Activity Dispatching for Cooperative Trails-based Applications

Ben Steichen, B.Sc.

A dissertation submitted to the University of Dublin, Trinity College,

in partial fulfillment of the requirements for the Degree of

M.Sc. in Computer Science

University of Dublin, Trinity College

September 2007

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Ben Steichen

September 13, 2007

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Ben Steichen

September 13, 2007

Acknowledgments

I would like to thank my supervisor, Dr. Siobhan Clarke, for her support and guidance during the course of this project. Also, I would like to acknowledge Eamonn Linehan and Cormac Driver for their help and assistance.

Ben Steichen

University of Dublin, Trinity College
September 2007

Activity Dispatching for Cooperative Trails-based Applications

Ben Steichen, M.Sc.
University of Dublin, Trinity College, 2007

Supervisor: Dr. Siobhan Clarke

The challenge of efficient resource allocation is omnipresent in all types of organisations and companies across the global marketplace. Although these resources have generally been very static in the past, dynamic human resources have now become the most important assets of many organisations. Allocating these resources efficiently is therefore of crucial importance for the success of an organisation. The rapid increase of mobile device usage across all organisational and social levels supports this development of growing mobility and is therefore one of the driving factors towards pervasive solutions for the improvement of daily processes. Recent advances in ubiquitous computing offer new opportunities, with the promise of providing a better modeling of the mobility, collaboration and context-awareness of the different actors involved. The interesting question is therefore how to merge the new concepts making up the notion of pervasive computing with the area of resource allocation.

This research dissertation describes the design, implementation and evaluation of a context-aware resource allocation application, which addresses the issues of user mobility in a dynamic environment. Activities to be performed by users are being aggregated by the application and dispatched to users' trails (i.e. collections of activities) based on the current context. The chosen scenario for the application is a hospital environment, where a set of mobile doctors has to perform a multitude of activities with varying priorities. The dissertation hence contributes to the field of pervasive healthcare, as well as the more general domains of ubiquitous computing and resource allocation by combining the research fields in order to provide an insight of what the main challenges are in the corresponding areas and what types of solutions could be applied to the existing problems.

Contents

List of Figures

Chapter 1

Introduction

The challenge of efficient resource allocation is omnipresent in all types of organisations and companies across the global marketplace. Traditionally, these resources have been very static-based, such as for example factory machinery. With a continuous global economic shift to the Service Sector since the 1960s, human resources have now become the dominant assets of most organisations.

At the same time, a development needs to be observed in terms of workforce mobility, as the Tertiary Sector is characterised by a variety of movements of the human resources involved. Allocating these resources effectively and efficiently is therefore of crucial importance for the success of an organisation, especially considering growing global competition.

The rapid increase of mobile device usage across all types of organisational levels supports this development of growing mobility and is therefore one of the driving factors towards new and intelligent solutions for the improvement of daily processes. Recent advances in pervasive computing offer new opportunities for improving traditional methods of resource allocation, with the promise of providing a better modeling of the mobility, collaboration and context-awareness of the different actors involved.

1.1 Motivation

The motivation for this dissertation is therefore the possibility of pervasive computing penetrating into daily human resource allocation processes of dynamic organisations. Although mobile devices are used increasingly, the enormous capabilities and possibilities are by far not being exploited to their full potential.

The different areas motivating this claim are the developments in mobility, collaboration and context-awareness. In each of those areas, a considerable amount of research is currently undertaken, however, the combination of the different areas is often neglected. The interesting question is therefore how to merge these new concepts making up the notion of pervasive computing with the presented area of resource allocation.

1.2 Mobile Devices

Mobile devices have become increasingly powerful, both in terms of usability and processing power. The capabilities are moving closer to powerful desktop computers, with technologies such as the Java Micro Edition not lacking much of the Java Standard Edition functionality. With the help of such a standardised API, it is hence possible to develop powerful, rich-client applications on mobile devices.

In addition to that, due to the mobility provided by these new types of devices, users are in possession of a powerful data processing machine at all times. This enables for example making use of a smart decision making application when a quick choice is required, improving greatly on limited human capabilities. The mobile device is therefore responsible for improving everyday processes, both in terms of quality and efficiency.

However, mobile devices are always limited in terms of size, as well as battery life and connectivity. These issues provide new types of challenges worth exploring. As displays and input devices are usually fairly small in order to provide mobility, the interface design of mobile applications needs to be taken special care of.

1.3 Collaboration

With increasing communication capabilities, mobile devices such as smart-phones or PDAs allow a new interesting way to achieve collaboration among its users. With connectivity technologies such as Infrared, Bluetooth or WiFi, it is possible to connect to a powerful central server or even form small ad-hoc networks.

This development leaves a lot of options of how to perform resource allocation, as it is now possible to share work much easier and in a much more dynamic way. End-users are now able for example to communicate any unforeseen events, as well as to exchange information with peer users.

1.4 Context-Awareness

There are several definitions for the term context. Dey [11] states that "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

According to Loke [31], context in pervasive computing is mostly retrieved via sensor networks surrounding the mobile device users. Furthermore, it is the application's responsibility to transform the sensed information to a relevant context. However, Soke also states that any information sensed by a mobile user can be regarded as context, also including for example a user's emotions.

With an increasing number of these sensor networks and sensing devices, such as for example GPS location receivers, mobile computing has entered a new era of being ubiquitous in people's daily lives. It is therefore possible to design new types of solutions, penetrating even deeper into daily processes, which might have been considered traditionally as not suitable for computing systems.

1.5 Distributed Issues

Pervasive computing also introduces a variety of new issues in terms of device diversity. Compatibility problems might arise as a consequence of differences in software or hardware architectures. In addition to that, different types of devices always have varying strengths and weaknesses in terms of usability and mobility. For example, a powerful server machine is very suitable for supporting a multitude of client machines but it therefore lacks massively in terms of mobility. On the other hand, a handheld device such as a smart-phone might be very mobile, but the input and display capabilities are very weak (small keyboard and screen). It is therefore of crucial importance to design computer systems, which distribute workloads according to device usability and processing capability.

The field of distributed systems is also concerned with problems such as devices getting disconnected from a network for a prolonged period of time and therefore being unavailable for communication with other nodes. Intelligent solutions need to handle these cases efficiently in order for the users to remain as oblivious as possible of such connectivity problems.

1.6 Contribution

Following the outlined issues in resource allocation and ubiquitous computing, this dissertation contributes to both of these research fields by combining the areas in order to provide a model of how pervasive computing can penetrate into daily scheduling processes. A specific focus will be in the area of pervasive healthcare, which depicts one of the many application scenarios of ubiquitous computing and resource allocation.

First of all, an investigation is made to find out the extend to which current designs and solutions model such resource allocation scenarios with regards to pervasive computing developments, again with a special emphasis on pervasive healthcare. Furthermore, the dissertation provides the design and implementation of a solution which addresses the majority of the issues described in this chapter. An evaluation of this solution then provides the research fields with an even deeper insight of what the main challenges are

in the corresponding areas and what types of solutions could be applied to the existing problems.

1.7 Approach

Over the years the areas of resource allocation and ubiquitous computing have both grown into very vast research domains. It is therefore necessary to narrow down the research interests to a more specific domain in order to conduct a precise background and state of the art research. For this dissertation, the growing area of pervasive healthcare has been chosen to illustrate current developments in ubiquitous computing and resource allocation.

By researching current issues and possible solutions in such a reduced subset it is possible to discover general issues, which also apply to different research areas in the same domain. The dissertation is therefore emphasising on pervasive healthcare without neglecting the generalisation of the issues and solutions encountered. Hence, the contribution outlined in section 1.6 will be for the specific field of pervasive healthcare, as well as the general domains of ubiquitous computing and resource allocation.

1.8 Thesis Outline

Chapter 1 provides the motivation for the dissertation, as well as an outline of the various research fields, which are addressed. In addition to that, the contribution and approach of the dissertation are clearly defined.

Chapter 2 outlines a more precise background analysis of recent developments and corresponding technologies in the domain of the dissertation topic.

Chapter 3 provides a state of the art analysis, outlining major achievements in the research domain so far. However, the analysed solutions are lacking maturity in terms of usability and functionality and therefore motivate further development in order to

extend the current technologies and applications.

Chapter 4 proposes an application scenario of where ubiquitous computing can be used to support resource allocation. This consists of a hospital environment, where a highly dynamic workforce (i.e. doctors) are required to be allocated to incoming tasks, which have varying priorities (e.g. emergencies). The design of an appropriate solution is then provided, encompassing the different components of the proposed application.

Chapter 5 describes the implemented solution, which tries to address most of the issues outlined in the preceding chapters. It encompasses the development of a central server application communicating with several clients in order to perform efficient, context-aware task allocation.

Chapter 6 provides an evaluation of the implemented application by analysing the proposed solutions to current issues in the corresponding research fields.

Chapter 7 draws a conclusion from the preceding chapters and outlines a summary of the findings. Areas of future work are then identified with respect to the limitations of the solutions proposed in the dissertation.

Chapter 2

Background

This chapter provides a review of the general domains to which this dissertation is contributing. This involves outlining established and current research trends in resource allocation, ubiquitous computing, context-awareness and pervasive healthcare. In addition to that, current development technologies are analysed, which are relevant to the design and implementation of the proposed activity dispatching application.

2.1 Resource allocation

The domain of resource allocation has produced a multitude of solutions to problems similar to the well-known Traveling Salesman Problem (TSP)[30]. In this problem, a salesman is required to visit a fixed number of cities by minimising the total travel time involved. Variations of the problem also include additional constraints, such as only being allowed to visit each city once and once only or having to return to the original city at the end of the journey.

Mathematical models behind the solutions for this problem have traditionally been concerned with the area of combinatorial optimisation, which is characterised by the following parameters:

- *Resources* should be optimised according to an objective function, which differentiates and ranks solutions according to defined criteria. The objective function might either favour the maximisation or minimisation of the resource

in question, depending on the type of problem space. In the TSP for example, travel time should be minimised, whereas the number of places visited could be maximised in a variation of the problem.

- *Constraints* limit the solution space by excluding certain possibilities and providing strict rules for solutions to be met in order to be regarded as valid.

- *Cost* constitutes the negative factor of a combinatorial optimisation problem, which has to be weighed against both the resources to be allocated and the constraints involved. The total cost of a solution should always be minimised.

According to Smith et al. [38], there are many approaches to finding the solutions of such problems, but they can be categorised into exact techniques, that return the optimal solution, and approximate techniques, that return the best solution found during the search. The need for the latter is due to the fact that for many real-world problems the former may take too long because the solution space is too vast. This is certainly the case when calculations have to be performed on devices with limited processing capabilities, such as for example mobile phones or PDAs. These approximate techniques are often labelled as *general purpose heuristics* or *metaheuristics* [5]. Examples of such techniques are *Neighbourhood Search* [35], *Tabu Search* [29] or *Simulated Annealing* [21]. In addition to that, Blum et al. [5] propose the hybridisation of these established solution techniques in order to combine the different strengths and minimise the weaknesses.

Combinatorial optimisation has historically been very static-based, with the resources, constraints and costs being well-defined and non-dynamic. However, as outlined in chapter 1, recent developments in resource allocation have shifted towards more dynamic scenarios, often consisting of mobile workforce management. This development brings a number of changes to the original concepts of combinatorial optimisation:

- *Resources* are not fixed as they are changing constantly in terms of availability and capability, as well as their respective optimisation preferences and requirements.

- *Constraints* are similarly highly dynamic and need to be checked constantly in order to produce valid solutions at all times.
- *Cost* factors in a dynamic environment are varying over time as well, as specific allocations might become more expensive to perform under changed conditions.

By analysing these modified factors, it has to be noted that no scheduling algorithm is able to predict such future changes of the resources, constraints and costs. The notion of a perfect solution therefore needs to be eradicated from the model, making dynamic resource allocation a non-trivial problem.

2.2 Dynamic Workforce Management

There are various application fields of dynamic resource allocation with respect to human resource management. The scheduling of field technicians constitutes the most dominant type of problem currently being researched. However, there are a multitude of other scenarios where efficient human resource allocation can be beneficial. In any type of business in the service sector, human resources constitute the most valuable assets and their efficient allocation is therefore directly proportional to company profits. In addition to that, non-profit organisations such as for example emergency services could greatly benefit from more responsive techniques in terms of resource scheduling in order to handle emergencies more efficiently.

Due to the outlined importance of efficient human resource scheduling in various types of companies and organisations, a corresponding business sector has emerged and currently experiences enormous growth. Research house Frost and Sullivan estimates @Road [27], which is one of the current market leaders in this field, to be a 1 billion business by the end of 2007. Huge investments in this particular company are therefore made by various types of multinationals, such as Intel, Hitachi and ABS Capital. This has contributed to @Road being the fastest-growing technology company in 2005, outpacing famous competitors such as for example the online market and auction service eBay [16].

2.3 Ubiquitous Computing

The field of computer science has so far experienced two main eras. First of all, *Mainframe computing* initially introduced the concepts of human-computer interaction, as well as the automation of data processing tasks. Secondly, the emergence of the *personal computer* brought this development to the average household, with ordinary people being able to process information on their own machines, as well as to communicate with other users across the globe through the Internet.

A new third phase extends this notion of computing gaining increased penetration into peoples' daily lives, additionally trying to make technology effectively invisible to the user [39][40]. This notion of *ubiquitous computing* can therefore be regarded as the natural evolution of computing systems towards a pervasive involvement in human processes. This development is certainly noticeable when analysing the market of mobile devices, such as mobile phones or PDAs.

CTIA (Cellular Telecommunications and Internet Association) reports that in 2006 there were 233 million wireless subscribers in the US, which equals to a penetration rate of 76 percent. This is an increase of 25 million (6 percent) from just 2005, with figures having been as low as 110/34 million (39/13 percent) in 2000/1995 [9]. An even more dramatic development can be noticed in Europe, with penetration rates for example being as high as 157.3 mobile subscribers for every 100 people in Luxembourg [37].

In addition to that, Gartner reports that in 2Q06 the worldwide PDA market grew 2.7 percent relative to the same period of 2005, to 3.7 million units. This was about 1 percent higher than their forecast of 3.67 million units and the highest second quarter for PDA shipments on record [19]. Furthermore, Apple has recently sold 270,000 copies of its highly anticipated iPhone in its first 30 hours. The research firm iSuppli predicts that 4.5 million such devices will ship during the 2007 calendar year, with figures rising to more than 30 million in 2011 [7]. Especially this particular shift towards more powerful devices enables the increasing development of rich-client applications.

2.3.1 Context-Awareness

A particular aspect of ubiquitous computing is the notion of context-awareness defined in section 1.4. Mobile devices increasingly gain different types of connectivity, from close-range technologies such as Infrared or Bluetooth to wide-area connectivity through WiFi or UMTS. This enables so-called sensor networks to be deployed throughout mobile users' environments in order to capture different types of contexts and therefore enhancing the aforementioned rich-client applications. Traditionally, the only type of context being deployed in live applications has been location with the aid of GPS tracking devices [3]. However, an endless variety of additional contexts exists, usually being very application and scenario-specific. In section 2.4.1, several instances of such specific types of contexts are presented from the area of pervasive healthcare.

2.4 Pervasive Healthcare

Any type of hospital constitutes a constantly changing environment, which needs to be managed efficiently in order to deal with daily emergency situations. Numerous dynamic actors are involved, ranging from administrative staff and technicians to nurses and doctors. In addition to that, constantly changing patients are another factor responsible for making the environment even more unpredictable. Furthermore, the design of mobile hospital applications is complicated even more by the fact that wireless communication technologies commonly used by mobile devices can conflict with medical devices and therefore might not be allowed to be used in certain locations [20]. Hospitals can therefore be regarded as perfectly suitable for these new and dynamic types of technologies making up ubiquitous computing, especially with the promising outlook of facilitating everyday processes in dynamic environments.

Mobile devices are being deployed increasingly with very high success rates [4] and the general field of pervasive healthcare has recently gained tremendous interest, with research centres being created that solely concentrate on this subject, e.g. Centre for Pervasive Healthcare at University of Aarhus [1][2][3][25]. However, real life testing of more advanced solutions has been fairly limited so far, as most research experiments have generally been carried out in laboratory settings [26]. Nevertheless, many projects

have outlined the fact that hospitals generally contain various sources of context, which are overloading human actors with too much information if no appropriate technologies are being deployed.

2.4.1 Context-Awareness in Pervasive Healthcare

Munoz et al. [36] propose four critical contextual elements that a context-aware system would have to consider in supporting a hospital's information management and activity coordination system. These factors have all been confirmed and extended by other research groups specialising in pervasive healthcare.

Location

As already mentioned in 2.3.1, the most explored type of context for most application experiments has traditionally been location. In a hospital, this encompasses both staff (e.g. doctors, nurses, technicians, etc.) and patient location. As GPS device performance is reduced in indoor settings, several alternatives for user-tracking have been proposed.

Munoz et al. [36] have developed a Location-estimation agent, which resides in all users' PDAs and consequently obtains each user's position by triangulating the signal strength from at least three 802.11b access points. Bardram et al. [3] propose an alternative method involving Bluetooth-enabled devices such as phones, PDAs or tablet PCs. In addition to that, they also outline that their technology is extensible to any type of device due to the large availability of Bluetooth USB sticks.

Both groups agree that such tracking techniques are fairly imprecise, but in most experiments they were able to retrieve which room the searched person was in.

Time

In a dynamic hospital environment, time constitutes the most important context, as well as the most complex one. Different tasks to be carried out by the medical staff need quick attention, whereas other activities can only be performed after a

certain time has elapsed. It is therefore of crucial importance to perform precise time-management, making use of doctors, patients and tasks being each associated with a defined timeframe. This type of context therefore needs intelligent and powerful applications, which reason about the different parameters and constraints involved. Examples of such solutions will be discussed in chapter 3.

User role

Another important context occurring in a hospital setting is the notion of user roles. Certain users can only be allocated to a subset of the tasks, therefore restricting the total number of human resources available at task distribution. Again, this context requires an intelligent application backing the overall information system.

Artefact location and state

The final context identified by Munoz et al. [36] is the so-called "Artefact location and state". This encompasses any additional information concerning the different actors involved. Examples of such state measurements in hospital environments have been developed for patient health monitoring. Holzinger et al. [26] propose for example transponders being worn by each patient, containing information about history elements such as previous treatments. In addition to that, these transponders can be used for more dynamic data, such as blood pressure or heart rate values. Furthermore, doctors can be equipped with specific readers in order to be able to quickly retrieve this information.

2.5 Background Conclusion

This chapter has outlined the different challenges in the general domains of resource allocation, ubiquitous computing, context-awareness and pervasive healthcare.

It has been revealed that a lot of research potential exists in each of these areas, therefore providing an ideal base for the creation of sophisticated technologies and applications.

Chapter 3

State of the Art Review

This chapter provides an overview of existing solutions in the domains of resource allocation, context-awareness and pervasive healthcare.

First of all, a review of single-user applications is made, outlining their strengths and shortcomings in terms of dynamic task handling. This section particularly includes the Trails Application, which this dissertation project is extending in order to accomplish the outlined contribution.

In addition to that, multi-user applications are presented, which additionally address the issues of collaboration and dynamic task allocation across multiple users and devices. However, several weaknesses can be identified again, especially in terms of client self-reconfiguration.

The final section then provides a review of current applications in the field of pervasive healthcare, revealing particular issues, which have not been addressed appropriately to date.

3.1 Single-User Applications

The following single-user applications have been focussing mainly on the handling of environmental context, as well as the corresponding changes occurring over time,

requiring intelligent and responsive solution techniques.

3.1.1 Mobile Tourist Guides

A lot of research projects in the field of context-awareness have initially been using mobile tourist guides as an example application scenario. The reason for this is the fact that tourists represent highly mobile entities, being constantly surrounded by various types of context, such as location or attraction opening times.

The GUIDE project [6] is an example of such a mobile tourist guide application being developed for the city of Lancaster. Rather than remaining limited to laboratory settings, this application has been deployed over a period of four weeks by 60 different users. The main conclusions have been very positive, with visitors finding the location-aware navigation and information-retrieval mechanisms useful and reassuring. However, the application specifications and functionalities are fairly limited in terms of context-variety and dynamic task addition.

The amount of tasks within one schedule is limited to 9 activities in order to guarantee acceptable response times. In addition to that, this number cannot be changed dynamically during use and more importantly the only context being considered during schedule creation is user and task location. Furthermore, no notion of collaboration among users is being considered, making the application a pure single-user oriented solution. All users connect to a central server, which is responsible for tracking locations and generating task schedules. Therefore, no reconfiguration can occur during temporary disconnection phases.

P-Tour [32] represents another mobile tourist guide application with very similar specifications and functionalities. Although more complex schedule creation algorithms are used by this application, it still only considers location and time in terms of context-awareness. In addition to that, due to the nature of the genetic algorithms used, the thin-client design is very unlikely to be changed to a richer, self-reconfiguring client because of limited mobile device capabilities.

This again leads to the problem that a client is almost useless in the case of the mobile device getting disconnected from the central server. Furthermore, no notion of collaboration can be identified and the number of tasks is static after a user's initial selection of interests has been performed.

3.1.2 Application Framework for Mobile, Context-Aware Activity Scheduling

The Distributed Systems Group (DSG) at the University of Dublin has developed a framework, which provides generic designs for mobile, context-aware trails-based applications [13][14]. This *Hermes* project has addressed the different areas of context acquisition, modeling and reasoning, as well as collaborative context and trail management. Two example applications extending this framework have been implemented to date, consisting of a campus activity planner for new students at Trinity and a citywide riddle hunt game for mobile users.

Driver [12] has implemented an extension of this framework, emphasising on the challenge of efficient schedule creation on mobile devices. The implemented solution enables schedule reconfiguration to be performed on a mobile device without the need for a powerful central server. It therefore has a significantly improved functionality compared to previous solutions such as the tourist guides explained before, which hugely relied on external decision-making. The framework builds on the notion of a *Trail*, which is a collection of activities. Three different scheduling algorithms can be used for trail reordering, Brute Force, Simulated Annealing or a Genetic Algorithm. Each of those techniques make use of user preferences, which encompass weights set for the different parameters such as proximity, duration or urgency. Activities are compared according to those factors, with clashes (i.e. activities with equal timeframes) being resolved appropriately as well.

Context is handled dynamically, with for example a user's changing location hugely influencing the reconfiguration outcomes. As the solution is designed as an application framework, additional contexts can be added very easily. Driver proposes different example applications for the framework and confirms the context extensibility by

implementing for example a music festival guide application, which successfully adds a stage time change context.

The different activities of a trail are held in a Trail Repository, being loaded at application startup from a local properties file. This provides data persistency on the mobile device, which had not been addressed in previous mobile context-aware applications. The framework therefore outperforms preceding solutions as it does not require any external server application in order to fulfill the task of activity scheduling.

The user interacts with the application via a simple text-based interface, which constantly displays the currently scheduled trail. The activities are ordered by start time and the user has the possibility to interactively indicate when a specific task has been completed. This results in an immediate trail reconfiguration, confirming the dynamic nature of the schedule creation. In order to test context-awareness, a simple location simulator has been created, which periodically updates a user's position. Again, significant changes of this property trigger a trail reconfiguration, as certain activities might become more favourable due to an enhanced proximity score.

However, the number of tasks being stored and handled on the mobile device is static, i.e. no new tasks can be added dynamically to a user's trail after the Trail Repository loading stage has occurred. In addition to that, the framework does not address any notion of multiple users being responsible to perform a set of activities in a collaborative way. As the focus of the application had been isolated trails reconfiguration on mobile devices with limited performance capabilities, no communication component had been included in the framework.

Finally, due to its extensibility to cater for various types of context, the application framework could be used in the field of pervasive healthcare in order to facilitate dynamic schedules for healthcare professionals. An additional server component has been proposed by Cormac et al. [15], which would extend the existing framework to cater for new task aggregation and task dispatching. Exactly this component has been provided by this dissertation project, the details of the design and implementation being presented in chapters 4 and 5.

3.2 Multi-User Applications

The following sections outline the implementation of existing multi-user applications, allowing users to communicate and collaborate in context-aware environments.

3.2.1 Active Campus Explorer

The *ActiveCampus Explorer* [22] developed at UC San Diego addresses some of the shortcomings outlined in the previous section, especially in terms of multi-user interaction. The application supports context-aware activities, such as instant messaging and location-aware maps annotated with dynamic hyperlinked information. Users are hence able to track the location of fellow students, provided that these run a copy of the application on their own mobile devices. A study, which has involved hundreds of users over a whole year of deployment has revealed that students have embraced the application, as only a small minority has changed the initial privacy settings in order not to share their location.

However, there are several shortcomings of this application. First of all, once again only location is being considered as relevant context. In addition to that, users do not collaborate in order to fulfill a set of tasks together, as the application only allows students to track fellow users and consequently communicate with each other.

3.2.2 @Road

A very mature dynamic workforce solution has been developed by *@Road*, which is a fast-growing, California-based technology company (see section 2.2). Their idea comprises a small black box containing chips that receive signals from the Global Positioning System (GPS) being placed inside service trucks. The position of each vehicle is then transmitted back to headquarters via data networks operated by wireless phone carriers like Nextel, Sprint and others. This results in a powerful system for tracking vehicles and people in real time with minimal effort. For businesses with large human resources, such as telecommunications carriers, the consequent cost-savings are very promising.

In addition to this tracking system, *@Road* also offers very mature scheduling products to enhance business productivity even further. The *TaskForce* package, initially developed by a British-based company called *Vidus*, "is an intelligent, automated field service management solution that continually synchronizes the right human and physical resources required to fulfill consumer demand, even among complex variables and unpredictable circumstances" [27]. By implementing this solution, businesses have been able to:

Increase...

- productivity to 30 percent
- field force utilisation to more than 90 percent
- control-to-field employee ratio from 1:8 to 1:50

Reduce...

- annual total cost of operating a mobile field workforce by 15 percent
- number of control sites by 85 percent
- call duration using Intelligent Appointer by 17 percent

The *TaskForce* solution makes use of two scheduling stages, *Batch Scheduling* and *Dynamic Scheduling*. The former is run overnight to construct high quality schedules for the following day using a customised Simulated Annealing algorithm. The latter also makes use of customised local search algorithms, with additionally taking into consideration that solutions have to be found much faster.

A powerful visualisation tool is provided for managers overlooking the field technicians, allowing a constant overview of the created schedules in the form of a map tour representation or corresponding Gantt charts. In addition to that, field technicians can indicate when a task has been finished, which might trigger a reconfiguration in the case of an early/late completion.

Although the solution is very mature and has been deployed by major organisations such as British Telecom, e.on or ntl, it does have several shortcomings in terms of device independance. The application assumes almost constant device connectivity with respect to a central server application. Although this is certainly a valid assumption for the case of field technicians, a disruption-prone environment such as a hospital (or almost any other indoor location) would require mobile devices to be capable of more independent decision-making.

3.3 Pervasive Healthcare

In this section, current developments in pervasive healthcare are presented, revealing that the amount of research activity in this domain has been rapidly growing over the last few years.

3.3.1 Hospitals of the Future

Bardram et al. propose a multitude of solutions in order to create what they call the "Hospital of the Future" [1]. Their vision is a highly interactive hospital, where a multitude of mobile devices are deployed in order to facilitate communication and collaboration among doctors, patients and other hospital actors.

One of their earliest projects has been the development of an *Interactive Hospital Bed*, which comprises an integrated computer and touch sensitive display. Various sensors have been fitted in order to track status events such as for example a patient lying in the bed, a doctor standing beside the bed, as well as various other measurement events using RFID tags.

The *AwareMedia* solution addresses the need for collaborative applications within operating theatres. Using the Java Context-Awareness Framework (JCAF) [2], the application supports the co-ordination, scheduling and social awareness amongst clinicians involved in surgeries. Large interactive displays are hereby used to visualise the different actors and their corresponding status, e.g. "patient arrived", "patient anaesthetised" or "operation started". In addition to that, a message chat allows

communication amongst clinicians, allowing nurses for example to call surgeons to an operation ward. Furthermore, current schedules are displayed, showing both planned and acute surgeries.

Another context-aware application developed by Bardram et al., the so-called *AwarePhone* [25], is running on mobile phones and allows doctors to retrieve context information about their colleagues. Communication with a central server is performed in order to receive the location and activity status of the medical staff deploying such a device. The application also allows a user to view all the scheduled activities of fellow doctors, hence supporting personal decision-making about whether it is currently possible to contact a specific person or not.

However, the application does not perform schedule reconfiguration of any sort. Additionally, doctors are not able to indicate themselves whether they have completed a task or not.

3.3.2 QoS DREAM framework

The *QoS DREAM (Quality of Service for Dynamically Reconfigurable Adaptive Multimedia)* framework [34] developed by Mitchell et al. focuses on high-quality multimedia communications in dynamic environments such as hospitals. Their middleware has four main conceptual components:

- An operating system support layer, which is responsible for resource management for audio and video streaming.
- A Dynamic multimedia streaming platform.
- An event-based programming paradigm.
- A set of integrated APIs for building applications.

A "follow-me" video application has been developed on top of the framework in order to show the functionality possibilities. This application allows a clinician to establish an audio-video conference call with another clinician without being aware of his location.

The clinician being called is notified via his Active Badge, which performs user tracking via a network of infrared sensors deployed around the building. The actual call is then carried out at the nearest multimedia communication station. These stationary devices have been deployed throughout the hospital in order to cater for high-quality conference calls.

The application hence addresses the issue of context-awareness, although again being limited to location as the sole context type. In addition to that, no notion of schedule creation/reconfiguration is addressed by this project, nor does it handle doctors' badges becoming disconnected from the network. Furthermore, the use of stationary terminals clearly restricts the mobility of the users, i.e. the medical staff.

3.3.3 Context-Aware Mobile Communication in Hospitals

Munoz et al. propose a collaborative handheld system, which extends the instant messaging paradigm by adding context-awareness to support the intensive and distributed nature of information management within a hospital setting [36]. Their implementation allows users to specify new tasks to be performed by colleagues. It is possible to explicitly send a message to a specific doctor, as well as to a list of qualified doctors, e.g. to all surgeons. The context types being implemented are location, time and user role, as well as artefact location and state (as explained in section 2.4.1).

Although the system makes use of contextual information in order to retrieve relevant messages about tasks to be performed, it does not make any attempt at scheduling these according to specified parameters. It therefore has to be regarded as a powerful messaging client, rather than a task allocation system. In addition to that, the client application needs to be able to connect to the central messaging server at all times in order to be useful to a doctor.

3.4 State of the Art Conclusion

The technologies, frameworks and applications outlined in this chapter have revealed that a lot of research has been carried out in each of the areas of context-awareness,

mobility and resource allocation.

The first set of applications analysed in this chapter has been lacking any notion of collaboration among users. In addition to that, most of these single-user solutions require constant connection to a central server in order to provide their services. However, the presented *Application Framework for Mobile, Context-Aware Activity Scheduling* is purely focussed on providing independent activity allocation on constrained mobile devices and therefore provides an ideal base application on which this dissertation project can build in order to construct the desired solution.

The second set of applications has been focussing on communication and collaboration among users in context-aware environments, but again the developed client applications always need server connectivity in order to remain useful. In addition to that, most applications have a limited set of contexts being considered, as they often solely handle location changes.

The survey of technologies and applications in the field of pervasive healthcare has confirmed the need for ubiquitous computing in dynamic hospital environments. Current solutions already show a certain degree of maturity, although dynamic resource allocation has not been the main focus for any of the presented solutions.

More adequate activity scheduling applications are hence required, which should collaboratively handle various types of contexts and simultaneously keep the ability to perform independent decision-making.

Chapter 4

Design

This chapter outlines the design stage of the Activity dispatcher application implemented in this dissertation. Firstly the scenario which the application aims to simulate is defined, and the different parameters and constraints are identified. A review of the potential technologies of the scenario is then presented.

The model specifications are formalised by creating an appropriate database design to hold the necessary information regarding the different entities. The next step involves the creation of a realistic application design, which allows a dynamic task allocation to be performed between the different actors of the presented scenario.

4.1 Model

The chosen scenario is a hospital environment, where a set of doctors are required to perform a variety of activities during their shifts. This constitutes a highly dynamic environment, as there exist a multitude of constantly changing factors. The aim of the application is hence to successfully allocate incoming tasks to appropriate doctors despite these constantly changing environment variables.

In order to work out appropriate algorithms for a valid task allocation, a clear high-level model of all the parameters and constraints needs to be established. What is more, certain variables should additionally be optimised by the application during the

scheduling process.

The following subsections will now outline the different entities, which are part of the hospital environment. In addition to that, the various interactions and constraints among these entities will be presented.

4.1.1 Doctors

As previously stated, the allocation of human resources constitutes one of the main challenges in a scenario involving a dynamic workforce. In the case of a hospital, the optimal use of the medical staff is therefore of crucial importance.

Each doctor has a set of variables, which needs to be considered carefully when allocating specific tasks. First of all, as a doctor fulfills activities throughout the day, his/her geographical position changes constantly and therefore this parameter needs to be monitored by the information system. In addition to that, each doctor has a certain set of skills, which allow him/her to perform a specific set of tasks. What is more, each doctor is only available for a specific period of time (i.e. a shift) during which he/she can perform activities.

4.1.2 Patients

The patients being treated in a hospital constitute a highly dynamic variable as well. A change in geographical position might not occur as frequently as for example for a doctor, but nevertheless this parameter needs to be monitored and considered constantly during task allocation. In addition to that, as modern hospitals are equipped with a variety of status-monitoring tools, a significant source of context-generation appears in this part of the hospital environment. If a specific status drops to a critical value, the system should quickly discover this information and handle the emergency in an appropriate and responsive manner. What is more, each patient holds a history of previous treatments, which establishes doctor-patient relationships within the hospital environment.

4.1.3 Tasks

A task constitutes an activity, which needs to be performed by a doctor in order to provide treatment for a specific patient. First of all, such a task requires a defined set of skills, meaning that only a subset of all the doctors can engage in this activity. In addition to that, each task needs to be performed at a specific location, involving the doctor to traverse a certain geographical space in order to get to the patient/activity. What is more, a task has a defined earliest start time, a latest finish time, as well as an estimated duration. All of those variables may have a certain leeway, which is specified at the time when the task is entered to the system. Other important attributes of a task are its priority level as well as whether the activity is mandatory or not.

4.1.4 Constraints

Two different types of constraints can be identified: hard constraints and soft constraints. Hard constraints need to be fulfilled in order for a schedule to be valid, whereas soft constraints can be loosened if a schedule is otherwise not possible.
The opening and closing times of tasks are hard constraints, which need to be respected by the task allocator at all times. Each activity needs to be scheduled within the given timeframe (including leeway), otherwise the activity becomes impossible to be performed. The correct matching of a doctor's skills for a specific task constitutes another hard constraint as it is absolutely vital that an allocated doctor is able to perform an activity. A soft constraint can be identified in respecting a doctor's shift start and end time, as it should always be allowed to create a (moderate) amount of overtime in order to perform high priority jobs.

4.1.5 Optimisation

The primary role of a dynamic workforce scheduler is undoubtedly the allocation of all jobs to appropriate human resources. However, it should always be attempted to make an optimal use of these resources by using optimisation techniques.

The variables, which should be maximised or minimised in the presented scenario are as follows (with respect to the above mentioned constraints):

1. An activity should always be performed as early as possible.
2. Higher priority activities should be scheduled with greater urgency.
3. A doctor should have a minimal amount of idle time and travel as less as possible.

4.2 Technologies involved

As already indicated in sections 2.3 and 2.4, a diversity of technologies has penetrated into all types of organisations and company settings. Hospitals are no exception in this development, and hence there are a number of technologies, which will play a role in the given scenario. The following sections will now outline the different technologies used by the actors involved in the hospital environment.

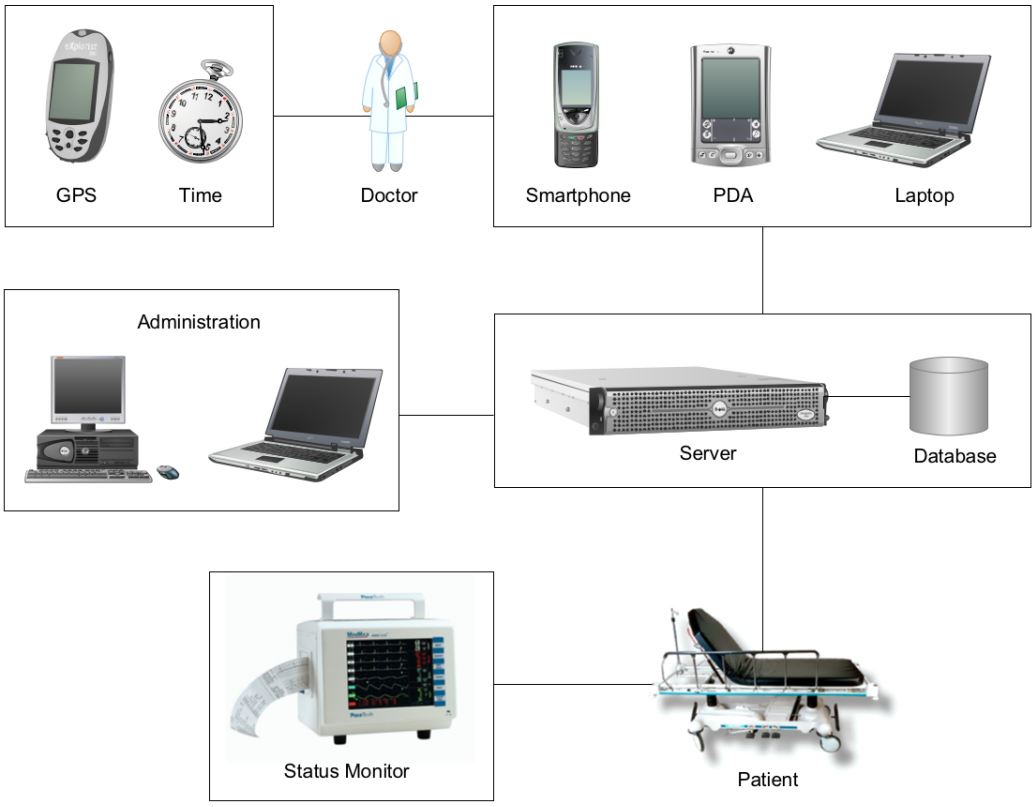

Figure 4.1: Technologies involved

4.2.1 Doctor

Doctors often possess a handheld device such as a PDA or smart-phone, which is capable of communicating with other devices. In addition to that, these devices will be able to run powerful applications as well as receiving context from external sources. These might include information about location using a GPS receiver, but also more context-specific data such as nearby doctors' schedules. As indicated, with the possibility to communicate using a variety of technologies, such as WLAN, Bluetooth or GSM, these small devices are capable of linking up with a central server, which is responsible for task allocation and which also houses all information about the environment. This could include information about a patient's status, the current schedule of the hospital staff or specific medical records.

4.2.2 Administration

Each organisation or company usually houses an administrative division, which is responsible for inputting and handling new data and hence creating an initial link between the organisation and its customers. The same applies for a hospital, where the administrative staff is responsible for welcoming new patients and entering new tasks. In this area, desktop and laptop computers will be the main input stations, also needing an interface for communicating with the server in order to store/manipulate the hospital records.

4.2.3 Patient

Again, as indicated in sections 2.3 and 2.4, research advances are being made in order to get sensors and mobility to new kinds of areas, which traditionally would have been much more static and less adaptive. Not only will doctors have their own mobile devices, but also patients (e.g. small transponders), and given a complete network connectivity, an exchange of information between the server back-end and the patients becomes also possible. In addition to that, patients are connected to a variety of status monitors, which, coupled with the aforementioned connectivity, results in the system having a good overview of that particular patient's health and vital signs status.

4.3 Database Design

After finalising the high-level model of the different parameters and constraints, a corresponding database model can be defined in order to be able to permanently store all necessary data needed by the application. This model reflects the different actors

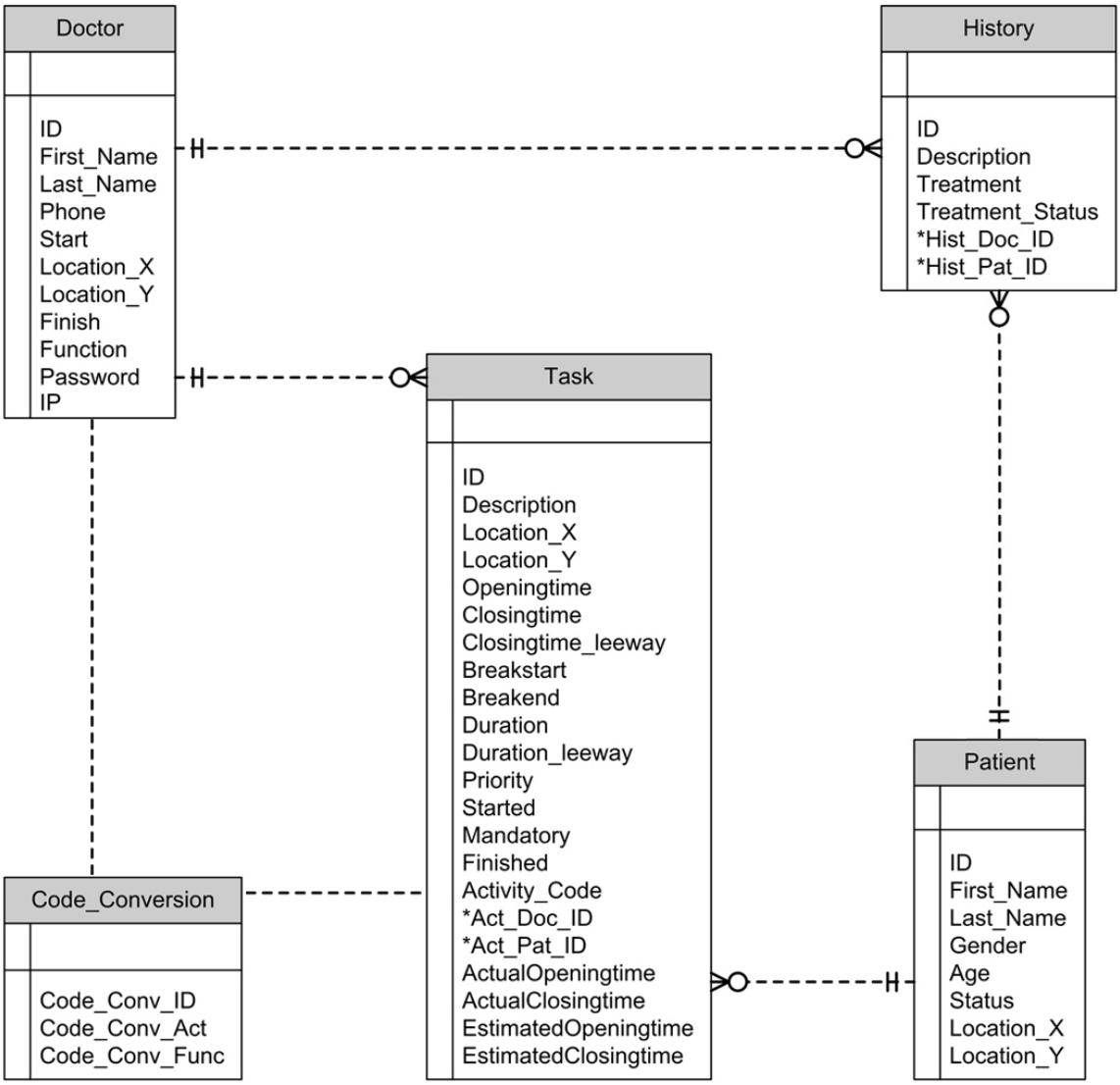

Figure 4.2: Database Design

of the hospital environment, i.e. the doctors and patients, as well as the additional information needed for a realistic modelling of the constraints.

Each doctor entry is uniquely identified by an ID and holds the specific geographical and shift-related information. In addition to that, a password allows a doctor to access confidential information when using an interface to the database (e.g. a web interface). In order to reflect a skill set, each doctor entry holds a function code, which defines his/her role in the hospital environment. As each doctor will have his own mobile device, the server will need to know the corresponding IP address in order to send messages about newly allocated tasks. For this purpose, an IP entry is held in the database for every doctor's device.

Similarly, each patient entry contains all the relevant personal data associated with a hospital patient. A status field reflects a patient's vital signs (e.g. heart rate or blood pressure), which might change quickly in the case of an emergency. What is more, a patient may be associated with a number of entries in the history table, which reflects past treatments. This relationship establishes a doctor-patient relationship, as each history item is also linked to the doctor who performed the treatment.

Incoming tasks are stored in the database as well and all the necessary information of this activity (e.g. opening time or location) is hence easily retrievable by the application. Each task has an activity code, which defines the set of skills needed by a doctor who wishes to perform the activity. If the task has been scheduled to a specific doctor by the application, the estimated start time can be retrieved by the chosen doctor, again by using an interface such as a web application. Once a doctor has started or finished a task, he/she can set the respective values as well, indicating to the application that this activity no longer needs to be considered. In the case of an application being finished, a new history item should then be created by the application. It has to be noted that the task table has been modeled to be easily compatible with the activity data structure used in the existing trails generation application [12].

Finally, a code conversion table links doctors' functions to activity codes in order for the application to quickly check if a specific doctor has the necessary skills to perform an activity or not.

4.4 Application Design

The task allocation of the proposed scenario constitutes a system which needs to be distributed over several entities. First of all, each doctor will need an own client application running on a personal, portable device (e.g. PDA), in order to view/modify his own shift schedule. In addition to that, a central server application is needed for the actual task aggregation, as well as the allocation of the tasks to the different mobile users.

The proposed application design distinguishes hence between one central server and several client machines. The following subsections will now outline the different parts of the system, as well as their respective roles and responsibilities.

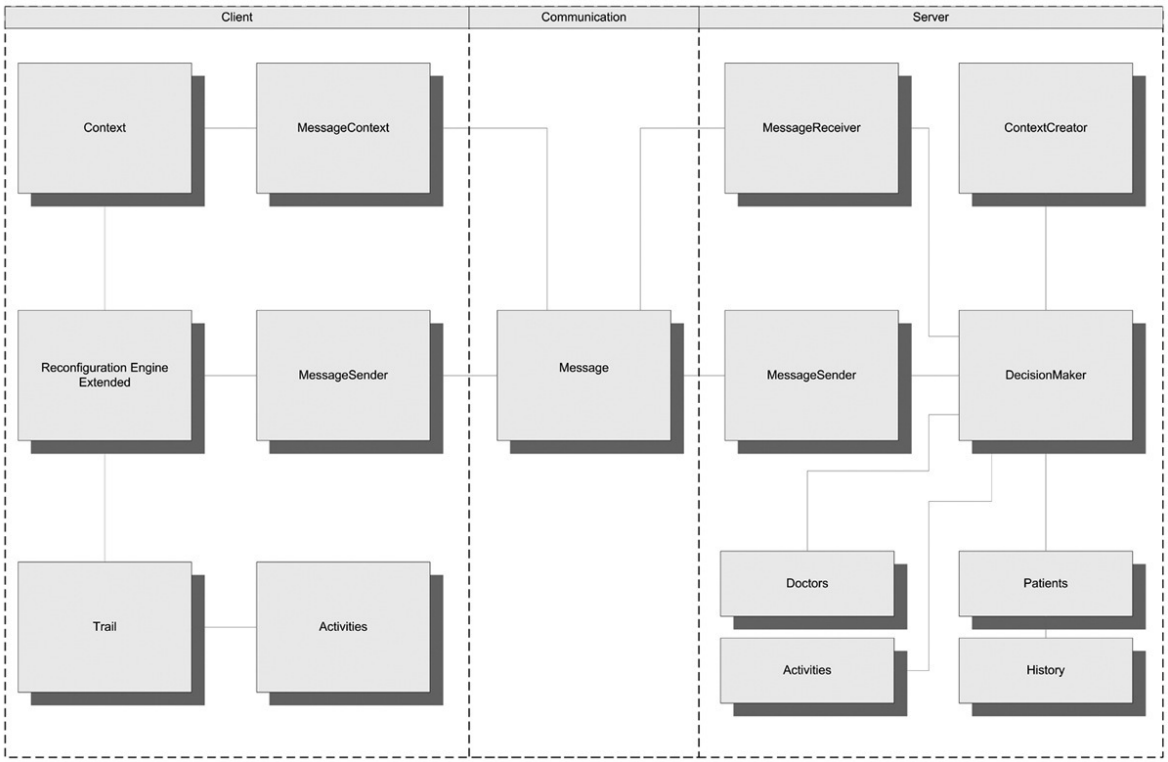

Figure 4.3: Application Design

4.4.1 Client

The core of the client application design is based on the existing trails generation application explained in section 3.1.2. However, a number of significant extensions are necessary in order to allow this single-user oriented application to interact with other entities.

Message Context

In order to keep the existing model of context creators (subjects) and context receivers (observers), a new type of such a context is incorporated into the client application design. The new context creator is a message receiver, which constantly listens on the network for any messages that are relevant for the specific doctor. Upon detection of such a notification by the server, the reconfiguration engine of the client will then be responsible to handle the message contents accordingly. There are different types of messages, which the client will need to decode (see 4.4.3). However, the concept of these messages should be extensible in order to cater for new types of messages, as well as a different message format.

Client sender

In addition to receiving a new message using the concept of context, the client also needs to be able to create messages itself in order to inform the server about significant context or trail changes. These messages should be of the same format as the messages being created by the server, in order to cater for proper code extensibility and consistency. Again, the reconfiguration engine is responsible for the logic behind the messaging, including the decision to create a new message as an appropriate reaction to an event.

Dynamic Activity Addition/Removal

The most important message being received from the server is the notification of a new activity to be performed by the client. In the existing trail generation application, dynamic activity addition was designed into the system, however it was not functional. It is hence of vital importance to build this feature of dynamically handling the addition/removal of an activity to/from the trail.

Trail Reconfiguration

As can be seen in Figure 4.3, the client application still contains the reconfiguration intelligence of the original trails generation application. This is due to the fact that clients might become disconnected from the server and hence need to decide themselves about whether to reconfigure the current trail based on the surrounding context. Not only is the client responsible for this so-called reconfiguration point identification, but also the actual reordering and feasibility checks of the trail activities. Therefore, the reconfiguration engine always needs to have a local copy of all activities which have been assigned to it, as well as access to all the relevant context information. These responsibilities will all be carried out in the original modules of the trails generation application, although some modules will need to be extended slightly for the given scenario.

4.4.2 Server

After having described the responsibilities and specifications of the client application, this section now outlines the design of the server part of the system. The main responsibilities are task aggregation and task allocation and in order to achieve this, a variety of modules need to be implemented.

Database Connection

First of all, as described in section 4.3, the application stores and retrieves its data from a database back-end. The responsibility of establishing a link to this database is solely on the server side, as clients should not have any direct access to this resource at any time for security reasons. Having this link on the server side provides increased security to the data, especially as the database will not only contain very confidential data, but also vital patient status information.

For efficiency reasons, the server application does not constantly make database connections during its operation, but it has local copies of the necessary data. It will hence have a good knowledge of all the patients, doctors and tasks at all times, and having access to this information using data objects, the decision making process

can be speeded up significantly. Not only will the database link give access to static data such as general information (e.g. name, gender, etc.), but also to dynamic data such as scheduled start time of activities. This should lead to a near perfect knowledge of the actual staff situation and the projected future movements of each doctor, as well as indicating if and when a doctor would be free to be allocated to a new task.

Patient Status Context

The server will also be responsible of receiving context from the different patients. As shown by Figure 4.2, patients are assumed to be connected to devices, which are monitoring status, as well as communicating this information to the server back end. It will therefore be the server's responsibility to acquire this context and handle it in an appropriate way. For testing purposes, a context creator component needs to be built in order to simulate this part of the environment.

Message Receiver

As indicated previously, clients will be creating messages intended to be processed by the server application and it is hence necessary to have a message receiver component on the server side. This should be very similar to the message receiver on the client side, and both the patient status context receiver and the message receiver are modeled using the mentioned subject/observer context concept.

Decision Maker

At the heart of the server application sits the decision maker component, which is responsible for handling new incoming messages, monitoring patient status context and reasoning about this information. The database component allows the decision maker to have a good indication of the environment variables, as this knowledge is of crucial importance in order to make task allocation decisions. This component is also responsible for creating new messages for clients, which have been given a specific task. It is therefore important to make sure that the task is feasible by the client and also to have a knowledge of the client's address in the network. However, if the decision maker's allocation will not be possible to be performed by the client, this fact will be spotted by the client application, resulting in a corresponding reply message and a

consequent reallocation of this activity by the decision maker. This concept shows that the server's knowledge of the current situation does not have to be perfectly accurate, which is very reasonable considering its real-world application scenario, where a lot of factors are changing very rapidly (including client network connectivity).

The system is designed to cater for emergencies, such as very alarming patient status values, by partly ignoring doctors' currently scheduled trails in such a case. An activity is immediately sent to the closest skilled doctor in terms of location. Again, if this results in previously scheduled activities becoming impossible for the chosen doctor, the client application logic will spot this fact and inform the server (resulting in a reallocation).

4.4.3 Messaging

As shown by Figure 4.2, both client and server applications agree on a universal message component in order to guarantee interoperability. The different types of messages being exchanged are:

1. New Activity Message: Sent from server to client to indicate that a certain task should be performed by the chosen doctor.

2. Position Change Message: Sent from a client application to the server to indicate a change in position

3. Activity Started Message: Sent from client to server to indicate that a specific activity has been started

4. Activity Finished Message: Sent from client to server to indicate that a specific activity has been completed

5. Activity Impossible Message: Sent from client to server to indicate that a specific activity is not possible to be performed by the chosen doctor

6. Trail Reconfiguration Message: Sent from client to server to indicate that a trail reconfiguration has occurred

7. New Task Message: Entered by either a doctor or an administrative user to indicate to the server that there is a new task to be performed

However, the message component is designed to be extensible to cater for new types of messages, allowing new types to be added as they might become necessary for a modified scenario.

4.5 Web Interface

The application is designed to be be run discretely on doctors' mobile devices. However, they will need to proactively communicate with the server occasionally, checking their schedules and also indicating manual changes. These manual interventions are: starting an activity, completing an activity and adding a new task. The full application design therefore contains an additional Web Application component.

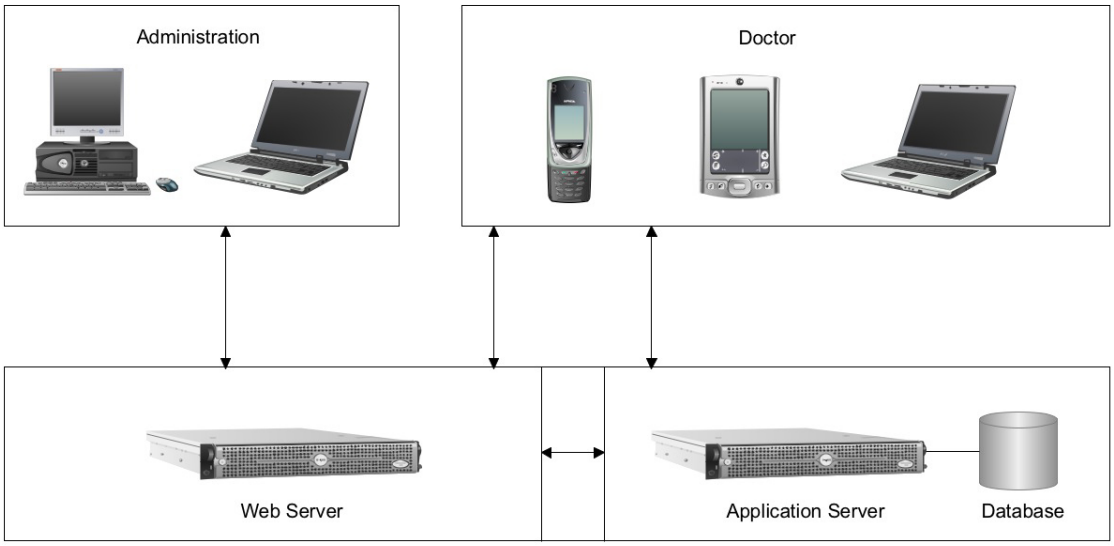

Figure 4.4: Application Design including Web Server

Choosing a web interface for this purpose allows universal access to the resources, hence also being available to an administrative division of an organisation. The Web-server can be deployed on the same physical machine as the application server, or on a separate machine in the network. Either way, both servers will be communicating

with each other in order to guarantee data consistency.

There are two main components of the Web Application, the set of web pages and the web application logic.

4.5.1 Website

As indicated, two types of potential users can be identified: doctors and administrative staff. The latter group has access to the task addition page, as well as the page providing a general overview of all doctors' schedules. However, it is not possible to view activity details such as patients' details, as this constitutes highly confidential data, which should only be accessible for doctors.

Both groups use a common index page, however in order to get to their personalised pages, doctors need to login using their id and password. These personalised pages give the doctors access to their own particular schedule of the day, as well as the possibility to look up the details of the scheduled activities. This also includes doctors being able to look up the patient details associated with an activity.

Furthermore, the website also provides a doctor with the facility to indicate to the server when a particular task has been started or completed, helping the decision making process significantly by updating the server's knowledge about a doctor's current state and position.

Doctors might occasionally want to enter new tasks as well, as a result of an unsuccessful treatment or simply as a follow-up. The site design allows this as the task addition page can be accessed without needing any specific login rights. In addition to that, during the addition of a task, a doctor can specify if it should be scheduled to a particular doctor (for example specifically to himself/herself). For this reason a doctor can, just as the administrative staff, look up the schedules of all doctors in order to get a good overview of the general current work load.

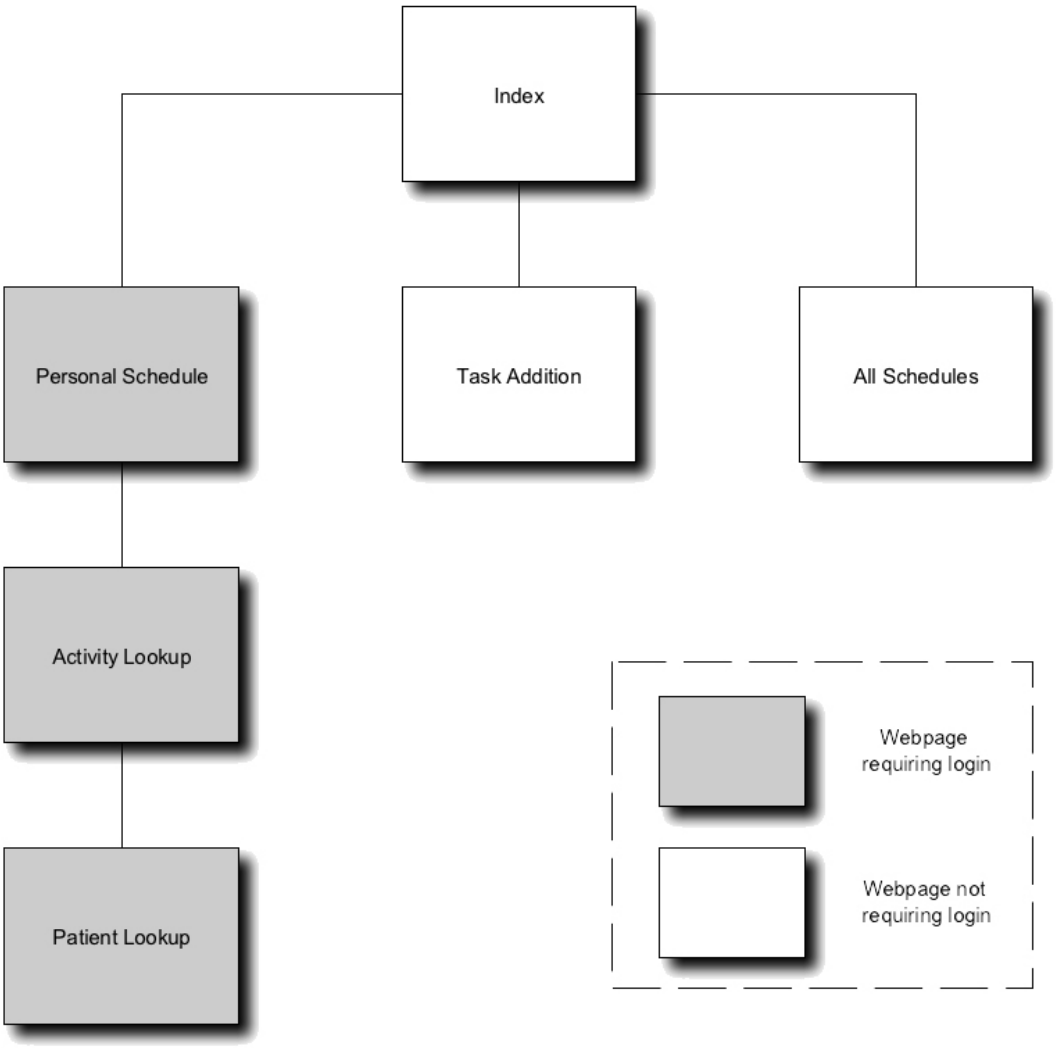

Figure 4.5: Site Map

4.5.2 Web Application Logic

Behind the user accessible web pages sits the web application logic, which provides a link to the database and also handles any changes triggered by the user. These changes, such as a task being started or completed, are reviewed by the web application engine and then communicated to the application server.

Again, the same message format as explained earlier is used, which makes the web application server an equal node in the network, acting as a regular client communicating with the application server. This explains why the web application server could sit on the same machine as the application server, but also on an entirely remote machine in the network.

4.6 Design Conclusion

This chapter has presented the design stage of the activity dispatcher application implemented in this dissertation.

First of all, the hospital scenario used for this project has been described, including the corresponding parameters and constraints. In addition to that, a database design has been created, which is responsible for holding all the relevant data needed by the application logic.

Finally, the application design has revealed the distributed nature of the proposed solution, as it comprises distinct client, server and web application components.

Chapter 5

Implementation

This chapter outlines the different components implemented following the design stage described in chapter 4. The main sections are the implementation of the database back-end, the client and server applications, as well as the web application component.

5.1 Database

The database design outlined in section 4.3 has been implemented on a PostgreSQL database [23] using Structured Query Language (SQL). Such a relational database allows easy data aggregation and retrieval, as well as the concurrent access needed for the proposed application. The SQL statements used to create and test the database have been designed conform to standard guidelines [8].

The table creation stage is relatively straightforward once a precise design such as the one proposed in Figure 4.3 has been formalised. A small set of test data has been used to test the correct functioning of all the tables in order to see if any parameters would be missing or needing small modifications. This same test data has been used throughout the application development stage.

Foreign keys have been used to establish links between tables, such as for example doctors and patients being linked to history items.

```
CREATE TABLE History
(
    Hist_ID                 serial,
    Hist_Description        varchar(100) NOT NULL,
    Hist_Treatment          varchar(100) NOT NULL,
    Hist_Treatment_Status   boolean NOT NULL,
    Hist_Doc_ID             int4,
    Hist_Pat_ID             int4,

    CONSTRAINT History_PK   PRIMARY KEY (Hist_ID),
    CONSTRAINT Doctor_FK    FOREIGN KEY (Hist_Doc_ID)   REFERENCES Doctor(Doctor_ID),
    CONSTRAINT Patient_FK   FOREIGN KEY (Hist_Pat_ID)   REFERENCES Patient(Patient_ID)
);
```

Figure 5.1: SQL Example: History Table Creation

5.2 Application

As previously stated, the application proposed in this dissertation builds on a Trails Generation Application [12], which performs context-aware trail reconfiguration for a single user on a mobile device (a trail being a collection of activities). Using the same development language as the original application would hence facilitate the application extension enormously, in this case being the Java programming language. In addition to that, Java represents a very suitable choice not only for the original application, but also for the additions proposed in this dissertation, as it is both platform-independent and portable to mobile devices [10].

The development has been performed using the Eclipse IDE [18], using Java Micro Edition for the client application and Java Standard Edition for the server application. There are certain limitations of the original client-only application, which have been addressed in order to cater for a more dynamic activity handling, as well as being able to communicate with a server back-end. However, the majority of the existing code has been reused, with additions only being extensions rather than modifications.

The following sections will now describe the two parts of the system, the client application and the server application.

5.2.1 Client Application

The original client application explained in section 3.1.2 encompasses very sophisticated trail reconfiguration features, which fully remain in the extended version of this dissertation project. However, as outlined in section 4.4.1, three main additions have been made to the original application: a message receiver context, a message sender and dynamic task additions.

Message Context Receiver

One very interesting aspect of the Trails Generation Application is the concept of a client's main component being a context-receiver. This notion of context is fully extensible and already implemented extensions of the Context base class were a Location Simulator and a Reconfiguration Timer, which triggers periodic reconfigurations. For this dissertation, a Message Receiver Context is added, which is responsible for receiving messages from the server. A Java thread is constantly listening on the network and upon receiving a relevant message, the context listener, i.e. the client's reconfiguration engine, is informed. The message is then disassembled and handled appropriately. The exact implementation of the messaging functionality is described in section 5.3.

Message Sender

Several events on the client side require a notification to be sent to the server, such as for example a task becoming impossible. In this case, a new message is created using static message creation and sending methods, again being explained in section 5.3.

Dynamic Task Addition

The original application is loading the set of tasks to be performed by the client from a Trail Repository, which makes use of a Java properties file. This list of activities is fixed from the first loading of the application and hence is not designed to be changing dynamically. An extension has therefore been built in order to dynamically write to the properties file, as well as changing all active data structures to be consistent with the modified set of activities.

5.2.2 Server Application

The server application has been built from ground up as the original application had no notion of a server back-end. The main responsibilities consist of performing information acquisition and task allocation.

Data Objects

First of all, the server needs to acquire knowledge about the different actors of the hospital, i.e. the doctors and patients. It does so by querying the database when starting up the application for the first time. All the data is stored in specifically created patient, doctor and task objects. The id attribute fields for the objects are the same as the ones used in the database and therefore each object can be uniquely identified from inside the application. Each time a change occurs in the database, the local objects are modified as well, in order to retain consistency with each other.

A patient object contains all the corresponding details held in the database, including a list of history objects for previous treatments. The status attribute is the most dynamic field as it changes upon receiving patient status context notifications (see 5.2.2)

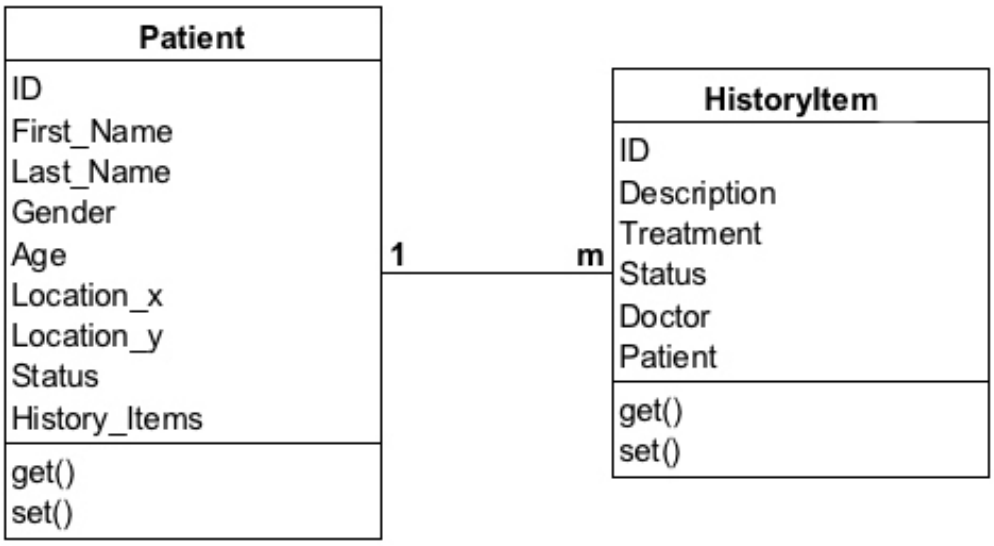

Figure 5.2: Patient Data Object

Similarly, doctor objects contain data fields such as for example name or role, as well as the current schedule based on the information acquired by the database. This schedule is represented by a TimeTable object, which is specific for each doctor.

This TimeTable class provides a variety of methods used by the decision maker in order to check if a doctor has enough time to perform a task. First of all, a TimeTable object contains an ordered list of so-called TimeTableItem objects, which each combine a scheduled task together with its scheduled start and end time. The ordering of this list is performed using the standard Java *Collections.sort()* method, being possible due to TimeTableItems being comparable (based on their scheduled start time). This list is also very dynamic, as tasks can be easily added or removed in the case of the decision maker scheduling a new task to a doctor, or if a message has been received that a task has become impossible on the client side. Before any task is added, however, the decision maker calls the *possibilityCheck(Task)* first, in order to check if there is enough time in the current schedule for this task to be performed, respecting the following times: doctor shift start time, doctor shift finish time, earliest task start time, latest task finish time. Remaining within those times also includes travel times from/to the activities preceding/following the checked task's considered start and end times.

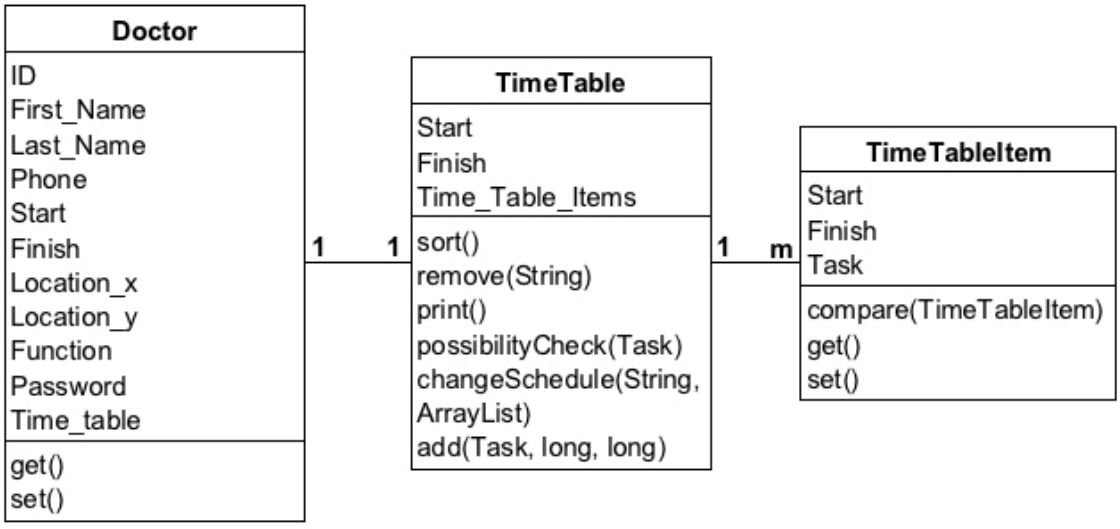

Figure 5.3: Doctor Data Object

The *changeSchedule(String,ArrayList)* method is called whenever the server receives a trail reconfiguration message from the client. This is the case when a reconfiguration has occurred on the client side, resulting in a change of schedule. As the server side needs to be as closely synchronised as possible with the client's application, i.e. the client's perception of the current schedule, the client therefore sends this newly calculated task order to the server. As indicated, the *changeSchedule(String,ArrayList)* method is then called, with the String being the received message and the ArrayList containing the tasks and their scheduled start and end times. The TimeTable object on which this method is called is then modified in order to reflect the changes occurred on the client side. By doing so, both client and server have the same knowledge about the doctor's timetable and therefore the server can make accurate decisions and predictions about the feasibility of new incoming tasks and emergencies.

Patient Status Context

As described in section 4.4.2, the server is responsible for monitoring a patient's status and handling the different changes which might occur. In order to test this feature, a Patient Status Simulator has been created, which is separate from the server application. Again, this feauture is extending the basic Context Generator class, with the server application being an Observer (i.e. Context Receiver). The simulator periodically changes a patient's status by retrieving a random patient record from the database and changing the status value. It then notifies all observers, in this case being the server application, and waits for a certain time (using the Java Thread sleep functionality) before again changing another patient's status. In the implemented solution, a status is only represented by a single number, but it could be modeled in a more realistic way in a future extension.

Decision Making Component

The DecisionMaker class combines all components described so far in order to perform correct task allocation. First of all, at application startup, all data objects, a patient status receiver, as well as a message receiver are instantiated. By loading the different data objects from the database, the component acquires complete knowledge of the different actors and it is hence also aware of doctors' current schedules. Upon receiving

a new message from the message receiver component (described in 5.3), the decision maker determines the type of the message and then processes the request straight away or passes on the message contents to the appropriate method.

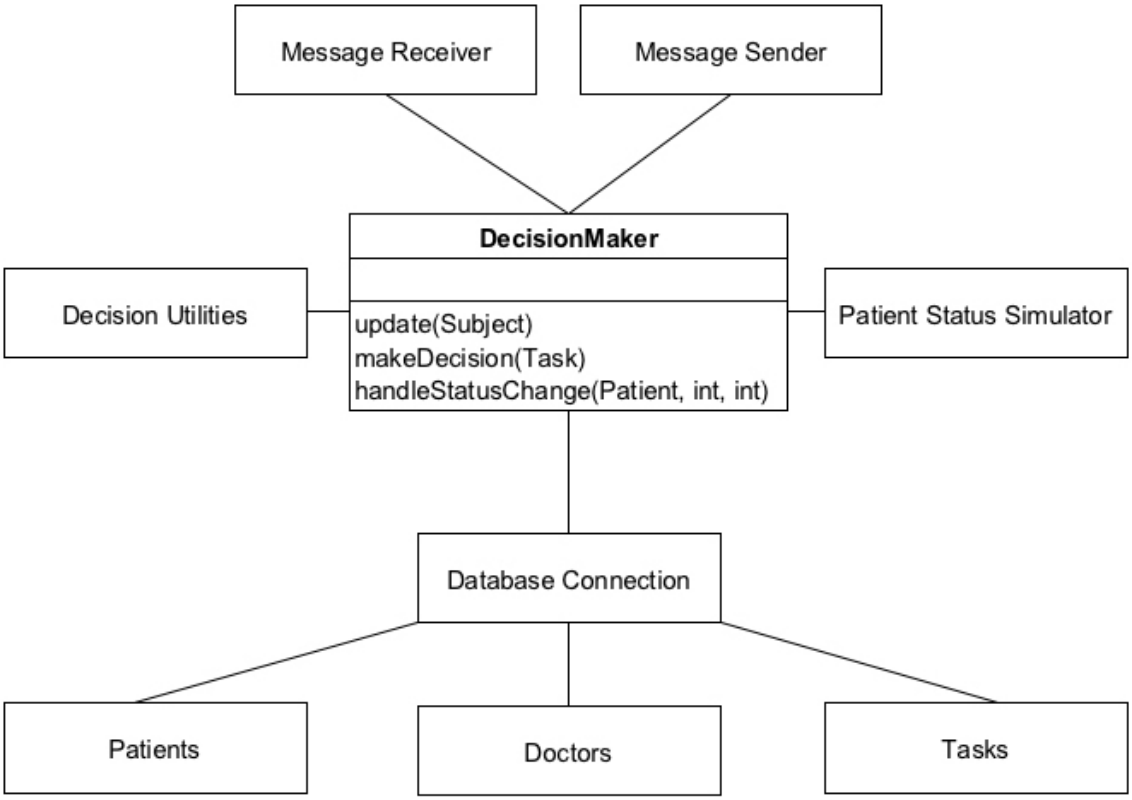

Figure 5.4: Decision Maker Component

In the case of a message indicating that a specific task has been started or completed by a doctor, the decision maker updates the database fields, as well as the data objects held in memory. A task completion message also triggers the creation of a new history item, which contains the information about the doctor, patient and treatment involved. Upon receiving a location change message from a doctor, again a simple update of both the database and data objects is performed. As indicated, the client is responsible for performing trail reconfigurations and it communicates these changes to the server. In this case, the decision maker calls the explained *changeSchedule(String,ArrayList)* method on the relevant doctor data object and also updates the database with the changed start and finish estimates.

If the server receives a message indicating a new task, the *makeDecision(Task)* method is called for this task in order to find a suitable doctor. Another type of message arriving from the client is the notification of a task having become impossible, and in this case the relevant task is not only removed from a doctor's TimeTable object, but also the relevant database links are dropped. Again, the *makeDecision(Task)* is being called, as the task now needs to be reallocated to a suitable doctor.

A patient status context change is handled by the decision maker as well, as the latter registers for receiving such a context change notification at the start-up of the application. The server analyses the change that has occurred by comparing the initial and the changed status values. The implemented application only handles changes which have a modified status of the highest value (5). In this case, a new emergency task is created for the relevant patient and the *makeDecision(Task)* method is called straight away. The opening time for the task is set to the current time and the priority is set to the highest value, which is important at both the actual decision making stage, as well as the reconfiguration stage on the receiving doctor's client application. This status change feature could easily be extended to cater for a more sophisticated service, also taking specific actions at less dramatic status changes.

The *makeDecision(Task)* method differentiates between urgent tasks (emergencies) and less important, more general tasks. In both cases, two/three main stages determine which doctor gets allocated to the task. However, whereas the first stage is common to both groups, the second one takes into account the differing needs for quick allocation and the third stage applies for non-urgent tasks only. A specific DecisionUtil class has been created to offer a more modularised decision making approach, as a future extension of the application might wish to add more stages.

During the first stage, all doctors are checked if their function matches the required skills needed for the activity. As the database contains a table that matches skills to functions, a simple call through the Database Connection component allows the retrieval of a list of suitable doctors. From this point on in the *makeDecision(Task)* method, all doctors not having the required skills are no longer considered for the task allocation.

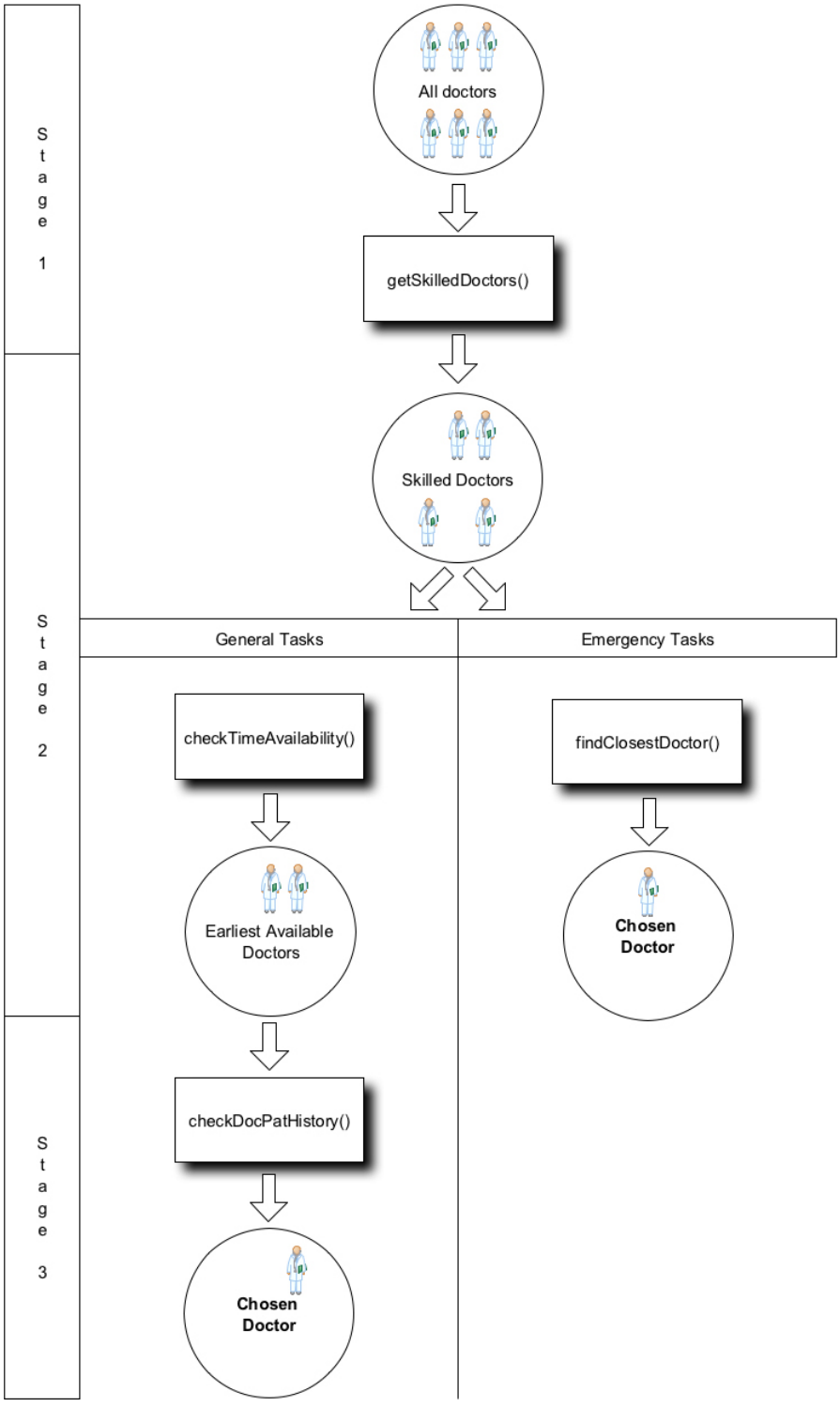

Figure 5.5: Decision Making Algorithm

For urgent tasks, the second stage involves checking the current positions for the remaining set of doctors, i.e. all the doctors who are skilled for the task. The task is then sent to the closest doctor. The decision for ignoring the current schedule of the doctors is based on the fact that an emergency has to be dealt with straight away. In addition to that, the client application receiving this task will schedule the job immediately, as it has highest priority. Again, the result of such a dramatic change of a doctor's schedule might have as a consequence that certain activities become impossible, even though they were previously scheduled to be performed later in the trail. In this case, the impossible activities, which should be less important, will be sent back to the server, resulting in a reallocation.

The second stage of the *makeDecision(Task)* method for less important tasks involves checking the skilled doctors' current schedules and finding suitable empty time slots where the task could be fitted in. For each doctor, the time available between each scheduled pair of tasks is checked, including needed travel time. Upon a successful finding of such a slot, the algorithm proceeds to the next doctor, remembering the earliest start time possible found in the previous doctors' schedule. At the end of the algorithm, the doctor with the earliest free time slot gets allocated to perform the task.

Should two doctors have their earliest free time slot at exactly the same time, the algorithm proceeds to an additional third stage, which compares the two chosen doctors in terms of doctor-patient relationship with the task's patient. This is performed by making a call to the database to check which doctor has a greater number of history items linked to the specific patient. At the end of this round, either one doctor has been found to be more suitable or the doctor with the lowest id of the two will be chosen.

After having found a suitable doctor, the server then passes the task into a message and sends it to the relevant client machine. In addition to that, all the relevant data objects, as well as the database records are then updated in order to reflect the change. However, it is likely that the client application will schedule the task at a slightly different time than the server application, resulting in a trail reconfiguration message. As explained earlier, the server will decompose this message and store the modified

times in order to remain synchronised with the client application.

5.3 Messaging

The most important addition to the original Trails Generation Application has been the described addition of a server component, which allocates tasks to the different clients. In order for the components to be able to communicate with each other, both applications need to agree on a common communication process. This section explains the composition of the communication messages, as well as the actual communication protocol used for transferring these messages.

5.3.1 Message Composition

As outlined in 4.4.3, there are 7 distinct message types being exchanged between the client and server applications. Each of those subtypes extends the Message superclass, which contains all the relevant data attributes in order to be processed at the receiving end. These attributes are as follows:

- Message Type: An integer value indicating the type of message (e.g. Position Change Message, New Activity Message, etc.).
- Message Creation Time: The time stamp of when the message was created.
- Sender ID: Each sender has a unique identifier, which corresponds to the ID value held in the database for the doctors. In order to distinguish itself from the clients, the server has an ID of 0.
- Sender IP: Each sender has an IP address, which is contained in the message for the receiver to update the corresponding ID-IP pair (for reply messages).
- Receiver ID: See Sender ID.
- Receiver IP: Used by the communication component (see 5.3.2).
- Message Body: For each of the 7 message types this attribute changes significantly. It contains all the relevant data needed by the receiver to process the particular type of message.

In both the client and server applications, the listed attributes are defined by constructing a message object. However, in order to send a message over the network this object needs to be transformed into a formatted String. The implemented solution does so by using a Comma Separated Value (CSV) format. Hence, the different attributes are each clearly separated by a comma, which allows an easy rebuilding of a message object using standard Java String manipulation methods on the receiver side.

The body contents contain several sub-attributes as well, which are each clearly separated in the Message Body String by a '/' character when a Message object is constructed. These different sub-attributes differ for each message type:

1. NewActivityMessage: As this message indicates to a client that a new activity has been scheduled to its trail, all the relevant information about this activity needs to be included in the message body. This includes information about the location, the earliest start time, the latest finish time, etc.

2. PositionChangeMessage: This message's body only contains the changed location values for the doctor. The server additionally retrieves the doctor's id from the Sender ID value contained in the message.

3. ActivityStartedMessage: Similarly, the body of an ActivityStartedMessage only contains the ID of the task, which has been started. As this ID is a global identifier (equal to the value held in the database), the server has no problem in recognising exactly which task has been started.

4. ActivityFinishedMessage: The body of this message contains the ID of the completed task, as well as the doctor's indicated treatment performed on the patient.

5. ActivityImpossibleMessage: Again, for this type, only the ID of the impossible task is needed in the message body.

6. TrailReconfiguration Message: This message indicates to the server that a reconfiguration has occurred on the client machine. The body therefore contains the current order of tasks, together with the corresponding start and end times.

7. NewTaskMessage: This message, being received at the server from either a client or administrative user contains all the relevant information about this activity, similar to a NewActivityMessage message explained above.

The choice for using CSV messages has enabled an easy assembling and disassembling of message Strings. However, this component could easily be replaced with a different formatting functionality (such as for example XML encoding and decoding). Provided that client and server agree on a common standard, the formatting of the messages is fairly irrelevant to the application, as long as the composition and decomposition can be performed in an efficient and extensible manner.

5.3.2 Communication

After having formed Message objects and corresponding Message Strings, the application needs to send the message contents to the relevant receiver using an agreed protocol. On the server side, a database check is made before each creation of such a message in order to determine the current IP address for a specific doctor. Both client and server communicate on an agreed port via UDP socket connections and as the receiver's IP address is contained in the message object, a static call can be made to a Sender component. The *Sender.send(Message)* method is responsible for dividing up the message contents into Datagram Packets and sending these across the network.

On the receiver side, a Java Thread is constantly listening on the agreed port for incoming UDP messages. This Message Receiver component has been modeled on both the client and server side as a context creator. Upon receiving a message, a notification can therefore be sent straight away to any listening context observer, which is the decision maker component for the server application and the reconfiguration engine component for the client application.

The choice for using UDP socket connections has been made primarily to enable an easy and fast communication component development. However, the communication design could be extended or exchanged very easily, allowing more sophisticated and reliable techniques to be employed (see section 6.3).

5.4 Web Application

As outlined in section 4.5, a Web Application component acts as the user interface for both clients and administrative users. The application is running on a local Apache Tomcat Server [17], with the pages having been developed using the Java Server Pages (JSP) and JavaBeans technologies. This allows the design of dynamic pages, with the possibility of handling user input in an efficient and secure way. In addition to that, using a Java-based technology allows complete usage of the powerful Java API, as well as the seemless communication with other Java applications [24].

The front-end is composed of 7 JSP pages, which allow the different users to look up current schedules, as well as adding tasks. As the pages are viewed by users with varying devices in terms of display capabilities (e.g. small PDA screens), a very simple and concise design has been used.

When a user navigates to the Index page of the application, he/she can chose between adding a task, looking up all the current schedules or logging in using their doctor id and password.

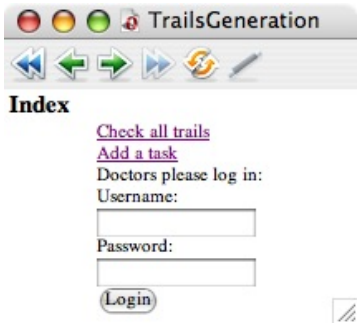

Figure 5.6: Index Page in Opera Small Screen View

5.4.1 Adding a task

As indicated, doctors and administrative users can add new tasks and this page is therefore designed to be accessible by the different types of browsers in which the users might be running the application.

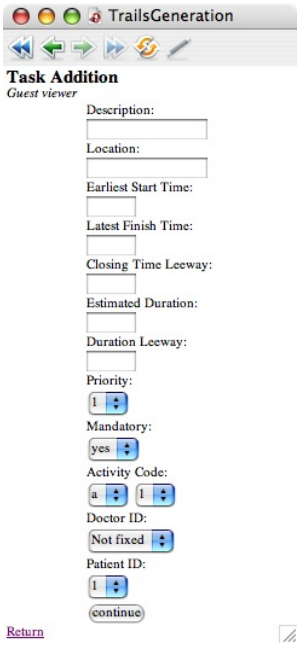

Figure 5.7: Task Addition Page in Opera Small Screen View

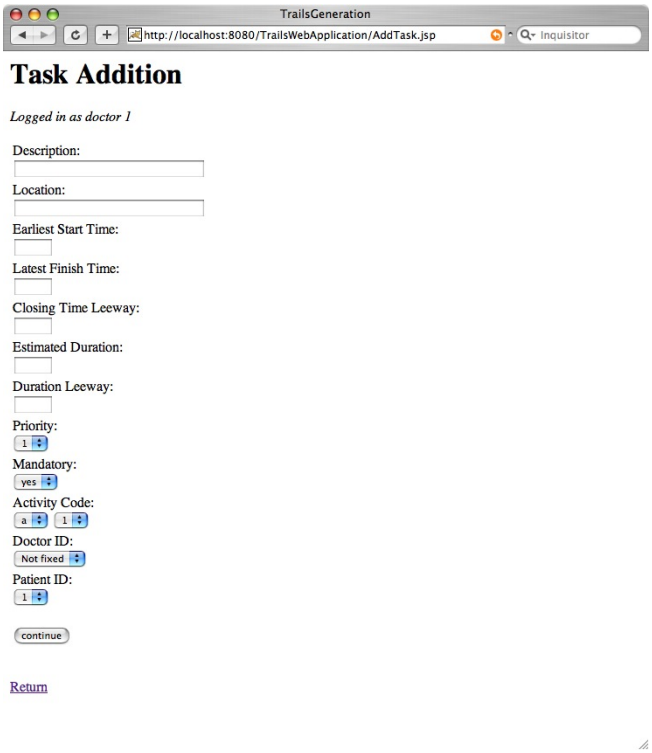

Figure 5.8: Task Addition Page in Safari

5.4.2 Overview page

Again, doctors and administrative users can both have a look at the general schedule overview. This consists of a list of all doctors and their allocated tasks.

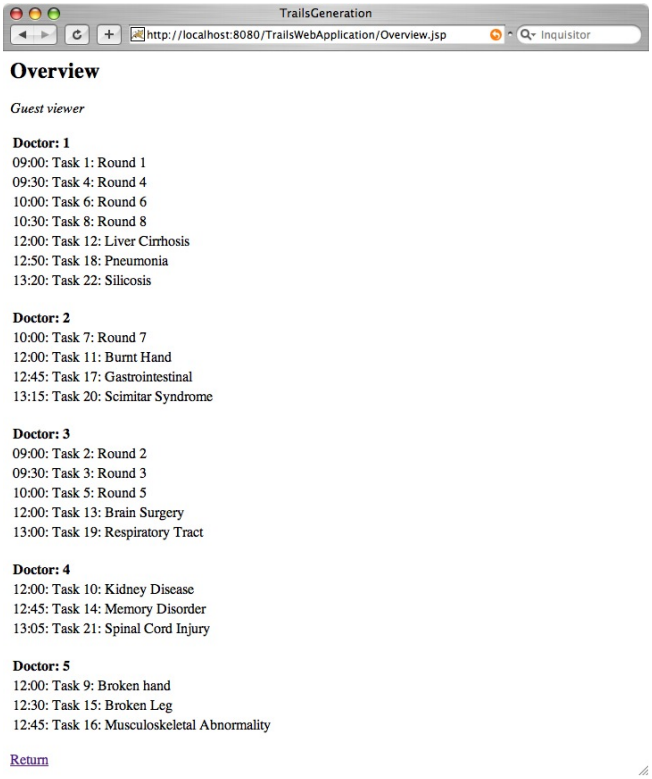

Figure 5.9: Overview Page in Safari

5.4.3 Doctor Area

The login facility allows a doctor to enter a more personalised area of the website. Here, it is possible to not only view the scheduled tasks of your own trail, but also the details of each task (including the corresponding patient details). In addition to that, a doctor can indicate when a task has been started/finished, hence interacting actively with the application.

The login functionality makes use of the Java *HttpSession* functionality, which allows a doctor to remain logged in for a prolonged period of time until he/she wishes to logout.

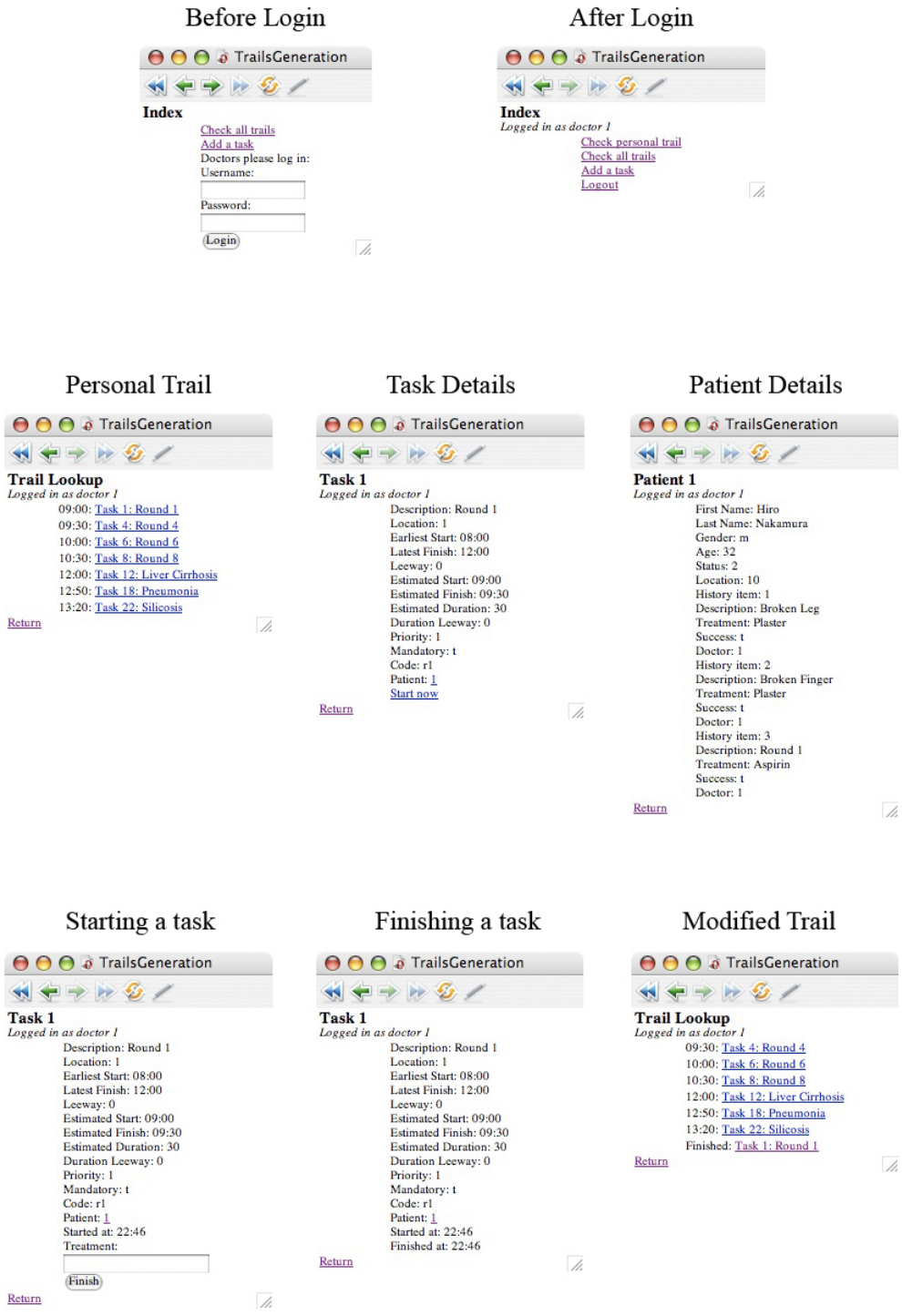

Figure 5.10: Doctor Area Functionality

5.4.4 Web Application Logic

Behind the JSP pages resides the Web Application Logic, which is responsible for providing a link to the database, as well as communicating with the Application Server.

Database Interaction

When a user navigates to a page, which displays information that is stored in the database, the Database Connection component is called from the JSP page. This component makes use of the Java Database Connectivity (JDBC) in order to retrieve the queried data. Similarly, if a new task is added by a user or gets started/completed by a doctor, this information gets stored to the database straight away using the same component. However, in these latter cases, an additional message is sent to the application server in order to perform a notification of the changes made to the data.

Messaging

The Messaging component of the Web Application uses the same agreed processes and standards for creating messages as the client and server applications. The message objects, as well as the message sender classes are therefore identical to the ones used on the client and server applications. The sender ID being included in the task started/finished messages is the doctor ID of the logged-in user who performed the action. The Web Application does not require a message receiver, as its sole responsibility is to notify the application server about changes being performed by users.

5.5 Implementation conclusion

This chapter has outlined the different components implemented following the design proposed in chapter 4.

First of all, a database back-end has been presented, which holds all the relevant data required by the activity dispatcher application.

In addition to that, client and server applications have been described, which comprise the application logic of the implemented solution. This includes components for data acquisition, as well as task allocation and information communication.

Finally, the outlined web application component acts as a clear interface to the users of the system, allowing an active interaction with the application.

Chapter 6

Evaluation

After having detailed the design and implementation of the activity dispatcher application, this chapter delivers a thorough evaluation in order to assess the contribution of the dissertation project. First of all, a more technical evaluation is performed in terms of decision latency and correctness. The consequent sections then perform a higher-level discussion by evaluating to what extend the application addresses the shortcomings of existing solutions outlined in chapter 3.

6.1 Decision Latency and Correctness

In order to test the functionality of the application, a set of test data has been created, encompassing 22 tasks, 5 doctors, 15 patients and 10 history items. Several test runs have been performed, each providing very positive results.

When allocating the 22 tasks to all 5 doctors with regular intervals between each insertion, the decision-making process takes an average time of 110 milliseconds before a message is sent to the client[1]. A similar figure of 101 milliseconds can be measured when the tasks are allocated with only 3 available doctors. This slightly lower value can be traced back to the linear nature of the task allocation algorithm, which checks through each doctor's schedule in order to find the most suitable candidate.

[1]Test machine: Apple Macbook Pro, 2.33 GHz Intel Core 2 Duo, 2GB RAM

When inserting the tasks in reverse order, very similar values can be measured again (103/115 milliseconds), therefore confirming the almost instant decision-making process.

In terms of allocation correctness, a manual check of the different parameters reveals that each of the 22 tasks get allocated to a suitable doctor in terms of function-skill matching. In addition to that, the allocated times meet the shift and task timeframes, therefore respecting all the constraints outlined in chapter 4.

As described in chapter 5, emergencies are allocated slightly quicker because doctors' current schedules are bypassed with the application only considering skill-matching and current location values. The average allocation duration values for emergencies lie around 50 milliseconds, hence revealing an even more responsive decision-making process.

However, this difference is hardly noticeable for such a small set of test data. It would therefore be highly recommendable to perform additional, more exhaustive tests in order to evaluate the algorithms under heavy load. This could potentially reveal the need for more sophisticated non-linear algorithms. However, for the purposes of this dissertation project the implemented solution provides a more than adequate proof of concept for context-aware resource allocation.

6.2 Mobility and Context-Awareness

The state of the art analysis in chapter 3 has shown a lack of mobility and context diversity in many of the current solutions. Several designs include stationary terminal machines, which require users to travel a certain distance in order to be able to perform specific tasks. In addition to that, most context-aware applications have only addressed the notion of user location, hence neglecting the many additional context sources available in dynamic environments.

First of all, as outlined in chapter 4 the implemented activity dispatching application comprises both a server and a client application. Users are able to run the latter on

their mobile devices, due to the application being conform to the resource-efficient Java Micro Edition specifications. It can therefore be run on a multitude of ultra-portable devices, such as PDAs or smart-phones, but also on more powerful devices such as laptop or desktop computers. Hence, universal mobility has been addressed very successfully by the application.

Secondly, the solution builds on the notion of context extensibility of the original Trails Generation Application and therefore caters for an unlimited number of contexts to be added as required. Examples of such extensions are the implemented patient status context and the message receiver component. The application therefore improves greatly on current solutions and confirms the notion of context being ubiquitous in dynamic environments.

6.3 Communication

One of the main additions to the original Trails Generation Application has been the ability to communicate with a central server. As outlined in chapter 5 the various messages are sent using UDP socket connections. Although this method provides the fastest possible delivery to the destination, it is very prone to message loss. A more suitable implementation could alternatively use TCP connections, consequently establishing reliable communication links between client and server. As the communication component has been built in an easily extensible/replaceable manner, this change would require minimal programming effort.

However, as described in section 2.4, a hospital environment often experiences wireless interference, as well as frequent problems in terms of unavailable connectivity, e.g. some devices are not allowed in specific locations such as operating wards. These cases are also certainly not limited to the hospital scenario as they represent typical challenges of highly dynamic environments. More adequate solutions therefore need to make use of additional delay and disruption-tolerant protocols.

During the time a device is disconnected from the network, the server could alternatively bundle outgoing messages and deliver this package upon device

reconnection. Depending on the urgency of the messages involved, as well as the duration of the disruption, the server could also again decide to remove specific task allocation messages from the message-package in order to reallocate the activity to another available doctor. Again, as the application has been built with easy extensibility in mind, such a solution could be added without having to change large amounts of the code-base.

6.4 Trail Reconfiguration Independence

During the application design stage, one of the most important decisions has been whether to embed the main trail reconfiguration functionality on the different mobile devices or on the central server application. However, as outlined in chapter 3, one of the main advantages of the original Trails Generation Application over other existing solutions has been its independence from external devices.

By keeping this feature, a user is therefore still able to reconfigure his/her trail in the case of becoming disconnected from the central server. Consequently, the application combines both the single-user and multi-user applications presented in chapter 3 into one integrated solution. This makes the application unique in terms of mobility, as well as adaptability in dynamic environments.

6.5 Accessibility

The original Trails Generation Application provided the user with a simplistic text-based command-line interface. However, the application extension described in this dissertation makes use of powerful web application technologies in order to significantly improve on the accessibility and user-friendliness of this previous solution.

The web pages have been tested on a number of different web browsers (Safari, Opera and Firefox), as well as various operating systems (Mac OS X, Windows XP and Ubuntu Linux). This proves the universal accessibility of the interface, especially considering that numerous generic browser extensions exist in order to help users with

disabilities (e.g. screen readers supporting visually impaired users).

In addition to that, the web interface has been tested using the Opera small screen view, which simulates the limited display capabilities of a mobile device. As the application performs perfectly in this view, it is very safe to assume that any mobile device with a reduced form factor such as a PDA or smart-phone can handle the application without any usability concerns.

6.6 Reusability

The proposed application has been built on top of a framework, which has specifically been designed to be reused and extended. In addition to that, the implemented communication modules (i.e. message, sender and receiver components) have been designed in order to be easily enhanced by more sophisticated extensions.

However, the database components, as well as the corresponding data structures and algorithms have been specifically tailored to the described hospital scenario and therefore could not be reused easily for a modified environment. Although the different parameters and constraints for schedule creation are therefore directly linked to the scenario, the main concepts can nevertheless be reused for similar application fields.

6.7 Security

When designing and implementing computer systems, one often neglects the importance of considering an appropriate security model. In addition to that, it is crucial to develop a model of the possible threats and countermeasures as early as possible in order to build an inherently secure system.

As this project is tailored towards a healthcare environment, most data needs to remain confidential because it is related to personal information. In addition to that, communication messages are sent over a public network infrastructure, hence consisting of the second area requiring security measurements.

First of all, as described in chapter 5 the web application is initially accessible to a variety of users, i.e. administrative staff, as well as medical doctors. However, in order to provide the outlined data confidentiality the application forces users to enter their username and password before they can access private information such as patient details or history records. It is therefore not possible for unauthorised users to access any information without possessing such login details. In the current solution, it is only possible for unauthorised users to have a look at the general schedule overview, as well as to add a new task.

However, a simple extension of the application could provide a complete resource restriction by forcing every user to login, including administrative staff. This enhancement requires minimal programming effort and has only been omitted in order to enable easier testing. This part of security handling can therefore be regarded as adequate for the given scenario.

The second threat in terms of potential security attacks lies at the low-level communication between client and server applications. In order to cause any serious disruption, a malicious user would need to know the message format, as well as the used protocols and port numbers. The system is therefore relatively safe, although more sophisticated solutions would be needed in the case of such a solution being deployed in a real hospital environment.

6.8 Real-World Applicability

The chosen hospital environment scenario has provided a good example application for the desired type of solution. However, in order to evaluate its real-world usability, an actual case study involving real doctors would have to be carried out over a prolonged period of time.

However, due to the shortcomings outlined in this chapter, especially in terms of communication disruption and security, further development would be required initially before such a real-world study could be performed.

6.9 Evaluation Conclusion

This chapter has provided an evaluation of the implemented solution with respect to various aspects and difficulties in the fields of mobility, context-awareness and collaboration. It has been shown that the application addresses several issues, which have not entirely been solved by an integrated solution before.

As shown by Figure 6.1, the implemented Activity Dispatcher for Trails-based Applications combines the independent decision-making capabilities of the original Trails Generation Application with communication and resource allocation features proposed in other context-aware solutions (see chapter 3). In addition to that, multiple contexts are handled, with the possibility to extend this number even further to cater for additional context sources. The dissertation project has hence been successful in creating a highly dynamic context-aware resource allocation solution.

Feature / *Application*	Device Mobility	Context-Awareness	Multiple Contexts	User Collaboration/ Communication	Universal Accessibility	Reconfiguration Independence
Single-User Applications						
Mobile Tourist Guides	✓	✓				
Trails Generation Application	✓	✓	✓			✓
Multi-User Applications						
ActiveCampus Explorer	✓	✓		✓		
atRoad TaskForce	✓	✓	✓	✓		
Pervasive Healthcare						
Interactive Hospital Bed		✓	✓	✓		
AwareMedia		✓	✓	✓		
AwarePhone	✓	✓	✓	✓		
QoS DREAM		✓		✓		
Context-Aware Communication	✓	✓	✓	✓		
Activity Dispatcher For Collaborative Trails-based Applications	✓	✓	✓	✓	✓	✓

Figure 6.1: Comparison of Application Features

Chapter 7

Conclusion

This chapter outlines the various conclusions that can be drawn from the dissertation project by taking into consideration the findings from all the previous chapters.

In addition to that, several areas of future work have arisen during the project, which require further research in the fields of context-awareness, mobility, collaboration and security. A quick overview of these problems is presented, together with potential solution attempts.

7.1 Code Reuse

As outlined in chapter 4, the implemented Activity Dispatching Application has been built on top of the Trails Generation Application framework described in chapter 3. Due to the high extensibility of this solution, a code reuse of almost 100 percent has been achieved.

Any additions to the framework have been performed by extending existing classes or by adding new components. The application therefore adheres to good programming guidelines, as these generally recommend modules to be "open for extension but closed for modification" [33].

7.2 Resource Allocation

Over the years, the field of resource allocation has become a well established research domain, with numerous publications being concerned with the optimisation of various types of scheduling problems. However, it has been shown that the challenge of dynamic resource allocation has been far less popular and that only recently this field has gained more interest due to the potential commercial benefits.

Current solutions address the need for efficient human resource scheduling, although the developments in ubiquitous computing have not yet fully penetrated into the research area. This dissertation project has therefore contributed to this field by combining the problem of dynamic scheduling with the possibilities provided by pervasive technology solutions.

7.3 Context-Awareness

The presented background research has revealed that the area of context-awareness has been addressed increasingly by a multitude of solutions in various application fields. Various types of sensor networks have been proposed, allowing the notion of ubiquitous computing to gain increased popularity.

However, many research projects have been confined to remain high-level application designs, rather than delivering real-life deployment solutions. In addition to that, the notion of location changes has been the dominant context being addressed, hence ignoring the numerous additional types of context being present in dynamic environments.

The implemented solution has addressed this latter issue by including a multitude of contexts and also allowing additional types to be added as required. However, only a limited amount of testing has been performed to evaluate its actual usability, although a real-world scenario has been used throughout the project (i.e. a hospital environment).

7.4 Mobility

User mobility has been shown to create a continuous development towards increasingly pervasive technologies. The statistical facts provided in chapter 2 indicate that modern society embraces new solutions enhancing their mobility in everyday processes.

It is therefore not surprising that more and more applications are designed to be deployed on these devices, which are inherently characterised by limited display and data processing capabilities. However, it has also been revealed that current solutions often require powerful server back-ends in order for the mobile applications to be of any use.

This dissertation project has also proposed such a solution, which can be deployed on various types of mobile devices, such as PDAs or smart-phones. In addition to that, a server back-end supports the notion of distributed task management among a multitude of mobile users. However, the implemented client application remains functional on users' devices even in the case of becoming disconnected from the central server. It therefore improves greatly on most of the solutions presented in chapter 3.

7.5 Pervasive Healthcare

The domain of pervasive healthcare has emerged following the recent developments in the outlined fields of ubiquitous computing and context-awareness. Various types of sensor networks have been proposed, building on the fact that hospitals constitute highly dynamic environments. In addition to that, several collaborative solutions have been implemented, which facilitate daily processes of healthcare professionals.

Mobile solutions have been proposed as well, mainly providing easier communication among colleagues across different locations. Context-awareness has been included in most of these solutions, therefore allowing the modelling of very realistic applications. However, the challenging task of efficient resource allocation has been widely neglected, although the medical staff constitutes the main asset of any hospital setting.

This dissertation has proposed a solution to address this problem of efficiently scheduling incoming activities, including high priority tasks, which have to be performed by a dynamic workforce (i.e. the medical staff). The implemented application handles various types of contexts with respect to the hospital environment and allocates activities according to the constraints imposed by the different parameters. The project has therefore contributed to the domain by proposing possible solution techniques in order to facilitate resource allocation in a highly dynamic hospital scenario.

7.6 Future Work

As already indicated, several areas of future work have been identified, which have not been addressed mainly due to time constraints.

7.6.1 Communication Security

The security evaluation in section 6.7 has revealed that the solution lacks maturity in several security aspects, especially in terms of communication security between the client and server applications.

Several techniques, such as for example private/public key technologies [28] could be used in order to enhance the application, making it therefore more appealing for a real-world deployment. This is certainly the case for hospital environments, where data integrity and confidentiality have to be ensured at all times.

7.6.2 Server-side Trail Reconfiguration

The proposed solution allocates activities according to several constraints, such as skill-function matching or time availability checking. Once a suitable doctor has been found by the application, a new task notification is sent to the corresponding doctor's mobile device. It is then the client application's responsibility to continuously assess the feasibility of the added activity, as well as the general trail order.

Due to the schedules changing constantly over time, a certain unbalance might occur, requiring specific doctors to perform significantly more tasks than some of their colleagues. This development could be tracked by the server application, allowing a task reallocation to be performed in extreme cases of certain doctors getting overloaded compared to others.

In addition to that, as mobile devices are limited in their data processing capabilities, the server application could generally support clients' trail reconfigurations. This resource-expensive task would then only be required to be performed on the mobile device in the case of communication disruption.

7.6.3 Communication Disruption

As outlined in section 2.4, a hospital environment often experiences wireless interference, as well as limited connectivity due to wireless devices being disallowed in specific areas. It is therefore necessary to address this issue of a high connection disruption rate. An example extension has been proposed in section 6.3, which would enhance the application in order to provide a more adequate handling of such communication difficulties.

Bibliography

[1] Jakob Bardram. Hospitals of the future – ubiquitous computing support for medical work in hospitals. In *UbiHealth – the 2nd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications*, 2003.

[2] Jakob Bardram. The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications. In *Proceedings of the 3rd International Conference on Pervasive Computing*, May 2005.

[3] Jakob Bardram, Thomas Hansen, Preben Mogensen, and Mads Soegaard. Experiences from real-world deployment of context-aware technologies in a hospital environment. In *UbiComp: Ubiquitous Computing*, 2006.

[4] Blackberry. Baylor healthcare case study. http://www.blackberry.com/.

[5] Christian Blum. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308, 2003.

[6] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a context-aware tourist guide: the guide project. *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, 2000.

[7] CNETNews.com. Study: iphone tops july smart-phone sales. http://news.com.com/.

[8] Thomas Connolly and Carolyn Begg. *Database Systems*. Addison Wesley, 2004.

[9] CTIA. Wireless quick facts. http://www.ctia.org/.

[10] H. M. Deitel and P. J. Deitel. *Java: How to program.* Prentice Hall, 2005.

[11] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing Journal*, 2001.

[12] Cormac Driver. *An Application Framework for Mobile, Context-Aware Trails.* PhD thesis, University of Dublin, Trinity College, April 2007.

[13] Cormac Driver and Siobhan Clarke. Hermes: A software framework for mobile, context-aware trails. In *Workshop on Computer Support for Human Tasks and Activities at Pervasive, Vienna, Austria*, 2004.

[14] Cormac Driver and Siobhan Clarke. Hermes: Generic designs for mobile, context-aware trails-based applications. In *Workshop on Context Awareness at MobiSys, Boston*, 2004.

[15] Cormac Driver, Eamonn Linehan, Mike Spence, Shiu Lun Tsang, Laura Chan, and Siobhan Clarke. Facilitating dynamic schedules for healthcare professionals. In *Pervasive Health, Innsbruck, Austria*, November 2006.

[16] Forbes.com. The fastest-growing tech companies. http://www.forbes.com/technology/2005/02/11/05fastechland.html/.

[17] The Apache Software Foundation. Apache tomcat. http://tomcat.apache.org/.

[18] The Eclipse Foundation. Eclipse - an open development platform. http://www.eclipse.org/.

[19] Gartner. Pda market reaches record level in 2q06. http://www.gartner.com/.

[20] V. Gehlot and E.B. Sloane. Ensuring patient safety in wireless medical device networks. *IEEE Computer*, 39(4), 2006.

[21] Fred Glover. Tabu search. *ORSA Journal on Computing*, pages 190–206, 1989.

[22] W.G. Griswold, P. Shanahan, S.W. Brown, R. Boyer, M.Ratto, R.B. Shapiro, and T.M. Truong. Activecampus: experiments in community-oriented ubiquitous computing. *IEEE Computer*, 37(10), 2004.

[23] PostgreSQL Global Development Group. Postgresql: The world's most advanced open source database. http://www.postgresql.org/.

[24] Marty Hall. *Core Servlets and JavaServer Pages*. Prentice Hall, 2000.

[25] Thomas Hansen. Interaction with multiple devices in hospitals. In *Workshop: The Spaces in-between: Seamful vs. Seamless Interactions, UbiComp Conference, Tokyo, Japan*, 2005.

[26] Andreas Holzinger, Klaus Schwaberger, and Matthias Weitlaner. Ubiquitous computing for hospital applications, rfid-applications to enable research in real-life environments. In *Computer Software and Applications Conference*, 2005.

[27] @Road Inc. @road: Mobile resource management. http://www.atroad.com/.

[28] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, 2002.

[29] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimizing by simulated annealing. *Science, Number 4598*, 4598:671–680, 1983.

[30] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley Sons, 1985.

[31] Seng Loke. *Context-Aware Pervasive Systems*. Auerbach Publications, 2007.

[32] A. Maruyama, N. Shibata, Y. Murata, Keiichi Yasumoto, and M. Ito. P-tour: A personal navigation system for tourism. *Proceedings of the 11th World Congress on Intelligent Transport Systems*, 2004.

[33] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1988.

[34] S. Mitchell, M. Spiteri, J. Bates, and G. Coulouris. Context-aware multimedia computing in the intelligent hospital, 2000.

[35] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers in Ops. Res.*, 24:1097–1100, 1997.

[36] M.A. Munoz, M. Rodriguez, J. Favela, A.I. Martinez-Garcia, and V.M. Gonzalez. Context-aware mobile communication in hospitals. *IEEE Computer*, 36(9), 2003.

[37] OECD. Oecd communications outlook 2007. http://213.253.134.43/oecd/pdfs/browseit/9307021E.PDF/.

[38] V. J. Rayward-Smith and G.D. Smith. A feasability report on a mobile workforce scheduling management. Technical report, SYS Consulting Ltd, 2004.

[39] Mark Weiser. The computer for the 21st century. *Scientific American*, 1991.

[40] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 1993.