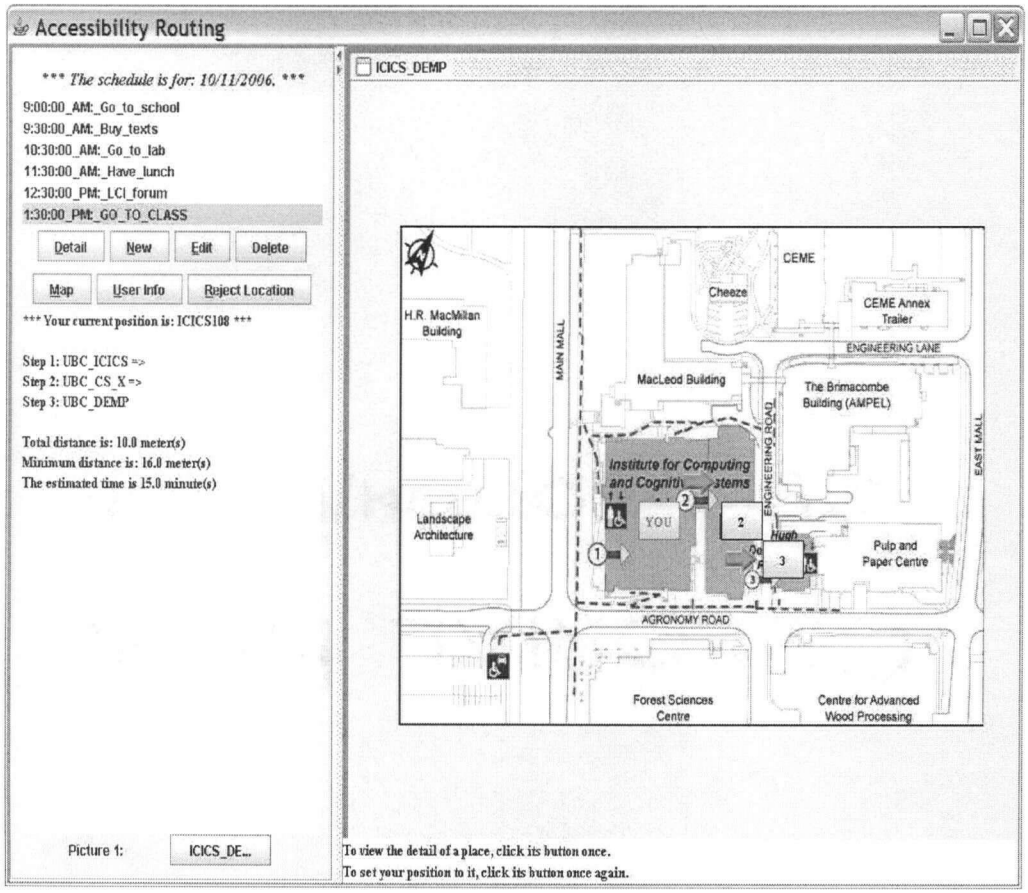

Figure 3.15: A screen shot of the Route Planner when a high level path is shown

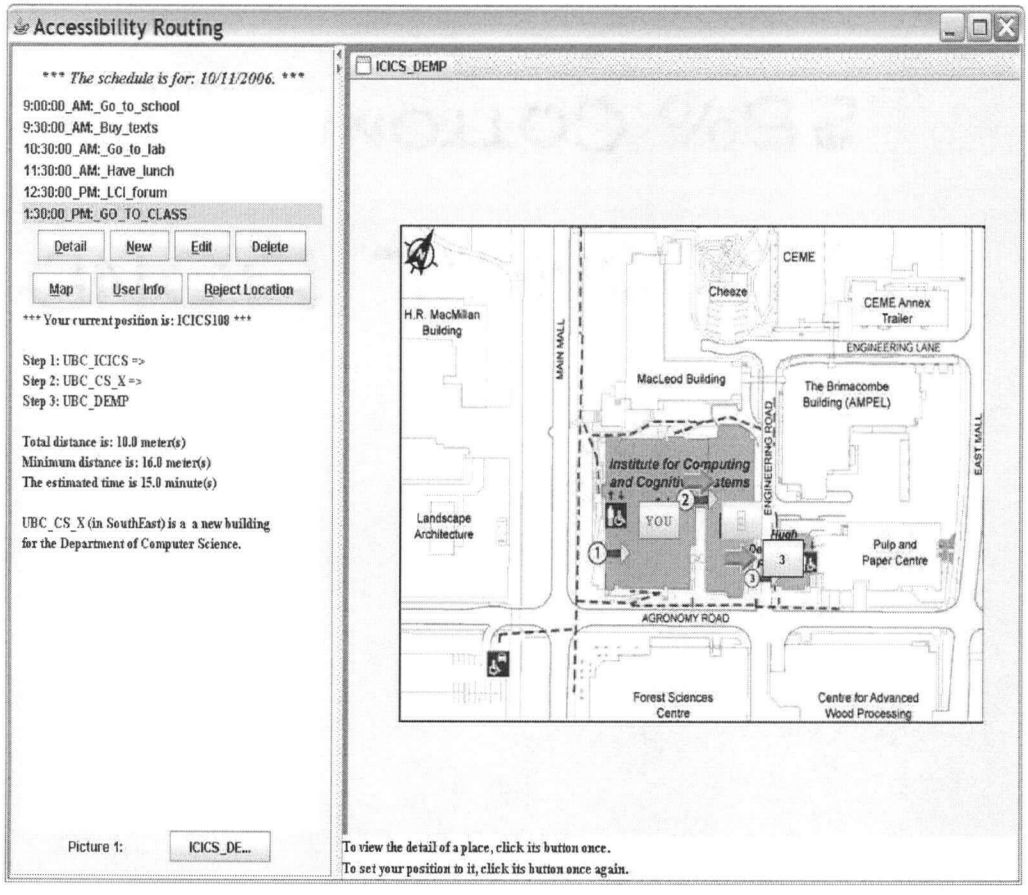

Figure 3.16: A screen shot of the Route Planner when the user wants to view the location information of a place

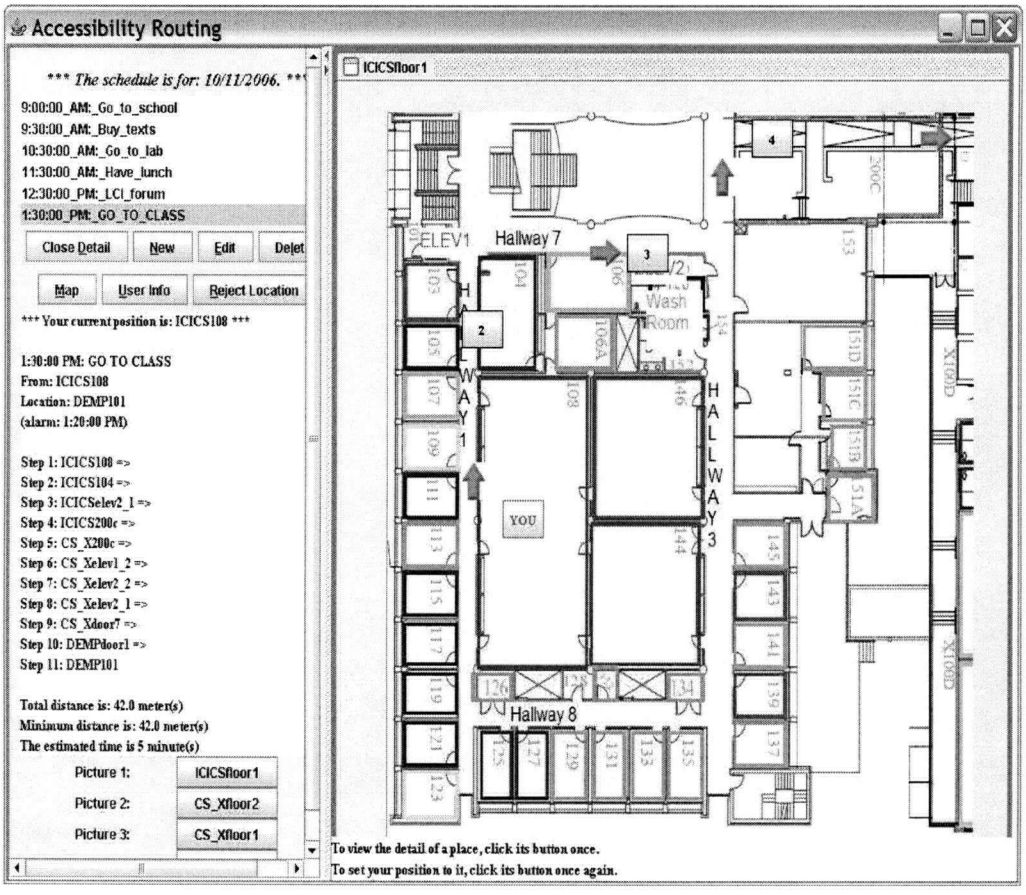

Figure 3.17: A screen shot of the Route Planner when a low level path is shown

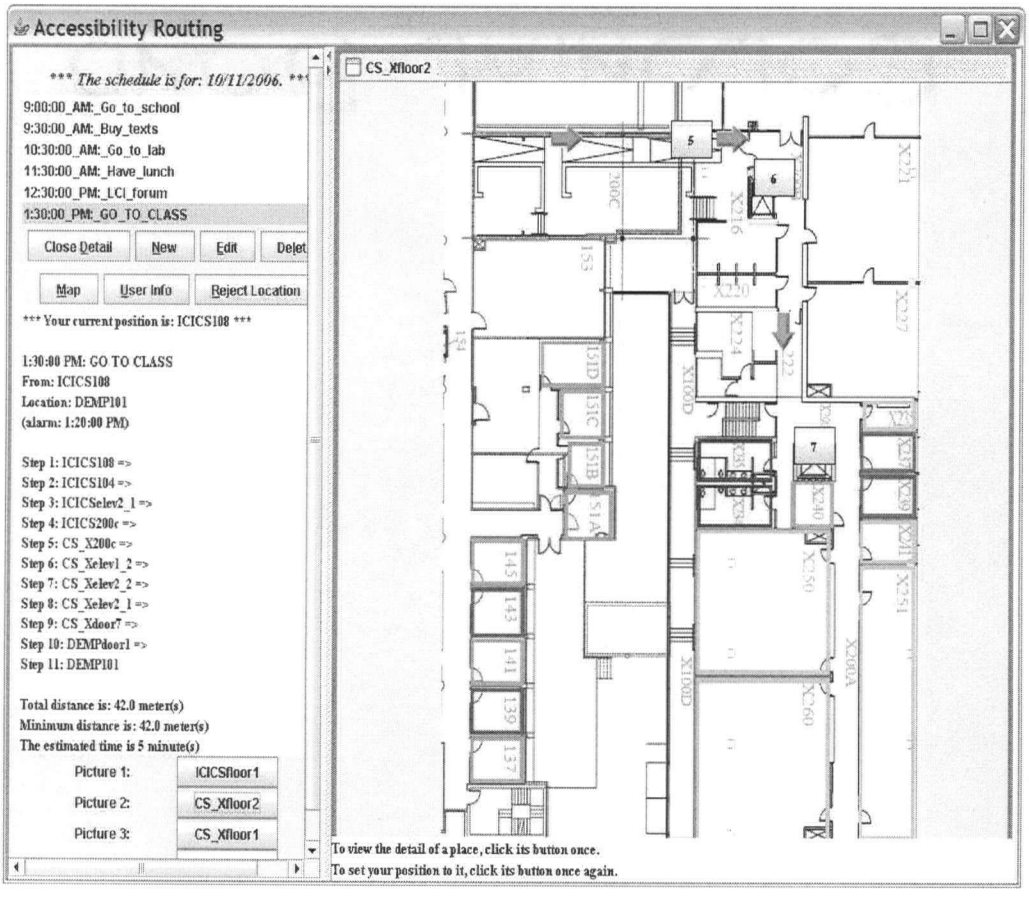

Figure 3.18: A screen shot of the second picture of the low-level path

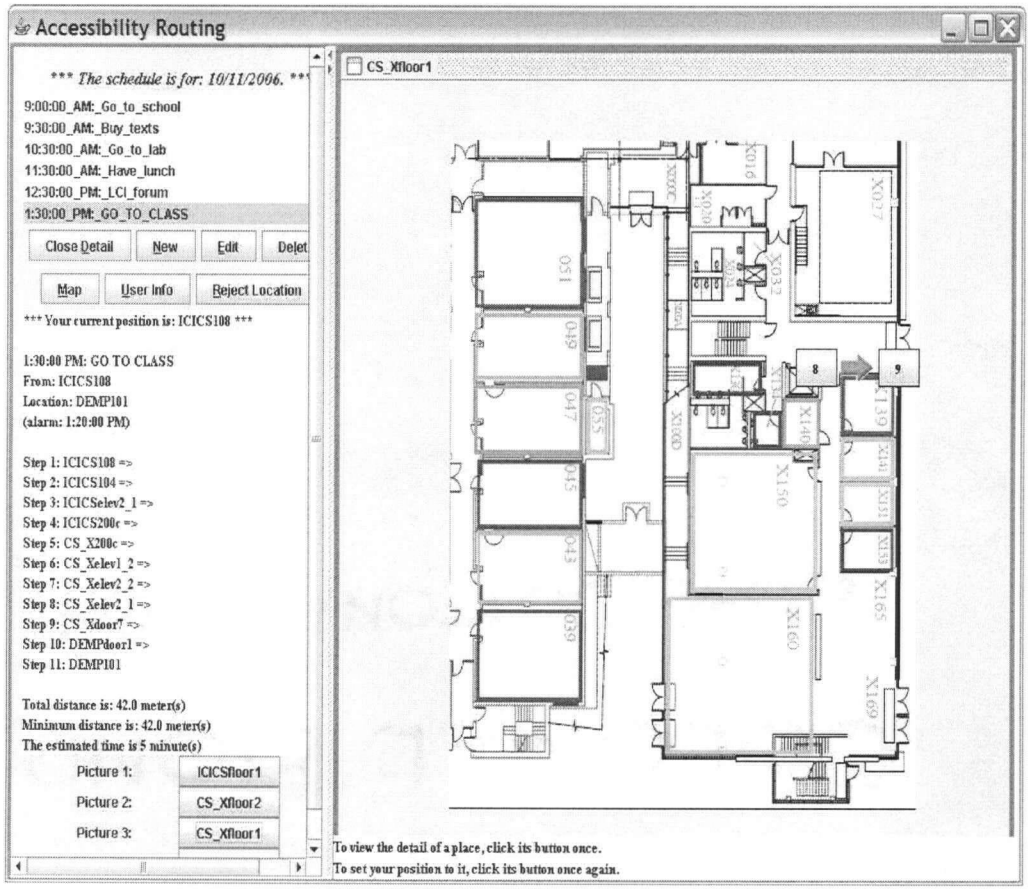

Figure 3.19: A screen shot of the third picture of the low-level path

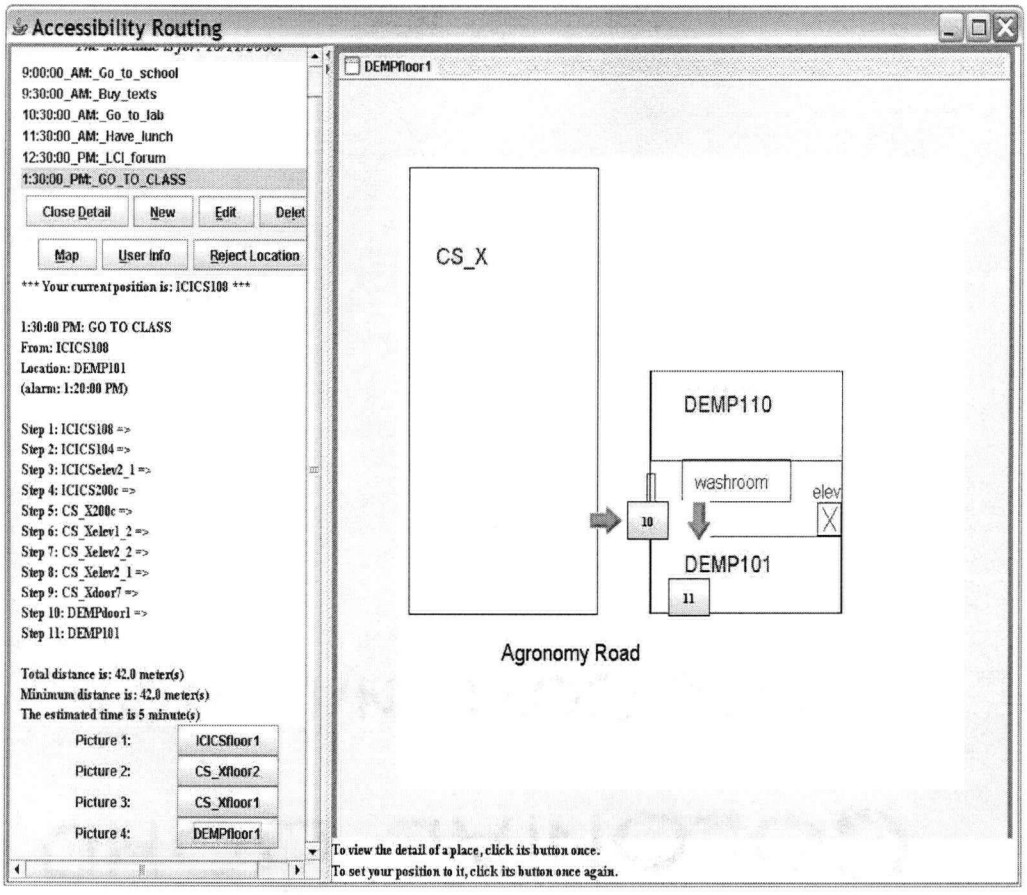

Figure 3.20: A screen shot of the fourth picture of the low-level path

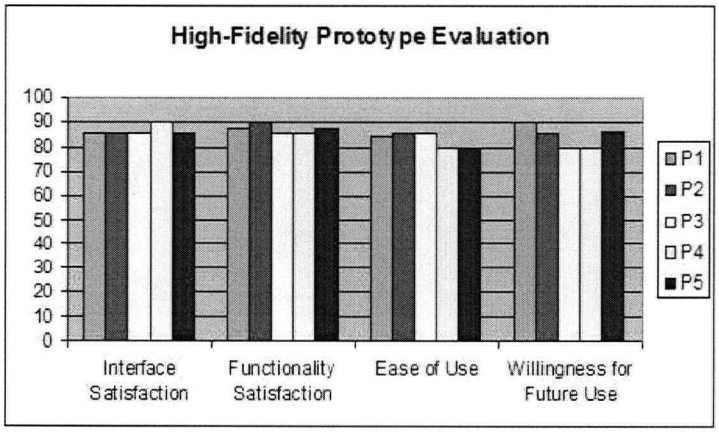

Figure 3.21: Evaluation of the high-fidelity system prototype

Chapter 4

Functionality

4.1 Scheduler and Reminder

The scheduler is based on a list of events, which can be imported from Microsoft Outlook. Each event item contains three mandatory attributes, namely "Subject", "Start Date" and "Start Time", and several optional attributes. A Java class *TransferSchedule.class* is implemented to read the file containing the event list and to transfer the list to an XML file. The XML file is then parsed and several lists are created. These lists include a list of event summaries that are shown in the main screen, a list of reminder times that are compared with the current time, and a list of details that contain all the information the user entered as well as path detail, if a path is found. With these lists, the implementation is simplified and the following functions are easily achieved.

4.1.1 Basic Scheduler Functionality

Since the event list is transferred from Microsoft Outlook, many functions provided by it can be used to supplement functionality of our scheduler. Therefore, our scheduler provides only several basic functions on managing the schedule, based on viewing or modifying the lists that are created when the *TransferSchedule.class*

parses the schedule list.

Adding a new event adds an XML item to the XML file in a sorted order. Then, the *TransferSchedule.class* parses the XML file again, so all lists are also updated. To insert the item in sorted order, we suppose that the original list is sorted and that we can linearly search for the correct position. Since the number of events in a one-day schedule is usually small, linear search is appropriate.

To delete an event, the client selects the event name from the list and presses the "Delete" button. An index number i is obtained when the client selects the event. The i-th items of the event summary list, reminder time list and detail list are removed from those lists.

Editing an event's information is equivalent to deleting the event and then adding another one. Therefore, when the client wants to edit an event's information, it is displayed in a window similar to that used for adding a new event (see Figures 3.10 and 4.1). If the user presses the "Edit Event" button, the event item is deleted first, and then an new event with the information filled in, as shown in Figure 4.1, is added.

When the client presses the "Detail" button from the main screen, the highlighted item's index number i is obtained. Then the i-th item from the detail list is shown.

4.1.2 Reminder Function for Individual Clients

Our system supports both manual reminder and automatic reminder functionality. The manual reminder time is obtained when the client creates an event item, while the automatic reminder time is calculated by the system for each individual client based on his/her speed. A thread is created to act as a time counter. It reads the system time every second and compares it with each reminder time in the lists. If the system time matches one of the manual reminder times, the system will prompt with a simple message, as shown in Figure 4.2 (a).

Figure 4.1: The frame for editing an event's information

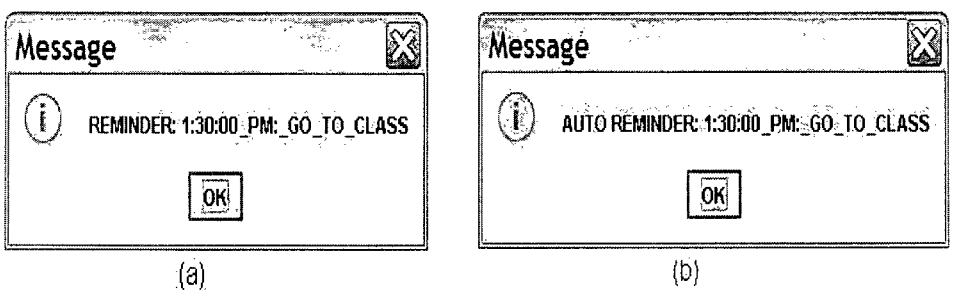

Figure 4.2: (a) A prompted message for manual reminder; (b) A prompted message for automatic reminder

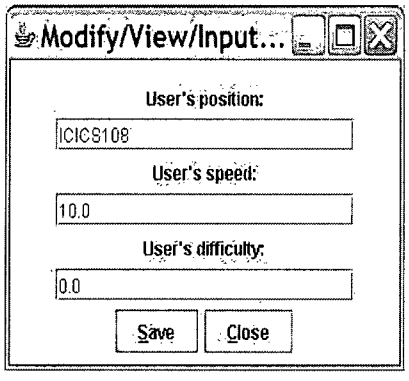

Figure 4.3: The frame for inputting user's information

When the system is launched, the administrator or the client himself/herself can enter some basic client information into the system database. This information can be learned using Machine Learning methodologies, such as Logistic Regression and Particle Filtering. For simplicity, our system will take the needed information as inputs (see Figure 4.3). For each event, if a path is found to the destination, the required time for the client to arrive there is calculated as follows:

$$required\ time = \frac{distance}{speed}. \tag{4.1}$$

The *required time* is rounded to minutes and subtracted from the event time. The result is the automatic reminder time, and is compared with the system time as well. A automatic message is shown, as Figure 4.2 (b).

4.1.3 Cooperation between Scheduler and Route Planner

From the above discussion, it can be noted that the scheduler and the route planner cooperate closely. This section summarizes the cooperation between them. The scheduler handles all the event information. The destination list is maintained by the scheduler. When the information is updated, the scheduler checks to see if the destination has been changed. If it has, the scheduler corrects the destination list. With this list, the route planner can find a suitable path and estimate the distance.

Therefore, the required time can be estimated as well. Based on the required time, the scheduler provides an automatic reminder to each individual client.

4.2 Hierarchical Shortest Pathfinding

The notation used in this section is described here. A map is represented by a graph $G = (N, E, l)$, where N is the set of nodes, E is the set of edges, and l is the number of hierarchical levels. The cardinalities of N and E are denoted by n and e. Each $node \in N$ contains a set of neighbours $neigh(node)$ and is assigned to a level. Each edge $(i, j) \in E$ is associated with another pair of nodes $(exit, entrance)$ where $exit$ and $entrance$ are nodes of one level lower, or they may be $null$. The weight on an edge (i, j) is denoted by $w(i, j)$, and the weight of a path P is $w(P) = \sum_{(i,j)\in P} w(i, j)$. We want to find the *shortest path* P that has the minimum weight $w(P)$ from a node s to another node d in a graph G.

Definition 1: **Search** *on a graph G: The process of finding a path* $P = \{s, i, ..., d\}$, *a sequence of nodes on G, from the start s to the destination d.*

Human beings may approach complex problems by dividing them into easier sub-problems, each of which can be further divided into smaller problems or solved by a quicker search. More than forty years ago, Minsky realized that a successful division of a complex problem will greatly reduce the search time by a *fractional exponent* [19]. Such divisions can be considered as "islands" in the search space, where these islands can be abstracted from groups of nodes. In this section, an abstraction of a group of nodes is found when these nodes are at the same level and within the same enclosure. The abstraction then is their enclosure. For example, a building is an abstraction of the rooms in it. Conversely, details at the room level constitute the refinement of the building.

Definition 2: **Hierarchical Pathfinding**: *The process of finding a sequence of paths* $\{P_m, P_{m-1}, ..., P_0\}, m \leq l$, *where* P_i *contains nodes of* i-*th level and that are an abstraction of nodes on* P_{i-1}.

The maximum length of an abstract path then is defined as $maxlength(P_i) = w(P_i) + \sum_{node \in P_i} upperbound(node)$, and the minimum length of the path is $minlength(P_i) = w(P_i) + \sum_{node \in P_i} lowerbound(node)$, where $upperbound(node)$ is the maximum of shortest distances between any pair of places within the node and $lowerbound(node)$ is the minimum distance from the entrance to the exit. The upper bound of a node can be overestimated, whereas the lower bound can be underestimated. Hence, $upperbound(node)$ can be the total of low-level edge lengths in an abstracted node, *node*, and $lowerbound(node)$ can be zero.

4.2.1 Previous Work

One of the well-known search algorithms is Dijkstra's, a non-heuristic version of A*. However, it is not as efficient as some A* algorithms with good heuristics. A* [10] and IDA* [13] are both heuristic and complete algorithms on locally finite graphs (graphs with a finite branching factor), but neither of these two algorithms is well-suited into a dynamic environment.

Hierarchical pathfinding has been explored since the mid 90's, and several excellent algorithms have been developed. In [24], Rabin provides a high-level description on path-finding using a two-level hierarchy. This algorithm uses only two levels of abstraction, but real world maps are divided into numerous levels. In [11], Holte *et al.* explain how abstraction could lead to speedup on finding a solution for a graph-oriented problem. Similar to their work, Hierarchical Pathfinding A* (HPA*) [8] and Partial-Refinement A* (PRA*) [28] both construct multi-level abstractions using grid-based representation, and greatly reduce the amount of time required to find a near-optimal solution. Instead of estimating a near-optimal solution, our

algorithm finds all possible candidates and keeps them until some are proven to be non-optimal.

If the number of sub-level nodes of each node is fixed, the size of the map increases exponentially as the number of levels increases. The running times of existing search algorithms grow exponentially as well, so they may not be practical in reality. In this thesis, we propose the Hierarchical Shortest Path (HSP) algorithm, as well as a hybrid of the HSP and A* (HSPA*) algorithms. HSP finds a threshold that is the least upper-bound of the length of a high-level path, and refines the paths that may be shorter than this threshold. It stops refining a path when the path length exceeds the threshold, and finally returns the shortest path among those that remain. HSPA* works in a similar way, except that it uses A* for refinement as soon as it reaches a specific level. The experimental results show that HSP and HSPA* are suitable for various applications.

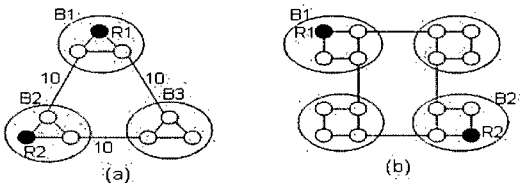

Figure 4.4: Examples: (a) and (b) are maps of campuses comprising three and four buildings, respectively, where each building contains a number of rooms

4.2.2 The Algorithm

The hierarchical approach taken in this section involves multi-level representation of a real-life map. For example, these levels can be campuses, buildings, floors and rooms. Assume that a person wants to travel from room R_1 in building B_1 to room R_2 in building B_2 in the world of Figure 4.4 (a). If the person begins his pathfinding from B_1 to B_2, then he could easily find the shortest path from B_1 to B_2 with one edge. If the walking distances within these buildings are negligible, then the person can take the shortest path between B_1 and B_2, and needs only to refine the path

from R_1 to the exit of B_1 and the path from the entrance of B_2 to R_2. Hence, the person does not need to search paths within B_3.

Therefore, in order to save time in planning, we can search for short paths between high-level sites first, and then refine the paths within the selected sites for more details. However, the high-level shortest path does not guarantee the shortest overall path, because the paths within each site can vary in length. Consider the previous example, where the walking distance within buildings B_1 and B_2 is significant. Then, a path from R_1 to the exit of B_1, and a path from the entrance of B_2 to R_2, may increase the length of the overall path by a large amount. As a result, we can prune away a path P only when it is guaranteed to be non-optimal, i.e., the minimum length of the path exceeds the maximum length of another path:

$$\exists P' : minlength(P) > maxlength(P'). \tag{4.2}$$

The main algorithm, 4.1, consists of four major steps: first, find the topmost level abstract nodes wherein the start node and the destination node are different, and abstract the graph G to that level (line:1,2); second, find a high-level shortest path between ancestors of s and d at the $\hat{l}$-th level, based on which a threshold is obtained (line:3,4); third, find all possible high-level paths with minimum lengths less than the threshold (line:5); finally, refine each high-level path to the lowest level and return the one with minimum length (line:6,7). This algorithm is complete since it maintains all possible paths and ignores a path only when it is definitely longer than another possible path. The following will explain the algorithm in further detail.

Abstract of a Graph: As mentioned above, the desired level in the first step is needed for abstracting the graph. For example, if a person travels from a room in a city to another room in another city within a country, the two cities are the topmost places he should consider. In other words, we seek out the first same ancestor of the two places to find the one level below that ancestor that is desired. Algorithm 4.2 presents the procedure for finding the first same ancestor of two places

Algorithm 4.1 : main (G, s, d)
Input: A graph $\boldsymbol{G = (N, E, l)}$, the start node s and the destination node d
Output: A shortest path $\boldsymbol{P}$ from s to d
1: $\hat{l} := sameAncestorLevel(s, d) - 1$
2: $\hat{G} := abstract(G, \hat{l})$
3: $\hat{P} := A^*(\hat{G}, ancestor(s, \hat{l}), ancestor(d, \hat{l}))$
4: $threshold := maxlength(\hat{P})$
5: $S := possiblePaths(\hat{G}, ancestor(s, \hat{l}), ancestor(d, \hat{l}), threshold)$
6: $\hat{S} := refine(S, l, threshold)$
7: **Return** $P := min\{\hat{S}\}$

Algorithm 4.2 : sameAncestorLevel (s, d)
Input: The start node s and the destination node d
Output: The level l' of the first same ancestor of s and d
1: $lower = lowerLevelNode(s, d)$
2: $higher = higherLevelNode(s, d)$
3: **while** *lower* is not at the same level as *higher* **do**
4: $lower = parent(lower)$
5: **end while**
6: **while** *lower* is not a sibling of *higher* **do**
7: $lower = parent(lower)$
8: $higher = parent(higher)$
9: **end while**
10: **Return** $l' := level(lower)$

and returning the level of that ancestor. Note that pathfinding between different level architectures is also allowed. After the desired level $\hat{l}$ is found, the graph G is abstracted to that level:

$$\hat{G} := (\hat{N}, \hat{E}), \tag{4.3}$$

where $\hat{N}$ and $\hat{E}$ are subsets of N and E, respectively, and the nodes $node \in \hat{N}$ and $i, j : (i, j) \in \hat{E}$ are only those at level $\hat{l}$.

A High-level Shortest Path and a Threshold: Either Dijkstra's or the A* algorithm is used on $\hat{G}$ to find the shortest path $\hat{P}$ between the ancestors of s and d at the $\hat{l}$-th level. Then, $maxlength(\hat{P})$ is the threshold that we are looking for. The time required to find the high-level path and the threshold is insignificant if the number of high-level nodes is relatively much smaller than that of low-level

Algorithm 4.3 : possiblePaths ($\hat{G}$, $ancestor(s, \hat{l})$, $ancestor(d, \hat{l})$)

Input: The abstract graph $\hat{G}$, the ancestors of s and d at level $\hat{l}$, and the threshold t

Output: A set of paths at level $\hat{l}$ with minimum lengths less than t

```
1: CurrentPaths ← { {ancestor(s, l̂)} }
2: Output ← { }
3: while CurrentPaths changes do
4:   for each P ∈ CurrentPaths do
5:     if last node ln of P is ancestor(d, l̂) then
6:       Output ← Output ∪ {P}
7:       PossiblePaths ← PossiblePaths ∪ {P}
8:     else
9:       for each nb ∈ neigh(ln) do
10:        P̂ = P ∪ {nb}
11:        if minlength(P̂) is smaller than t then
12:          PossiblePaths ← PossiblePaths ∪ {P̂}
13:        end if
14:      end for
15:    end if
16:  end for
17:  CurrentPaths ← PossiblePaths
18: end while
19: Return Output
```

nodes.

High-level Possible Paths: After a threshold is obtained, we could search on $\hat{G}$ for all possible high-level paths and prune away those with minimum lengths exceeding the threshold. This process is shown in Algorithm 4.3.

Hierarchical Refinement on High-level Short Paths: After high-level paths are found, a refinement step is executed, as shown in Algorithm 4.4. It refines each possible high-level path to one level lower each time by recursively finding the shortest path from the entrance to the exit of each node on the high-level path. Then, the lowest level path is constructed by concatenating all partially refined paths. The refinement step ceases if the length of the current path exceeds the threshold.

Algorithm 4.4 : refinement (S, l, t)
Input: A set S of high-level paths, the lowest level l and the threshold t
Output: A set $\hat{S}$ of low-level paths at level l

```
1: Ŝ = { }
2: for each hierarchical high-level path P ∈ S do
3:    P̂ = {s}
4:    for each node ∈ P do
5:       find the edge from last node of P̂ to the entrance of node
6:       if level(entrance) ≥ l then
7:          P' = HSP(entrance, exit) where entrance and exit are within node
8:          P̂ = P̂ ∪ P'
9:       end if
10:      if minlength(P̂) > t then
11:         ignore P̂ and continue the outer loop
12:      end if
13:   end for
14:   Ŝ = Ŝ ∪ {P̂}
15: end for
16: Return Ŝ
```

4.2.3 A Hybrid of HSP and A* (HSPA*)

Note that there may be more than one high-level path of shorter length than the threshold, but that is unusual in many realistic cases. The refinement step is fast, if the number of possible short paths is small. Otherwise, the time spent on refining these paths may be excessive. Hence, we could refine our algorithm to select between HSP and A* depending on the number (α) of these paths. Furthermore, from the experimental results shown in the next section, it is evident that HSP is not as fast as A* if the number of levels is small and the number of sub-nodes is not significantly large. Therefore, we could use A* algorithm when a high-level path is refined to a specific low-level (β).

4.2.4 Analysis of Running Time

Let b be the number of sub-nodes within an abstract node. If b is fixed, then the total number of lowest-level nodes is $n = b^l$, where l is the number of levels. A* and

Algorithm 4.5 : hybridRefinement (S, l, t, s, d)
Input: A set S of high-level paths, the lowest level l, the threshold t and the start node s and the destination node d
Output: A set $\hat{S}$ of low-level paths at level l

```
1: if |S| < α then
2:   Ŝ = { }
3:   for each hierarchical high-level path P ∈ S do
4:     P̂ = {s}
5:     for each node ∈ P do
6:       find the edge from last node of P̂ to the entrance of node
7:       if level(entrance) ≥ l + β then
8:         P' = HSP(entrance, exit) where entrance and exit are within node
9:       else
10:        P' = A*(entrance, exit)
11:      end if
12:      P̂ = P̂ ∪ P'
13:      if minlength(P̂) > t then
14:        ignore P̂ and continue the outer loop
15:      end if
16:    end for
17:    Ŝ = Ŝ ∪ {P̂}
18:  end for
19: else
20:   Ŝ = {A*(s, d)}
21: end if
22: Return Ŝ
```

IDA* with good heuristics can solve difficult problems in polynomial time [27, 14], i.e., $\mathcal{O}(n^c) = \mathcal{O}(b^{cl})$ where c is a constant. Therefore, the algorithms are exponential to the order of l. On the other hand, HSP only executes searches on $\mathcal{O}(b)$ nodes on each level if there is only one abstract path found. In this case, HSP has a running time of $\mathcal{O}(b^c\, l)$. Even if there is more than one abstract path, say d paths, the running time of HSP is $\mathcal{O}(b^c\, d^l)$. As long as $d \ll b$, HSP is still much more efficient than A* or IDA*. In reality, where distances within a site are usually considered insignificant, one high-level abstract path is expected to appear. Hence, the running time is $\mathcal{O}(b^c\, l)$ for the HSP algorithm in real-world examples versus $\mathcal{O}(b^{cl})$ for A* and IDA*, i.e., linear in l versus exponential in l. Moreover, l is usually small while

b is large. Thus, $\mathcal{O}(b^c\, l)$ or $\mathcal{O}(b^c\, d^l)$ should be relative smaller than $\mathcal{O}(b^{cl})$.

4.2.5 Comparison and Results

In real life, each site is connected to its neighbours from exits to entrances. For example, if two buildings are adjacent, then we can exit from a door of one building, and enter through a door of the other building. Hierarchical maps in our experiments are constructed with this connection in mind. Weights of the edges and the upper bound of the shortest distances between any pair of places within a site are included in the map, which is created in XML form. One of the reasons that we use XML form is due to its consistency and flexibility; i.e., you have to define each element with an opening and an ending tag, while you can use an arbitrary tag name. Another reason is that nested tags in XML can represent hierarchical relations easily.

All experiments described in this section were run on a Linux Intel(R) Pentium 4 machine running at 3.20GHz with 2 GB of RAM, using Eclipse version 3.1.

Examples First, we analyze the algorithms using the example shown in Figure 4.4 (a). Each building contains three floors, while each floor contains three rooms, and so on. In reality, the number of nodes at one level lower is usually much more than three, where the time saved from pruning away unnecessary high-level paths is more obvious. Here we use this example because of its simplicity in building an XML file and its simulation of a simple world. The total number of lowest level nodes is $n_0 = 3^l$, and since it grows exponentially to the order of l, Dijkstra's and A* algorithms are inefficient here. The A* heuristic used in our experiments is the Euclidean distance, which is popular to use in real-world route-finding problems. However, this heuristic may not be useful for a multilevel structure, since the heuristic values for low-level nodes may be similar. For example, the Euclidean distances from nearby rooms in one city to a room in another city may be indistinguishable. In contrast, HSP, which prunes away most insignificant high-level paths, runs well in these examples.

As shown in Figure 4.5 (a) and (b), both multi-level representation running

times of Dijkstra's and A* algorithms blow up quickly, while the HSP running time grows slowly. A semi-log graph of the three algorithms' running times is drawn in Figure 4.6 (a). We can observe that the slope of the HSP curve is significantly less steep than that of the other two curves. A further observation from Figure 4.6 (a) is that HSP is not superior among the three algorithms all the time. When the number of levels is small, HSP wastes time on recursive calls. Therefore, it is better to use a hybrid of HSP and A*, the HSPA* (as described in Section 4.2.3). The running-time results for HSPA* are compared with those of the other three algorithms in Figure 4.6 (b). Note that HSP and HSPA* have similar performances in these examples, although HSPA* is more stable.

Table 4.1 shows the running times of the four algorithms in the example where every abstract node has four sub-nodes, as shown in Figure 4.4 (b). In this example, more than one abstract high-level path is found at each level, but the running times of HSP and HSPA* are still promisingly good. Examples with larger number of sub-nodes (b) are also explored, and the outcomes are shown in Table 4.1 as well.

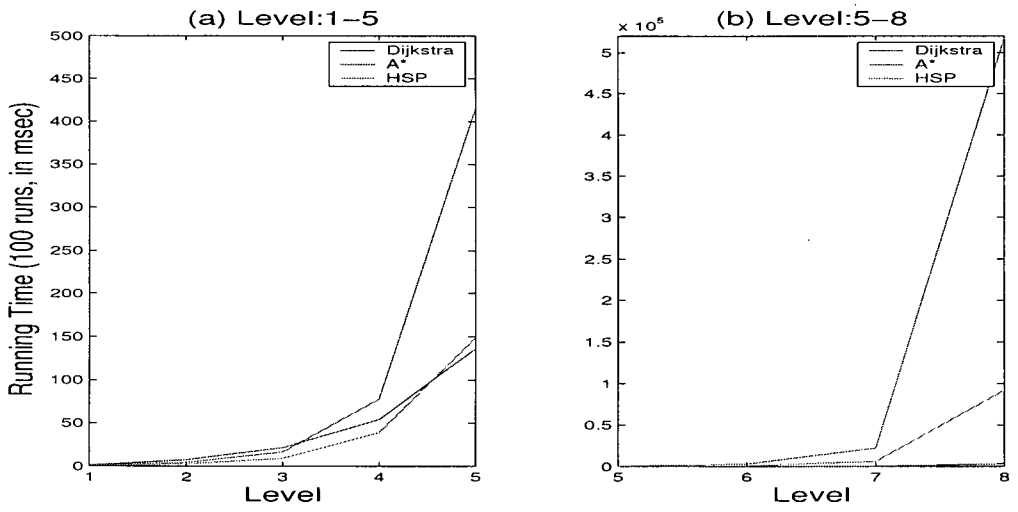

Figure 4.5: Running time plots: (a): the running times of three algorithms from level 1 to 5; (b): the running times of three algorithms from level 5 to 8

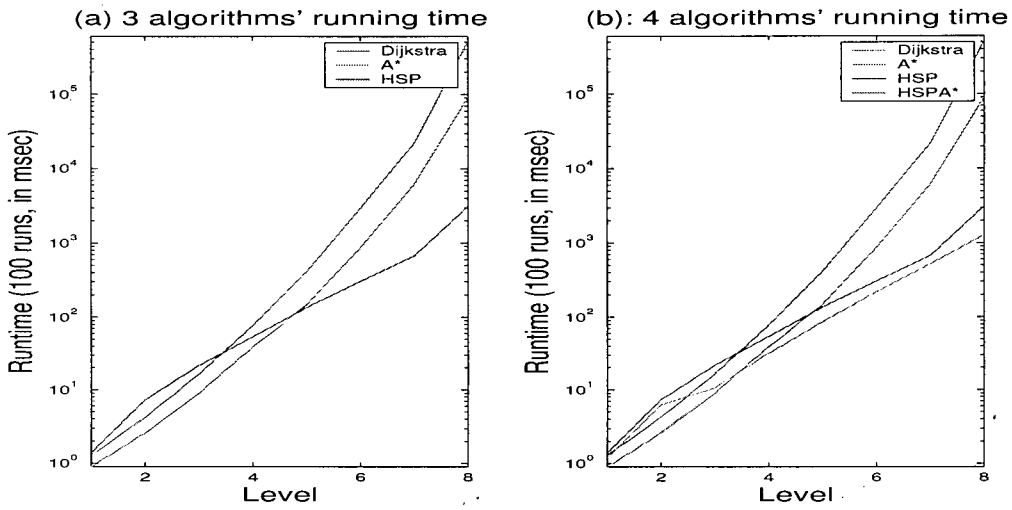

Figure 4.6: Running time plots: (a): the running times of three algorithms in log-scale; (b): the running times of four algorithms in log-scale

4.2.6 Route Planner for People with Disabilities

Our route planner, which shows accessible routes for wheelchair users, contains data of several buildings at UBC: ICICS, the X building and Dempster, the three main buildings in our department. Based on an accurate schedule, the destination can be easily estimated. Then, the route planner computes the possible paths to the destination using HSP and shows both a high-level and a low-level path to the client. Figure 3.15 shows a high-level path from ICICS to Dempster, where most classes are held. After the user clicks the "Detail" button, the low-level path corresponding to the high-level path is shown, as in Figure 3.17.

Table 4.1: Running times: The running times (in milliseconds) of four algorithms using examples where every abstract node has four or more sub-nodes (b)

(b) :			(4)				(5)	
Level:	*1*	*2*	*3*	*4*	*1*	*2*	*3*	*4*
Dijkstra	2	6	35	295	2	11	94	1278
A*	1	4	22	149	1	10	56	717
HSP	1	8	23	130	2	12	42	97
HSPA*	1	8	24	174	1	11	35	88
(b) :			(6)				(7)	
Level:	*1*	*2*	*3*	*4*	*1*	*2*	*3*	*4*
Dijkstra	3	24	350	7817	3	25	466	15601
A*	2	23	218	3462	2	21	231	8368
HSP	2	28	131	354	2	31	137	368
HSPA*	2	24	99	296	2	28	116	316

4.3 Alternative Path for Individual Clients

In this section, we will explain the design and method used to recalculate an alternative path and display it to the client. Users can simply declare to the system that the current location is too difficult for them to get through. In the next run of pathfinding, the system will not generate a path with higher or equal difficulty to the rejected location. Although we just use a simple computation to eliminate difficult locations for each user independently, we could apply sophisticated machine learning methods to train the model representing the user's ability. This approach is considered as future work.

4.3.1 Reject the Current Location

If a place is difficult to get through, the weights of edges connected to it are increased by some portion:

$$\hat{w} = \frac{w(e)}{access(p))}, \tag{4.4}$$

where $e = (x, p)$ and $access(p)$is the accessibility of the place. This technique works for all clients. However, different people have different strengths, so the system should be adaptive to individual users.

Assume that a student from LCI uses the Scheduler and Route Planner system and that he/she wants to attend his/her class in Demspter room 101. He/she follows the path the system suggests and arrives at the hallway between ICICS and the X building, as shown in Figure 4.7. He/she then finds out that the hallway is actually a ramp, which is hard for him/her. By clicking the "Reject Location" button, the student can let the system know that a node on the path exceeds the acceptable difficulty level of the user. A confirmation message, as shown in Figure 4.8, is displayed after this action. Then the system verifies that the location is more difficult than the assumed difficulty the user can accept, and updates the information of the user. From this point on, whenever the route planner tries to find a suitable path for the current user, it tests whether a place is acceptable to the user. If not, it ignores it; otherwise, it runs as the usual HSP or HSPA*.

4.3.2 Recalculate a New Path

After the user rejects the location, he/she can obtain a new path from his/her previous location to the destination. First, the user can roll back to the previous step by double-clicking on the previous button. Then he/she can edit the event information, and change the "From" location to his/her current position. He/she can simply remove the text in the "From" field, and the system will assume the start location is where the user currently is, as shown in Figure 4.9. Figure 4.10 shows the screen after the user finishes editing. A new high-level path is found, as well as a low-level path. Figures 4.11 to 4.13 show the sequence of screen shots of the system, displaying different pictures of the low-level path.

Table 4.2: An example of an event item in XML element form

```
<event>
 <Subject>Go to school</Subject>
 <Start Date>11/25/2005</Start Date>
 <Start Time>9:00:00 AM</Start Time>
 <Reminder on off>8:50:00 AM</Reminder on off>
 <Location>#99busstop</Location>
</event>
```

4.4 Database Management

In this section, we will discuss database management for schedule operation and map representation. This is a significant issue, because our scheduler is based on event information data and the route planner establishes paths over the map database. The event information database is supposed to be of a small size, but the management of it still needs great care since many functions can access it. On the other hand, the map database is huge. Section 4.2 discusses the methodology that we use to reduce the time needed to compute a path. Here, we focus our discussion on data storage and modification.

4.4.1 Schedule Importing and Exporting

There are three ways to create a whole day's schedule. First, the system can start with an empty event list, and the client adds event items one by one. Second, the administrator or the client can create an XML file that contains events in the form shown in Table 4.2. Third, the administrator or the client can input the event information in Microsoft Outlook and export a text file, which can be imported into our system.

When one launches Microsoft Outlook and selects "Calendar" on the bottom left corner, a time table appears on the right-hand side. Clicking on a time slot, a new window for creating an appointment appears, as shown in Figure 4.14. This scheduler includes more functions than our scheduler does. For example, the user

can schedule a meeting with his/her lab-mates and the Microsoft scheduler will automatically send email to all attendees. Rescheduling the meeting time is also allowed. If the client using our system prefers Microsoft scheduler, he/she can keep it on and use our route planner to find paths.

After a list of events is entered, the subjects and destinations of each event are shown on the time table (see Figure 4.15). Selecting "File" from the menu bar, and then selecting "Import and Export" from the drop-down menu, will trigger a sequences of screens as shown in Figure 4.16. Following these screens, a file named "schedule.txt" is created. This file is a list of event information, in the format that our system accepts. Our system can export a similar text file, and Microsoft Outlook can import it by similar steps as explained for exporting.

The Add, Delete and Edit functions all modify the data of the schedule, as explained in section 4.1.1. They modify the lists created by *TransferSchedule.class*. This class is implemented to transfer the lists back to an XML file, and then this updated file can be transferred back to a text file.

4.4.2 Map Representation

In our map database, architectural elements are stored in a hierarchical manner. A campus contains a number of buildings that in turn contain a number of floors. Rooms are considered to be the lowest level architectural element in our system. They are contained in floors. Table 4.3 is an example of an XML element representing a campus with sub-elements (nodes) representing the architectural elements contained in the campus. Each architectural element has its own neighbourhood and location. This information is represented by nodes as well. Other information can also be stored, such as the difficulty of accessing the site and other detailed information regarding the place.

In reality, the number of buildings in a campus is large, and there may be numerous rooms in each building. Therefore, the map database can be too

large to maintain and update. A Java class *ConstructMap.class* is implemented to parse the map XML file and create lists of architectures. Then, neighbours are connected by edges, with lengths specified in the XML file. Given the name of a place, *ConstructMap.Class* can look it up from the lists that contain all architectural elements. *ConstructMap.Class* also provides method for adding elements representing new places into the existing map XML file. It requires another XML file containing the new elements in the same format as in the example in Table 4.3. *ConstructMap.Class* will find out whether a campus or other architectural element is added, and connect the new place with its neighbours. The user can also modify the existing map information by clicking the "Map Info" button from the main screen. A frame for modifying and viewing map information is then displayed, as Figure 4.17. There are four levels of architectural elements for this application. More levels can be added by an easy implementation. After clicking the "Select a campus" or other buttons, the drop-down menu on the left-hand side is activated, so the user can select the place that he/she wants to modify or view. Figure 4.18 shows the resulting screen for displaying the information for the UBC southeast campus.

Table 4.3: A simplified example of the representation of a campus in XML form

```
<camp name="UBC southeast">
 <camp neigh name="UBC southwest">
  <distance>100.0</distance>
  ...
 </camp neigh>
 <x min>200</x min>
 ...
 <build name="UBC DEMP">
  <build neigh name="UBC ICCS">
   ...
  <build neigh>
  ...
  <floor name="DEMPfloor1">
   <floor neigh name="DEMPfloor2">
    ...
   </floor neigh>
   ...
   <room name="DEMP101">
    <room neigh name="DEMPdoor1">
     ...
    </room neigh>
    ...
   </room>
  </floor>
 </build>
</camp>
```

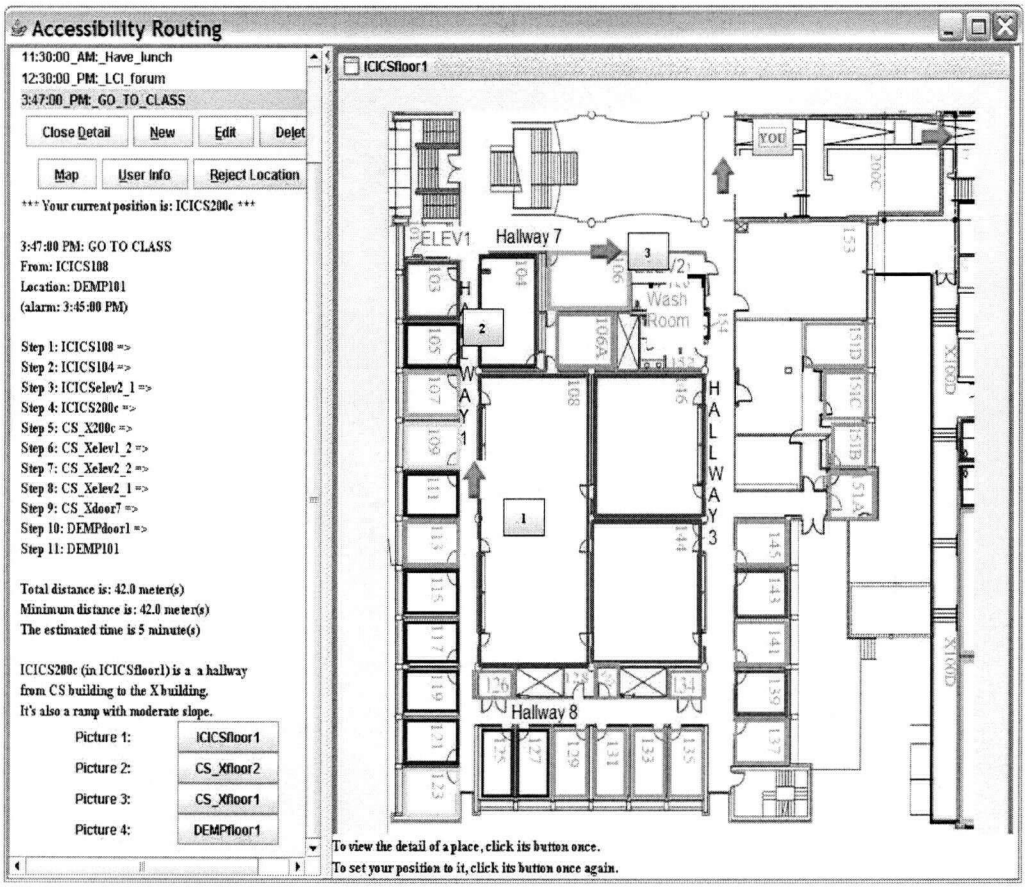

Figure 4.7: A screen shot of the system when the client arrives at a certain position

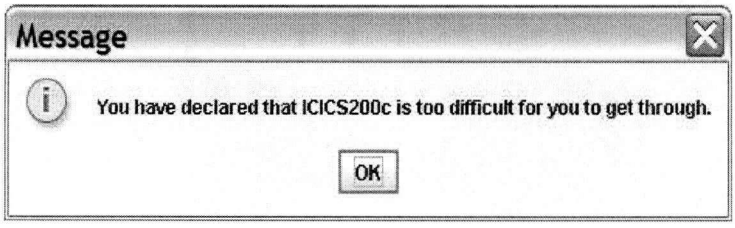

Figure 4.8: The rejection message if the client clicks the "Reject Location" button

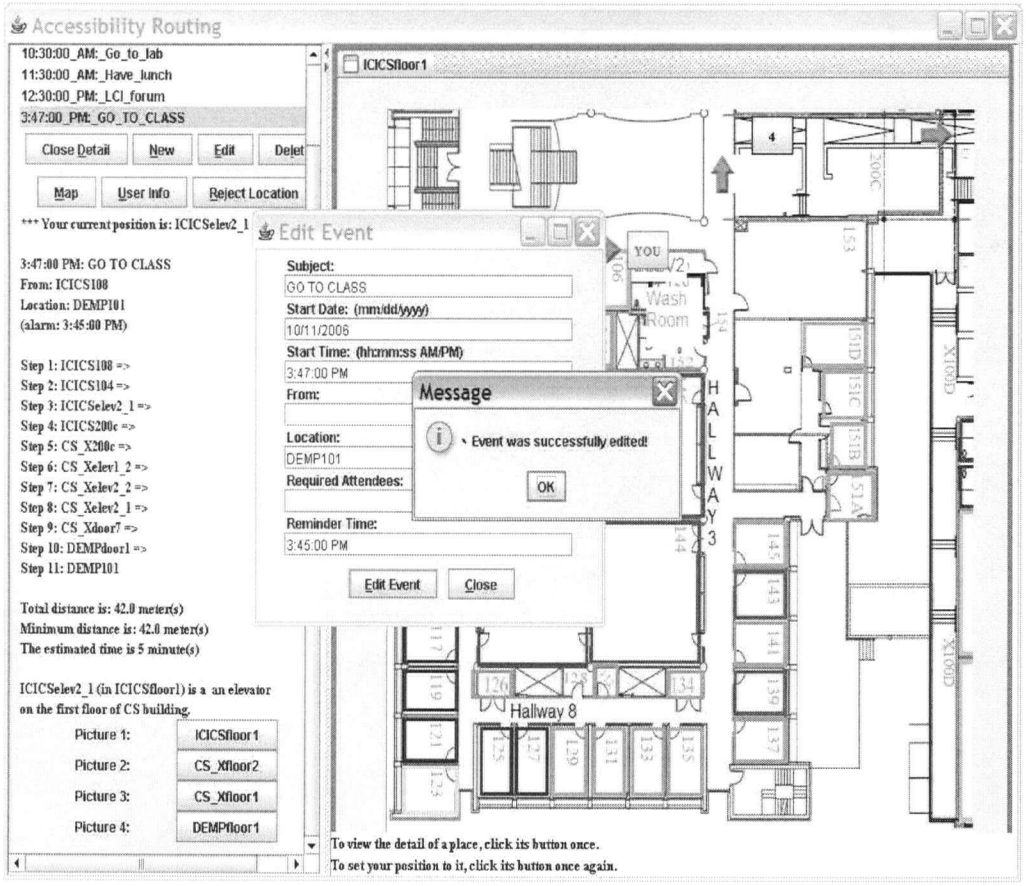

Figure 4.9: A screen shot for updating information for an event

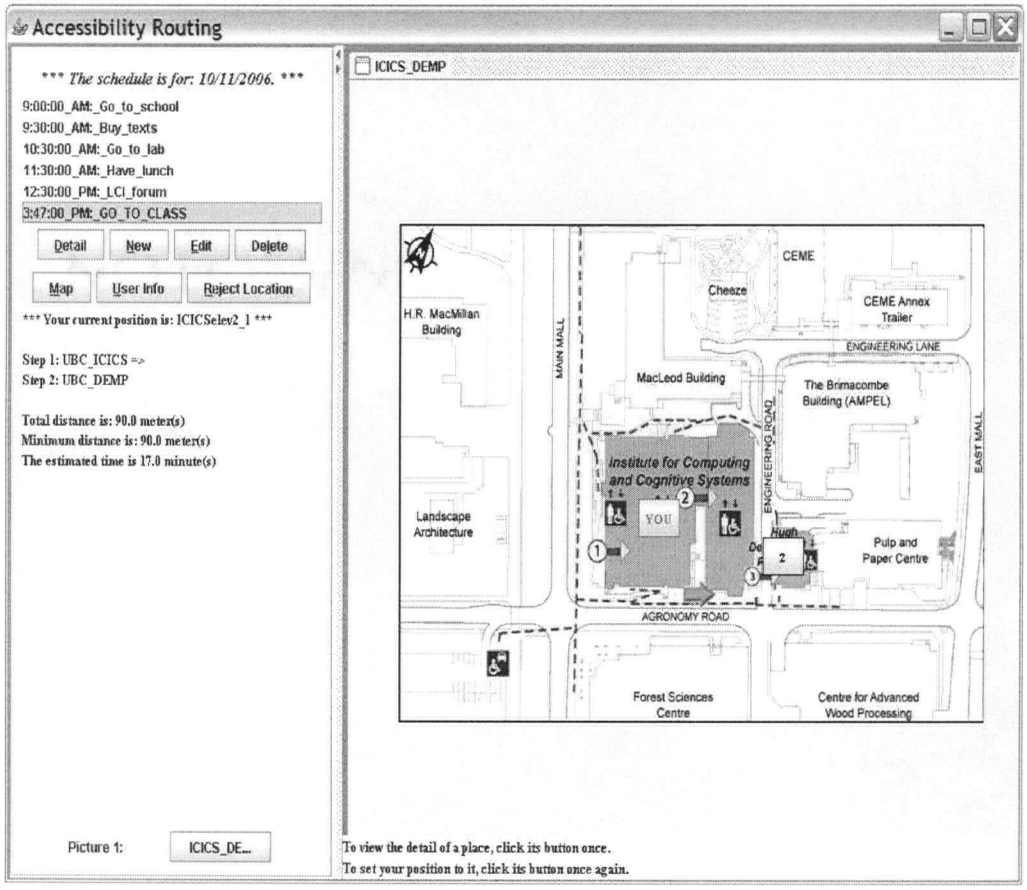

Figure 4.10: A screen shot of the system when a new high-level path is found

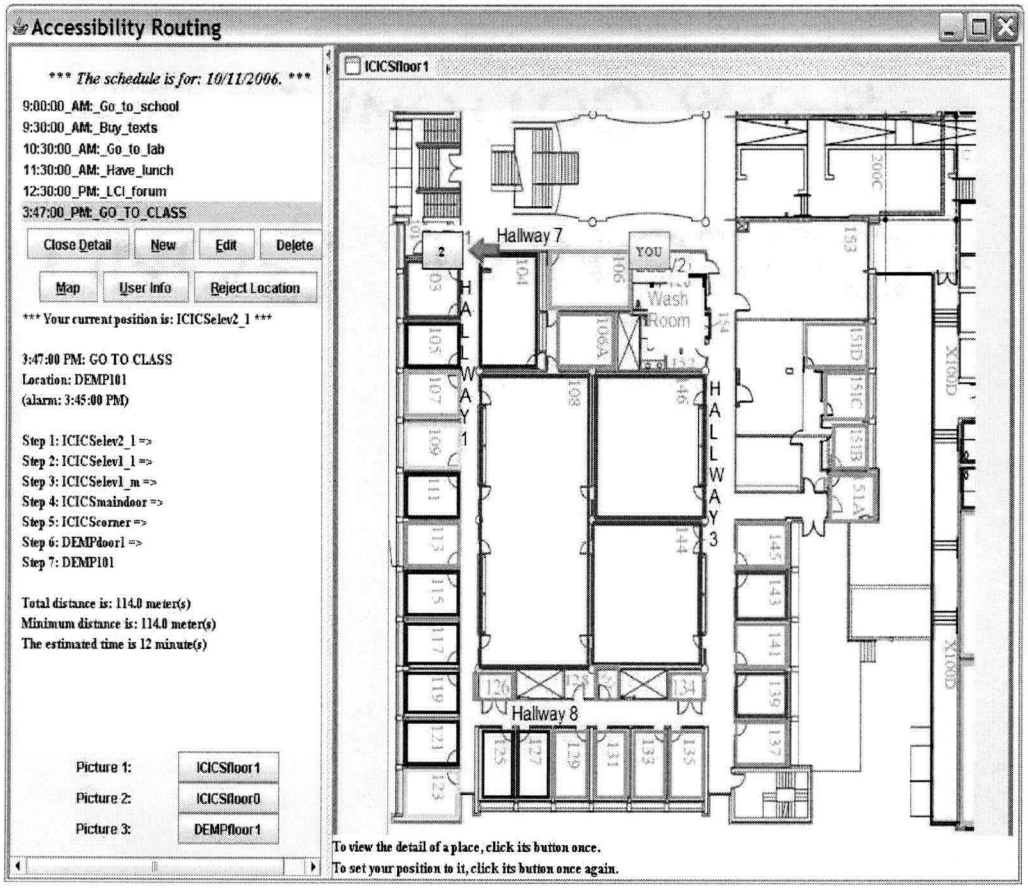

Figure 4.11: A screen shot of the system when it displays a new low-level path

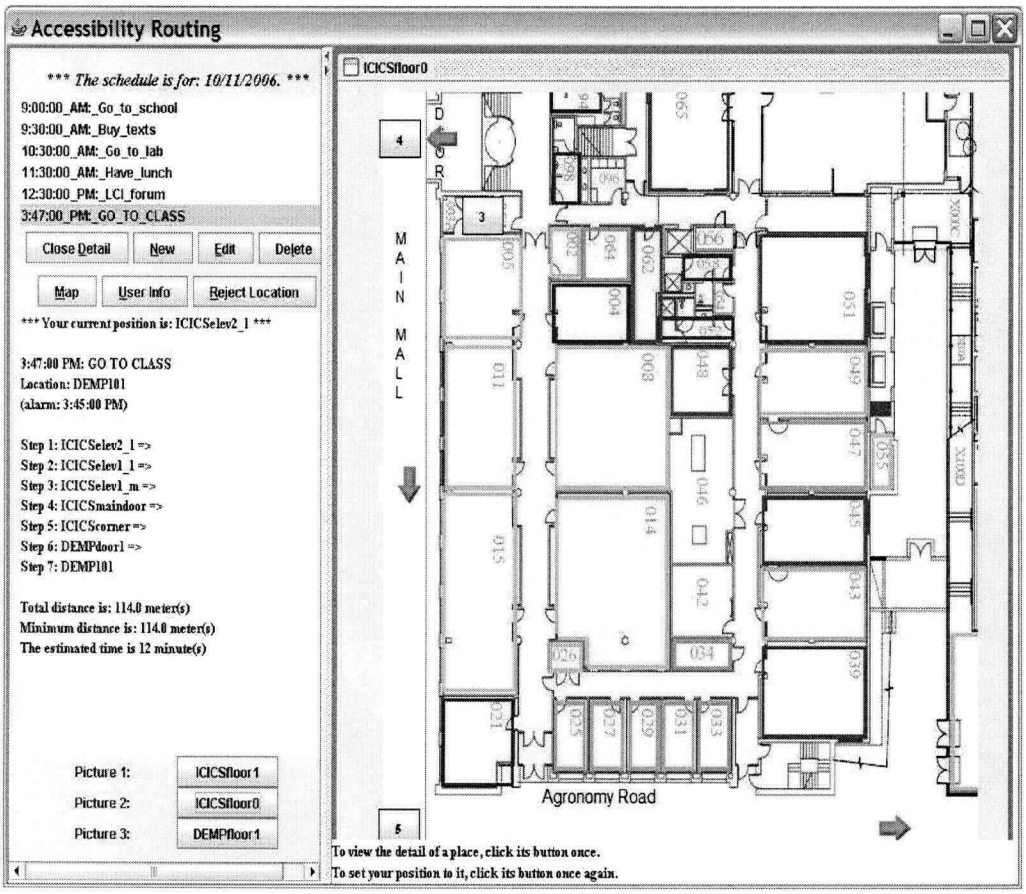

Figure 4.12: A screen shot of the second picture of the newly found low-level path

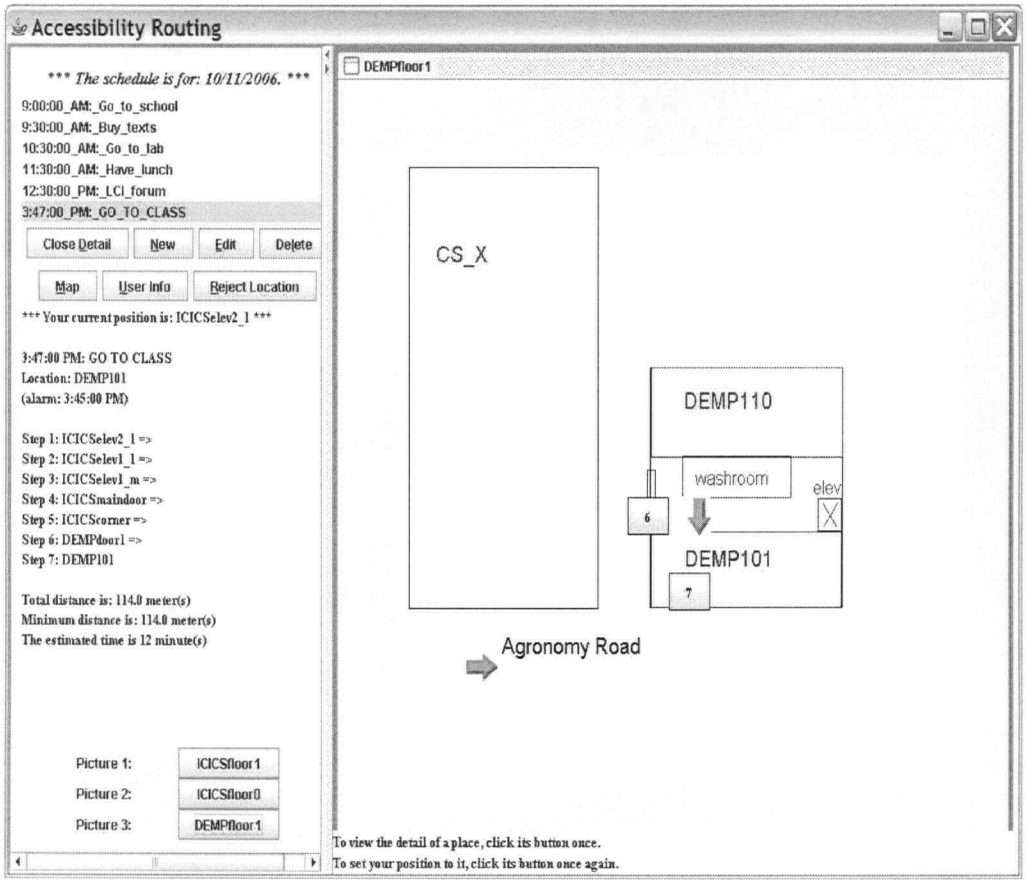

Figure 4.13: A screen shot of the third picture of the newly found low-level path

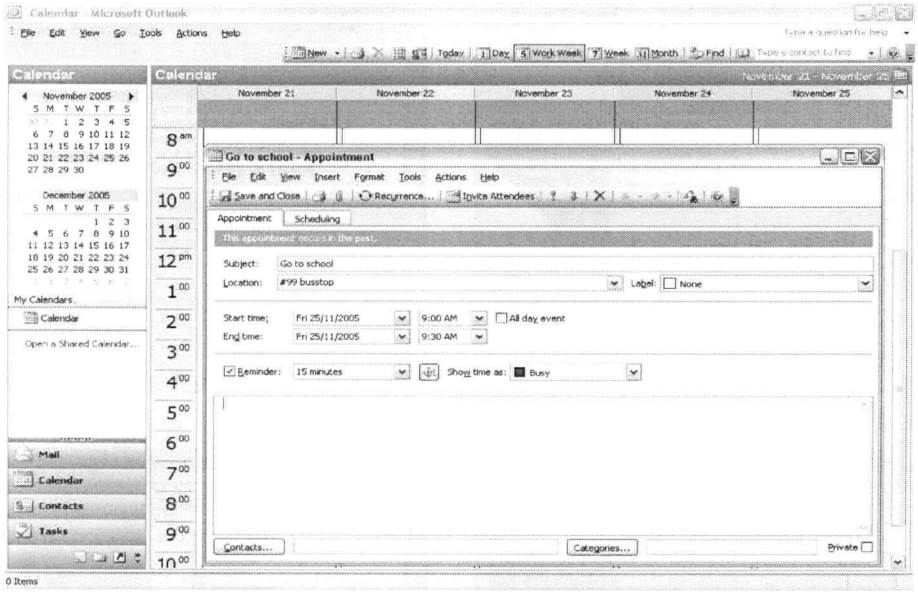

Figure 4.14: A screen shot of the calendar of Microsoft Outlook with a frame for adding a new appointment

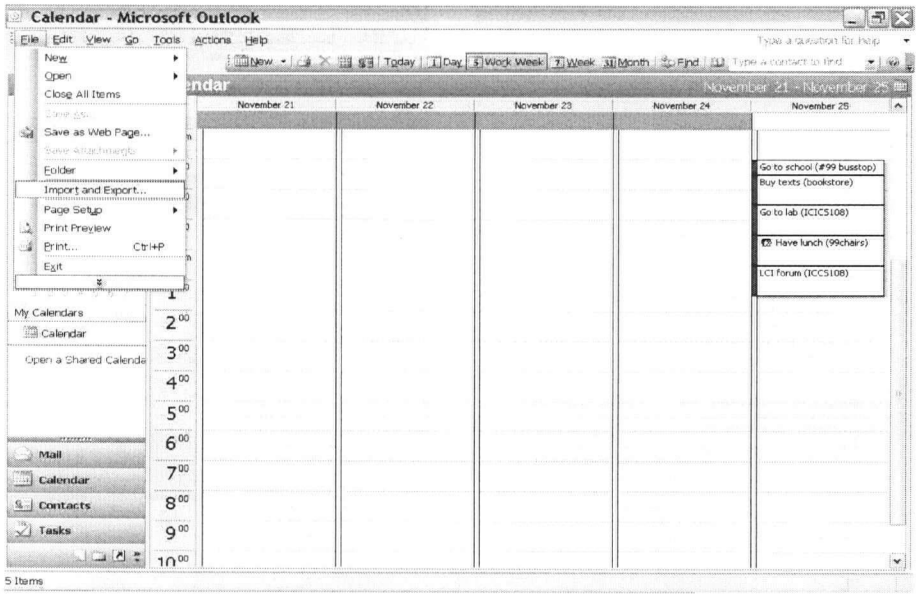

Figure 4.15: A screen shot of Microsoft Outlook calendar with some events entered

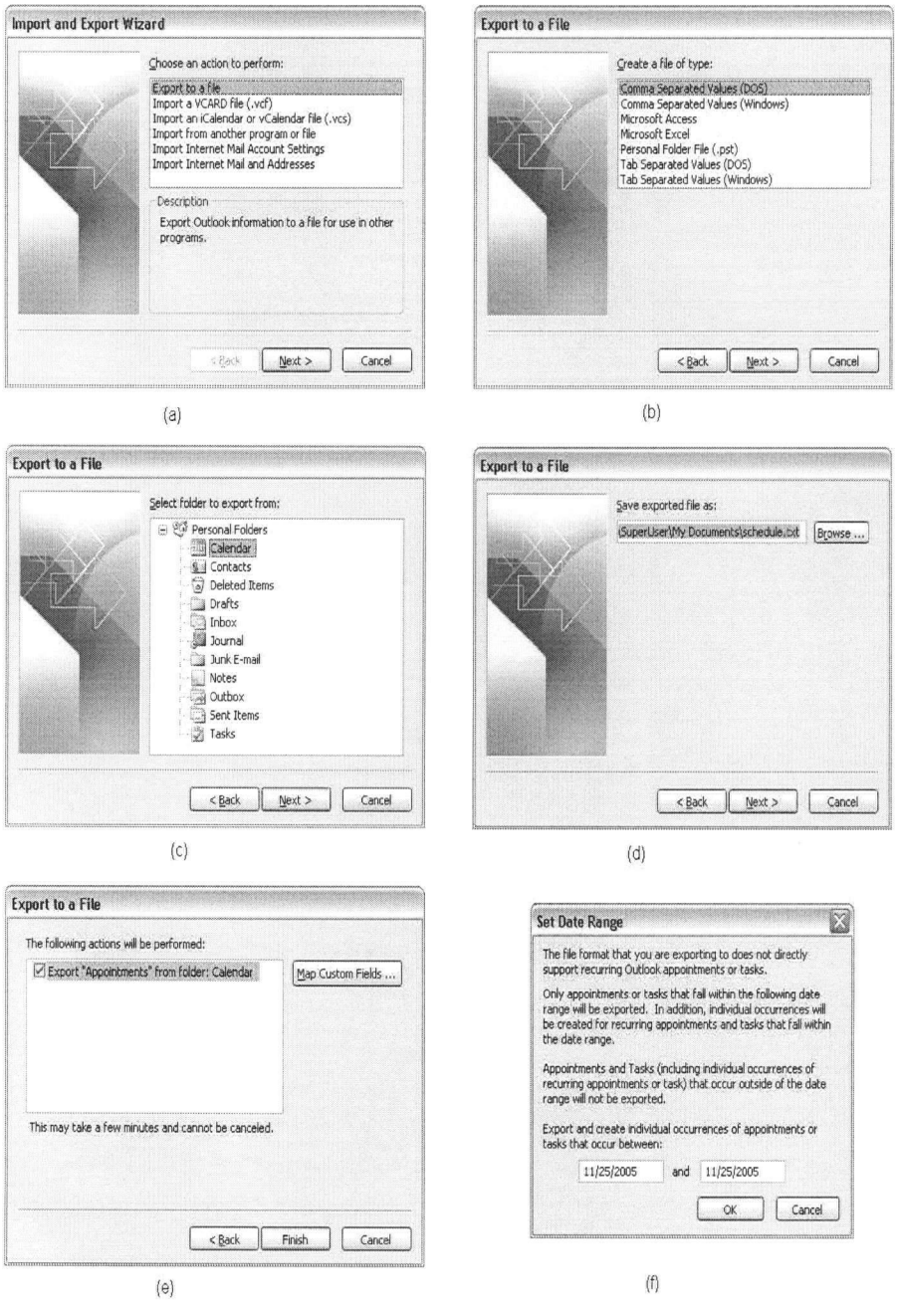

Figure 4.16: A sequence of screen shots when exporting event information from Microsoft Outlook calendar

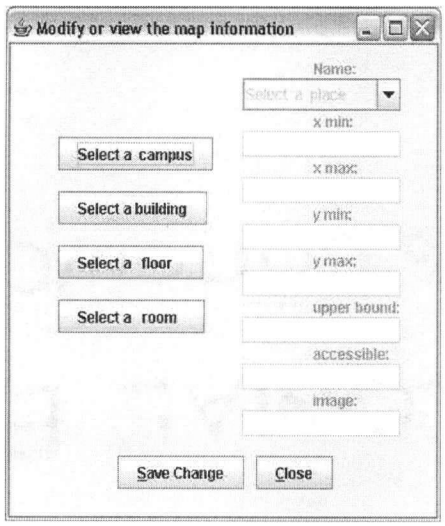

Figure 4.17: A screen shot of the frame for modifying or viewing map information

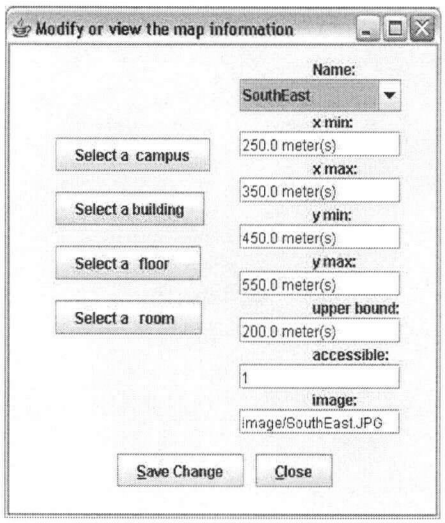

Figure 4.18: A screen shot of the frame for viewing campus information

Chapter 5

Implementation

We implemented our project in Java, which is a widely-used, object-oriented (OO) programming language. Object orientation is an approach to organizing both the problem and its solution as a collection of objects, together with their data structure and behavior, in the software development process [23]. There are seven characteristics to OO representation: identity, abstraction, classification, encapsulation, inheritance, polymorphism and persistence. These characteristics assist developers in embracing some or all notions involved in object orientation.

Unified Modeling Language (UML) is used in this chapter to generate a design for the Scheduler and Route Planner System, since UML can competently illustrate the concepts of the OO process. UML can be used throughout the software development process, because OO concepts apply to all stages of the process. In particular, UML can be used to describe different design alternatives in a visual manner.

5.1 Inheritance and Polymorphism in Java

Among all of the characteristics of OO representation, inheritance and polymorphism are two major ones allowing code reuse. In a large project, code-reusing is exceptionally useful. In our project, the object classes representing an architectural

element are hierarchical. *Campus* is at the highest level, while *Room* is at the lowest. To insert one more sub-level, we can apply the inheritance in Java using the following syntax:

```
public class NewClass extends Room{
// class definition here.
}.
```

Functions from the superclass or any ancestor are inherited. However, if a function behaves better in another manner, a new function declaration and definition can be implemented using the same prototype as that of the existing function. The existing function is overloaded by the new one for the new class. This is the main concept behind polymorphism.

In the design process, we used a UML diagram to describe objects, classes, behaviors and the relationships between them. The drawing can be done with paper and pencil, so we can easily modify and improve the design, or even change it entirely. Figure 5.1 shows the first cut at the design of our system. It includes nine boxes representing nine classes needed for our project. Four levels of hierarchy of architectures in our sample world are related using the "is-a" arrow. Most functions are implemented in the topmost level, *Campus*. *Campuses* consist of a *Path* and a *CamPred*, which is a class representing a set of campuses with their predecessors' names captured. *Graph* draws the interface items of the system. In *Graph*, lists of *campuses*, *buildings*, *floors* and *rooms* are maintained for the client. *ConstructMap* helps *Graph* to build the map database, while *TransferSchedule* builds the list of events.

With the gradual development of the project, two more classes, *Along* and *ImageList*, were considered helpful. *Along* is the direction and name, if applicable, from a place to its neighbor. Hence, we can provide more detail to users with the *Along* class. *ImageList* reads in an XML file and parses a list of images that are available for users to view. Therefore, the administrator can manage all images

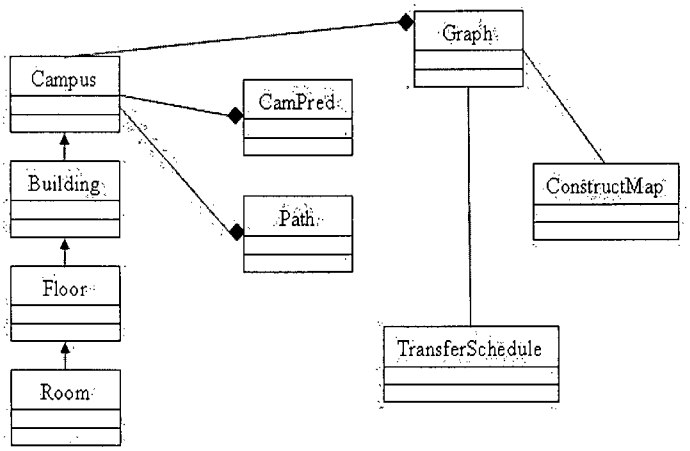

Figure 5.1: First cut at the Scheduler and Route Planner System design

easily. Figure 5.2 shows the final cut at the system design, which involves some possible class attributes and behaviors.

5.2 System States and Activities

In addition to the static view of the system, UML diagrams provide a dynamic view to illustrate system states and activities [23]. State and activity diagrams are two general approaches to describing a dynamic model of the system. Since most behaviors of our system are straight-forward, we will illustrate only the activity diagram for computing the shortest path from a start node to a destination node and for prompting reminder messages. In a UML activity diagram, a filled circle represents the start node, while a black circle inside a white circle represents the end node. A rectangle represents a state, with an arrow showing transitions from one state to another. A diamond indicates that a decision should be made, and two different steps will follow.

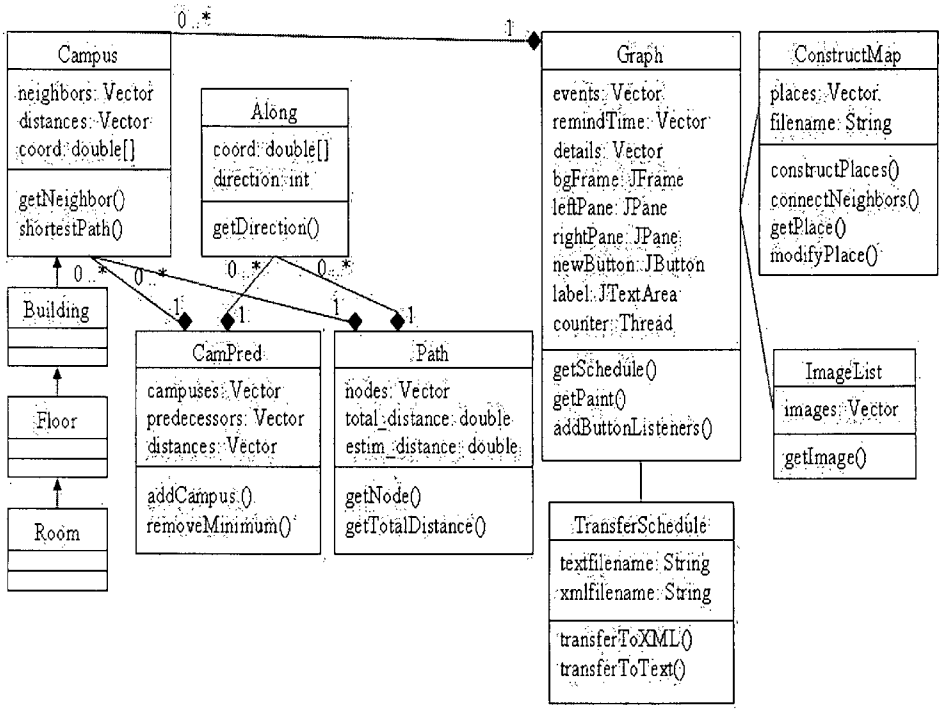

Figure 5.2: Final cut at the Scheduler and Route Planner System design

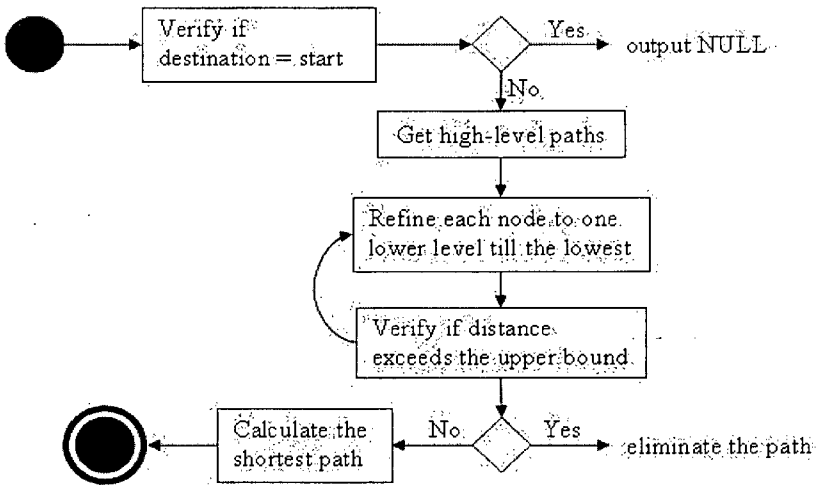

Figure 5.3: The activity diagram for the Hierarchical Shortest Pathfinding algorithm

5.2.1 Computing the Shortest Path

To compute the shortest path from a start node to a destination node, the system verifies of whether the destination node is the start node first. Figure 5.3 shows that after the verification is made, *NULL* is output if it is true. Otherwise, determining high-level paths and other activities will be involved. After one or more high-level paths are found, a refinement is done recursively for each high-level path until the lowest-level architectural element is reached. Some paths may be pruned away in the recursive loop if their lengths exceed a upperbound limit. The system calculates the minimum length among all the paths left. Following the arrow on the diagram, the shortest path will be found at the end node.

5.2.2 Prompting Reminder Messages

Figure 5.4 shows the activity diagram for prompting both manual and automatic reminder messages. The manual reminder times are input by the users, so the system

can obtain it from the *schedule.txt* file. An automatic time requires some calculation after the shortest path is found for each event. The program obtains the system time and compares it with the reminder times. If times match, the system sends prompting messages to remind the user. This process executes until the program ends.

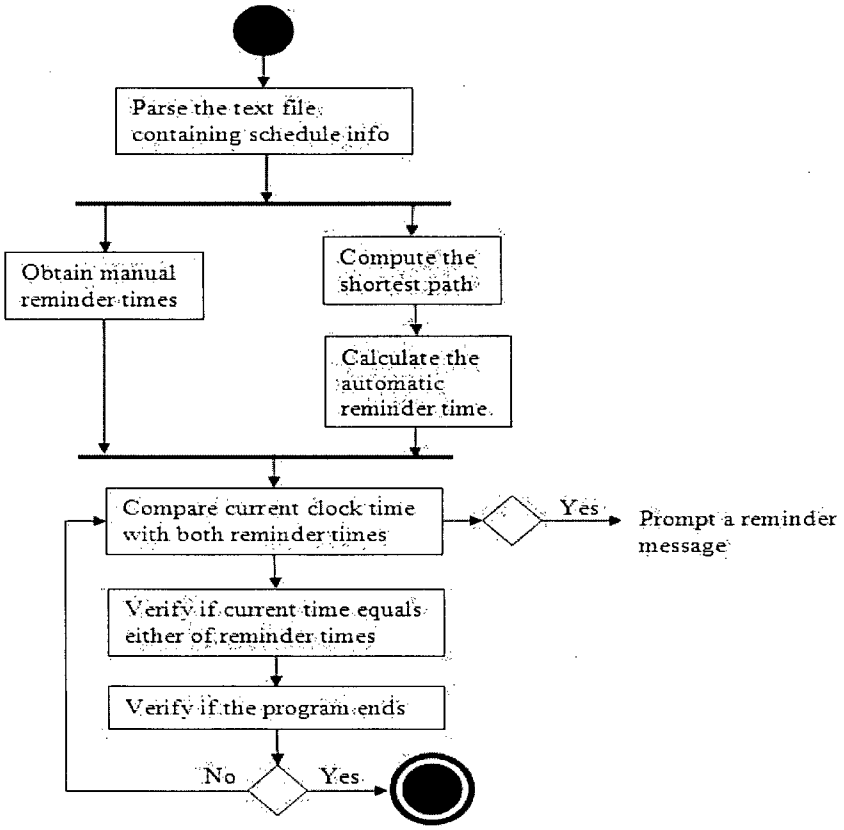

Figure 5.4: The activity diagram for the manual and automatic reminder function

Chapter 6

Experiments and User Study

The literature on wheelchair navigation, discussed in Chapter 2, suggests developing a autonomous or semi-autonomous system to physically control wheelchair movement. However, problems may arise when wheelchair users are travelling in an unknown place. The purpose of our project is to solve these problems and provide a user-friendly interface design. Situations vary depending on road conditions. In this chapter, we describe an evaluative study designed to address questions regarding the usability of our system, comparing the performance of wheelchair users and non-wheelchair users.

6.1 Motivating Scenarios

The Scheduler and Route Planner system can be useful in different cases. Two examples are discussed as follows. The first one describes a scenario for a wheelchair user who has used the system for a while. The user can reject a path that the route planner suggests to him/her, which modifies the system's known navigating ability of the user. The second example involves a new wheelchair user, who may need a more detailed description of the nodes on each path.

1. Mary has been using a wheelchair for ten years. She is a staff member at Disability Resource Center at UBC, and her activities mostly take

place in the northeast quadrant of the campus. Mary is going to attend a talk on cognitive assistive systems at the Department of Computer Science (CS) located in the southwest quadrant of the campus. She has little information about the building housing the CS department. With the Scheduler and Route Planner system, Mary inputs the destination of the event, and a path is found with detailed descriptions of which ramps and elevators she should take. Since Mary has been using the system for several months, it notices that some slopes and roads are too rough for her and eliminates paths invoking those slopes and roads. She arrives at the CS department and attends the talk.

2. John, a student at UBC, injured his leg one month ago. He usually carpools to school with his friend, but his friend is out of town today so he is going to school by bus. He plans to buy some text books, but he doesn't know the accessible wheelchair routes from the bus stop to the bookstore. Therefore, he mounts a tablet PC with our system installed on his wheelchair. The route planner provides John with the easiest path, as well as relevant road conditions. John follows the route and arrives at the bookstore. After John buys the text books, the scheduler alerts him that he has a class in ten minutes. He checks the route planner again for the best path available to get him there on time.

6.2 Evaluators and Task Guidelines

In this evaluative study, we invited two wheelchair users, Dave and Yasaman, and two non-wheelchair users, Chao and Jerry. Dave has used a wheelchair for 31.5 years and Yasaman for 4 years. Table 6.2 shows a summary of three participants' relevant information, with the same column attributes as in Table 3.1. Evaluators are asked to perform a number of actions that assume they are students in the Department of

Computer Science at UBC, and are working in their research lab, the Laboratory for Computational Intelligence (LCI). Each participant takes about half an hour to complete the following list of tasks. Their performances are measured in terms of correct and incorrect button clicks, total time, number of major questions, etc.

Table 6.1: Summary of participant information

Participants	Use Wheelchair	Age	Gender	Education	Computer Use
Dave	Yes	50	Male	16	Often
Yasaman	yes	25	Female	14	Often
Chao	No	25	Male	18	Often
Jerry	No	25	Male	16	Often

1. Assume that you are at ICICS 108 (named ICCS108 in the database), and the current time is 1:10:00 PM. You will have a class at Dempster 101 (named DEMP 101 in the database) in twenty minutes.

2. Create a new event using the scheduler function from our system. Enter the information of the next class, as follows:

   - Subject: Go to class
   - Start Date: 09/17/2006
   - Start Time: 1:30:00 PM
   - From: ICCS108
   - Location: DEMP101
   - Reminder: 1:20:00 PM

3. Wait until a reminder is triggered. Then view the high-level path, which provides you with a general idea about the path.

4. If you are not sure of the exact path from LCI to the classroom, view the detailed path by clicking the "Detail" button. You can also click on the room button to view its information.

5. Following the path, you arrive at a steep ramp that you may not be able to pass through. Reject the current location.

6. Change your position to the previous step by double-clicking on the corresponding button. (Note: this will let the system know where you are along the path.)

7. Edit the event information (i.e., remove the text in the "From" field).

8. Review the high-level and low-level paths. Follow the direction and arrive at DEMP101.

One research investigator accompanied with each evaluator during the test. The researcher provided an introduction to the system and help when the evaluator needed. The researcher also observed the performances of all evaluators, and compared the performance differences between wheelchair users and non-wheelchair users by the required time, the number of false button clicks, the number and contents of questions, etc. In addition, interest in system appeared by each evaluator was also observed by the researcher during the discussion of evaluating the system and suggestion or recommendation of future work.

6.3 Evaluation Results and Recommendation

We conducted survey with each evaluator regarding the usability of the Scheduler and Route Planner system. All evaluators were interested in the system design, but suggested the following modifications and improvements:

- Although an alternative path is provided to users after they reject a difficult location, alternative paths could be provided in general as different options for the users.

- The application covers only a small local area, and a major portion of the paths are indoors. Therefore, transportation is not shown. However, transportation

methods may affect the pathfinding and user speed. Hence, an indicator of transportation methods is suggested.

- Information about whether a door is powered, as well as the door width, could be added to the database.
- The North direction should be included on each picture.
- More directional arrows are needed when an edge is long or involves turns.

After the user study test, the participants were asked to evaluate four aspects of the system: interface design, functionality satisfaction, ease of use, and willingness for future use, as explained in Chapter 3. Figure 6.1 displays the results.

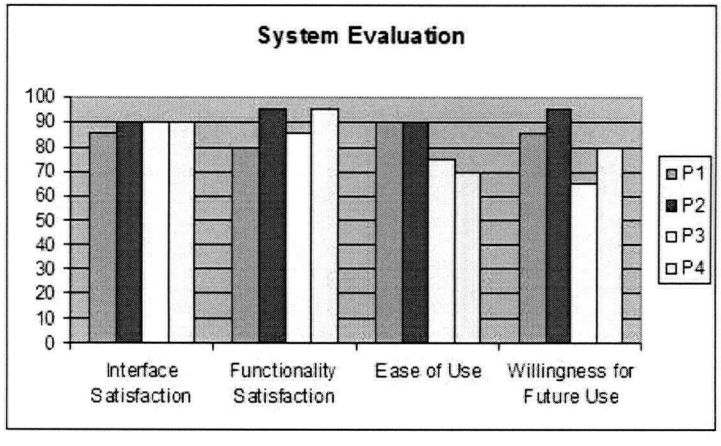

Figure 6.1: Evaluation of the Scheduler and Route Planner system

Chapter 7

Conclusions and Future Work

The major goal of this research was to discover how modern technologies might help wheelchair users carry out daily activities. A significant sub-goal was to build an efficient system that helps wheelchair users in route-finding. We met this sub-goal through the design and creation of the prototyped system and the evaluative studies. In this chapter, we summarize how this sub-goal was accomplished and why it is an significant step towards the major goal. Finally, we discuss anticipated future work.

7.1 Conclusion

In this thesis, we have focused our efforts on building the Scheduler and Route Planner system, which can find both an abstracted high-level path and a detailed low-level path. There has been extensive research in the cognitive science field that aims to provide help in daily activities. Some researchers devoted their effort on building systems to provide pathfinding and direction-reminding help for people with mental problems or for forgetful clients. However, these systems provide only limited information about direction, but not related to accessible routes. Hence, these systems are not applicable for wheelchair users. Although systems exist to help wheelchair users avoid obstacles, these system do not provide future path-planning. In the system design and evaluation phases, we used interactive methods,

which involved potential users, both wheelchair and non-wheelchair individuals, to evaluate each level of system prototypes and the actual system. This evaluative study lets us realize user's need towards interface specification and functionality requirements. We apply efficient path-finding algorithms, the HSP and HSPA*, in our system, which runs on hierarchically presented maps containing only wheelchair accessible routes.

In addition, the scheduler provides basic functions for users to keep track of their activities. With Microsoft Outlook providing augmentative functionality, our system worked well during the evaluative tests. Manual and automatic remainders are applied to announce events in the near future. An automatic reminder is created by the system after it calculates the approximate distance to the next destination. The system takes user speeds as inputs and estimates accurate auto-reminder time. The system is also designed to learn user abilities, which are assumed to be strong originally and are updated along the execution process. An alternative path is displayed, instead of a shorter path that contains non-acceptable locations for individual user.

With all of the functionality stated above, the new cognitive assistive system is able to contribute to helping users. The evaluative user study presented in Chapter 6 shows that the evaluators appeared interested in using the system in the future. Nevertheless, we can extend our research in several areas, which will be discussed in the next section.

7.2 Discussion and Future Work

There are six areas where further research is required. First, the HSP and HSPA* algorithms can be refined and tested further to achieve even better performance. Second, the Java GUI interface implementation needs improvement. Third, the database could be enlarged to include more buildings and information. Fourth, advanced machine learning methods could be applied to refine the user model. Fifth,

the system could be redesigned for other classes of users. Finally, we could enhance our system to make it more marketable.

7.2.1 Optimizing the HSP and HSPA* Algorithms

In this thesis, the Hierarchical Shortest Pathfinding (HSP) algorithm is presented and its running time analyzed. The speedup from eliminating unnecessary high-level paths is remarkable, and good performances of HSP is expected in real-world pathfinding problems. There are a few directions that this research could be extended beyond the work shown in Section 4.2. First, more examples are anticipated to analyze the performances of the four algorithms. We analyzed the running times of the HSP and HSPA* algorithms based on artificial examples. We shall apply the algorithms on more actual examples, such as the whole UBC campus. Second, as described in Section 4.2.3, the HSPA* algorithm contains two parameters, namely α and β. The best-fit values of these two parameters may vary depending on different problems. Therefore, it may be worthwhile learning well-suited values for these two parameters using stochastic local search methods. Third, comparison of HSP, HSPA* and other hierarchical pathfinding algorithms could be further investigated in terms of their running times and performance.

7.2.2 Improving Interface Implementation

The locations of buttons representing path nodes may not be ideal. These buttons show the step numbers in the found path, since the button size is not large enough to contain a full name of a place. They should show the names of their corresponding places, so that the users do not need to check the path description to see what position they are in. A better layout method could be used to solve these problem in the future. In addition, as pointed out by the evaluators, there is only one arrow from one node to the next node, which may not be clear if the road is long and involves turns. In other words, we should enlarge the database to include the

information of the arrows and refine the *Along.java* class to accept multiple arrow positions.

7.2.3 Improving Database Management

Our current database includes only a few buildings on UBC campus, so route-finding only functions in this limited area. We could build maps containing the whole UBC campus, or the Vancouver city, or even larger areas. We could also include information about whether a door is powered, its width, etc. All the data could be located in a shared online database, so that users can exchange information by uploading and downloading files. Networking techniques and skills would be needed, and extra care would be required to analyze which data file has the latest update and to combine parallel changes.

7.2.4 Applying Advanced Machine Learning Method

In Section 4.3, we mentioned that more sophisticated machine learning methods are anticipated for modelling the specific user model. This refinement would allow the system to better understand the users, and provide more suitable paths for them. The new user model could include the transportation methods, so the system could provide better estimated travelling time.

7.2.5 Redesigning for Different Classes of Users

The system can be extended or redesigned for different classes of users. With voice communication, our system is also suitable for the visually impaired. Also, the hierarchical pathfinding algorithm can be redesigned to generate procedures of performing activities. The procedures can be viewed as general or detailed steps, which would be helpful for people with mental problems, such as Alzheimer's disease. In addition to people with impairments, normal people can get help from our system if it is installed on a handy device and applied on a large map.

7.2.6 Future Development in the Market

Beyond the work in this research, our system can be extended to advanced software which can be installed on a smaller device, such as PDA and cellular phone. Since cellular phone is easy to carry and widely used, our system, with different implementation and interface design that suit for cellular phone, could be more marketable.

Bibliography

[1] Clever project, http://l3d.cs.colorado.edu/clever/.

[2] Overwhelming need for wheelchairs, http://wheelchairfoundation.ca/.

[3] Aware home project, http://www.cc.gatech.edu/fce/ahri/.

[4] Nursebot project, http://www.cs.cmu.edu/~nursebot/.

[5] Assisted cognition, http://www.cs.washington.edu/assistcog/.

[6] Accessibility routing, http://www.maps.ubc.ca/PROD/accessRoute.php.

[7] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[8] A. Botea, M. Müller, and J. Schaeffer. Near optimal hierarchical path-finding. In *J. of Game Develop.*, pages 7–28, 2004.

[9] G. Fischer, E. Arias, S. Carmien, H. Eden, A. Gorman, S. Konomi, and J. Sullivan. Supporting collaboration and distributed cognition in context-aware pervasive computing environments, 2004.

[10] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Trans. on Systems Science and Cybern*, pages 4:100–107, 1968.

[11] R. Holte, T. Mkadmi, R. M. Zimmer, and A. J. MacDonald. Solving problems by searching: A graph oriented approach. In *Artificial Intelligence*, pages 85(1–2):321–361, 1996.

[12] H. Kautz, D. Fox, O. Etzioni, G. Borriello, and L. Arnstein. An overview of the assisted cognition project. In *American Association for Artificial Intelligence*, 2002.

[13] R. Korf. Depth-first iterative-depening: an optimal admissible tree search. In *Artificial Intelligence*, pages 27(1):97–109, 1985.

[14] R. Korf, M.Reid, and S. Edelkamp. Time complexity of iterative deepening-a*. In *Artificial Intelligence*, pages 199–218, 2001.

[15] L. Liao, D. Fox, and H. Kautz. Learning and inferring transportation routines. In *American Association for Artificial Intelligence*, 2004.

[16] C. McCarthy and M. Pollack. A plan-based personalized cognitive orthotic. citeseer.ist.psu.edu/mccarthy02planbased.html, 2002.

[17] J. McGrenere, R. Davies, L. Findlater, P. Graf, M. Klawe, K. Moffatt, B. Purves, and S. Yang. Insights from the aphasia project: Designing technology for and with people who have aphasia. In *Proceedings of ACM Conference on Universal Usability*, pages 112–118, 2003.

[18] D. P. Miller and M. G. Slack. Design and testing of a low-cost robotic wheelchair prototype. In *Autonomous Robots*, volume 2, pages 77–88, 1995.

[19] M. Minsky. Steps toward artificial intelligence. *IRE Transactions on Human Factors in Electronics, HFE-2*, pages 39–55, 1961. Reprinted in Computers and Thought, Ed. E.A. Feigenbaum and J. Feldman, McGraw-Hill, 1963.

[20] K. Moffatt, J. McGrenere, B. Purves, and M. Klawe. The partricipatory design of a sound and image enhanced daily planner for people with aphasia. In *Proceedings of ACM CHI*, pages 501–510, 2005.

[21] J. Ojala, K. Inoue, K. Sasaki, and M. Takano. Development of an intelligent wheelchair using computer graphics animation and simulation. In *Computer Graphics Forum*, volume 10, pages 285–295, 1991.

[22] D. Patterson, O. Etzioni, and H. Kautz. The activity compass. In *First International Workshop on Ubiquitous Computing for Cognitive Aids*, 2002.

[23] S. L. Pfleeger. *Software Engineering.* 2 edition, 2001.

[24] S. Rabin. A* aesthetic optimizations. In *Game Programming Gems*, pages 264–271, 2000.

[25] R. Simpson, S. P. Levine, D. A. Bell, L. A. Jaros, Y. Koren, and J. Borenstein. The navchair assistive navigation system. In *IJCAI-95 Workshop on Developing AI Applications for People with Disabilities*, pages 167–178, 1995.

[26] R. Simpson, S. P. Levine, D. A. Bell, L. A. Jaros, Y. Koren, and J. Borenstein. Navchair: an assistive wheelchair navigation system with automatic adaption. In *Springer-Verlag*, 1998. Lecture Notes in Artificial Intelligence: Assistive Technology and Artificial Intelligence, V. Mittal, H. A. Yanco and J. Aronis (eds.).

[27] S.Russell and P. Norvig. Solving problems by searching. In *Artificial Intelligence A Modern Approach*, 1995.

[28] N. Sturtevant and M. Buro. Partial pathfinding using map abstraction and refinement. In *AAAI-05*, pages 1392–1397, 2005.

[29] H. Wakaumi, K. Nakamura, and T. Matsumura. Development of an automated wheelchair guided by a magnetic ferrite marker lane. In *Journal of Rehabilitation Research and Development*, volume 29, pages 27–34, 1992.

[30] H. Yanco. Wheelesley: A robotic wheelchair system: Indoor navigation and user interface wheelchair. In *New York: Springer*, pages 256–268, 1998. in Assistive Technology and Artificial intelligence, V. Mittal, H. Yanco, J. Aronis, and R. Simpson, Eds.

Appendix A

Consent Form for All Evaluators

Participants were asked to sign a consent form prior to participating in the evaluation phase. The next page is the standard consent form used in this study. We paid the evaluator $10 as an honorarium.

THE UNIVERSITY OF BRITISH COLUMBIA

Department of Computer Science
2366 Main Mall
Vancouver, B.C., V6T 1z4

July 31, 2006

Consent Form

A scheduler and Route Planner for Wheelchair Users

Principal Investigator
Alan Mackworth, Ph.D.

Student Investigator
Suling Yang

Project Purpose and Procedures
This project is designed to develop a system of a scheduler and route planner installed on a small device, such as a tablet PC, for people using wheelchairs. This project also forms the basis of the student investigator's thesis. The purpose of conducting a user study is to improve the system interface design. You will be asked questions about interface prototypes. This session will take no longer than ten minutes. Your answers to the questions asked by the investigator and your suggestions may be used for a research presentation in the Department of Computer Science at the University of British Columbia.

Confidentiality and Contact Information
The identities of all participants will remain anonymous and will be kept confidential. We are very grateful for your participation. If you have any questions or require further information about this project, you may contact either of the investigators.

Consent
Your participation is entirely voluntary and you may refuse to participate or withdraw from the study at any time. Your signature below indicates that you have received a copy of this consent form for your own records, and that you consent to participate in this project. You do not waive any legal right by signing this form.

I, _____, agree to participate in this project. My participation is voluntary and I understand that I can withdraw at any time.

_____
Participant's Signature Date

_____
Student Investigator's Signature Date

Appendix B

Questionnaire for All Evaluators

B.1 Questionnaire for the Evaluators Who Participated in the Design Phase

Participants in the evaluative study for design specifications were asked questions regarding their opinions and suggestions about our proposed system. Text page shows the questionnaire for the evaluators who participated in the design phase.

B.2 Questionnaire for the Evaluators Who Participated in the System Evaluation Phase

Participants involved in the system evaluation phase will be asked the same questions as were the design evaluators. Participants for this phase will also be asked the following questions:

- Did you encounter any problems in using the device that has the Scheduler and Route Planner system installed on it? If yes, please explain.
- What functions are not too clearly shown but you think are useful? How would you change the design of these functions?

- What functions are worth adding to the system? What level of usage of these functions would you expect?
- Did you need the investigator's explanation or any help with the test? If yes, what problem(s) or question(s) did you encounter?

THE UNIVERSITY OF BRITISH COLUMBIA

Department of Computer Science
2366 Main Mall
Vancouver, B.C., V6T 1z4

July 31, 2006

Questionnaire

A Scheduler and Route Planner for Wheelchair Users

1. Personal Information:
   Age ____ Year(s) of using wheelchair ____ Year(s) of education ____
   Frequency of computer use: a. Often b. Sometimes c. Seldom d. Never ____

2. Are you satisfied with the locations of interface components? If not, what makes you unsatisfied?

3. Are you satisfied with the number of clicks to access a certain function? If not, what makes you unsatisfied?

4. Can you give a grade out of 100% for the interface design?

5. Are you satisfied with the functionality that the system provides? If not, what makes you unsatisfied? Can you give a grade out of 100% for this aspect?

6. Do you think the system is ease to use? What aspects should be improved? Can you give a grade out of 100% for the ease of use of our system?

7. Would you use our system if it is produced in the future? Can you give a grade out of 100% for your willingness to do so?