DYNAMAP, A MOBILE AND CONTEXT-AWARE PEDESTRIAN NAVIGATION APPLICATION

By
Eric Gervais

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

September 2005

© Copyright
2005, Eric Gervais

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-494-10117-2

Our file Notre référence

ISBN: 0-494-10117-2

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

We are living in an ever-changing computing world. The post-PC era as predicted by Mark Weiser seems to have run its course, thanks to a number of technological advances. Wireless communication is becoming more and more omnipresent. People can use their cell phones in digital mode in the most remote places, while one can check his/her emails in cafés with Wi-Fi access points. Cell phones come with embedded GPS, and portable music players can carry gigabytes of data. This has led to intense research activity in the fields of mobile ubiquitous computing, context-awareness, location-based systems, navigation, and GIS. Usability has regained its place at the top of the list of priorities, and everyday new ways are found to better integrate technology in the tasks of a user's everyday life. We present here the *DynaMap*, a mobile and context-aware pedestrian navigation application. We show how such an application helped us in understanding the various ideological, theoretical, and technical issues pertaining to the development of mobile GIS systems. We review in particular the design of a mobile real-time client based on predefined constraints imposed by the user (usability), the domain (spatial data handling), and the environment (limited processing power and connectivity). We discuss the difficulty in the integration of the user's context in a way that is 1- system generic and 2- practically applicable in a mobile GIS environment. We show how we circumvented these difficulties. Finally, we present the results of a usability study measuring the user's perceived behaviour of a given set of map rotation algorithms.

Acknowledgements

I would like to acknowledge the following people for their help and generosity:

- Jörg-Rüdiger Sack and Doron Nussbaum. Thank you for the support you provided me throughout my work. I benefited in many occasions from your wisdom and research experience.
- All of the IMA research group members with whom I spent countless hours in the research lab: Hua Guo, Youngsoo Roh, Jiehua Yi, Hongsheng Liu, Ting Wen. Thank you guys for your friendship which I hope will continue well beyond in the future. Thank you Youngsoo and Hua in particular for the technical support you provided me with when it was needed.
- My good friend Hugo Pellerin for simply being there in good and bad times.
- Marianne Desjardins. Thank you for being in my life and for constantly and patiently offering me your moral support.

I would also like to take this opportunity to acknowledge these sources of funding which made possible the research presented in this document:

- The National Science and Engineering Research Council of Canada (NSERC) Postgraduate Scholarship
- The High Performance Computing Virtual Laboratory (HPCVL) Scholarship
- Sun Microsystems

Contents

Acknowledgements	i
1 INTRODUCTION	1
1.1 Contribution	3
2 RELATED WORK	5
2.1 Organisation of the Literature	5
2.2 Previous Work	8
2.2.1 Mobile Ubiquitous Computing	9
2.2.2 Context-Awareness	14
2.2.3 Location-Based Systems	21
2.2.4 Navigation	31
2.2.5 Geographical Information Systems	37
2.2.6 What's Next	39
3 THE DYNAMAP APPLICATION	44
3.1 One Scenario	45
3.1.1 An application too good to be true?	46
3.1.2 The emergence of DynaMap	48
3.2 Motivations	50
3.3 The IMA framework	52
3.3.1 Intelligent Map Agents	52
3.3.2 Framework's Aims and Benefits	55
3.3.3 <i>DynaMap</i> 's role	55

3.4	Requirements	58
3.5	Design	59
3.5.1	The choice of development tools	60
3.5.2	The <i>DynaMap</i> threads	63
3.5.3	Modules and Abstraction Classes in <i>DynaMap</i>	66
3.5.4	The components of <i>DynaMap</i>	69
3.5.5	Raster versus vector spatial data	77
3.6	End-Product	78
4	CONTEXT-AWARENESS IN DYNAMAP	82
4.1	Context Management System	82
4.1.1	The Context Toolkit	88
4.2	What Context?	92
4.2.1	Contextual Influence	93
4.2.2	Context-Aware Realisation	96
4.3	End-Product	100
5	MAP ROTATION	103
5.1	Problem Definition	104
5.2	Proposed Solution	105
5.2.1	STATIC	106
5.2.2	DIRECT	106
5.2.3	R.ANGLE	107
5.2.4	MIN	109
5.2.5	MAX	110
5.2.6	MIN_MAX	111
5.2.7	ADPT	112
5.3	Evaluation	120
5.3.1	Hypothesis	120
5.3.2	Lab Study	120
5.3.3	Field Study	128
5.4	Summary	129

6 CONCLUSION AND FUTURE WORK	130
6.1 Future Work	131
Bibliography	133

List of Tables

2.1	Areas of interest relevant to <i>DynaMap</i> 's development	7
2.2	Examples of context-aware systems classified by their characteristics.	16
5.1	Map Rotation Schemes available in the DynaMap	106
5.2	Algorithms - Sample Data (1/2)	115
5.3	Algorithms - Sample Data (2/2)	116
5.4	Differences Between Map Rotation Algorithms (1/3)	117
5.5	Differences Between Map Rotation Algorithms (2/3)	118
5.6	Differences Between Map Rotation Algorithms (3/3)	119
5.7	Paired Related Samples T-Tests for Sequences and Algorithm for Question 1 [76]	125
5.8	Pair Related Samples T-Tests Comparing Curve and Zigzag Sequences by Algorithm for Question 1 [76]	126

List of Figures

2.1	Organisation of the literature	6
2.2	Computer Systems Timeline	8
2.3	Examples of Currently Available Mobile Devices	10
3.1	Initial Hardware used by <i>DynaMap</i>	49
3.2	The Jpeg GeoInfo Encoder	50
3.3	Screenshot of the Initial DynamicMap Application	51
3.4	Intelligent Map Agent Environment	53
3.5	IMA Framework - Layered View ⁵	57
3.6	IMA Framework - Three Tier View ⁵	57
3.7	DynaMap Hardware Structure	61
3.8	DynaMap Threads	64
3.9	Code Modules of <i>DynaMap</i>	66
3.10	Modularity and Abstraction Diagram of <i>DynaMap</i>	68
3.11	Layered View of the Components of <i>DynaMap</i>	69
3.12	The <i>DynaMap</i> Central Controller State Diagram	70
3.13	The <i>DynaMap</i> GIP State Diagram	72
3.14	The <i>DynaMap</i> Client/Server Architecture	73
3.15	Flow of Events in the GUI Abstraction Process	76
3.16	Raster vs. Vector Data	77
3.17	Screenshots of <i>DynaMap</i> in Action	79
4.1	Context Communication and Storage with CDL	84
4.2	Sample Context Data Encoded with <i>DynaMap</i> 's CDL	85

4.3	The <i>DynaMap</i> Context Manager State Diagram	87
4.4	The <i>DynaMap</i> Context Data Exchange	88
4.5	The <i>Context Toolkit</i> Components	91
4.6	Context Acquisition Diagrams	97
4.7	Context Acquisition Widgets	97
4.8	The User's Preferred Path Context Acquisition Diagram	100
4.9	Screenshots of Context in Action in <i>DynaMap</i>	102
5.1	The Lab-Study Questions	122
5.2	NMEA 0183 Data Generator	123
5.3	Means for Algorithm and Sequence for Question 1 [76]	124
5.4	Mean Ratings for Curve and Zigzag Sequences and Algorithms for Question 1 [76]	126
5.5	Means for Algorithm and Sequence for Question 2 [76]	127
5.6	Mean Ratings for Curve and Zigzag Sequences for Question 2 [76] . .	128

Chapter 1

INTRODUCTION

In recent years spatial data have become increasingly available and used. This can be seen in a number of areas: Geographical Information Systems (GIS), medical computing, image processing, etc. Mapquest [3] and Google Maps [2] are examples of tools now frequently used to obtain location and direction information. Here we are focusing mainly on a particular type of spatial data, i.e., those that are geographically referenced. Such data define objects by their geographical coordinates. For example, geographically referenced data is used when navigating to and from work in the form of driving directions, or when determining what the fair value of a house or a cottage is. The increased dependency on spatial data motivates the development of new technology. In computer science the field of Geographic Information Systems (GIS) specializes in organizing, storing, querying, and retrieving geographically referenced data.

Since its inception GIS has constantly evolved. A lot of progress has been made towards processing and manipulating large amount of spatial data efficiently. Due to the increased size of available data and the complexities of some of the applications parallel computers are required to give users rapid access to requested information. This has led to an increase in the number of practical solutions used nowadays such as satellite image processing technology and/or advanced accounting of the natural resources available in a country for example. However in many cases most of us still use traditional paper-based maps to help with the task at hand over any sort of GIS

solution. There are two main reasons for this: 1- the traditional paper-based map can easily be taken to where it is needed, and 2- GIS-based solutions, although more powerful, tend to be more difficult to operate and comprehend, or in other words are less usable than the paper counterpart.

Currently we can identify a number of recent advances that have the potential to revolutionize the usage of geographic information:

- The growth and advancement in communication technology, in particular wireless communication, significantly contribute to the fact that many new computing devices have the capability to communicate with each other (e.g. WLAN [46], 3G [23], Bluetooth [13], etc).
- The concept of ubiquitous computing, which emerged in the early 1990s [107], is beginning to be realized thanks to the proliferation of computing devices such as Personal Digital Assistants (PDAs), cell phones, portable digital music players, mobile video-gaming platforms, etc.
- Computing devices increasingly offer the potential for context-aware and location-aware computing. More and more computing devices are being equipped with external sensors (e.g. GPS in cell phones). These sensors provide additional information to the device about the user's environment allowing them to consider user's context in the tasks that they perform [29].

We foresee continuous uninterrupted access to geographic information and the ability to process geographically related information anytime anywhere. We envision the emergence of software-based personal assistants who, using this unlimited access to geographic data, provide relevant and personalized information to their user. Applications for this type of software are far-reaching. For example we can imagine an advanced tourist assistant providing suggestions of places to see and visit based on the knowledge the assistant has about its user. We can also imagine business executives relying on their software-based assistants to negotiate the location and the time of their next meeting, introducing the concept of collaboration. However, we observe that in many applications there is a notion of the user having to go somewhere, having to reach a particular location. Concluding, geographic (spatial) data

are increasingly available and required by mobile and other applications. We believe that computer-assisted user navigation is a fundamental building block towards the emergence of software-based GIS assistants.

1.1 Contribution

In a collaborative effort¹, we have designed and implemented a mobile and context-aware pedestrian navigation application: *DynaMap*. *DynaMap* runs on a PocketPC^(TM) PDA. Through the help of a GPS, an electronic compass, and a connection to a spatial data server it provides personalized map visualization and navigation services. Using recent innovations from the fields of mobile ubiquitous computing, context-awareness, location-based systems, and GIS, we have designed a system which is generic enough to be re-applied in the context of other mobile GIS applications. Finally we address the usability issues of real-time map rotation based on the user's current orientation. In collaboration with the Human Oriented Technology (HOT) lab at Carleton University we researched several map rotation schemes and studied how a user interacts with and perceives *DynaMap* for each of the schemes. We have completed the first portion (lab-study) of a 2-part user design testing process and completed all software expansions required for the second user design study (field-study).

First we begin in Chapter 2 with a review of the relevant work. We then provide the design of *DynaMap*'s design in Chapter 3 and show how it meets its architecture modeling requirements as well as its navigation-assistance tasks. Next we focus on the handling of the user's context within the application's architecture in Chapter 4. We describe the context management system of *DynaMap*, and show how it can be applied in the specific case of a student-aware navigation assistance program. Finally we discuss seven map rotation schemes which can be used by navigational tools such as *DynaMap* in chapter 5. We identify the issues pertaining to map rotation, and propose several solutions, including the results of a user study. We then conclude and

¹Sponsored in part by Sun Microsystems, Stantive Computing, NSERC, with collaboration of Jörg-Rüdiger Sack, Doron Nussbaum, Hua Guo, Youngsoo Roh, Jiehua Yi, Hongsheng Liu, Ting Wen, and with external collaboration by M. Duckham

provide suggestions for future work in chapter 6.

Chapter 2

RELATED WORK

2.1 Organisation of the Literature

The work presented in this thesis spans different sub-fields, which come from computer science, and to a lesser extent, also from cognitive science and psychology. Since the early nineties a large number of publications have been written, which have had a direct impact on the design and development of DynaMap. This information was organized in a way that allowed a better understanding of the global structure of the issues to be dealt with. First, a very high level set of categories was designed as can be seen in Figure 2.1.

These broad categories are mobility, context, location, navigation, and geographic information systems. The overlap displayed between each of them in Figure 2.1 illustrates that each category is not mutually exclusive with respect to the other ones. For example location is often seen as a particular type of user's context. Another example is navigation which is often achieved in the real life with the integration of a geographic information system. However each category is distinct in the sense that it has its own particular set of issues and solutions. Re-using the same examples, a common issue unique to location information is to find a good technology that works well in both indoors and outdoors environments, which obviously is not a topic related to the general discussion on context. Similarly navigation, which is the act of going to a destination, is influenced by human-psychology, which is not at all related to

Figure 2.1: Organisation of the literature

the field of geographic information systems. We put *DynaMap* inside the intersection of all the categories because it relies on ideas and concepts from all of these fields. Finally, we use these five categories to classify the different areas of interest that were identified relevant to the development of *DynaMap*. These sub-categories are given in Table 2.1.

Mobility
<ul style="list-style-type: none"> . Future Vision of a Ubiquitous World . Service Discovery . Technical Issues: <ul style="list-style-type: none"> . . . Small Display Sizes . . . Data Transfers Optimization . . . Energy Preserving Methods
Context
<ul style="list-style-type: none"> . Definition . Acquisition Process . Context-Aware Systems . Pattern Recognition and Learning . Usability Studies
Location
<ul style="list-style-type: none"> . Acquisition Process . Location-Based Systems . Usability Studies
Navigation
<ul style="list-style-type: none"> . Definition . Mobile Navigation . Psychology Aspects . Map Display
Geographic Information Systems
<ul style="list-style-type: none"> . Technical Issues . Mobile GIS

Table 2.1: Areas of interest relevant to *DynaMap*'s development

We have identified the areas of interest based on our experience with the development of *DynaMap* and based on our review of the research work accomplished. We find that the exercise of categorization and identification of the different areas of interest relevant to *DynaMap* helped us better understand the nature of our work. Please refer to appendix A for a breakdown of the reviewed literature by the areas of interest. We proceed in the next section with an overview of the previous work accomplished organized following the categories and areas of interest identified in Table 2.1.

2.2 Previous Work

Since their conception, computer systems have evolved tremendously. From the first batch systems of the fifties, to the first multiprocessing systems of the sixties and the time-sharing systems of the seventies, IBM then revolutionized the computing world with its introduction of the personal computer (PC) for the masses in the eighties. After a reign of twenty years, we have entered the post-PC era, the era of *ubiquitous computing* [108] (see Figure 2.2).

Figure 2.2: Computer Systems Timeline

We begin here our review of the previous work accomplished. We start in Section 2.2.1 with mobile and ubiquitous computing, followed by Section 2.2.2 on context-awareness. We then go over location-based systems in Section 2.2.3 and over navigation in Section 2.2.4. We discuss geographical information systems in Section 2.2.5 and “what’s next” in Section 2.2.6. Finally we end this chapter by providing a concluding remark.

2.2.1 Mobile Ubiquitous Computing

Foundations

Weiser proposed a shift in computing paradigm through his observations relating to how we had failed to make computing truly adapted to human needs [107]. His claim was that “The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it” [107]. He envisioned that computers would become an inherent tool used by humans similarly to the way that literacy is being used. According to him, “*literacy technology* does not require active attention, but the information to be conveyed is ready for use at a glance”.

Weiser observed that the main issue with PCs is the fact that they always require the complete focus of the user, making the initial task that the user wanted to achieve become of second priority. In an effort to provide a solution to this problem, Weiser proposed a new computing paradigm that he coined *ubiquitous computing* [107]. In this world, instead of having one computer, the PC, which serves all computing needs of the user, there would be a multitude of small, interconnected devices, each with processing power. Each device would then serve one purpose while minimizing the user’s need to understand how this is achieved. By being all interconnected, the devices achieve super-connectivity and can potentially rely on each other to accomplish their tasks. In order to prevent a user from having to go to a particular location in order to access these devices, or from having to carry them everywhere, it is assumed that a multitude of devices would be available regardless of the user’s location. This configuration was later termed *anywhere, anytime* computing.

Weiser’s futuristic vision was supplemented by the first mobile ubiquitous systems which took the forms of mobile tabs, pads, and boards [107]. The mobile devices of increasing sizes were meant to be prototypes of mobile and ubiquitous information purveyor such as current location, calendar, diary, notes, or slides in a meeting for example. Almost at the same time came the *Active Badge* [105] from the Olivetti labs. The *Active Badge* is an in-door personal location system based on the Infrared frequency. Badges worn by the users are detected by sensors spread through out

a building. The location of the user is then sent to a central server which can be queried. An example application is the one where a telephone operator queries the server to redirect calls to the recipient's nearest phone.

From Vision to Reality

Since the mid 1990's a multitude of smaller and smaller electronic gadgets have appeared. We are now more than ever entering in a mobile world by using cell phones, global positioning systems (GPS), personal digital assistants (PDA), and even tablet PCs (2.3). What we are observing now is the convergence between these devices. For example, cell phones are equipped with GPS systems, and PDAs are also phones. The integration of these devices in our daily tasks allows researchers to further investigate the potentials of pervasive systems.

Figure 2.3: Examples of Currently Available Mobile Devices

Through innovation we are able to get closer to the pervasive computing world of Weiser. Many technological issues must be addressed in order to create a truly mobile and ubiquitous environment. For example one simply has to look at the display size of mobile devices, being smaller than anything system designers have

been accustomed to over the years. Many traditional user interface (UI) assumptions have to be discarded and re-established in a mobile setting not only due to the smaller screen size but also because of the different contexts in which one uses the devices. For example a user involved in an activity requiring most of his/her focus, such as driving a car, will have different UI requirements than one who can devote his/her full attention to the tool. Brewster proposed to use sound [14] whereas Oviatt suggested multimodal interfaces [82] to overcome the problems of screen size and user focus. Both solutions worked well in specific applications, but more work is needed to determine when to apply them in more general terms. For the particular case of graphics rendition Suomela et al. came up with different views (2D, perspective, 3D) and provides a list of advantages and disadvantages for each one in a mobile setting [99]. In *DynaMap* we decided to offer 2D maps due to simplicity and the fact that users are accustomed to such maps. However it would be interesting to see how users behave when presented maps in the other views.

The diversity of the devices also means that there is more than one way to interact with them. Nowadays the use of hardware buttons and touch-sensitive screens with a precise stylus seems to be the norm, but we can expect new methods to be developed like in the head-mounted display in the mobile authoring system of Baillot et al.[6]. Often the same application can be run from a multitude of different devices like in the mobile VindigoTMGIS application [16]. In that case the application must adapt to the changing constraints of each device, e.g. in the mobile navigation system of Baus et al.[10]. Fischmeister proposed in [35] a generic UI description language which is understood by all devices, which then knows how to generate such UI on themselves, hence alleviating the system's designer task. We were inspired by Fischmeister work and used a similar technique to gather system settings in *DynaMap* (see Section 3.5.4). Rist et al. propose instead that graphics be modified on the fly to fit different devices [88]. In any case, there will always be displays of all sizes with different user input techniques. This review process where old assumptions have to be reconsidered is refreshing, and will be pursued for some time.

Display size is by no means the only issue encountered by mobile and ubiquitous

system designers. To keep devices truly mobile they depend on wireless communication technology. Although communication networks are never perfectly reliable, wireless communication suffers more from noise, slower speed, and intermittent connectivity. Since most of the time application designers have little that they can do about which communication channel they can use, they have to instead look at ways to reduce their dependability on data exchanges. The first step is to reduce the amount of data to be transferred, and compression algorithms are often used, e.g. see [104] [16] in a mobile setting. Another solution is to modify the data structures to be transferred in a way that enables them to be sent progressively, or in other words, that the first amount of data sent can be used right away without having to wait for the completion of the transfer ([67] and [104] are theoretical and applied solution related to this technique). Radio streaming on the Internet is an example of a progressive data transfer where the data is used by the application as it is received regardless of whether or not the transfer has completed or not. If the information being transferred is related to the user's location, then there exist ways to cache the result of the transfer using cache replacement rules based on the delta from the user's previous location [87]. The inverse consists of trying to predict the next location of the user and therefore trying to determine future data requests to be communicated. This technique known as prefetching (or hoarding) reduces wait times experienced by the user due to data transfers, if done successfully [66]. In addition, depending on the type of query made (e.g. shortest path query) and on the type of data being retrieved (e.g. graph data, geometric data, etc.) one can take advantage of some of the latest results obtained in the field of external memory data structures and algorithms [79] [21] [22] [4] [55] [44]. In *DynaMap* simple communication optimization techniques are used such as caching the transmitted data requested, and ad-hoc prefetching of map segments expected to be required in the near future.

There are many other issues related to mobile and ubiquitous systems for which researchers have provided solutions. We terminate this section with an overview of some of the ones we find to be the most interesting. First of all one of the most basic problem is the one of finding a good hardware design for mobile devices. This is a crucial part of development since everything else is built on top of it. Schmidt

proposes the *Smart-Its* [93] as a standard hardware platform that can be integrated into any object that needs to become mobile and ubiquitous (application dependent; Schmidt uses the example of a cup). The *Smart-Its* optimized design offers wireless communication and low power consumption while providing decent processor power and memory storage. That brings us to another issue, the one of energy scarcity in mobile devices due to their reliance on battery power. As we know batteries can only supply energy for a bounded time period before recharging is needed. In order to increase the usability of an application one must take great care to preserve power and thus extend the time between charge ups. There is a new trend in the field of algorithms and data structures to not only evaluate performance of various solutions by their speed of execution and their memory usage, but also by their impact on power resources. For example An et al. provides a study on “energy-performance trade-offs for spatial access methods on memory-resident data” [1]. Another issue is the one of service discovery. Since devices are mobile they cannot always rely on known resources to complete their tasks. Instead they must adapt to new computing environments as they are being carried from one place to another. Due to the impossibility to predict the configuration of every environment, a device must use a discovery protocol to find out what services are available. A number of protocols have been defined to provide such discovery process. These include SLP [57], HAVi [50], UPnP [73], and Jini [74]. We refer to [36] for a review of the work done on this topic. Finally there is the issue of privacy, which is quite relevant in this context since devices are often learning intimate facts about their users (e.g. their location) in the name of so called better technology integration. A number of authors have raised some issues in the fear that all the information pertaining to a user would fall in the wrong hands [31], while others have noticed that to gain acceptability, technology will have to demonstrate the benefits to the user for him/her to let go of his/her private information [59] [17] [105] [7] [63]. We do believe this to be true, but we also note that in many cases of privacy breaches it did not matter whether or not the best technology at preserving user privacy was being used, but whether the user took care at doing his/her best at preserving his/her privacy. We believe that it is the human who should be in control of which private information is being released.

In any case, it was quickly observed that in order to achieve *calm computing*, computing that requires the minimal user's attention, as referred by [108], applications running on mobile devices need to become more proactive in understanding the user, and his/her context.

2.2.2 Context-Awareness

Foundations

One key concept that emerged from mobile ubiquitous computing is the need to change the user's relationship towards the multitude of tools available nowadays. We can no longer expect the user to learn or train him/herself for each new mobile device that becomes available. Instead improvements must be made in key areas such as ease-of-use and peripheral output [108]. A user cannot handle many items if each one requires his/her full attention. The onus is therefore on the technology to integrate more naturally into the user's context of action. Besides *calm computing* [108], many other names are used to describe the same concept such as *invisible computing* [80] and the *disappearing computer* [109] (listed here for reference only).

Schilit [91] first formalized the notion of user's context and context-aware computing in his research on the Xerox PARCTABS, which were first introduced by Weiser [107]. The tabs are palm-size hand-helds that can communicate with other nearby smart tabs, pads, boards, etc. It was observed that the usability of the device decreased quickly if the user had to reconfigure it every time it was brought into a new environment with different apparatus to communicate with. This has lead to the first definition of *context-aware computing*: “software that adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time” [91].

Not entirely satisfied with this rather technical definition, other researchers have tried to find a more general definition of what context and context-aware computing are. Of special significance is the work of Dey. Dey defined context as “any information that can be used to characterise the situation of an entity” and context-aware computing as “a system that uses context to provide relevant information and/or

services to the user, where relevancy depends on the user's task" [29]. These definitions, which are based on the notion of relevance to the user, eliminate the need to enumerate the environment in which a user operates. Indeed, depending on the situation, context is as simple as a user profile composed of certain user's preferred settings (font size, user's mother tongue, etc.), while in other cases it may consist of more complex information like the user's immediate computing surroundings (available printers, internet access, etc.), or even historical information such as the past one hundred queries that the user made in his favourite search engine for example. What information is worth to be labeled as relevant to the user is dependent on the current situation, and is consequently unbounded.

Context Classification

As the understanding of user context, and context-awareness became clearer, more work was conducted at defining the characteristics of context. For example, in their survey paper Chen and Kotz [17] differentiate between

Active Context Awareness: an application automatically adapts to discovered context, by changing the application's behavior,

and

Passive Context Awareness: an application presents the new or updated context to an interested user or makes the context persistent for the user to retrieve later.

This distinction allows for different levels of context-awareness in an application. Barkhuus and Dey [7] offer a third level, that of *personalization*, which mainly consists of a deterministic number of settings that a user can set to his/her liking.

Finally in his Ph.D. thesis, Schmidt [93] offers a three dimensional context categorisation: *Self* (*device state, physiological, cognitive*), *Activity* (*behaviour, task*), and *Environment* (*physical, social*). Table 2.2 presents examples of different context-aware systems classified by the above mentioned characteristics.

	Self	Activity	Environment
Active	A cell phone that turns itself off before it runs out of battery	A navigation application that provides the simplest route when the driver is driving in a dense traffic area such as downtown	A PDA that automatically configures itself with the nearest printer
Passive	A refrigerator that reminds the user to buy milk when the box of milk is almost empty	A word processor that recommends some preformatted documents when it finds out what kind of a document the user is writing (e.g. a professional letter)	A car that reminds the user to slow down when it detects that the police is doing radar control in the area

Table 2.2: Examples of context-aware systems classified by their characteristics.

Context-awareness is currently at the center of much research work, and like any evolving technology it is easy to get lost in the various definitions and notations that are given. Other names given to context-awareness include *sentient computing* [53], *situated computing* [54], and *pervasive computing* [15]. In this document we use the terms *Mobile Ubiquitous Computing* to refer to the emergence of smart, always on, and interconnected devices, and *Context-Aware Computing* to refer to the concept of integrating the user's context as an integral part of a system's behaviour. With respect to context-awareness, we use the term throughout this document as it is defined by Dey [29]. We also refer to the categorisation of context given by Chen and Kotz [17], as well as Schmidt [93] (see Table 2.2).

The problem of understanding context and defining it did not prevent context-aware systems from being developed. After the auto-configuring tabs from Xerox PARC, many other projects followed. For example, in their work Nagao and Katsuno [75] used personalized mobile agents to implement what they termed the Hyperdialog tool. The tool enables two members, e.g. in a social gathering event, to exchange information digitally such as email addresses and/or websites, etc. But even more

interesting is the capability of these mobile agents to automatically communicate to each other, exchange information about their user, and search for other users in the proximity who might share common interests.

Of course, one of the most often cited context-aware system implementations is the *GUIDE* [19] project developed at Lancaster University. This particular project consists of a number of tablet PCs that can be borrowed from the Lancaster tourism office for use by tourists. These tablet PCs, which are each equipped with WLAN receivers, are running the *GUIDE* client application whose task is to assist the tourist in his/her visit of the city. A number of non-overlapping WLAN cells are deployed around the city, which then provide two important items to the *GUIDE* system: 1- it allows the *GUIDE* client application to know in which part of town it is currently located by using the cell coverage map and, 2- it allows for within-cell broadcasts of local tourist information to the clients. Finally, an effort is being made at providing the users with tailored tourist information based on their profile, which they must enter in the system prior to using it.

Context Acquisition

There are naturally many other context-aware systems that were implemented over the years, but it is outside the scope of this document to list them all. However, the development of these systems helped identify many of the difficulties that one must face to build a context-aware application. One of the most basic problems is the one of acquiring context. Unless the only context used by an application is of type *personalization* (user profile, entered directly by the user once) otherwise a lot of work must be done for collecting context, interpreting it, and acting upon it.

Indeed, most software engineers will stop at this point in the reflection and ask themselves: “How do I go from the various abstract and encompassing definitions of context-awareness to an actual concrete implementation of such system?”. The answer to this very basic question is still open, and is the source of intense research activity.

The first step in realising context-awareness is to find a way to model the user and his/her environment. To that effect, a number of solutions have been proposed,

two of which have passed the test of time. First of all, Dey in his *Context Toolkit* [28] proposed that one should borrow from the widget metaphor, which is already used extensively in the world of graphical user interfaces. The widget metaphor is to encapsulate in a reusable piece of code, termed a “widget”, functionality that is most likely to be needed in multiple places. Every time the functionality is needed by an entity, that entity simply has to use the widget as a black box. In the context of a GUI, widgets are used to encapsulate various user input mechanisms (text fields, radio buttons, drop-down boxes, etc). Widgets in the *Context Toolkit* represent a pre-defined piece of context to collect (either simple or complex, e.g. location or user’s intention). The benefits from using them are multifold. First of all, a widget abstracts out the complexity of dealing with low-level sensors. For example, an application might require to know the location of the user, in which case it would make use of a location widget. Since the interaction with such a widget is standardised, the application does not need to know how the user’s location is extracted from which sensor. Additionally, in the event that the actual underlying sensor were to be replaced by say a more accurate one, the context-aware application would not need to be modified as long as it makes use of the standardised widget (which however must be replaced by one that can deal with this new sensor). In Dey’s model, a context-aware application would have a “library” of context widgets available to be used. The fact that this model has been implemented, and tested in a number of scenarios explains its popularity [28]. One of the other benefits of the *Context Toolkit* is how it provides for a complete context architecture from context acquisition, to context encoding, and context propagation. For a more in-depth overview of the *Context Toolkit* refer to Section 4.1.1.

The second approach, proposed by Schmidt [93], is a bottom-up object oriented view of the world. Schmidt argues that instead of trying to understand and model the whole set of entities that makes up the user’s context, one should instead focus on making each object a context provider based on the features of that object. An example is the *MediaCup* [11] where coffee mugs are equipped with sensors and circuit boards which enables the cup to provide information about itself such as current capacity, temperature, etc. The cup itself is indifferent to who uses this information

that it provides, which is one of Schmidt’s aims: to remove the need to model the complete world around the user. In order to achieve such a ubiquitous and context-aware environment, where each object is aware of its states and can provide them to the outside, Schmidt and his colleagues developed the component called the *Smart-Its* [93] that we introduced in Section 2.2.1. This small circuit board provides the most basic functionality for objects to become context providers.

The bottom-up approach of Schmidt does not necessarily contradict the widget metaphor of Dey. They simply are two different models based on different assumptions, and one could attempt to bridge the two paradigms. One thing so far is common, however, and that is the central role of sensors in each model. This has often been criticized as being too simplistic and in somehow helping the current trend where most context-aware applications only make use of low-level context information (e.g. location, temperature, current time, etc). Barkhuus proposed in [8] that a clear distinction be made between “User level” information, “Context level” information, and “Sensor level” information in any context-aware design and development.

Context Interpretation

Another issue when dealing with context information is the interpretation of context. Namely how to use context and how to draw appropriate conclusions from it. For example some cars nowadays offer an automatic door locking mechanism as an extra security measure in case of accidents. Initially it was decided by the system designers that the doors should lock automatically after the engine had started. However there were cases when such behaviour was not wanted. For example a driver starts the car with the door opened and without shutting the engine steps out of the car closing the door to fetch something that s/he has forgotten to bring along. The result is that the car locks the doors with the driver being outside of the car. Therefore effort is required in order to facilitate context interpretation at design time. Using Dey’s *Context Toolkit*, a developer can build *Aggregators* and *Interpreter* widgets, whose aim is to respectively combine different pieces of context into one, and to deduce new context from previously acquired ones. There exist also advanced context inference engines based on artificial intelligence [65] as well as context fusion platforms [77]

that can merge together context information coming from different sources whether or not they are contradicting each other.

Context Distribution

The distribution of context consists of making available to the entities that requested it the newly acquired and interpreted context. Again we refer to Dey's *Context Toolkit* as one possible solution to this problem. The toolkit is based on IP networks, allowing anybody with a network connection to have access to the widgets. A discovery protocol is used to permit automatic widget detection. Finally, the notion of the blackboard metaphor [110], i.e. where the user's entire context is available to all clients in a central fashion, is also supported via an extension. Indeed, the blackboard infrastructure is often assumed in the context-aware literature, but is achievable in practice only for systems working in a setting involving a few different pieces of context. We note at this juncture that the overall idea of using the network as the communication layer does not only allow each context provider to run independently, but it also provides for a standard interaction procedure that facilitates the overall integration of the context-aware application with the outside world. The problem of global integration of independent systems is a major obstacle in the general effort to bring to life the concepts of mobile ubiquitous and context-aware computing [27].

Context-Awareness Usability

Over the past few years new context-aware applications have evolved from merely sensor-based systems, into smart and more advanced inference engines. For example refer to Marmasse [68] and similarly the work of Ashbrook [5]. In both cases they developed a learning mechanism used by their system to discover places that are deemed relevant to the user. The inference is based on the user's travel pattern, as well as the amount of time spent within a pre-defined radius. Once a location is discovered, the user is asked to confirm whether or not this place is relevant to him/her, and slowly a user-centered location model is built.

Recent research into the usability of context-awareness seems to indicate that the

current philosophy that “more context is better than less” does not equal increased usability. The problem mainly lies in how context-aware applications often take the user out of the control loop, when the intended objectives were to simply better integrate the application into the user’s everyday life [32]. If the user is no longer the one that controls the application then that application must always correctly interpret the user’s context in order not to behave in ways that are in contradiction with the user’s desires [7] [18]. However, it is well accepted that no matter how advanced our technology is, it is still quite far from a truly intelligent system that can fully comprehend the human world (refer to the automatic car locking system described previously as an example). Another issue is how to deal with uncertainty due to noisy context [18] [27]. Baillot et al. [6] even recognised this to be a problem in their own system used for authoring physical environments. Finally, assuming we are dealing with accurate context information, then comes the difficult task of not making any error of interpretation [18]. Yet again, the general solution to all these usability problems comes from a return to the basic principle of letting the user be in charge, since s/he is the only one who can with certainty decide what the next step should be [32]. In other words, when in doubt, developers should favour the use of passive context, and keep active context for cases that are not dramatic if an error occurs.

It is important to emphasise at this point that most usability studies whose focus is on context-awareness show a high appreciation rate from users, despite the previously mentioned usability bugs. We give as example the *GUIDE* system [19], the *WalkMap* [99], the work of Brewster on overcoming lack of screen space with the use of sound [14], and the scenario-based usability study conducted by Barkhuus and Dey [7].

2.2.3 Location-Based Systems

User location is playing a key role in the world of mobile, ubiquitous, and context-aware computing. We only have to look at the acceptance criteria of location-based systems (LBS) as defined by Geake [39]:

- speed of delivery

- completeness of delivered information
- compactness or clarity of delivered information
- currency of delivered information
- relevance of delivered information
- simplicity of use

All these points are in line with the defining thoughts of context-awareness. There is in fact a movement in the research community which argues that location is too predominant in the research on understanding context-awareness, and that a more encompassing view of all possible contexts should be favoured [92] [28].

Location Acquisition

A large number of applications are developed where user location is an input to their system. An explanation for the growing popularity of LBS comes from the fact that user location can now be detected by computing systems in a variety of ways. Current location acquisition technology is divided into two sectors: 1- outdoors location and 2- indoors location. The main reason for this physical dividing line is due to the difference in the coverage area for which the user's location must be detected. For most outdoors systems it is considered fine to provide location information within a few meters of precision, while some indoors systems require sub-centimeter accuracy. This can be explained by the observation that user location is a relative concept dependent on the coverage area. For example to locate the user within one or two meters in a coverage area of ten thousand square kilometers is considered accurate, whereas the same precision in a twenty square meter room is not. Fortunately indoors environments tend to be much more controlled than outdoors ones. This gives more options to engineers in developing location obtainment technology. However, it is often forgotten that outdoors vs. indoors location represents fundamentally different types of information altogether. Indulska and Sutton [59] classify location information into three categories:

- physical
- logical
- virtual

According to them, physical location comes in the form of some coordinate system, or frame of reference. Longitude and latitude, (x, y) coordinates, or “you are three meters away from the sensor” are all examples of physical location. Logical location refers to location in a logical location model. For example, a hierarchical location model based on the containment topological relationship (I am in the psychology lab, which in turn is on the third floor of the social science building, which is...). Finally virtual location is defined as location information obtained from deducing or inferring location from other types of information. For example “Tom must be at home because I just called him on his land line”, or “Jack is in the psychology lab because he has just logged on to one of the workstations there” are examples of virtual location. With this type of location classification, we can see that outdoors LBS are more involved with physical location. Alternatively, indoors LBS deal more with logical location and are insensitive to whether this information was acquired directly from logical sources, or via a physical or virtual transformation.

There exist currently many ways to acquire location information. It is an undisputed fact that the most commonly used location technology for outdoors environments is the global positioning system (GPS). In 1973 the U.S. department of defense designed a system, based on radio signals sent via satellites, to determine the location of anyone on Earth with great accuracy. In 1978 the first satellite was launched, and by the mid-1990s the system was fully operational with a total of twenty-four satellites. In order to obtain his/her location, a user must have a GPS receiver, which must have access to the signals of at least four satellites. Due to the frequencies and amplitude of the signal emitted by the satellites, the receiver must have line of sight, forcing the global positioning system to be used only outdoors, and in open areas. In addition, the U.S. military used to scramble the signals emitted by the satellites reducing the accuracy at which the receiver could pinpoint its location. This was done in fear that the technology could be used by enemies of the United-States, and

was initially detrimental to making GPS a mainstream location acquisition system. However, some people tried to circumvent this loss in accuracy by planting at fixed known locations extra signal emitters. GPS receivers could be augmented to take into account the signals for these extra emitters in order to more accurately produce their own location. This system is known as the differential GPS (DGPS), and has been used in some of the earliest Location-based systems [101] [98]. The scrambling of the satellite signals has now been reduced to the point where high-end GPS receivers can obtain a location up to two to three meters in accuracy. This level of precision is good enough for most LBS and studies have shown that the little bit of extra accuracy possible via DGPS is no longer relevant in most practical cases [94]. Nowadays GPS is the most common position system for outdoors LBS and is therefore the location acquisition method of choice for several projects found in the literature ([85], [10] to name only a few).

However, GPS is not the only outdoors location acquisition system that exists. More and more location-based systems are using the cellular phone network to establish the user's location. The coverage area of cellular phone networks is defined by the union of the coverage areas of each individual deployed antenna. The area covered by an antenna has some geometrical shape, which can be approximated by measuring the strength of the signal, and analysing the impact of the topology of the terrain into which it is located. Since a cellular phone receiver communicates with only a single antenna at any given time, the one with the strongest signal, means that the location of that receiver is assumed to be in the area covered by that antenna. In the best case, nowadays, the location of the receiver can be estimated plus or minus fifty meters, in urban settings. Although that level of accuracy is still too low for most LBS, the fact that the same technology can be used for both communication, and for location acquisition makes it interesting. Among the positive points is a reduction in necessary external hardware, and a decrease in the energy required to run the overall system. Beside accuracy, there is however one more negative aspect to this approach and that is the fact that the user's location is detected from a central entity. This central entity is the phone network administration utility, which then knows the location of all the users. This setup raises prospects of privacy infringements.

More recently, another approach is being investigated, and that is the one of assisted GPS (A-GPS). This technology came up with the passing of a US Federal Communication Commission (FCC) rule that stated that all wireless carriers had to be able to provide geolocation of an emergency 911 call [33]. According to the rule, network-based solutions had to be accurate within one hundred meters sixty-seven percent of the time, and within three hundred meters in ninety-five percent of the time. Handset-based solutions had to be twice as accurate from the network-based solutions for the same percentage of the time [33]. It is at that time that the A-GPS, which is essentially the combination of the cellular phone with the GPS, was born. In that system, when satellite signals are sensed, the GPS functionality is used to provide location detection, with the help of the cellular phone network. This results in a more accurate positioning, similar to DGPS, and in one which requires less signal strength from the satellites. In cases when no satellite signal is detected, cellular phone network positioning is used with its limited accuracy. However, since GPS time can be used to help establish a synchronized clock, the positioning precision of an A-GPS cellular phone is still better than the one without GPS integration [33]. Since many LBS are already using both cellular phone networks as their communication carriers, and GPS for positioning, the arrival of the A-GPS seems like a natural evolution.

A current disadvantage of cellular-based phone networks is their limited available bandwidth compared to other wireless communication technology such as 802.11 WLAN [46]. Cell-phone companies are not yet ready to provide a high bandwidth wireless data stream to a large number of mobile and ubiquitous users, even with the upcoming of the new 2.5G and 3G protocols [23]. That scarcity of bandwidth resources is a direct consequence of its high price, making most LBS with on-demand data transfer features rather expensive to use. The need for a better and cheaper wireless communication stream has lead some to turn to privately deployed wireless networks using IEEE 802.11 WLAN for example. That implies a significant amount of manual labour to deploy the antennas and any other required equipment in order to create the desired network. This however gives system operators a complete control over the coverage area of each antenna, and can therefore achieve a fairly accurate

positioning system in the same way that it is achieved for cellular-based phone networks. Example of LBS which used that approach are the *GUIDE* system from the city of Lancaster, UK [19], and that of the *MUSA-Shadow* developed in Constance, Germany [35]. There are however a few trade-offs in using a private network: 1- The overall coverage of the network is limited due to the high cost of deploying it, and 2- the energy burden which is put on the batteries running the devices using these high-bandwidth networks is high. Data transfers, and more particularly wireless ones, are indeed costly in terms of energy usage due to the increased voltage requirements that come with transmitting.

Note that we are at a cross-point here since WLAN networks, when used as a location detection facility, may be used both in indoors and outdoors settings. Indeed, one of the most popular methods of providing location information indoors is through some kind of proximity sensor, like in the WLAN case. However proximity is usually detected at a much smaller range than that of outdoors proximity detection. The first piece of work that started the location-based system movement was actually describing a new indoors location acquisition technique: the *Active Badge* developed in the Olivetti labs [105]. The system consisted of a network of infrared sensors deployed at various locations throughout the building. Users would be wearing a badge, whose signal could be detected by the sensors. The system was used by a telephone operator, who redirected calls to the nearest phone where that person's location had been determined. Many other research projects subsequently used the *Active Badge* as their location detection technology. The main drawbacks of this system came from the infrared system, which provides, at the most, room level accuracy since an infrared signal bounces off walls and furniture. Similarly, radio frequency (RF) based systems have been developed, such as bluetooth [13], but their location detection capabilities are again limited. This time though it is for the opposite reason: RF signal goes through walls; therefore the emitters must limit the intensity of their signal so that it should not be picked up in other locations. This results in a bothersome increase of RF sensors to detect these weaker signals in a continuously covered area.

In an attempt to solve all of these previously stated issues, the Olivetti research labs team proposed yet another location acquisition system: the *Bat Ultrasonic Location System* [106]. They used time measurements of sound pulses to establish a receiver's location. The system consists of at least three or more receivers fixed at known locations on the ceiling of a room. A transmitter, identifying a mobile user or object, then transmits a sound pulse picked up by the receivers. The receiver then evaluates the distance of that object to them by measuring the time taken between the emission of the pulse and its actual reception. A centralized system captures all these time intervals, and through multilateration calculations (like in the GPS case) determine the exact position, plus or minus fifteen centimeters, of the transmitter. The *Bat Ultrasonic* indoors location detection system is to date the most accurate one available. However it does suffer like its infrared, or RF counterparts, from potential privacy breaches since location is determined via a central administrative entity as opposed to from the devices themselves (whose location is to be determined).

Although the wireless signal proximity sensing methodology is one of the most common ways to detect the user's location indoors, there are quite a number of other ways to do so. The most basic one is to let the user specifically confirm his/her location. The Maxim *iButton* [56] is an example where a user gets near a reader with a computer chip embedded in a metallic button. That chip uniquely identifies the user, and the reader then recognises that the user is located at that reader's position. This system has been used in the *Context Toolkit* [28], and other similar machine readable ID systems (e.g. bar codes ID tags being scanned) have been used in other cases [75]. Such system, although intrusive and providing only logical location, has the advantage of having an almost non-existent error rate. Among other types of indoors location systems we also note the *Context-Aware Floor* [93] where a floor is sensitive to the load being put on it. Sensors are installed at certain positions on the floor, and are capable of determining the location of a single moving entity. Although this method suffers from multiple limitations, it shows the level of innovation currently found in the field of LBS and context-awareness. For more location acquisition ways we refer to the survey on context-awareness by Chen and Kotz [17].

Location Models

Obtaining location of a user is only the first step of any LBS. The end goal is to actually act upon it; to change behaviour based on that information. In order to work with location, an application must first understand it. For this reason all LBS must define their location model. There are different ways to model location, and there are trade-offs between them. Therefore a LBS designer must take good care of choosing a location model that satisfies the need of the system. Bauer et al [9] have listed the following items as requirements of what a location model language should be able to do:

- describe the geometry of objects relative to different coordinate systems,
- express symbolic information, e.g. names and descriptions of objects,
- express all possible relations between objects,
- support the description of objects in different levels of detail and provide the means to express connections between sub-models in different levels of detail, e.g. containment.

To which we add the following communication requirement:

- support transfer of information via the given communication channels.

Again, different location models will serve different purposes, and some location models are better at certain things than others. For example, pure positioning is the location model of many LBS. For several LBS, exact geometry and placement of objects is what matters the most, and such location models tend to be closely related to a certain coordinate system such as Longitude/Latitude and Cartesian systems. But exact location in space is not necessarily what matters to all applications. In certain cases what is important is more the topological relationship between predefined locations such as in RAUM model of Beigl et. al [11]. Their model is based on a hierarchical tree representation of the space. The parent-children relationship in the tree represents a topological relationship in the space. This specification of the

actual topological relationship is defined at design time by the engineers of the LBS, as long as it has all the mathematical properties defined in [11]. A common example is the containment relationship where the root would be the parent of all things, e.g. the universe, and going down the tree means entering inside the space of the parent. Another example of a location model is the *Semantic Location* proposed by Pradhan [84]. In his model the meaning of location is more important than its actual position, or than its spatial relationship to other places. Therefore he proposes a model based on uniform resource identifiers (URI) which could include various contextual information about the place encoded (including possibly its position). He argues that this way, a semantic web of locations is created, opening the door to current web technology for accessing the information. He also argues that privacy issues disappear in his model due to the anonymity of web users, and to the lack of possible inference on their actual physical location. Finally, Marmasse and Schmandt [68] came up with an interesting model, one that is based on the user's important places. Based on previous latitude and longitude readings, they have developed a pattern classification engine capable of extracting places relevant to the user based on the number of occurrences the user has spent at these locations. Their method used in the *comMotion* [68] system then asked to user to confirm the importance of the newly found location, and to categorize it under certain criteria. Their idea is based on the notion that context should drive the model and not the other way around.

Applied LBS Examples

As we can see, a large amount of work has been done to understand and improve location-based systems. Over the years a huge number of LBS have been brought to life; only a few of those are enumerated here. We refer to [28], [17], and [93] for an extended list of the work accomplished. We provide a sample of the most relevant work here, classified by different application types/objectives.

There is a large number of LBS's aimed at easing day-to-day tasks of office workers. We recall the "find-me" telephone operator service provided with the first indoors location acquisition technology: *The Active Badge* [105] (introduced earlier in this section). Similarly, Dey developed a few office related applications to demonstrate the

benefits of his *Context Toolkit* in [28]. In particular there is the *In/Out board* which detects who is in the building and who is not (actually based on the work of Xerox PARC [107]), and also the *Intercom* system which allows intercoms to be used on a one-on-one basis, based on the participants' location, as opposed to the broadcast scheme that we are accustomed to. Then there are also a few LBS prototypes aimed at improving a user's shopping experience. For example there is the *Shopping Jacket* which alerts the user if s/he walks past of shop that might interest him/her [85], and similarly there is the *MRM Server* which provides location/user aware advertisement via surrounding displays [61]. Proactive displays have also been used in other settings such as in conferences [72]. The tour guide type of LBS has already been visited with the GUIDE system of Cheverst et al [19] as we have shown in Section 2.2.2. Along similar lines is the tourist guide of Simcock et al. which offer a smooth integration to different tourist areas as it is deployed by the system designers [94]. Field engineers in the UK working in the utility industry in the public sector now have access to a GIS mobile context-aware system. This application was developed in Manchester to increase efficiency of collaborative or single work among team mates [52]. In case of emergency we might soon be located by just a few meters to our actual position thanks to our A-GPS cell phone which would guide the rescue team to us [33]. The emergency team might even be helped by a tailored navigation system for a rescue effort such as the one found in Montréal, Canada at Urgences Santé [90].

Most of the previously cited examples of LBS are in prototype stages, and still under research. There are however a number of already commercially available LBS on the market. For example, car navigation systems such as [58] have been offered for a while in high-end vehicles (see next section for more information on navigation). As well, certain personal digital assistant (PDA) computers, or cell phones, now offer information regarding services offer in your vicinity like in the *Vindigo* mobile device [16]. In any case, whether LBSs are in prototype stages or not, their usefulness has been studied and documented [63] [45], and are destined to grow both in number of services available, but also in the value that these services bring us.

2.2.4 Navigation

Definition

Navigation is defined as “The theory and practice of navigating, especially the charting of a course for a ship or aircraft” by *The American Heritage Dictionary of the English Language*. They also define navigating as “To plan, record, and control the course and position of (a ship or aircraft)”. Our understanding of navigation goes along the same lines, with the exception of having no limitation on the type of vessel used (ship, aircraft, car, on foot, etc.). Our definition of navigation therefore is:

Navigation: The act of planning, recording, and controlling the course and position from a known starting location, to a known arrival location.

The task of navigation is therefore well suited to be assisted via a mobile ubiquitous and context-aware system, and that is because of three reasons:

1. User’s location is central to the task of navigation. It is also part of the user’s context.
2. As a user is navigating, therefore moving into some environment, only a light and self-sufficient, hence mobile device can be carried around to help with the task at hand.
3. Beside location, Baus et al. [10] have shown that other contextual information about the user is essential to provide him/her with tailored navigation instructions. They give the example of navigation instructions given to a time-pressured business man, versus the ones given to a more relaxed tourist (see also Nivala in [78] for other examples of how context can influence navigation).

Applied Examples

It is therefore not a surprise that there are quite a few already developed mobile ubiquitous and context-aware navigation systems. We have in fact already mentioned that in-vehicle navigation systems have been around for a while already in high end cars [58]. One of the first navigation applications is the *map-in-the-hat* of Thomas et

al. [101] where a user, having previously entered a number of waypoints to be followed, is given through a head-mounted display the status of the orientation/position to the next waypoint. Although limited, their application was a pioneer in the field. Later Tarkiainen et al. [100] developed a turn-by-turn pedestrian navigation system to be used by WAP browsers (e.g. on cell phones). User location was entered manually, and so was the confirmation that the last step was accomplished. That aside, the simplicity of their application, and the care taken to provide pedestrian-tailored route information proved to be a success among the users. Then came the work of Baus et al. [10] with the *REAL* system, which they portrayed as being resource-adaptive. Consisting of both an indoors (infrared sensors) and an outdoors (GPS) component, the *REAL* system offered to date the most context-aware navigational features. For example route description is tailored according to screen resolution, size, and colour capabilities, and as well to location accuracy, user orientation, and user speed. Finally, the work of Suomela et al. [99] consists of the *WalkMap*, a navigation tool whose focus is on map visualization. They experimented with three different views of the geographic data: birds-eye view, perspective view (like birds-eye view but at a fixed angle other than 90 degrees), and virtual view (full three dimensional rendition of the data). They argued that the perspective view suffered the most from location acquisition inaccuracies, and favoured the virtual view as it was more general, and could always be used to simulate the other two if the user preferred them. On that note there is also the work of Harrie et al. [48] who developed a fish-eye-view of map data. Fish-eye views are well known in virtual environment navigation since they distort the resolution of the viewed data so that the center is viewed at a higher resolution than the edges. This provides a high level of detail at the center (where the user is), and a lower level of detail as we move away from the user's location. Therefore at the same time the user is provided with a local and a global view of his/her surroundings, at the expense of having a distorted view of the environment. It has not yet been shown whether, in the context of real world navigation, users like fish-eye-view rendering of maps. Additional context-aware applications that assist users in their navigational tasks are not listed here since it is not the focus of this thesis.

There are however many human cognitive aspects that must be taken into consideration when one wants to assist someone navigates. Especially in the case of pedestrian navigation, where there are a few characteristics that are of particular importance. Suomela et al. [99] identified the skills required for a successful navigation as:

- the ability to locate oneself on a map,
- the ability to recognise the destination,
- the ability to plan and execute the route to the destination.

For example, in their work on the turn-by-turn WAP-based navigation system [100], Tarkiainen et al. mentioned that pedestrian users had difficulties to evaluate distances, especially large distances. They also realised that landmarks are essential in pedestrian navigation instructions. These observations are confirmed in a more formal study done by May et al. [69] where the following guidelines for pedestrian navigation were drawn:

1. Use landmarks in navigation instructions.
2. Distances and street names are not easily understood.
3. Offer redundant information for confirmation purposes.
4. Street names are perfectly acceptable to be used as confirmation information.
5. There is no need to provide the “next” instruction a long time in advance.

These recommendations are of high value, and might help in the generation of future geo-datasets oriented towards pedestrians. In a similar effort Corona and Winter [26] [25] proposed a unified ontology for pedestrian navigation. Their approach consisted of first finding the central concepts of pedestrian navigation. They then used the lexical tool *WordNet*¹ to obtain a categorization into families by abstraction, and

¹<http://www.sogsci.princeton.edu/~wn/>

hierarchical dependencies. They adopted the NaVigation Markup Language (NVML) as the structure of their ontology.

The general consensus is that landmarks are a critical component of any pedestrian navigation. But what is a landmark, and what makes a landmark good for navigational purposes? According to Sorrows and Hirtle: “landmarks in navigation serve a different purpose by identifying choice points where navigational decisions are made, identifying the origin and destination points, providing verification of route progress and influencing expectations, providing orientation cues for homing vectors and suggesting regional differentiating features” [95]. Their criteria for a good landmark are defined by its distinctiveness, its prominence, its accessibility, and its prototypicality. Vinson proposes that virtual environments follow the below suggestions in order to improve navigability [103]:

1. It is essential that the virtual environment contains several landmarks.
2. Include all five types of landmarks (path, edge, district, node, reference point) in your virtual environment.
3. Make your landmarks distinctive.
4. Use concrete objects, not abstract ones, for landmarks.
5. Landmarks should be visible at all navigable scales.
6. A landmark must be easy to distinguish from nearby objects and other landmarks.
7. The sides of a landmark must differ from each other (for orientation).
8. Landmark distinctiveness can be increased by placing other objects nearby.
9. Landmarks must carry a common element to distinguish them, as a group, from data objects.
10. Place landmarks on major paths and at path junctions.
11. Arrange paths and edges to form a grid.

12. Align the landmarks main axes with the path/edge grids main axes.
13. Align each landmark's main axes with those of the other landmarks.

Although it might not be possible for a designer to follow each single suggestion when defining her virtual environment (e.g. a pedestrian data visualisation), but each little bit done to help improve navigability of that environment will increase general user acceptance of the overall system. Finally, we report on the work of Hampe and Elias on landmark automatic extraction [47]. They developed a data-mining algorithm that identifies which entities can be used as landmarks for pedestrian navigational instructions using the German authoritative topographic cartographic information system (ATKIS) data set. Gartner [37] even suggests that entities identified as landmarks in the real world should also include proximity sensors. This would serve two purposes: 1- to provide location information to the user and, 2- to confirm whether the user is on track with the navigation instructions provided.

Mobile Cartography

Navigation entails that users build some sort of mental model of the space that they are traveling in. Maps and the field of cartography in general have always been helpful assistants in providing navigators with a better understanding of their environment. According to Steiner: “Cartography is the art, science and technology of making maps together with their study as scientific documents and works of art” [97]. A new sub field is now emerging: *Mobile Cartography* to help discover the various details in mobile map display/behaviour that makes them more useful to the eyes of their users. Reichenbacher says that: “mobile cartography deals with theories and technologies of dynamic cartographic visualisation of spatial data and its interactive use on portable devices anywhere and anytime under special consideration of the actual context and user characteristics” [86]. He claims that the user and his/her context need to be central in mobile cartography, which is akin to the example in route description given by Baus et al [10] (quoted previously at the beginning of this section).

A particular issue being at the center of a small controversy is the actual position on the device's screen of the user's current location. This information is normally

displayed as some kind of “You Are Here” (YAH) symbol. Some people argue that the environment (map) should be static, and the YAH symbol should move as the user travels in the space. The tourist guide application of Simcock et al. [94] is an example of this scheme. However in general it is difficult to adopt this scheme since in most cases it is impossible to fit on the screen the entire map data. The standard, which was for a long time to put the user at the center of the screen and update the screen according to the user new locations, is now being challenged. More and more people are arguing that the user should instead be located on the bottom third of the screen so that two thirds of the screen are dedicated to what is currently in front of the user, and only one third to what is behind (and thus cannot be seen) [38]. However not everyone agrees with this, like Suomela et al. [99], where in their *WalkMap* application they have put the YAH symbol at the bottom quarter mark of the screen, leaving three quarters of screen real estate for what is in front of the user. There are even some who argue that since a mobile navigation application is about helping the user understand her surrounding environment in order to get to her destination then the YAH symbol should literally be at the bottom of the screen [112]. In this way only what the user sees in front of him/her is displayed on the device’s screen. We argue that this aspect of visualisation should be decided by users themselves. For example, under different circumstances (context) a user might prefer to be placed at the center of the screen at certain times, and in the bottom third of the screen at other times.

Another interesting viewpoint coming from mobile cartography is that displayed maps should be rotated to fit the user’s current heading (head-up) vs. a North-up map which is never rotated. Gartner reports that the results of a study conducted by Radoczky shows that eighty-six percent of all participants preferred the head-up oriented map, as opposed to only nine percent who preferred the North-up map [37]. We note that the authoring system of Baillot et al. [6] as well as the REAL system of Baus et al. [10] used head-up maps. In their weather forecasting for ships Hartigan uses a north-up map, but argues that had the system supported it, they would have rather used a head-up map [49]. Not all is fine however with head-up maps as reported by Suomela et al. [99]. They have noticed that in their *WalkMap* system that users

were complaining of constantly rotating maps due to insignificant change in their heading. They solved that problem by requiring a change in user's heading of at least ten degrees in order to generate a new map rotation. Again we argue that the user's context should be used to determining whether a map needs to be rotated or not. We provide in Chapter 5 a study on different possible map rotation algorithms developed in *DynaMap* and give the results of a usability study measuring in which context is a given algorithm more relevant.

2.2.5 Geographical Information Systems

Navigating in the real world implies using real world data. Geographical information systems (GIS) is a mature field, which specializes in representing and analysing the geographical world in which we live in. Roger Tomlinson, known as the father of GIS [41], literally helped create the field of GIS in 1963 by leading the Canadian geographical information system (CGIS) project. Since then this field has progressed tremendously. New data structures and algorithms were invented to deal with the intrinsic requirements of spatial data. We are now better at storing, indexing, performing complex computations such as overlays and shortest path queries. Advances in a number of related fields like computational geometry, graph theory, and parallel computing is in part the reason why we can perform such complex operations efficiently. GIS, due to the increased availability of data, tend to be bulky, resource intensive, rich in features, and complex to operate. These have proven to be an obstacle in trying to bring GIS more accessible to the mainstream user.

Mobile GIS

With the advent of mobile ubiquitous and context-aware LBS, there was yet another reason to make GIS applications thinner and more accessible. This is not a straightforward process and many issues need to be resolved. One problem resides in the big gap between the large scale computer systems normally running GIS vs. the thin and mobile clients used by LBS. A decision has to be taken regarding the roles of each

entity. We know it is impossible to simply rely on the mobile devices to do everything; similarly the GIS servers, even with all of its processing power, cannot do all the work and simply use the LBS devices as dumb terminals. This would overtax the GIS servers, and would result in large delays while data is being transmitted back and forth. At the same time current systems must take into consideration the expected increase in processing power of mobile devices in the future. Perhaps a system where roles be easily moved between the GIS server and the thin client is the long term solution. Even for simple things such as the visualization of data, we realise that work must be done in order to overcome the lack of screen space. In that specific case Oviatt provides an interesting insight into the use of other types of interfaces than just graphical ones to interact with maps (e.g. speech recognition) [82]. We also refer to Virrantaus et al. [104] and to a lesser extent to Stockus et al. [98] for a more in-depth discussion on the problems in bridging the technological gap between GIS and mobile ubiquitous and context-aware LBS.

Applied Examples

Most location-based systems out there that display the user's location on some kind of map rely on GIS data. However, beyond this simple example of GIS-LBS collaboration, there are a number of applications that have achieved, with success, to bring more advanced GIS features to mobile clients. We first note the work of Hope et al. [52] where they have designed a mobile context-aware GIS for the utility industry in the UK. Their aim was to increase fault restoration response time of workers without compromising their safety. Their system makes use of an advanced GIS containing data on the utility resource network (e.g. electricity lines) that could be queried remotely by the workers. Another example of GIS-LBS collaboration is the mobile local marine weather forecasting of Hartigan [49]. Their system called the *Maritime Nowcasting Service* (MNS) consists of a full GIS marine weather forecasting lab, a wireless transmission mode (2G or satellite), and a collection of service subscribers (ships at sea) each equipped with thin clients. Based on the client location, MNS provides them with real-time up-to-date on-demand weather forecast and weather alarms for their area. Finally, in yet another example of GIS-LBS collaboration we introduce

the work of Baillot et al. [6] where this time a client device is used to acquire on-site geodata, and subsequently feed a bigger GIS system. Their mobile authoring of physical models system suffers from inaccuracies in their acquisition mechanism, but nonetheless the idea of using a LBS as a GIS data acquisition clients is interesting.

Geographical information systems still have a long way to go before they become fully mobile compliant, but the recent work shows that advances are made at a fast rate. More work needs to be done on standardizing GIS server databases access, and queries. Standardization is an engineering tool to facilitate exchange of expertise between system designers. This is the topic of our next section.

2.2.6 What's Next

The field of mobile ubiquitous context-aware systems is quite young and suffers from growing pains. At the moment most of the work published comes in the form of prototypes or new ideas worth considering. Slowly we see the light at the end of this sometimes incoherent tunnel, and progress is made to drive the field further than the “prototype” stage.

Applied Examples

The work of Fischmeister et al. [35] [34] is an example where a good system design was one of their primary objectives. They built a software agent architecture for LBS which is modifiable/adaptive, scalable and fault-tolerant. They used Java to implement their decentralized architecture in order to allow code mobility. Another interesting piece of work is the tour guide utility of Simcock et al. [94] which focuses on extensibility. They design their system so that users can easily modify the spatial data being used, and the tours being proposed. We are still far from automatic adjustment to the environment, but this is a step in the right direction. Finally we would like to point out the work of Yau and Karim [114] in designing a middleware for context-sensitive communication. Their concept was to extend the CORBA-ORB (from the Object Management Group (OMG)) to support multi-platform, across programming languages context-sensitive communication. It is unclear however at this point how

much current mobile devices can support CORBA code and an ORB with its heavy network protocol considering the cost of wireless communication. That aside, the adoption of an architecture like CORBA comes with the credibility of the OMG.

Software Agents

According to the American Heritage Dictionary, an agent is *One empowered to act for or represent another*. Although this definition makes sense when one thinks about travel agents or real estate agents, it fails to truly characterize software agents since the notion of “empowered” or “representative” is unclear in the realm of computer processes.

Software agents have also been described as an arrangement of one or more classes with a specific goal, where a class is an object-oriented programming concept. This concept known as Agent-Based Software Engineering [40], although interesting, is too restrictive to characterize software agents in general. The principles by which agents are defined may serve as a good metaphor whose purpose is to help software designers in their tasks.

We provide here the characteristics that are commonly found in software agents and that are the most widely agreed upon [113].

Autonomous To be autonomous means that the software agent can act on its own initiative.

Persistent A software agent is an entity that evolves over time. As opposed to traditional programs, a software agent does not have a fixed objective, which, once reached, denotes the termination of computation. A software agent has a persistent goal (e.g. finding good stock prices on the stock market, setting up meeting times for the user, etc).

Adaptive Some software agents may be adaptive in the sense that there can be behaviour changes as their environment changes. This concept borrows from the AI field, in particular algorithmic learning and reasoning.

Pro-Active Some software agents may take pro-active actions according to the current state of their environment. In other words these agents will take action in anticipation of incoming events.

Social A software agent is said to be social if it interacts in some ongoing fashion with entities in its environment. These entities for example could be other users or some other agents, etc.

Mobile This characteristic is truly optional. A mobile software agent is one that can move itself (behaviour and data) to other hosts, which could be the same, or some other platform.

With these characteristics in mind we envision what a software agent is. According to [113] a software agent is “a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives”. It is not necessarily acting or representing some user or some thing, although it may, but it is always acting on behalf of a persistent goal. It does so in a way that is autonomous, reactive, sometimes adaptive, sometimes pro-active, sometimes social, and optionally mobile. What differentiates a software agent from the average computer program is the objective it is trying to achieve. The objective of a software agent consists of an ongoing task which, if it were not for the presence of a software agent, would require intensive resource usage. We could say that a software agent is a computer program involved in a slave relationship towards its task. A good software agent is one who works towards its objective in a manner described by the six properties we mentioned above.

Software agents have been the focus of intense discussions as to whether they should be seen as being a breakthrough for the advancement of context-awareness. Reichenbacher says: “Future research in mobile cartography includes the tasks of automatic information search and retrieval supported by mobile intelligent agents” [86]. There are two projects where software agents were used to accomplish the task at hand. First, one of the earliest context-aware systems by Nagao and Katsuno is their work on “situation-aware personalized mobile agents” [75]. They used the Java based agent programming system and language (APSL) to create a platform of

agents representing users. These agents would be responsible for carrying tasks on behalf of their users, according to their user's current location. For example a user agent could perform queries on the library search engine on behalf on the user, and provide the results to that user when s/he gets to the library. Fischmeister et al. also used agents in designing the *MUSA-Shadow* agent platform for location-based systems [35] [34]. In their system they model the user world into a virtual world of agent anchor points corresponding to locations the user could go to. Once a user went to such a location, its assigned agent moved (using the code mobility features of java agents) to the anchor point corresponding to that location. From that anchor point the agent discovered the services offered at that location and through context-awareness on its user, registered only to the services deemed relevant. We note that so far the world of software agents is closely tied to the Java programming language. Indeed, an interesting agent platform known as Java Agent DEvelopment Framework (JADE) [83], and its mobile light weight extension: LEAP have achieved a comfort level in terms of developer acceptance and appreciation.

Software Usability

Finally we have observed that the one software engineering field that thrives in mobile ubiquitous context-awareness computing is definitely the one of software usability. The user experience being such an important focus, this comes with little surprise. On that note we list here some of the work where a usability study has been performed. First of all, the *GUIDE* system has been tested for different purposes, first from a user appreciation perspective [19], then from a comparison between active and passive context [20]. User perceived lost of control, and its occurrence in context-aware software was the center of the research of Barkhuss and Dey in [7]. The research conducted by Kaasinen arrives at similar results: too often in context-aware applications is the user taken out of the control loop [63]. Many authors have expressed their doubts about the claimed benefits of mobile ubiquitous context-aware computing. We can think of Cheverst et al. [18], Davies and Gellersen [27], as well as Erickson [32]. In terms of usability studies performed on current mobile ubiquitous and context-aware application, some results have been obtained. Brewster showed how users found that

sound could supplement graphics to make up for the lack of screen real estates in small displays [14], and along a similar line of thoughts Oviatt showed how more efficient were users in using multimodal interfaces (speech and graphics) to interact with dynamic maps [82]. Research conducted by Suomela et al. came to the conclusion that users did not prefer a perspective view to a map in a navigation context [99]. Finally a study on field experiments to evaluate mobile guides by Goodman et al. proposes a number of suggestions in order to obtain adequate usability results [45].

As shown in this chapter a lot of work has already been accomplished in the fields of mobile ubiquitous computing, context-awareness, and navigation. With the proliferation of devices and the advances in wireless communication technology, we are slowly entering the post-PC era. We have a better understanding of what defines user context and of the issues related to its integration in applications. We see repercussions of this new knowledge everywhere, including in navigation, where a number of new tools are being proposed. This has lead researchers to further study what information is more relevant when providing navigation instructions to a user in different settings (pedestrian, driver, at the mall, etc). Finally geographical information systems are becoming more light weight and thus more mobile. There are still a number of outstanding issues however, such as how to integrate together many of the solutions presented in this chapter. There is little concrete help offered to system designers, which often end up re-doing everything from scratch. We also have little knowledge about the usability of many of the mobile GIS solution proposed. In an attempt to answer some of these questions we introduce in the next chapter *DynaMap*, a prototype application that offers real-time map visualization services integrating the user context.

Chapter 3

THE DYNAMAP APPLICATION

Vast amount of research is carried out in areas related to mobile and ubiquitous computing. However, we observe that even though a lot of progress has been achieved in specific domains, there are still many issues and problems of incorporation and implementation of ubiquitous computing solutions. The way current context-aware systems are being implemented is still following the “prototype” model where issues are addressed on a case-by-case basis, saying little on how to solve them in a general setting [27].

We now have access to state of the art acquisition of location technology for both indoors and outdoors environments (see Section 2.2.3). We also have a much better understanding of how to model context and on how to integrate it into the user’s workflow in a way that minimizes usability problems (see Section 2.2.2). Finally we better understand the impact of working in a mobile environment with limitations on the available processing power, available memory, screen real estate, and communication bandwidth (see Section 2.2.1). However there has not been a formal statement made, to the best of our knowledge, on how to integrate these independent solutions into a fully featured mobile system’s design that works in the real world. We would like such design to be as generic as possible without losing its usefulness in the abstraction process, and also without compromising performance.

We introduce in this chapter the *DynaMap* application. This application provides to a pedestrian user personalized navigation services. We further constrained the domain to that of a student user in a campus environment. From this applied problem, we propose a system design that meets the requirements. We demonstrate, at the same time, how the proposed design is generic enough to be applied to other mobile ubiquitous and context-aware applications by addressing issues related to spatial data handling in a mobile environment. As a proof-of-concept we implemented our design and tested its functionalities with real users.

In this chapter we look at the *DynaMap* application under many angles, namely its history, its aim, its environment, its design, and its usage. We show how *DynaMap*'s conception consisted of many important design decisions, unveiling the true complexity hidden in the system. But first, in order to better understand the context around *DynaMap*, we start with a fictitious scenario where a student named Charlie uses an application like *DynaMap* in his daily routine.

3.1 One Scenario

Charlie is a first year student at Edulearn¹ University. He just moved into the city for his studies and is excited about his new life. There are many new things that Charlie must adjust to: the new city, the new campus, more advanced studies, new people, etc. To assist him, he decided to bring along with him his Personal Digital Assistant (PDA), which is equipped with a GPS.

He downloaded a special software system from the University's website which is aimed at easing the transition of new students to University life. The software system includes a map of the campus and a navigation tool, which helps students to reach the different buildings on campus. From time to time the software system accesses the Internet and updates itself with new information. For example, the software knows Charlie's class schedule. It reminds him of an upcoming assignment or of a class that got cancelled. When a book that he borrowed from the campus library is due back, the system sends a reminder to Charlie. The application also knows that he rents a

¹fictional name

room in the residences, and offers him information about services that are available to him, such as cafeteria hours and menu, study rooms, school sponsored events, etc.

Initially, the software enabled Charlie to be on time for classes. When Charlie needed to get to his next classroom, it provided him with navigation instructions overlaid over the displayed campus map. As Charlie gets more familiar with the campus, he relies on the system to inform him whether his usual path is blocked by some hazard or construction work (e.g. stairs closure in the winter time), and is presented with alternate routes. At night, when it is raining, or when it is cold out, the application provides him with navigational cues on how to avoid being outdoors by using the University tunnel system. Charlie is also impressed that the map changes according to the time of the day. For example, in the morning buildings with coffee shops are highlighted. Around noon, buildings that serve food are displayed using a different colour. And in the evening, the location of on-campus pubs and recreational areas are shown.

That's not to mention the personalization capabilities of the software. Charlie's friends have different ring tones for the various alarms of the applications (assignment deadlines, class cancellation, etc). Charlie found it confusing to have blinking items on the campus maps, so he opted to have highlighting by a colour scheme, which he finds visually pleasing. He provided his own icons for each of his classes, so they reflect his opinion about the course, or the professor teaching it. It became a bit of a competition among his friends to come up with the funniest icons for the classes they share. Charlie would not consider for a minute to stop using the application he downloaded. It is simply too useful to give up.

3.1.1 An application too good to be true?

Not too long ago such a scenario, as described in the previous section, would have been classified as a work of science fiction, or at least as embellished reality. However today such a system does not sound so esoteric anymore.

From a conceptual point of view we can recognise many models from this scenario. First of all there is a notion of ubiquity since the functionality of the system is

available at all times throughout the campus. The application always knows about changes in the campus' physical and logical layouts, as well as changes in the user's preferences or behaviour. It is also always there to provide assistance to Charlie. The argument for context-awareness is also quite explicit. We note how Charlie's current location, his travel habits, his class schedule, the time of the day, or even the current weather all have an influence on the behaviour of the system. Campus navigation is one of the features of the program, and the user's current location can be used as a starting point. That alone makes the application location-based. In addition, it also knows about Charlie's important places on campus such as the residences, his classrooms, the library, etc. Of course, to maintain such a dynamic information system, based on spatial properties of the campus, there must be a GIS in the backend. We also note that the application is personalized to Charlie. Someone else using Charlie's application on Charlie's PDA for the first time would find the behaviour unpredictable, user-unfriendly, or even useless. That could be solved if the user is abstracted out from the application via the notion of personal software agents. The agents, representing a user, would be responsible for moving themselves onto whichever applications their user is currently accessing. As we can see, the previously described scenario relates to most of the research fields we have touched in Section 2.2.

From a technical point of view, we can see that such an application is now feasible. First of all there is the communication component. We see that the system can work offline, as long as it had a chance to update itself at least once (upon download for example). However, in order to remain up-to-date, then a wireless network, such as the IEEE 802.11 standard [46] could do the job (since we are talking about a limited area consisting of the campus). From a location acquisition point of view, we know for a fact that GPS can interact with most PDAs nowadays, so this is not a problem. There is the issue of indoor location acquisition for which there are also a number of possibilities (see Section 2.2.3). Finally, in terms of power consumption, we note that now most PDAs can last a few days without recharging. In terms of processor load, then again, most PDAs nowadays have enough processor power for real-time 2D graphics, and shortest-path computations on limited data sets. So from a technical

point of view, the previously mentioned scenario is practically feasible.

From a software engineering point of view, we can see that the challenge resides in bridging the required proposed solution mentioned in Chapter 2 into one complete integrated system. In order to have a system that is modular, easily maintainable, scalable, and powerful, care must be taken in the design.

3.1.2 The emergence of DynaMap

It is after imagining many scenarios such as the one described previously (other scenarios include tourist guide application, collaborative fire-fighter units system, multi-tier meeting scheduler, etc) that we came up with the project of developing *DynaMap* as a research project. What was originally assigned as an honours project to further investigate the implementation issues of such ideas² quickly grew into the work presented in this document. The original applied problem was to implement a small navigation application to run on a PDA [42]. The application was to display the user always centered on the screen, and the map always rotated according to the user's orientation. Due to the number of ways in which this project could be achieved the work completed was mainly to do research on how to concretely solve this applied problem, such as finding the best hardware configuration.

At the time the chosen platform was the Casio Cassiopeia E-200 for its physical properties (storage capacity, processor power, colour display, and the number of available peripheral input slots) as well as for its advanced operating system (multi threading): Windows CE 3.0 from Microsoft Corporation. The chosen GPS was the eTrex Summit from Garmin due to its price and its incorporation of an electronic compass that provided orientation independently from location (see Figure 3.1 to see the initial hardware). After the hardware was chosen and purchased, a GPS connection was programmed, as well as the parser to extract the needed information from the GPS' output. The general multi-threaded design of the application was provided with little implementation however. A general description of future issues to be dealt with was also provided.

²The scenarios and proposed ideas were the result of personal discussions between members of the IMA research group

Figure 3.1: Initial Hardware used by *DynaMap*

After the completion of the honours project we decided to extend the work into a research project. The work consisted of implementing the basic layers of the navigation application. The GPS connection and parser were incorporated into the general PDA implementation of what was then called *The DynamicMap* application. The necessary threads were created, and a communication system between them implemented. A JPEG-based map visualisation component was implemented as well as the software needed to encode geographical coordinates into JPEG images (see Figure 3.2). Finally, we implemented the real-time updating of the map so that it is always centered on the user's current location and always rotated according to the user's current orientation. See Figure 3.3 for a screen shot of what the original *DynamicMap* application looked like.

The main objective of *DynamicMap* was to prove that such an application could run on a small PDA in a way this is acceptable from a user point of view in terms of performance. We will show in the rest of this document how we improved and extended *DynamicMap* into *DynaMap* by re-designing it in a way that is generic and reusable while still maintaining the performance accomplishments. The rest of this chapter provides the objectives of *DynaMap*, shows the framework surrounding its development, provides a number of requirements that must be met by *DynaMap*, goes through some of the most important design decisions, and finally concludes with

3.2 Motivations

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 3.3: Screenshot of the Initial DynamicMap Application

ubiquitous location-based information, in a user's context sensitive way.

The main goal of the *DynaMap* project is to design and implement a system that:

- solves an applied spatial problem (map visualization and navigation) in a mobile environment (outdoors),
- incorporates location based context,
- proposes a framework that supports real-time ubiquitous services,
- allows for a general user context modeling within the system,
- offers pedestrian navigation visualisation services.

There are many sub-objectives inter-related to the main ones which we have just listed. From these objectives we derived a number of requirements which can be found in Section 3.4.

These objectives were dictated from the needs of a larger scheme, for which *DynaMap* is a sub-project: *The Intelligent Map Agents* (IMA). The IMA project provides the framework behind *DynaMap*, and this framework is the topic of the next section.

3.3 The IMA framework

DynaMap is a part of a bigger project based on the notion of intelligent map agents. This undertaking consists of establishing a global framework in which multiple GIS agents would be able to assist users in given spatially defined tasks. The framework is an actual research project carried out at Carleton University as part of the *Intelligent Map Agent* (IMA) research project [89]. We give a brief overview of the IMA architecture and how *DynaMap* is integrated into the system. We will first begin by introducing the framework, then describe “intelligent map agents”. We provide the aims and the benefits of having such a framework, and conclude by showing how *DynaMap* contributes to the development of the framework.

One observation that we made in Section 2.2.5 on the nature of GIS was how context-aware applications would benefit from the power of decision making tools provided by GIS. The greatest difficulty in this case, both from a technological and a theoretical point of view, lies in the big gap that exists between the large scale computer systems normally running GIS vs. the thin and mobile clients common in context-aware computing. In an attempt to find solutions to these problems, the IMA research group proposed the IMA framework. The purpose of the framework is to bring to mobile users, via their mobile devices, a class of applications whose degree of complexity is higher than what the computing capacity of such devices would normally be able to accomplish alone.

The IMA group used the notion of intelligent software agents to look at ways at bridging the two worlds that are GIS and mobile ubiquitous context-aware computing. The conceived solution entails a framework where lives a collection of intelligent agents, each representing its user in his/her various tasks. Services would be distributed across a networked infrastructure and agents would automatically find out about them. The agents could then use the services as they see necessary.

3.3.1 Intelligent Map Agents

As expected, an intelligent map agent is also a software agent (refer to Section 2.2.6 for a discussion on software agents). Therefore an intelligent map agent is characterized

by the same six attributes defining software agents: autonomous, reactive, adaptive, pro-active, social, and mobile.

However we want to be more specific than that so that we can show that intelligent map agents define a family, a class of their own within the encompassing set of all software agents.

A crucial observation is to notice that intelligent map agents are all focused on their users. That is they, like the dictionary definition previously given in Section 2.2.6, “are empowered to act or represent” the user. That means that there are no such things as intelligent map agents that are simply goal-oriented, without at least the notion of a human user.

With that in mind we can now start describing the common attributes that characterize the environment of all intelligent map agents. The elements that are part of a typical intelligent map agent can be seen in Figure 3.4.

Figure 3.4: Intelligent Map Agent Environment

User The user is central to an intelligent map agent. It is the entity that the agent is representing and acting for.

Geographic Data Intelligent map agents all have in common the fact that they involve geographic data. This is true for all GIS applications whether they are agents or not.

Data Sources Depending on the application, other data sources may be required. These data sources can be anything from dedicated databases to the plain world-wide-web. Legacy systems are also part of this category.

Network All intelligent map agents have to interact with other agents or other system components. This is done by using a computer network, e.g. the Internet. With the emergence of wireless networks more and more IMAs are bound to be in a wireless network environment.

Other Agents (optionally) In some cases an intelligent map agent may require to interact with other software agents. This depends on the actual application.

We can now start looking at what internally differentiates an IMA from the more generic software agent. At the core of our vision of IMAs lies this concept of relevance. According to [96] relevance is defined as follows:

An assumption is relevant in a context if and only if it has some contextual effect in that context.

Extent condition 1: *an assumption is relevant in a context to the extent that its contextual effects in this context are large.*

Extent condition 2: *an assumption is relevant in a context to the extent that the effort required to process it in this context is small.*

We notice that the notion of relevance was first introduced in the definition of context-awareness given by Dey [29] (see Section 2.2.2). An IMA is therefore a specialized software agent for two reasons:

- it revolves around a spatially-related task,
- it is user context-aware.

We can now define more concretely an intelligent map agent as: “a software agent based on spatial data that acts relevantly on behalf of its user”. In addition, the intelligent map agent’s environment consists of networks, possibly other data sources, and other intelligent map agents.

3.3.2 Framework's Aims and Benefits

The IMA research aims to provide a framework to ease the development of mobile ubiquitous and context-aware GIS applications. Through such framework the intelligent map agents have access to a range of available services, either offered by the framework itself, or by other agents available via the communication and sharing services provided by the framework. The development of such framework is currently under way and many design decisions have been taken. One decision is to divide it into a three-tier system: backend, service, and client. We can see an illustration of the framework three-tier view in Figure 3.5⁵.

The direct advantage of the framework is to solve many common issues of mobile GIS applications. It facilitates the development process of mobile GIS applications that make use of the framework services. The second type of benefits comes from the fact that when designing applications, developers have to separate the “services” that are offered, from the “clients” that use them. Originally these services were aimed at the clients in one context, but the framework makes those services also available to any agent who requires them. Each agent running on the IMA framework is therefore a possible client to all services, giving potential features to applications that were not part of their original design.

The IMA framework, having to provide core services to intelligent map agents, is based on the JADE platform for software agents [83]. The JADE platform provides an environment for Java agents to interact with one another, and to move from one place to another. It also has an extension, LEAP, which allows agents to run on smaller mobile devices. For these reasons, the IMA framework includes an extension of the JADE platform with an emphasis towards services aimed at mobile GIS applications (spatial data transfers, location acquisition, etc). We can see an illustration of the framework architecture model in Figure 3.6⁵.

3.3.3 *DynaMap*'s role

The role of *DynaMap* within the IMA framework is to demonstrate some of the possibilities of the framework. It enables us to envision issues that are to be expected

in the development of the framework as well as services that it should provide.

Therefore *DynaMap* is part of a bigger system. Its design is to be reused by future framework applications. Its architecture will serve as a prototype, and if it happens to meet the requirements it was designed for, it will serve as a model for both the architecture of the framework, and that of future applications.

More specifically *DynaMap* mainly fits into the client-tier of the IMA framework, as can be seen in Figure 3.6 and 3.5⁵. Informally, *DynaMap*'s roles are to gather the information relevant to the user, and to display it in the most useful way. It is therefore a client of services and data sources offered by the framework. However certain components of *DynaMap* belong to the service tier (shortest path computation, user context information sharing, etc) and to the data tier (spatial data server, context information storage, etc). Its main contribution to the framework is to serve as a model to future clients to be developed. Being developed in parallel with the framework, *DynaMap* allowed the IMA group to better understand what was expected in all of the framework's three tiers. It allowed better define and understand the roles of each layer in the overall system. The next step forward from a framework point of view is to develop more clients, services, and also data sources in order to more exhaustively show the benefits of the framework.

⁵Source: The IMA research group internal publications

Figure 3.5: IMA Framework - Layered View⁵ Figure 3.6: IMA Framework - Three Tier View⁵

3.4 Requirements

Requirements' gathering is the first step when designing a new system such as *DynaMap*. We therefore start by providing the requirements that influenced the design of *DynaMap* (including the objectives defined in Section 3.2). We first provide them in a concise tabular form to facilitate traceability back to the requirements when further explaining the design decisions later on. Additional explanation is provided at the end of this section.

Reference	Name	Description
REQ#1	GIS	Solves an applied spatial problem (map visualization and navigation) in a mobile environment (outdoors).
REQ#2	LBS	Incorporates location based context.
REQ#3	RT Architecture	Proposes a framework that supports real-time ubiquitous services.
REQ#4	Context	Allows for a general user context modeling within the system.
REQ#5	Navigation	Offers pedestrian navigation visualisation services.
REQ#6	O/S Independent	Provides a system architecture that is platform generic (including mobile platforms).
REQ#7	Modular	Provides a system design based on modules that can be easily extended modified and maintained.
REQ#8	Performance	Provides information to the user in real-time.
REQ#9	Robustness	Shall gracefully handle unforeseen situations.
REQ#10	Hardware	Requires hardware components that are widely available.

REQ#11	Software	Reuses acclaimed solutions. Do not reinvent the wheel.
REQ#12	Abstraction	Favours abstraction of concepts to facilitate code reuse.
REQ#13	Standard	Follows approved standards when available.
REQ#14	Framework	Shall be compatible with the IMA framework.
REQ#15	User	Shall never compromise the user experience of the system.

We note that the requirements provided here are mainly high-level. These are the requirements that relate to any generic mobile ubiquitous and context-aware GIS application. They capture the defining lines for these types of applications.

The requirements are grouped into subcategories. For example, requirements REQ#1 to REQ#5 consist of the system objectives as defined in section REQ#6. REQ#7 to REQ#9 are the non-functional requirements. REQ#10 to REQ#14 are all associated with design and code reuse, and finally REQ#15 ensures that in the end, what matters is the user experience.

As we will see, these requirements greatly influenced the design of *DynaMap*, which is laid out in the next section.

3.5 Design

Any application which attempts to integrate a multitude of different technologies and concepts such as *DynaMap* requires a well thought-out design. The fact that *DynaMap* is to be held as a prototype and a model to follow in the context of the IMA framework calls for an even more thorough design process. In the rest of this section we first review the decisions behind the choice of tools used for the development. We then discuss the design decisions taken for *DynaMap*. We explain the need for a multi-threaded application as well as how to accomplish it. We describe the

abstraction process used and the separation of the code into modules. We present each of the components of the system and describe their role and the reason why we decided to have them. We provide the underlying mechanism used to facilitate GUI independence from the rest of the system. Finally we show how our design works in a real-life environment by describing the usage of *DynaMap* by actual users.

3.5.1 The choice of development tools

Determining the development tools may have a profound impact on the development of a new system. We take the opportunity and describe two decisions taken regarding the development tools used for *DynaMap*: the chosen hardware, and the chosen integrated development environment (IDE).

Hardware

We have already mentioned in section 3.1.2 what PDA (Casio Cassiopeia E-200), and what GPS and electronic compass (Garmin eTrex Summit) were initially chosen to allow us to begin the development of the *DynamicMap* application. However, a number of issues required us to make changes to that initial selection.

First of all came the omnipresent problem of obsolescence. The time interval between releases of new and improved mobile devices into the mass market is notably short. In less than a year our PDA was obsolete in terms of storage capacity and processor power. That became a direct consequence of the second issue: lack of service from the manufacturer. Our PDA suffered from a few technical issues such as the incompatibility of the manufacturer provided serial cable for the PDA and the lack of a functioning backlight. The manufacturer decided not to address those issues and instead oriented their focus towards producing newer and more advanced devices by discontinuing our PDA. Finally, we realised that due to the un-usability of the serial port of our PDA it was becoming impossible to access the GPS and, at the same time, provide a network connection. We therefore decided to obtain a new PDA.

In order to reduce the impact on development we opted to minimise change.

We therefore chose the Hewlett-Packard iPAQ H5550 running the same operating system as before, Microsoft PocketPC^(TM), only in a newer version. In addition to having an established market presence this PDA also came with an embedded IEEE 802.11b [46] network card. We then only had to connect the GPS to it, which we were quickly able to do. Finally, the extra storage space available and the faster processor reduced the limitations of the previous PDA. Figure 3.7 illustrates the current hardware structure of *DynaMap*. We believe that this hardware configuration meets the popular availability requirement, REQ#10, defined in section 3.4.

Figure 3.7: DynaMap Hardware Structure

Development Environment

The choice of the right development environment has a long lasting impact on any software project comparable in scale to *DynaMap*. Many factors come into play in such decision, among others we can think of the intended operating system, the background of the development team, and obviously the particularities of the project in question. In our case, we knew that the operating system would consist, in the near

future, of the Microsoft PocketPC^(TM), and that the development team was proficient in both C++ and Java in terms of programming language. The particularities of *DynaMap* revolves around REQ#15: the user experience, which is somewhat related to REQ#8: real-time behaviour.

The challenge was therefore to decide between a Java and a C++ development environment. The arguments in favour of Java are its cross-platform support, its rapid development turnaround, its built-in code robustness enforcement, and its acceptance by the academic community as the programming language of research. However, in the world of mobile devices not all of these claimed advantages of Java are true, at least not currently. For example, many Java virtual machines (JVM) for mobile devices are platform centric due to the huge differences between different devices. Also few vendors offer a complete Java programming environment tailored specifically for mobile devices, leaving the programmer with more work to write, debug, and distribute his/her code onto the device of his choice. Finally, one of the main issues with Java programs is that they tend to be slower due to the introduction of another layer in the process, the virtual machine (JVM), as well as the occurrence of the garbage collector which operate outside the scope of the developer.

On the other hand, if care is taken, C++ code can be made so that it compiles with little modifications onto different platforms. The fact that C++ code is compiled into machine level instructions eliminates any extra layer in the process, and ensures the fastest possible speed of execution. We also note the absence of any garbage collector in C++, leaving the developer with complete control over how, and when resources are allocated and freed. Of course this comes at the cost of the programmer's added responsibility of writing sustainable code. Finally, it occurs that Microsoft favoured code reuse when they offered the PocketPC^(TM) platform in the sense that they took their Windows operating system and scaled it down to what could be supported by smaller devices. This resulted in the availability of the usual Microsoft C++ libraries to help the developer faster implement and better integrate their applications. It also meant that development in C++ on the PocketPC^(TM) platform was possible through the eMbedded^(TM) Visual C++ IDE, which Microsoft provides free of charge. This IDE allows rapid development through its provided simulators, its direct connections

with the devices, and its impressive range of debugging services and documentation.

Due to all of these reasons, and due to the confidence of the development team in their ability to write concise, clear, and sustainable C++ code, it was decided that the development of *DynaMap* would be done in C++ with the help of the eMbedded^(TAL) Visual C++ IDE from Microsoft.

3.5.2 The *DynaMap* threads

We decided to partition *DynaMap* into multiple threads. The fact that *DynaMap* relies on many independent systems (GPS, Spatial data servers, sensors, etc.) makes the prospect of using a single thread of execution worrisome. What would happen if the entire execution of *DynaMap* had to be halted while it waits for the spatial data server to respond with its reply to a data query? This scenario seems intolerable from a user's perspective. Indeed, requirements REQ#3, REQ#8, and REQ#15 respectively on RT architecture, performance, and user experience, all speak in favour of the use of multiple threads. One of *DynaMap*'s tasks is to display to the user a map centered and rotated in real-time according to the current user location and orientation. This real-time requirement in particular speaks against any possibility of a system that hangs, or is slow to respond to events, whereas the performance requirement is more likely to be met if independent processes are allowed to run separately. Finally the user experience can only be improved when one reduces the impact of a failure of one independent process onto the other processes. Since *DynaMap* is to serve as a model for such systems in the IMA framework, its design should encompass established solutions to such problems, such as multi-threading.

The task of multi-threading is not the same at the one of finding objects and defining classes. Threads should not overly and unduly be created. Threading in *DynaMap* is done by separation of concerns. The concerns in this case are as follows: GUI, server(s), GPS, context sensor(s), application's logic. These entities have all in common that they act independently from each other. For example, the role of the GUI is to serve as an input/output mechanism to the user, quite independently from the application's logic execution, or the GPS. By creating a thread for each of

these entities we model that kind of independence in the computing process. For an illustration of the current threads available in *DynaMap* refer to Figure 3.8.

Figure 3.8: DynaMap Threads

The general idea behind the multi-threading design of *DynaMap* is to have a master thread: the central controller, which contains the application's logic, to ensure that the overall flow of events goes well. Through the messaging feature of PocketPC^(TM) threads, the central controller has a well-defined message based protocol with each of the other threads (displayed in Figure 3.8 as the floating text around the thread objects). Most of the thread messages are generic and are used as meeting points (data exchange) between two threads. For example from a thread point of view the GPS does only two things (see Figure 3.8): 1- some hand-shaking at thread creation time with the central controller (GIP_INITIALIZED message), and 2- It notifies the central controller when it has new location or orientation information available (GIP_DATAAVAILABLE message). It is however possible to define application specific messages to support additional functionality of a component. For example in the context of a usability lab study (refer to Section 5.3.2) we have integrated the

GET_LAT_LON message between the GUI and the central controller (Figure 3.8). This message is called whenever the user points and holds the stylus somewhere on the displayed map for a few seconds. The GUI then sends this message to the central controller with the screen coordinates of the query. The central controller sends back the LAT_LON_ANSWER message containing the latitude and longitude coordinates of the point on the map, which the GUI can then display to the user. This functionality was added for debugging purposes.

This multi-threading architecture works well in *DynaMap*. Threads are kept simple and are used for the sole purpose of decoupling autonomous processes. The only two threads in this system which are not allowed to hang are naturally the central controller, and also the GUI thread. The former is to avoid a complete system freeze, while the latter is to always provide response to the user and therefore improve the user experience (REQ#15). Indeed, the central controller role is to respond to messages sent by the other threads, either by taking a few deterministic steps, or by dispatching work to another thread. In this way we ensure that the central controller does not become a bottleneck in terms of performance. This architecture also allows the central controller to detect when another thread is not responding, using a timeout value, and to act accordingly (restart the thread, or notify the user). This makes the system more robust (REQ#9). We enforce a safe behaviour in our threads by making a clear distinction between the data accessed by a single thread and the data used in thread communication. The latter is always accessed via some mutual exclusivity mechanism such as semaphores or mutexes. Finally it is often claimed that debugging a multi-threaded system is more complicated. This is true and care has to be taken during development to keep the thread related code clean and simple. However, if multi-threaded debugging is supported by the IDE, such as with the Microsoft eMbedded^(TM) Visual C++ IDE, then debugging is not as much of an issue anymore.

3.5.3 Modules and Abstraction Classes in *DynaMap*

In *DynaMap* we decided to constrain the code into partitions of well-defined objectives (modules). This has resulted in a code structure with components of specific roles. For example, the GUI module of *DynaMap* is solely responsible for relaying information between the user and the application's logic (through graphical display, and input devices), whereas the GPS (location acquisition) component is solely responsible for obtaining the data from the location provider, parsing it, and sending the relevant bits to the application's logic. The other components of *DynaMap* (see Figure 3.9) are the Server, responsible for the data acquisition process, the Context Manager, responsible for collecting and pre-processing the user context, and the Central Controller which includes the application logics and manage the flow of events in *DynaMap*.

Figure 3.9: Code Modules of *DynaMap*

The advantage of using modules is to have grouped together tasks of a similar nature. For example, several problems need to be dealt with in extracting location information from the GPS. We can think of the establishment of a communication channel, the reception of the location information, the interpretation of that information, etc. If for some reason it was decided that a new location acquisition technology

should be used, then the code affected by that change is found all in one module. As long as the inter-module communication protocols are respected, no impact is felt on the other modules. Finally, the impact of such change is further diminished within the module itself if the designers took the time to abstract out the different concepts found in that module.

In our case, we are faced with the *DynaMap* paradox when it comes to abstracting out the different concepts. On one side we have an application that should serve as a prototype/model for other similar application development. This is a strong argument in favour of abstraction. On the other hand we have a well-defined application whose concrete objectives are to provide user's context-aware navigation services and whose real-time requirements are strongly established. This calls for a system with optimized code and therefore leaves little room for generality and reuse via abstraction. Therefore the challenge in the design of *DynaMap* was to find the right level of abstraction in order to meet all of the objectives equally. We decided to abstract out only those parts of the system which offer an important architectural concept, and to otherwise favour optimized code development in the other parts in order to maintain a good level of real-time responsiveness.

We refer to Figure 3.10 as an illustration of the concepts that were abstracted out in *DynaMap*. We use the UML 2.0 class diagram notation, i.e. the filled diamond means "Composition", the empty triangle with the solid line means "Generalization" (inheritance), and the empty triangle with the dotted line means "Realization" [81]. The diagram shows, using inheritance and abstract interfaces, which components are abstracted out in the code. The names shown are the actual class names used in the code. Finally the doubled-sided arrows indicate thread communications.

As we can see in Figure 3.10, the abstraction process comes right after the step of code modularity. This comes from the fact that abstraction in *DynaMap* is done in favour of code reuse, therefore it is tightly related to the different concepts defined in *DynaMap*. The idea shown in Figure 3.10 is that each component of *DynaMap* can be abstracted out. For example, the central controller is where the application's logic resides, the GUI is where interaction with the user is performed, the GPS is where location information is collected, the spatial data server is where external data

Figure 3.10: Modularity and Abstraction Diagram of *DynaMap*

is queried, and finally the context manager is where the user's related information is assembled. All of these abstracted out concepts could be reused in other applications. Their structure, common across applications, is preserved via this abstraction step. These components are, in the code, made abstract which means that these classes contain no implementation, simply a definition of concepts. To obtain a working system, these components need to be realised (via an inherited implementation in C++, or interfaces in Java), which is what was done in *DynaMap*. For example the application's logic, to offer navigational services, has been coded in the central controller and the GUI behaviour, supported by the PocketPC^(TM) platform, has also been integrated. We therefore believe that a small one-step abstraction price had to be paid in *DynaMap* with little impact on performance, but to the huge benefit of a clear design and increased code reusability.

3.5.4 The components of *DynaMap*

We now describe in more details the components that make up *DynaMap*. As introduced in the previous section these components are the Central Controller, the GPS, the Context Manager, the Data Sources Controller (Server), and the GUI. We can see in Figure 3.11 the different layers involved in *DynaMap*. The top layer represents the end-units of the system, that is the GPS, the PDA, the spatial data server, and the sensors. Directly underneath are all of the *DynaMap* components, working together to provide context-aware navigational assistance to the user. These components use the PocketPC^(TM) messaging service to communicate among each other. This service offers a reliable way for independent entities within the PDA to send data to each other, and therefore becomes the third layer of our system termed “Internal Communication”. Finally, there is the External Communication layer consisting of the Serial BUS used to communicate with the GPS and of the TCP/IP network layer to communicate with both the spatial data server as well as the various context sensors. *DynaMap* performs its work at different level throughout these layers. We will now see what the purpose of each component is and how they fulfill it.

Figure 3.11: Layered View of the Components of *DynaMap*

The Central Controller

The central controller contains the application’s logic. As its name implies, it is also the central entity from where decisions are taken and where work is dispatched to the other entities of the system (Figure 3.12).

Figure 3.12: The *DynaMap* Central Controller State Diagram

The central controller’s flow of events consists of three main states: waiting for context, waiting for external spatial data, and waiting for the exchange of information with the GUI. In the initial state, context, the controller waits for new contextual information such as new user’s location or a change in the weather forecast for example. Once such an event occurs, the controller analyses the incoming information and acts upon it. When the controller requires more spatial data to complete its tasks, it requests it from the data source proxy and then waits for that data to arrive. Finally, using the newly acquired spatial data, it performs its last computation step, prepares a new buffer for the GUI, and sends it off.

The central controller is of course tailored for the specific requirements of *DynaMap*. However, the state diagram of Figure 3.12 is fairly general and can easily

be applied to other mobile context-aware GIS applications. In that diagram, the application's logic of *DynaMap* is represented by the action taken in between each state, i.e. what it does with the acquired context, what information it requests from spatial data sources, and how it prepares the display for the GUI. However the states themselves are generic and offer a general way of handling context, spatial data, and the other components of the system. In the Central Controller module these states translate into function calls of the same names. When introducing new application the system designer only has to fill these functions with the application's logic code.

The GPS

The GPS component, also known as the geographical information provider (GIP) component provides the user's location and orientation information to the central controller. It does so in the form of latitude, longitude, and a heading value. It abstracts out to the central controller which location acquisition technology is used to capture these values. In the case of *DynaMap*, we decided to use a GPS with an embedded electronic compass. Therefore the GIP component does the following two tasks: 1- set up the serial connection to the GPS, and 2- parse the incoming data, which follows the NMEA 0183 encoding standard. We can see these steps in the form of a state diagram in Figure 3.13.

We made a conscious decision to treat geographical information separately from any other contextual information. The reason for doing so is because the user location is central to *DynaMap* and all other mobile context-aware GIS applications for which it may serve as a prototype. By treating this information separately from the other user's context information we are better able to optimize for performance, specifically addressing requirement REQ#8 on performance. Also from the experience of others who developed similar applications [10] [19] [94], we realised the importance of geographical data and of the need to model it properly, which made us create the GIP component.

Figure 3.13: The *DynaMap* GIP State Diagram

The Context Manager

The context management component is responsible for collecting contextual information as it becomes available, and of notifying the central controller when new context information has arrived. We refer to Chapter 4 for more information on this component as the entire chapter is devoted to context-awareness in *DynaMap*.

The Data Sources Controller

The data source component performs queries to external data repositories on behalf of the central controller. Again, an immediate benefit of having the data source component is that changes in the external data repository do not require a rewrite of the central controller. By external data repository we mean anything that the mobile context-aware GIS application might need. In the case of *DynaMap* this consists only

of spatial data servers that could be accessible wirelessly throughout the campus of Carleton University. Since no such server was available in the IMA framework at the time, we decided to simulate one from within the PDA itself. We decided to run the server directly on the PDA since there was no extensive wireless network available throughout the entire campus of Carleton University. Figure 3.14 shows a diagram of the client/server architecture in *DynaMap*.

Figure 3.14: The *DynaMap* Client/Server Architecture

We opted to use HTTP to exchange the data between *DynaMap* and the external server (Figure 3.14). The simplicity of the protocol and the fact that it has been used extensively are the main reasons behind this decision. Other protocols where

considered such as CORBA from the OMG, and Jini from Java. However none of these other protocols where as simple and lightweight as HTTP, which are important features in a mobile environment. However as the development of more data sources become available we should revisit the decision taken behind the protocol used for communication. There are also a number of important factors involved in the decision of which communication protocol to use, such as how it handles security, or how queries are formulated and answers are returned. For this iteration of *DynaMap* we decided not to address security. Queries were sent to the data source by simply expressing via latitude/longitude coordinates the region of interest, which was returned by the server in the form of a jpeg image. The steps illustrated in Figure 3.14 worked well with our simulated server and are expected to behave similarly in a real environment. The steps are general and are reusable in future applications. Of special interest we note that the central controller maintains a cache of previously acquired spatial data. This reduces the number of times that data need to be retrieved from external sources. Similarly, the central controller will request more data than what it really needs in the hope that the extra data will become useful in the near future, hence reducing the number of requests to the data server, and providing a more seamless visualization to the user. This was inspired from the examples set by the work of Ren and Dunham on spatial data caching [87] and the work of Kubach and Rothermel on prefetching [66].

The Flexibility of the GUI Component

As opposed to most PC-based software, little can be assumed about the capabilities of the device to which an application like *DynaMap* might be ported to. Indeed, the slightest notion that developers have taken for granted, such as minimum display size, has to be reconsidered when modeling for mobile devices.

However, physical constraints are only one of the concerns in designing the GUI outline of *DynaMap*. The way in which the system has been modularized and abstracted out brings another issue: the one of sharing system-specific information with the user without breaching abstraction rules. How does the GPS component ask the user about the various connection details to the GPS (COM port, baud rate, etc.)? It

is tempting for a programmer to hardcode these artifacts into the GUI and therefore to solve the issue right away, but this practice is a direct violation of the principle of separation of concerns and abstraction. What would happen if a new location acquisition mechanism were used instead of using GPS? The new location module most likely would not care about COM ports and baud rates. Therefore, in the above scenario the GUI would require to be changed including the way in which the data is exchanged to and from it. This GUI model fails to constrain changes to the modules where they occur. We opted not to follow that route and instead propose a GUI model which preserves separation of concerns, and therefore evolves well with change.

Our solution resides in the adoption of a user interface description language. That technique in itself is not new. In fact Fischmeister introduced a similar system in the *MUSA-Shadow* system [35], and there also exists the User Interface Markup Language (UIML) as a standard way of encoding UI specifications [24]. We found these solutions to be not easily obtainable (as in the case of Fischmeister), or quite broad in their objectives (as in the case with UIML). In all cases their integration requires intense programming effort and little support is offered. Since the cost of using these solutions was quite high, we opted to develop our own language, tailored for a system like *DynaMap* and optimized for performance and efficiency. Indeed our solution not only focuses on the description of what is needed from the user (in a way that can be efficiently rendered), but also comes with a transfer protocol that facilitates the communication between components. Refer to Figure 3.15 for an illustration of the various steps involved in the form of a message sequence chart.

The flow is as follows: the central controller, upon initialisation, is aware that each component might require user provided information to function properly. It therefore asks each component for what they need to know from the user, which is returned by the components in XML format (See appendix B for the full specifications). The central controller then passes the aggregation of these documents to the GUI component, which can render them into familiar interfaces for the user. Whenever the user provides new information regarding these settings, the central controller is notified with the changes in question, which is then forwarded to the relevant component. The only notion common to all components is the establishment of a communication

Figure 3.15: Flow of Events in the GUI Abstraction Process

channel for user settings transfers. The semantics of those user settings is confined to the component that needs them. In this way separation of concerns is preserved with no obstruction to both modularity and abstraction. All of which comes to the low cost of first encoding these settings into a format which is understood from a rendering point of view by the GUI.

Finally the fact that we limit our approach to the transfer between modules of user settings allows us to keep the GUI specification simple. This is different from UIML whose goal is to be able to describe everything the GUI can do. We limit the exchanged data to be of simple types: integer, text, float, etc (refer to Appendix B). Since everything can be expressed via these simple data types we provide a general way of specifying user specified settings no matter what the application is. However

we do so efficiently because keeping our approach simple also means that less time is used encoding/decoding the GUI specifications and therefore more time is spent performing the core tasks of the application.

3.5.5 Raster versus vector spatial data

We terminate the discussion on the design of *DynaMap* with the slightly controversial decision to only support raster based spatial data during this development phase. We first explain the two types of spatial data available nowadays: raster and vector based.

Figure 3.16: Raster vs. Vector Data

Raster-based data is analogous to multi-dimensional arrays, where each array cell represents a location in the geographical domain. Usually a numeric value is stored in each cell, representing a certain property at that location. For example, one could store in each cell the height above sea level, making this data set into what is known as a digital elevation model (DEM). In recent history we have seen an enormous increase in the number of raster-based data sets that are available. The invention of new technologies such as digital cameras and data collecting satellites, whose output is in raster format, provides an explanation for such proliferation of raster-based data.

That being said, raster-based data is not perfect. One problem is the high amount of memory required to store data in such format. For example, many adjacent cells in a raster may store the same value. One way of reducing the memory requirement is to merge all these cells together into one object and to assign only once this value to that object. That new object can now be described by the coordinates of the vertices that

define it. We call such data representation vector-based. Vector data can therefore be points, line segments, polygons, polyhedral, etc. Another immediate advantage of using this data format is in the possibility to attach topology information to the data. For example, we can now maintain with each line segment representing, say, a road, the other roads that such line segment share an intersection with. Such type of information was difficult to store with raster-based data since objects such as roads are implicitly stored and cannot be directly indexed. Figure 3.16 shows the same data stored in a) raster format and b) vector format.

The main reason to mainly support raster based data in *DynaMap* is simplicity. Raster based data is conceptually simpler than vector data (it is also less powerful). It was also easier to obtain raster based spatial data in a standard format than it was to obtain vector based spatial data. However we decided that in the ideal scenario an application like *DynaMap* must be able to support any kind of spatial data formats. We therefore abstracted out the various operations that DynaMap required to be executed on the spatial data used and theoretically any format supporting the *DynaMap* imposed API will work. To this day we only tested *DynaMap* with raster-based data for the reasons mentioned above but it should be possible to use vector based data (or any other format for that matter) with little or no code modifications to the central controller.

As a final note we would like to point out that some small amount of vector data is being used in *DynaMap* in order to test our user's context management system (which is the topic of section 4). The data was however self-collected and self-encoded in order to satisfy our needs in terms of level of details, stored attributes, and topological relationships.

3.6 End-Product

The design of DynaMap ensures its viability and fulfillment of our objectives. The result of the design decisions is a system that is sound in terms of its real-time architecture (REQ#3, REQ#8), and that is easy to develop due to the support provided by the IDE. Its design relies on both code modularity and abstraction, as a

way to favour separation of concerns, and architecture reuse. Finally, this is achieved with currently available software and hardware.

Raster data is currently used, but not strictly enforced. As a matter of fact, some members of the IMA research group are currently investigating how to integrate a massive vector data set into the IMA framework. *DynaMap* will be extended to support such format when it becomes available. We will show in Chapter 4 how *DynaMap* encompasses the user's context.

Figure 3.17: Screenshots of *DynaMap* in Action

Figure 3.17³ provides an overview of the different features of *DynaMap*. We can

³The screenshots were generated in the context of this thesis. The fact that the title of the application still says “DynamicMap” is purely due to a mishap by the programmer.

see a screenshot of *DynaMap* used in text mode (Figure 3.17a). In this mode *DynaMap* gives to the user a summary of the information available from the GPS, which includes latitude, longitude, speed of travel, GPS current date and time, and heading (displayed both textually and graphically in the form of a compass). The user may change the format of the displayed information. The user may even decide to turn off some of them. The text mode is provided when no spatial data is available. It may also be used as a quick way to test whether the *DynaMap* has acquired correctly access to the GPS.

However, the most interesting mode is the graphical mode as seen in Figure 3.17b. In that mode a map is displayed such that the user appears in the center of the screen (shown as a small target symbol). The user's heading is shown both literally, at the top of the screen, and graphically, as a rotating arrow anchored at the user's location. Depending on the map rotation algorithm chosen (see Chapter 5), the map may or may not be rotated. In any case, a cardinal symbol is shown in the top-right corner of the screen that indicates in which direction is North. Finally, we have also integrated the breadcrumb feature to the graphical mode after a discussion with members of the Human Oriented Technology (HOT) lab at Carleton University. This feature is to display in the form of line segments up to thirty of the user's last positions. We hypothesize that this feature helps to improve the user's understanding of the map and reduces the scale of the mental interruption felt after a large map rotation.

Finally Figure 3.17c shows one of the multiple setting screens available in *DynaMap*. In that particular case we see the GPS settings. The other setting screens are the server settings, the context-related settings, and the miscellaneous settings (mainly affecting display). In Figure 3.17d we see one of the menus available from the menu bar, and in Figure 3.17e we find a user in the process of testing *DynaMap*. In order to have fewer objects for the user to carry in his/her hands, and to enforce that the GPS be kept in a stable horizontal position (to obtain more accurate readings from the electronic compass), we built a board where both the GPS and the PDA can sit.

The *DynaMap* application has been used extensively not only by the development team, but also by external users. The application has gone through a usability lab

experiment (refer to Section 5.3.2) where multiple users where asked to try the application in a simulated environment. Before the experiment took place, members from the HOTLab at Carleton University evaluated the application in a real-life setting. We are now in the process of preparing a field experiment where even more users will get to try *DynaMap* in a more realistic environment.

In this chapter we have introduced *DynaMap* from many different angles. First we provided a fictitious scenario giving an example which laid down the context in which such an application might be used. We then introduced the IMA framework, and explained the philosophy behind *DynaMap*. We then showed the various facets of *DynaMap* by displaying its development requirements as well as the design that met those requirements. We finally provided the results of its design and compared them to the results of other similar applications. There is however one more aspect of *DynaMap* which we have not yet approached, and that is the one of context-awareness, and more importantly how to manage it. This is the topic of next chapter.

Chapter 4

CONTEXT-AWARENESS IN DYNAMAP

Looking back at the scenario given in section 3.1 we note that, as portrayed up to now, *DynaMap* is still far from encompassing the user's context in a global and systematic way. The only user's context considered by *DynaMap*, up to now, consists of location and orientation. Indirectly, the history of the user's past locations, i.e. the breadcrumb feature, as well as the adaptive rotation mode based on the user's orientation patterns (see chapter 5 for more details) can also be included in the list of user's contexts to which *DynaMap* is sensitive to. However, we note that so far the user's context is integrated into *DynaMap* in a case-by-case basis, without any overall organisation and strategic planning.

4.1 Context Management System

Indeed history has shown that the integration of context in an application is often done on the side, without any organisation, in prototype mode only (e.g. [91] [75] [19] [68] [5] [6] [99] [10], refer to Section 2.2.2 for a review of the work done in context-awareness). We wanted to seize the opportunity offered by *DynaMap* to model Context Management so that it can be reused in other IMA clients, and possibly even in non-IMA clients. We therefore demonstrate here how we have integrated into

the design of *DynaMap* a Context Management System (CMS) .

We have already given, in Section 3.5.3 on the design of *DynaMap*, the different components of the application (in particular refer to Figure 3.9). The component dealing exclusively with context is the Context Manager. When combined with the Central Controller, which is responsible for acting upon the acquired context, these two components provide the context management services and therefore act as the CMS of *DynaMap*. We define a CMS to be a computing entity that can achieve the following features:

Reference	Name	Description
CON#1	Language	Provide a formal way to describe and exchange context
CON#2	Flexibility	Allow to easily integrate new or different context
CON#3	Logic	Can easily integrates into any application's logic
CON#4	Storage	Provide a way to store context, and to retrieve context from storage
CON#5	Profile	Provide a way to switch between context pertaining to different users

These features provide the necessary requirements for a CMS in a general setting. First of all, to achieve global acceptance a CMS must facilitate its integration into an application's logic. This is currently an issue as there is little help offered to system designers who wish to make their system context-aware. Secondly, as context-aware applications become more commonplace there needs to be a way for these applications to integrate new user's context as it becomes available. At the same time the context acquired by one application might be useful to another application. Therefore context communication and sharing needs to be supported by the application's CMS. Finally, because user's context is built over time one needs to be able to store that context

and later analyze and extract further context from the archived context. Similarly, an application might serve multiple users and it must be able to switch from the context of one user to another. We believe that a CMS supporting features CON#1 to CON#5 is complete and provide a generic context management solution to mobile context-aware applications.

The realization of the CMS within *DynaMap* covers all of the above listed features. The flexibility (CON#2) and in part the storage (CON#4) features are covered by the use of the *Context Toolkit* [28], and is the topic of next section.

We address the rest of the CMS features on two fronts. We start with the concept of user profiling. Because context is gathered over time, sometimes requiring machine learning, and pattern recognition algorithms, it must be preserved in between sessions when an application is executed. Context-awareness would be a much weaker concept if it consisted of only user-session information. Therefore a way has to be developed to store context. We propose in *DynaMap* a Context Description Language (CDL), based on XML, that allows context to be communicated, or stored. Figure 4.1 shows a scenario where a system based on CDL allows context sharing and archiving.

Figure 4.1: Context Communication and Storage with CDL

```

<ContextDescriptionData Comment="test">
  <ContextData ContextId="CTTID" Title="MyDog">
    <ContextUnit ContextSubId="NameID" Name="DogName" Type="Text">
      Fido
    </ContextUnit>
    <ContextUnit ContextSubId="AgeID" Name="DogAge" Type="Int">
      2
    </ContextUnit>
    <ContextUnit ContextSubId="WeightID" Name="DogWeight" Type="Double">
      20.3
    </ContextUnit>
    <ContextUnit ContextSubId="IsMaleID" Name="DogGender" Type="Bool">
      true
    </ContextUnit>
    <ContextUnit ContextSubId="FoodID" Name="DogFood" Type="Senum">
      <ElmChoice ChoiceId="foodchoiceone">yummy</ElmChoice>
      <ElmChoice ChoiceId="foodchoicetwo">soso</ElmChoice>
      yummy
    </ContextUnit>
    <ContextUnit ContextSubId="FriendsID" Name="DogFriends" Type="Menu">
      <ElmChoice ChoiceId="friendone">Daisy</ElmChoice>
      <ElmChoice ChoiceId="friendtwo">Jack</ElmChoice>
      Daisy;Jack
    </ContextUnit>
    <ContextUnit ContextSubId="BirthDateID" Name="DogBirthDate" Type="Date">
      2003-04-12
    </ContextUnit>
    <ContextUnit ContextSubId="TimeID" Name="DogSupperTime" Type="Time">
      17:00
    </ContextUnit>
    <ContextUnit ContextSubId="CompetitionsID" Name="DogCompetitions" Type="ContextUnit">
      <ContextUnit ContextSubId="CompetitionDateID" Name="Date" Type="Date">
        2004-23-10
      </ContextUnit>
      <ContextUnit ContextSubId="CompetitionScoreID" Name="Score" Type="int">
        94.3
      </ContextUnit>
    </ContextUnit>
  </ContextData>
</ContextDescriptionData>

```

Figure 4.2: Sample Context Data Encoded with *DynaMap*'s CDL

The CDL allows the designer to specify a context ontology that is used in the application, i.e., to define what context is used and how it is described in a current setting. *DynaMap*'s CDL stores the context in a recursive attribute-value pair scheme as can be seen in the example provided in Figure 4.2. The full specifications of *DynaMap*'s CDL may be found in appendix C. The proposed CDL in *DynaMap* provides a first step in standardizing and therefore facilitating how context is expressed between two entities. Eventually more work will be required to determine whether the current specifications address all needs of all context-aware systems. In addition a meta-language needs to be established to provide a mechanism to describe the ontology of the context information specified in a CDL encoded stream. The CDL therefore

covers the language (CON#1) and in part the storage (CON#4) requirements of the CMS. Once context can be expressed contextual computation does not have to originate where context is collected. For example certain shortest path computations in GIS are some of the most computation intensive operations done. Therefore such computations would normally be handled by a high performance server. If the shortest path is based on some user's context attributes, then those can be passed to the server using CDL. Along the same line of thought, user profiling is simplified once there exists a CDL. Indeed, for each user a different CDL profile (stored as a file, a database record, etc.) can be loaded, effectively switching between the contexts of different users. Which context is saved and reloaded is application dependent. In *DynaMap* we opted to have for each user a local file stored directly into the PDA in *DynaMap*'s CDL format (appendix C).

The other point addressed by the realization of the CMS within *DynaMap* is the one of integrating context-awareness in the flow of execution of the application's logic (CON#3). Care has to be taken here so that the integration of context follows the same principles of abstractions and modularity as defined in Chapter 3 in order to maintain a generic design that can be re-applied. In order to recall the current system architecture of *DynaMap* we refer to Figure 3.9 describing the code modules and to Figure 3.12 providing the central controller's state diagram. Both figures can be found in Section 3.5. In these illustrations we see that there is a "Context Manager" component. We also see that the central controller event flow is triggered via some external event informing it of new available context. The context manager component is the one sending that event. We illustrate in Figure 4.3 the state diagram of the context manager.

The purpose of the context manager is to handle sources of context information. It contains a list of locations where context information can be acquired and it manages the communication with these sources. This list might be deterministically specified or built over time using a discovery mechanism. This is application dependent. The context manager is then responsible to preprocess the acquired context from the sources and to present it to the central controller component in a pre-established format that is understood by both components. The context manager is also responsible

ContextManager: Waiting for Context

Figure 4.3: The *DynaMap* Context Manager State Diagram

for storing historical context as well as loading and saving the user profile. From the central controller's perspective, once it is made aware of new context information, it proceeds by taking decisions based on the application's logic, and then acts upon those decisions. We can see in Figure 4.4 a typical contextual data exchange between the central controller and context manager components as well as the context sources (called widgets as they are defined in the next section).

From *DynaMap*'s perspective the context manager has a predetermined list of context sources that are considered. This explains why the state diagram provided in Figure 4.3 contains the given states. The specific context information used by *DynaMap* is covered in Section 4.2. Once acquired and preprocessed, the *DynaMap*'s context manager provides the context to the central controller. In *DynaMap* the most common operation based on context data is to integrate personalized information in the visual display of the presented map. The details of the context induced behaviour in *DynaMap* is covered Section 4.2.1. However there is still one missing piece of the

puzzle: what are the context sources and how do we extract the context from them? The answer is provided in next section with the introduction of the *Context Toolkit* [28].

Figure 4.4: The *DynaMap* Context Data Exchange

4.1.1 The Context Toolkit

The *Context Toolkit* [28] was introduced in section 2.2. It consists of a set of software tools whose purpose is to support context acquisition, aggregation, interpretation, and distribution. As a result of his work on the *Context Toolkit*, Dey has not only formalized the notion of context and context-awareness, but he also provided a tool that is both simple to use and to understand. It is built on software engineering principles such as robustness, scalability, efficiency, etc. We decided to use the *Context Toolkit* in order to help us systematize our process of context acquisition.

The toolkit is based on the widget metaphor often used in graphical user interface

development. The idea is to separate from the application's logic the internal complexities of on-going context extraction. The second objective is to be able to reuse context in other applications by separating the process of context extraction from the other processes involved in the application. The idea there is that by decoupling context extraction, multiple applications can tap into this shareable resource.

The widget metaphor provides a solution that meets both of the toolkit's objectives. In the toolkit context is seen as information waiting to be retrieved. Specific context will have specific logic in how it is acquired. For example user location often requires an application to connect to a GPS, to parse the GPS output, and to encode the location information in a useful format. On the other hand a system architect designing the application's logic has little interest in "how" the context was obtained. The "how" part of context acquisition is a cumbersome task to the system designer, offering little added value to the application. The difficulty involved in extracting context provides an explanation why very few applications make use of context, and why very few types of contextual information are considered. Just like the way GUI widgets are used to simplify user input access, a context widget is something the system architect would like to use to simplify context access in the application. Referring back to the user location example, by using a user location widget an application would no longer need to include the GPS connection logic and the parsing logic, all of which would be included in the widget. As the user location technology evolves so does the user location widget successfully hiding away to the application the complexity behind context extraction. Now in the case of context that is accessible via a central location such as current traffic conditions for example, the context widget plays a second role: to be a common access point. In addition to containing the logic to extract context, the context widget manages who wants what context under what conditions providing a way to share common context across applications.

The *Context Toolkit* is based on four components as well as a communication convention. We show a simplified diagram of the components in Figure 4.5 in a way that shows the inheritance relationships between the components. We reuse the notation of Dey [28] to display the application, the BaseObject, the Widget,

the Aggregator, and the Interpreter components. This allows us to provide context acquisition diagrams in the same way defined by Dey [28] (see Figures 4.6 and 4.8).

The component central to the toolkit is the widget. It is the component responsible for extracting specific pieces of context, and for managing the subscribers to the information. Clients can connect to a widget and query the status of its context as a whole or in parts. They can also register themselves to be notified when a predefined context occurs. Finally, a widget comes with the capability to store the context it has extracted into an archive and provides a way for clients to query the archived context.

An example of the use of a context widget is as follows: imagine an application that needs to be aware when the user suffers from an anxiety attack. To measure anxiety the application uses a heart rate sensor. The code required to connect to the sensor and the code required for the communication protocol used is a cumbersome task for the developer to write. In addition, anytime another application wants to know of the user's degree of anxiety the same code as to be rewritten. By embedding the sensor in a context widget we automatically solve the problem of code duplication since the widget runs as a server and can accept multiple subscription requests to the context information. Moreover the applications can request to only be notified when the context is relevant to them, such as when the user's anxiety level is high enough to be classified as an anxiety attack. Finally the applications need only care about the user context and not about the particular characteristics of the sensor, which is all encapsulated in the widget. Widgets can be more complex than just a single sensor wrapper. For example a widget could be used to provide the current weather conditions in a city such as Ottawa, Canada. The widget could then use as its sources of information multiple websites providing that weather forecast information. Relying on data fusion techniques the widget could then make a final decision on the actual weather conditions, and provide that information to its subscribers in a standard way.

Another important component is the aggregator which is in fact nothing more than a specialized widget. The aggregator provides context built on top of context acquired by other widgets. It is both at the same time a context widget and a client

to other widgets. The aggregator is often used to generate higher-level context from sources of lower level context by containing the logic to merge and combine them. As a simple example one could use an aggregator to determine whether the user is in a rush or not by combining the following lower level context: user's schedule, user's speed of travel, user's heartbeat. An application that would only need to know whether the user is in a rush or not would only have to register itself with the aggregator, which would then take care of the rest.

A third component is the interpreter whose purpose is to convert context from one type into another. A physical to logical location converter (refer to section 2.2.3) would be an example where the use of an interpreter would be useful.

Finally, the fourth component is actually the parent of all the others: the BaseObject. The BaseObject provides the communication knowledge of the toolkit (including marshalling and unmarshalling of the data). It is used as a standalone component in applications that integrate the toolkit, like in *DynaMap*. The default communication convention used is HTTP as the data transfer protocol, and XML as the data format. Although HTTP and XML are two well-established standards in data communication, both of them can be replaced by other technologies if that is the wish of the system designer.

Figure 4.5: The *Context Toolkit* Components

Since the toolkit separates context acquisition from everything else little had to be done in the design of *DynaMap* to integrate it. Each piece of context is acquired

with the use of widgets, aggregators, and interpreters (refer to Section 4.2.2). This information is then transmitted to the central controller, via the Context Manager component. The Context Manager interfaces with the widgets through one instance of a BaseObject that it stores. We can see in Figure 4.4 the context exchange process involved between *DynaMap* and the context widgets that it uses.

Of interest from a technical point of view is the fact that the *Context Toolkit* was developed in Java, whereas *DynaMap* is a C++ application. However, since the communication to the toolkit widgets is done via HTTP (by default), the application that uses these widgets, *DynaMap* in our case, is indifferent to the programming language of the widgets. Only the BaseObject had to be converted into C++ since this is the component that contains the logic needed to communicate with the widgets. All widgets developed for *DynaMap* were done with the toolkit original code, i.e. in Java.

4.2 What Context?

In the scenario given in section 3.1 a student is using a university provided PDA assistant as a way to help with the transition to a new campus. We noticed many instances of context-triggered behaviour in the system. In addition to location and orientation awareness there is also the constant up-to-date knowledge of the application with regards to the campus (class locations, exam schedule, pathways status, etc), and to the student (courses taken, living arrangements, books borrowed from library, etc). But the context of interest is even wider, including such things as time of the day, weather conditions, and also personalization capabilities. In *DynaMap* we decided to consider the current time of the day, the weather, the student's class schedule, and the user's preferred path. We believe that the integration of this context will serve as proof-of-concept towards the architectural design of context and that is feasible within our limited timeframe. Of course many additional user's contexts could be integrated into the application but this is considered future work and is outside of the scope of this document.

4.2.1 Contextual Influence

In this section we introduce two sub-tasks of *DynaMap* whose behaviour is influenced by the user's context, namely map display and shortest path computations.

Contextual Map

Map display has been studied for many years and has been described as both an art and a science [97]. Cartographers have always been looking at ways of making maps more useful so that they contain information that is actually relevant to the user. With the emergence of context-awareness it is now more than ever possible to make maps tailored for a single user. We have an opportunity in *DynaMap* to provide real-time contextual maps.

The next question we need to answer is then: how can we personalize the map displayed to the user? From the context collected, described in Section 4.2.2, we decided to personalize the map in the following way. First, roads, paths and buildings are displayed with increasing colour intensity depending on the frequency such entity is actually used by the user. Therefore, the user's most relevant locations are shown more prominently. Secondly we indicate on the map the location of the user's next class if that class is one hour away or less from beginning. Along the same line of thought, locations containing food and entertainment services are also highlighted according to the time of the day. For example, in the morning, coffee shops are displayed, as opposed to cafeterias around noon and dinnertime, and to pubs in the evening.

All of our context related map display behaviours are triggered by active context (refer to Section 2.2.2) since they cause the map to be displayed differently without any user intervention. However, since those changes do not critically affect the use of the application we are still confident that this type of active context does not induce a "loss-of-control" feeling to the user. In terms of Schmidt's categorisation scheme [93], we can see that our contextual map comes from the environment, both from a social point of view (student), as well from the temporal environment (time of the day). One might argue however that some context is activity generated, more particularly the

one showing the location of the user's next class where here the activity is essentially the act of going to the classroom.

Contextual Shortest-Path

The shortest path problem consists of, given a source location and a destination location, to provide the shortest path from the source to the destination based on a cost model. The input to a shortest path algorithm consists of a graph (nodes and edges) representing the traveling network, as well as the position of where the source and the destination are. The cost model can be quite broad and not only consists of pure and simple Euclidean distance. Weights representing a certain cost (e.g. traffic density) of using an edge can be used, which turns the problem into a weighted shortest path problem.

The objective is to illustrate that user-context can be integrated in *DynaMap*. The algorithm used to compute the shortest path is of secondary importance here. We chose to use Dijkstra's algorithm [30] for its simplicity. However there exists a number of alternative algorithms to efficiently compute the shortest path in either internal or external memory [22] [55] [44] [43]. We leave as future work the exercise of identifying the proper external spatial data structure in conjunction with efficient algorithms for shortest path and other potential queries (window queries, contour line extractions, etc).

In *DynaMap* we consider four factors when assigning a cost to path edges:

- l = Euclidean edge length,
- u = edge relative usage ratio,
- w = edge weather exposure ratio, and
- d = edge darkness ratio.

The cost of edge e is therefore $cost(e) = l * 1/u * w * d$. The value l is the length of the edge and is therefore fixed. The value u is computed for each user as the edge usage counter divided by the total use counter (see Section 4.2.2 on the user's preferred

path). In other words the value u provides a relative measure of how frequently an edge is used versus the other edges for a given user. To compute the cost of the edge we use the inverse of the edge relative usage ratio. Therefore the higher the usage of an edge by the user, the lower the cost of that edge.

The weather and darkness ratio values w , d consists of the multiplication of two factors, a static factor w_s and d_s describing “influence”, and an attenuation factor w_a and d_a describing “current condition”. The influence factors are values between 0 and 1 determined by members of the Carleton University campus to describe how of an impact has weather and lighting on a specific edge. For example, the more an edge is likely to be unusable due to extreme weather conditions, the higher its w_s factor, and the less an edge is well lit at night, the higher its d_s factor. The attenuation factors, also values between 0 and 1, are assigned according to the current weather condition context and the current darkness condition context (e.g. extremely cold weather and total darkness have attenuation factors of 1, whereas warm weather and noon sunshine have attenuation factors of 0.3. Refer to Section 4.2.2 for the exhaustive list of attenuation factors used). In summary the w and d ratios provide a measure of how usable the graph edges are based on the current weather conditions and based on the current lighting conditions. The values are computed at shortest path computation time as follow: $w = w_s * w_a$, and $d = d_s * d_a$.

DynaMap offers the possibility to query for the shortest path from the user’s current location to a location provided by the user by pointing and holding the PDA’ stylus on the map corresponding to his/her desired destination. It is possible to do this in *DynaMap* by restricting the graph to only consist of the student’s campus, and by limiting the context to be the one described in the cost model. However a more general solution would be to have a GIS server with shortest path query capabilities. The application would provide the server the user’s context via a CDL formatted data exchange, and the server would return with the computed shortest path. Such architecture would remove the constraint of having the client application such as *DynaMap* to have to know about all possible context considered by the domain and to only focus on collecting the user’s context. However we consider this problem to be possible future work to be considered during the development of the IMA framework

(refer to Section 3.3).

Finally we point out that the graph considered in *DynaMap* is a 2D representation of the geographical domain. This does not cause a problem in most cases since we are mainly interested to provide 2D navigation assistance to the user as a proof-of-concept that user context can be integrated into *DynaMap*. However we live in a 3D world, and therefore the graph considered in *DynaMap* is not a perfect representation of the environment. However, 3D navigation could be supported with the addition of third coordinate to all instances in the graph, as well as with the addition that the GPS accurately provides the user's altitude.

In the case of contextual shortest path in *DynaMap* context is again active-context mainly because the shortest path computation is directly affected by the user's current context, and that, without the explicit consent of the user. Similarly to contextual maps we do not believe this to be a problem as we don't foresee users feeling a "loss of control" over this behaviour. However we do acknowledge that somehow the user should be able to better specify what kind of shortest path s/he might wish to have computed (fastest, simplest, shortest distance, etc). We also believe that somehow the user should have a final say over some of the weights assigned to the edges (for example some user might not care as much about the weather condition effect on the shortest path computation). These are what we consider additional context information of type personalization that could be supported by *DynaMap* but were not due to time constraints. In terms of Schmidt's categorization [93] contextual shortest path comes from the environment (time of day, weather) with the exception of the user's preferred path, which is of type activity since it comes from the user's traveling habits.

In all cases, in order to promote contextual influence on the behaviour of *DynaMap* the context itself must be collected. A detailed overview of how this is achieved in *DynaMap* is given in the next section.

4.2.2 Context-Aware Realisation

DynaMap acquires context with the use of the *Context Toolkit*. The toolkit provides one additional benefit: it forces system designers to identify the context required

and to break it down into pieces of information that can be collected. The designers then devise how to assemble the pieces together with the use of the objects provided by toolkit (refer to section 4.1.1). We have already defined the context of interest for *DynaMap*: the student's class schedule, the current time of the day, the weather forecast, and the user's preferred path. We will show in this section from what sources of information the context can be extracted and interpreted, as well as how it is handled by the toolkit before it is sent to *DynaMap*. In each case this will be illustrated with the help of a diagram, using the notation of in Figure 4.5.

Figure 4.6: Context Acquisition Diagrams

Figure 4.7: Context Acquisition Widgets

Class Schedule

In order to display to the user the location of his/her next class *DynaMap* needs to know in what classes is the student currently registered, at what time are those classes offered, and where they are taking place. We refer to this as the student's class schedule. We assume that this information is made available by the learning institution, in a way that allows a context widget to query it. A context widget provides access to that contextual information, and the BaseObject of *DynaMap*'s Context Manager subscribes to that widget. See Figure 4.6a as an illustration.

The actual implementation of that widget is to simulate a few class schedules that could be assigned to a particular user. Through a simple user interface illustrated in Figure 4.7a we can force a different schedule on the user. *DynaMap*, which subscribed to this widget, is informed every time the schedule of its user (represented by the student number) has changed. The current widget supports three users: 100232650, 100232651, and 100232652, as well as three different schedules: schedule A, schedule B, and schedule C.

Time of the Day

The current time of the day is contextual information typically relevant to a user. Not surprisingly the source of time comes from the clock local to the device, and a widget is assigned to keep track on its evolution. See Figure 4.6b as an illustration.

In *DynaMap* we again decided to simulate this widget to be able to better illustrate the change of behaviour it induces in the application (Map display, shortest path. See Section 4.2). We have therefore developed a simple user interface as illustrated in Figure 4.7b. *DynaMap* registers to that widget. Every time the time of the day changes, *DynaMap* is then notified. The current values available in the widget are: Breakfast(0.3), Morning(0.2), Noon(0.1), Afternoon(0.1), Supper(0.3), Evening(0.5), and Night(1.0). The numeric values indicate the darkness attenuation factors (d_a) used during contextual shortest path computation (see Section 4.2.1).

Weather

A user engaged in navigational activities is often affected by the current weather conditions. This is more so for a pedestrian user, who must endure directly the different moods of Mother Nature. This type of context acquisition system is modeled as shown in Figure 4.6c. The weather information from the weather database is extracted at regular small time intervals by the widget. We are only interested in the weather at the place where the user is located. Location is provided by a GPS, and is made available via a longitude-latitude widget. Since weather forecasts are available per regions, which are identified by the most prominent city in that region, we need an interpreter that can convert from longitude-latitude coordinates to a city name. An aggregator is then responsible for querying the city interpreter, and querying the weather widget for the weather forecast in the region of that city.

Again, for simplicity we opted to simulate this widget through the use of a user interface illustrated in Figure 4.7. We may change the current weather condition to: Sunny(0.1), Overcast(0.5), and Raining(1.0), as well as the temperature to: Cold(1.0), Chilly(0.5), Warm(0.1), Hot(0.5), and Burning(1.0). Every time any of these two value changes, the widget notifies *DynaMap*. The multiplication of the weather condition numeric value times the temperature numeric value provides the weather attenuation factor (w_a) used during contextual shortest path computation (see Section 4.2.1).

User's Preferred Path

The final user context included in this document is the one of the user's preferred path. A user is said to have used a certain path, if s/he was within a certain radius from that path (5 meters in *DynaMap*). Each user in *DynaMap* keeps a counter for each edge in the graph. In addition each user keeps a global counter. All counters are initialized to the value 1. Once a path is used, the counter for that path is increased by one and so is the global counter. Using these two counters we can compute for each edge the edge relative usage ratio (u) used during contextual shortest path computation (see Section 4.2.1). The ratio consists of the edge counter divided by the global counter

providing a relative measure of the usage of that edge versus the usage of the other edges. Edges with higher edge relative usage ratio (u) are displayed more prominently on the map and are favoured in the shortest path computation algorithm.

As we can see, a lot of work is done in *DynaMap* to decide which are the user's preferred paths. However one still need to obtain the current path being used. We refer to Figure 4.8 to illustrate the toolkit components involved in acquiring this information.

Figure 4.8: The User's Preferred Path Context Acquisition Diagram

Again, the GPS provides location information, which is extracted by the longitude-latitude widget. However, this time the longitude-latitude coordinates are being passed to an interpreter which has access to the spatial data, and can therefore decide which path (if any path at all) is being used by the user. That path is then sent to the BaseObject of *DynaMap*, and the edge counters are updated.

4.3 End-Product

The design of a Context Management System (CMS) that can be re-applied in general to other mobile context-aware applications is the main contribution of this chapter. We analyzed the features that are important to provide for any CMS in a mobile and

context-aware environment and came up with the following five: language (support context exchange), flexibility (easy to support new context), logic (easy to integrate in the application's logic), storage (can store context), and profile (can switch between users). We then proposed a Context Description Language (CDL) that facilitates context encoding, which in the end allows for a formal context exchange data format. Finally we re-introduced the *Context Toolkit* [28] but this time in more details so as to better understand its benefits towards context modeling, context acquisition, and context sharing.

Using the CDL and the *Context Toolkit* we have shown how to build a CMS within *DynaMap*. We explained how context is handled by *DynaMap*'s central controller component in a way that allow for any type of context to be supported. We have also introduced the Context Manager components and explained its roles of acquiring context (via the *Context Toolkit*), storing context and switching between contexts of different users (user profiling).

Finally we have shown how *DynaMap*'s CMS can be used in a real life environment by describing two areas of the application influenced by the user's context: contextual maps and contextual shortest paths. We chose four different types of user's contexts to illustrate how to achieve context-awareness in these two areas, and our choice was: the user's class schedule, the current time of the day, the current weather conditions, and the user's preferred path. Finally we showed how to model context acquisition for each one of them, and how to integrate this context within *DynaMap* to achieve contextual maps and contextual shortest paths.

Both contextual maps and contextual shortest paths have been partially implemented in order to validate the proposed CMS design. We were able to do so without implementing the entire proposed and modeled context described in this chapter due to the limited time constraint. Instead we focused on a scaled-down version of the user's preferred path in order to provide a proof-of-concept of *DynaMap*'s CMS. We may see in Figure 4.9 an illustration of how context information is currently used and handled by *DynaMap*. We can see the result of a shortest path computation based on a scaled-down cost model defined in Section 4.2.1.

DynaMap shares a number of similarities with other applications with respect to

Figure 4.9: Screenshots of Context in Action in *DynaMap*

its context induced behaviour. For example like many other LBS *DynaMap* makes use of the user's location and orientation [28], [17], [93], [10]. *DynaMap* also has a notion of the user's traveling habit as similarly defined in [68]. Analogous to tourist-type applications [19] [94], *DynaMap* uses domain-specific context information, namely that of a student through the class schedule context.

However where *DynaMap* differs is in the overall integration of context throughout the system architecture. It builds on the work of Dey on the *Context Toolkit* [28] and proposes an all-encompassing reusable CMS that can be applied in the IMA framework. However the principles outlined can easily be extended to non-IMA framework environments as well.

This concludes our section on *DynaMap*'s integration of context. We continue on the topic of user's experience in the next chapter since an attempt is made at discovering which map rotation scheme is best suited for its users.

Chapter 5

MAP ROTATION

We have all done this before, take a paper map, and rotate it so that what is in front of us appears at the top. This behaviour has been studied numerous times by psychologist and cartographers. The general explanation of such behaviour resides in the cognitive load imposed on a user in understanding a map, and in building a mental model of what is represented on the map. If the map is rotated so that it fits the user's orientation in his/her environment, then a lesser cognitive load is required to understand the map than if it had not been rotated in the first place. It has also been shown that the extent to which users will feel they should rotate a map varies between individuals and gender. A study done by Radoczky however, shows that eighty-six percent of all participants preferred head-up maps, as opposed to only nine percents that preferred North-up [37].

In *DynaMap* in order to accommodate everyone, we initially offered two choices to users: to have the map rotated (head-up) or to have its orientation fixed (North-up). If chosen, map rotation was performed every time a new user heading was provided by the electronic compass (embedded in our GPS). Therefore the map was always in-line with the user's orientation, plus an average of one second delay due to the sensing technology.

5.1 Problem Definition

When we tried *DynaMap* for the first few times however, we did not find the map rotation feature quite as useful as expected. Instead of having a better map understanding and a better positioning of our location on the map, we found that the map being constantly rotated confused us to the point of sometimes us becoming totally disoriented. This result can be explained by the following two reasons:

1. The instability of the map due to constant rotations caused by the variations in the user's heading.
2. The time required to reposition oneself on the map after each rotation.

Point 1 comes from the fact that it is almost impossible for the user to keep a constant orientation (between 0 and 360 degrees). The constant movement of the user, even when trying to be stable, and the margin of error of the sensor are both direct causes of this problem. Point 2 comes from the fact that the process of understanding a map requires the user to find a correlation between the real world and the one depicted on the map. A large sudden and unexpected rotation of the map can break the user's mental model of the map which is required to decipher the information provided on that map. One might argue that using animation to rotate the map would solve this problem but unfortunately this cannot be treated as a solution. Not only would animating rotations be a technical challenge on a PDA with limited processor and battery power, but it would also be difficult to find the correct balance between the time required for an animation, and the need to accurately show the map rotated according to the user's orientation. Of course, in the end, point 2 is much more a problem with large rotations than it is with small ones, since a user will not be thrown off as much by smaller rotations.

Point 1 has already been acknowledged by Suomela et al. [99] in their work on the *WalkMap*, an application testing different map projections. Their solution was to prevent rotations less than ten degrees. We find little comfort by this simple proposition. First of all there is no substantiation that ten degrees is the appropriate threshold value, and secondly, to apply a minimum threshold on rotation amplitude is

only one of many ways to approach the problem. We acknowledge that under certain conditions this solution might work well, such as for a user in a car mostly going in straight lines, but we believe that it may not work so well in many other cases. Indeed, pedestrian navigation is more complex than some other types of navigation due to the increased degree of freedom enjoyed by the users. For example, a user lost in an unknown environment would most likely use a navigation application like *DynaMap* by first orienting him/herself with an object in sight (building, intersection, etc) and then comparing what s/he sees with what is on the map displayed. In such an alignment case we certainly want map rotation to occur quickly and accurately. However, there are other cases where map rotation might be uncalled for. We can think of a map like those found in shopping malls. In such a case a user normally wants to find a certain location, from his/her current position on a global perspective map. If the map is rotated, even slightly, then the user's mental model of the environment is gone and has to be rebuilt.

We believe that due to the complexities of the tasks executed in pedestrian navigation, and due to the multiplicity of such tasks, that we need a more thorough evaluation of several map rotation strategies. We propose in the next section a few different map rotation algorithms, along with their analysis.

5.2 Proposed Solution

After our own experience in using an early version of *DynaMap*, and after discussing with people who were assigned short trials with the application, we came with a set of different rotation algorithms. Each one has their characteristics in terms of how and when the map is rotated, but also in terms of their efficiency and storage. These algorithms are briefly described in Table 5.1, and then further explained in the following subsections.

Name	Description
STATIC	Map is never rotated
DIRECT	Map is directly rotated according to the user's heading at every reading from the electronic compass
R.ANGLE	Map is only rotated at right angles
MIN	Map is only rotated if the difference in amplitude from the last rotation is more that the value specified in the variable MINTHRESH
MAX	Map is rotated by at most the amplitude specified in the variable MAXTHRESH
MIN_MAX	Scheme where the MIN mode is first applied, followed by the MAX mode
ADPT	Similar scheme as MIN_MAX with the difference that the value of the variables MINTHRESH and MAXTHRESH are automatically updated according to the history of the user's orientations

Table 5.1: Map Rotation Schemes available in the DynaMap

5.2.1 STATIC

This mode consists of North-up map. Therefore no map rotation occurs no matter what the user's heading might be. The benefit of this mode is that the user does not experience disorientation due to map rotation. However this comes at the cost of the user of having to reorient his/her mental map in order to align the real world with the map representation. No extra processor power, or storage is required in this mode.

5.2.2 DIRECT

This mode consists of the classical head-up map, in the sense that no intervention is made to prevent map rotation from happening. The benefit of this mode resides in the always-accurate orientation of the map with respect to the user's heading. The fact that map rotation is hypersensitive to the user's heading causes it to occur frequently, which can irritate users. Large map rotations are possible in this mode, which sometime causes a user to become disoriented after such a big change from one displayed map to another. The pseudo code of DIRECT rotation is listed in

Algorithm 1. The function `RotateMapToHeading()` used in line 2 takes a heading value and rotates the map such that the top of the map now corresponds to that heading value. For example, if the user's heading is 20 degrees, then the DIRECT algorithm calls `RotateMapToHeading(20)`.

Algorithm 1 DIRECT

```

1:  $X \leftarrow \text{UserHeading}()$ 
2: RotateMapToHeading( $X$ )

```

The processing power required by the DIRECT rotation mode is quite high in the sense that map rotation occurs frequently, i.e. at every new user heading. The amplitude of the rotation can be any arbitrary value between 0 and 360 degrees, preventing any optimization of the rotation code. The storage required by this mode is constant.

5.2.3 R.ANGLE

This mode allows only right angle map rotations. In this mode the map is either rotated North-up, East-up, South-up, or West-up. The direction into which a map is rotated depends on which of the four quadrants falls the user's heading. The quadrants are defined as ± 45 degrees from the cardinals. For example, a map is rotated East-up if the user's heading is within 45 and 135 degrees. A small buffer of 15 degrees along the borders is given to the user before a switch is made to another quadrant. This is done to avoid having back and forth 90 degrees map rotations as a user's heading drifts along a border case. The pseudo code of the R.ANGLE rotation mode is listed in Algorithm 2.

The benefits of the R.ANGLE map rotation mode lies in its simplicity and its correlation with the experience users have with paper maps. Indeed, people rotate paper-based maps mostly in only four ways: North-up, East-up, South-up, and West-up. Also, using this mode a user knows that there are only four ways a map can be rotated. The user should therefore be less surprised by a map rotation and s/he should require less mental processing to rebuild his/her mental model of the map afterwards. The drawbacks of this mode are the limited number of possible rotations

Algorithm 2 R.ANGLE

```

1:  $X \leftarrow \text{UserHeading}()$ 
2: if (LASTQUAD = NORTH) AND(( $X \leq 60$ )OR( $X > 300$ )) then
3:   RETURN
4: else if (LASTQUAD = EAST) AND( $30 < X \leq 150$ ) then
5:   RETURN
6: else if (LASTQUAD = SOUTH) AND( $120 < X \leq 240$ ) then
7:   RETURN
8: else if (LASTQUAD = WEST) AND( $210 < X \leq 330$ ) then
9:   RETURN
10: end if
11: if ( $X \leq 45$ )OR( $X > 315$ ) then
12:   LASTQUAD  $\leftarrow$  NORTH
13:   RotateMapToHeading(0)
14: else if  $45 < X \leq 135$  then
15:   LASTQUAD  $\leftarrow$  EAST
16:   RotateMapToHeading(90)
17: else if  $135 < X \leq 225$  then
18:   LASTQUAD  $\leftarrow$  SOUTH
19:   RotateMapToHeading(180)
20: else if  $225 < X \leq 315$  then
21:   LASTQUAD  $\leftarrow$  WEST
22:   RotateMapToHeading(270)
23: end if

```

meaning that often the map will not be perfectly in-line with the user's heading. Also this mode favours rotations of large amplitude, namely of 90 degrees.

In terms of processing power, this mode is the most efficient after STATIC. This is due to both the reduced number of rotations, and the fact that rotations occur only at four angles: 0, 90, 180, 270. Such map rotations can be highly optimized for speed since it can be done with simple pixel swapping operations. The storage required by this mode is constant.

5.2.4 MIN

In the MIN rotation mode the map is only rotated if the difference in amplitude from the last rotation is more than the value specified in the variable `MINTHRESH`. This is similar to the solution proposed by Suomela et al. [99] with the exception that the threshold value is user defined as opposed to being hard-coded to a certain value. The benefit of the MIN rotation mode is the decrease of insignificant rotations, where insignificance is defined by a minimum rotation amplitude. Assuming the user sets the `MINTHRESH` variable to a value that suits him/her, the MIN rotation mode results in more stable map display. However the MIN rotation scheme does not prevent large rotations from happening. In addition, the burden of finding the correct `MINTHRESH` value is left to the user. This means that whenever the user engages in a different navigational task s/he may have to reset the `MINTHRESH` value to something else (e.g. when the user goes from walking around to driving a car). The pseudo code of the MIN rotation mode is listed in Algorithm 3. A global variable named “LASTHEADING” is used in this algorithm to remember the heading to which the map was rotated in the previous rotation operation. For example the user heading in the previous rotation operation could have been 30 degrees, but the algorithm decided to only rotate the map to 20 degrees. In that case “LASTHEADING” is set to 20 degrees.

The processing power required by the MIN rotation mode is comparable but slightly better than the one of the DIRECT rotation mode. In both cases the map can be rotated at any arbitrary angle, however in the MIN case all rotations of lesser

Algorithm 3 MIN

```

1:  $X \leftarrow \text{UserHeading}()$ 
2:  $Y \leftarrow \text{abs}((X - \text{LASTHEADING}) \bmod 360)$ 
3: if  $Y > 180$  then
4:    $Y \leftarrow 360 - Y$ 
5: end if
6: if  $Y \geq \text{MINTHRESH}$  then
7:    $\text{RotateMapToHeading}(X)$ 
8:    $\text{LASTHEADING} \leftarrow X$ 
9: end if

```

amplitude than the MINTHRESH value are eliminated. The storage of the MIN rotation mode is constant.

5.2.5 MAX

In the MAX rotation mode the map is rotated at once by at most the amplitude specified in the variable MAXTHRESH. Nothing prevents rotations to be less than the MAXTHRESH value however (in comparison with the MIN mode). The benefit of the MAX rotation mode is the elimination of large rotations. In this mode large rotations are broken down into a number of rotations of at most MAXTHRESH. Therefore the MAX rotation mode is a way to simulate rotation animation that is feasible on a small device platform. However the MAX rotation mode suffers from being occasionally quite off with respect to the current user's heading. The impression that this scheme leaves on a user with a wrongly set MAXTHRESH value is that the map is always trying to catch up with reality. In addition, the burden of finding the correct MAXTHRESH value is left to the user. This results in the cumbersome operation of changing that value whenever the user engages in a different navigational task. The pseudo code of the MAX rotation mode is listed in Algorithm 4. This algorithm also makes use of the global variable "LASTHEADING" as described in the MIN algorithm, i.e. to store the heading to which the map was last rotated.

The processing power imposed by the MAX rotation mode is exactly the same one as of the DIRECT mode in the sense that rotation occurs almost at every new

Algorithm 4 MAX

```

1:  $X \leftarrow \text{UserHeading}()$ 
2:  $Y \leftarrow \text{abs}((X - \text{LASTHEADING}) \bmod 360)$ 
3:  $\text{ROTATERIGHT} \leftarrow ((X - \text{LASTHEADING}) > 0)$ 
4: if  $Y > 180$  then
5:    $Y \leftarrow 360 - Y$ 
6:    $\text{ROTATERIGHT} \leftarrow \text{not } \text{ROTATERIGHT}$ 
7: end if
8: if  $Y > \text{MAXTHRESH}$  then
9:   if  $\text{ROTATERIGHT} = \text{true}$  then
10:     $\text{ROTATIONHEADING} \leftarrow (\text{LASTHEADING} + \text{MAXTHRESH}) \bmod 360$ 
11:   else
12:     $\text{ROTATIONHEADING} \leftarrow (\text{LASTHEADING} - \text{MAXTHRESH}) \bmod 360$ 
13:   end if
14: else
15:    $\text{ROTATIONHEADING} \leftarrow X$ 
16: end if
17:  $\text{RotateMapToHeading}(\text{ROTATIONHEADING})$ 
18:  $\text{LASTHEADING} \leftarrow \text{ROTATIONHEADING}$ 

```

user's heading update, and that rotation cannot be optimized. The storage of the MAX rotation mode is constant.

5.2.6 MIN_MAX

The MIN_MAX rotation mode consists of first applying the rules of the MIN mode, followed by the application of the rules of the MAX mode. Therefore in the MIN_MAX rotation scheme, map rotation occurs only when the rotation amplitude is more than the value of the MINTHRESH variable, and if there is a rotation, it is limited to be at most the value specified in the MAXTHRESH variable. The benefits of the MIN_MAX rotation mode are the union of the benefits of the MIN and the MAX modes. In other words the user is presented with a more stable map where large rotations are reduced assuming that the variables MINTHRESH and MAXTHRESH are set correctly. The drawbacks for the MIN_MAX modes combine the drawbacks of the MIN and the MAX modes. That means that the burden of finding correct values

for the MINTHRESH and MAXTHRESH is again left on the user. The pseudo code of the MIN_MAX rotation mode is listed in Algorithm 5.

Algorithm 5 MIN_MAX

```

1:  $X \leftarrow \text{UserHeading}()$ 
2:  $Y \leftarrow \text{abs}((X - \text{LASTHEADING}) \bmod 360)$ 
3:  $\text{ROTATERIGHT} \leftarrow ((X - \text{LASTHEADING}) > 0)$ 
4: if  $Y > 180$  then
5:    $Y \leftarrow 360 - Y$ 
6:    $\text{ROTATERIGHT} \leftarrow \text{not } \text{ROTATERIGHT}$ 
7: end if
8: if  $Y \geq \text{MINTHRESH}$  then
9:   if  $Y > \text{MAXTHRESH}$  then
10:    if  $\text{ROTATERIGHT} = \text{true}$  then
11:       $\text{ROTATIONHEADING} \leftarrow (\text{LASTHEADING} + \text{MAXTHRESH}) \bmod 360$ 
12:    else
13:       $\text{ROTATIONHEADING} \leftarrow (\text{LASTHEADING} - \text{MAXTHRESH}) \bmod 360$ 
14:    end if
15:  else
16:     $\text{ROTATIONHEADING} \leftarrow X$ 
17:  end if
18:   $\text{RotateMapToHeading}(\text{ROTATIONHEADING})$ 
19:   $\text{LASTHEADING} \leftarrow \text{ROTATIONHEADING}$ 
20: end if

```

The processing power required by the MIN_MAX mode is a compromise between the one of the MIN and the MAX mode. Fewer rotations are triggered due to the rules of the MIN mode. However, when a rotation is triggered, it is often broken down into multiple ones due to the rules of the MAX mode. Since rotation can occur at any degree, there is no possible optimization. The storage required by the MIN_MAX mode is constant.

5.2.7 ADPT

The ADPT (adaptive) rotation mode is similar to the MIN_MAX scheme with the only difference that the value of the variables MINTHRESH and MAXTHRESH are automatically updated according to the history of the user's past headings. The

benefits of the ADPT mode are the same as those of the MIN_MAX rotation mode with the convenience to the user of not having to figure out what the values of the MINTHRESH and MAXTHRESH variables should be. The drawback of the ADPT mode is that *DynaMap* can only attempt to guess what the values of MINTHRESH and MAXTHRESH values should be. The quality of the guess depends on the number of past user headings that are available for analysis. However, there is a direct trade-off between the speed of adjusting to a new user's behaviour and the accuracy of the guess. In *DynaMap* a total of ten past records are used for analysis. For some users this setting might result in an ADPT mode that does not react quickly enough to changes in their task (e.g. from walking to driving a car), while for some other users this setting might result in an ADPT mode that does not perfectly find the correct MINTHRESH and MAXTHRESH values. The pseudo code of the ADPT rotation mode is listed in Algorithm 6. In line 3 of the algorithm the function `AverageRotAmplitude(USER_PAST_HEADINGS)` simply computes the average rotation by which the map was rotated in all the past rotation amplitudes stored in `USER_PAST_HEADINGS`.

Algorithm 6 ADPT

```

1:  $X \leftarrow \text{UserHeading}()$ 
2:  $\text{insert}(\text{USER\_PAST\_HEADINGS}, X)$ 
3:  $Z \leftarrow \text{AverageRotAmplitude}(\text{USER\_PAST\_HEADINGS})$ 
4:  $\text{MINTHRESH} \leftarrow \min(Z + 5, 60)$ 
5:  $\text{MAXTHRESH} \leftarrow \max(Z - ((Z/60) * Z), 5)$ 
6:  $\text{DoMinMaxRot}(X)$ 

```

Some explanation is in order to justify the formula used to calculate the MINTHRESH and MAXTHRESH values. Both are based on the average difference in heading induced by the user over the past ten recordings. The MINTHRESH value is straightforward since it is set to five degrees higher than this average. That means that only when the difference in user's heading is greater by at least five degrees to what is considered normal, is the map rotated. The MINTHRESH value is capped at 60 degrees to avoid it to become so high that it prevents rotation altogether.

The setting of the MAXTHRESH variable is a bit more complex. First of all,

because the MAXTHRESH variable provides the maximum amplitude for a map rotation, we ensure that it never goes below 5 degrees so as to not unduly prevent rotations to terminate. Once that is understood, a symbolic value of 60 degrees is used to do the rest of the computation. The value of 60 degrees represents the highest average difference in user's heading that we care for. If the average is actually higher than 60, then the minimum value of 5 degrees is simply used. In the other case, when the average is between 0 and 60 degrees, the percentage of the average with respect to 60 degrees is computed, and that value is removed from the average. For example, if the average difference in user's heading is 30 degrees, then MAXTHRESH is set to $30 - ((30/60) * 30)$ which is 15 degrees, or half of 30 degrees. That means in that case, that if a rotation occurs it will be broken down in at least two, if not more. The formula to compute MAXTHRESH has been designed specifically to be used in the MIN_MAX rules where first it is decided through the MIN rules whether rotation occurs at all. If we examine the formula, we notice that as Z approaches 60, MAXTHRESH approaches 0, and as Z approaches 0, MAXTHRESH approaches Z . Since a lower Z represents a more stable user, this formula makes map rotation occur more freely when the user is stable, and tries to break down the rotation into multiple ones when the user is more irregular. This is the behaviour that a user would expect from the ADPT rotation mode.

Similarly to the MIN_MAX rotation mode, the processing power required by the ADPT mode is a compromise between the one of the MIN and the MAX mode, with the addition of the average computation, and the MINTHRESH and MAXTHRESH modes (constant time). The storage requirement consists of the space required to store the user's past heading. In the case of *DynaMap*, this is set to the constant value of ten.

In an attempt to illustrate the differences between all the proposed map rotation algorithms, a comparison chart is provided in Tables 5.2 5.3 5.4, 5.5, and 5.6. Tables 5.2 and 5.3 shows actual computation values for each of the algorithms. As for Tables 5.4 5.5 and 5.6, each algorithm has been run through the same user sequence of locations and headings, consisting first of the user going in a straight line towards the North at 10m/s (in fluctuating headings of 10° and 350°), followed by a stationary

user rotating at angles of 10° , 360° , 20° , 30° , and 90° . Only the last five rotations are shown, when the user is standing in one location.

	DIRECT	R.ANGLE			MIN (10)		MAX (15)	
User Head- ing	Map Rot. Heading	Last Quad	Cur. Quad	Map Rot. Head- ing	Last Head- ing	Map Rot. Head- ing	Last Head- ing	Map Rot. Head- ing
60.1	60.1	-	EAST	90	-	60.1	-	60.1
60.1	60.1	EAST	EAST	90	60.1	60.1	60.1	60.1
60.3	60.3	EAST	EAST	90	60.1	60.1	60.1	60.3
60.3	60.3	EAST	EAST	90	60.1	60.1	60.3	60.3
333.1	333.1	EAST	NORTH	0	60.1	333.1	60.3	45.3
333.1	333.1	NORTH	NORTH	0	333.1	333.1	45.3	30.3
299.8	299.8	NORTH	WEST	270	333.1	299.8	30.3	15.3
302.2	302.2	WEST	WEST	270	299.8	299.8	15.3	0.3
302.2	302.2	WEST	WEST	270	299.8	299.8	0.3	345.3
30.7	30.7	WEST	NORTH	0	299.8	30.7	345.3	0.3
36.2	36.2	NORTH	NORTH	0	30.7	30.7	0.3	15.3
36.2	36.2	NORTH	NORTH	0	30.7	30.7	15.3	30.3
80.3	80.3	NORTH	EAST	90	30.7	80.3	30.3	45.3
80.3	80.3	EAST	EAST	90	80.3	80.3	45.3	60.3
137.3	137.3	EAST	EAST	90	80.3	137.3	60.3	75.3
137.3	137.3	EAST	EAST	90	137.3	137.3	75.3	90.3
140.9	140.9	EAST	EAST	90	137.3	137.3	90.3	105.3
140.9	140.9	EAST	EAST	90	137.3	137.3	105.3	120.3
134.2	134.2	EAST	EAST	90	137.3	137.3	120.3	134.2
132.5	132.5	EAST	EAST	90	137.3	137.3	134.2	132.5

Table 5.2: Algorithms - Sample Data (1/2)

	MIN_MAX		ADPT				
User Head-ing	Last Head-ing	Map Rot. Head-ing	Avg	Min Thresh.	Max Thresh.	Last Head-ing	Map Rot. Head-ing
60.1	-	60.1	60.1	60	5	-	60.1
60.1	60.1	60.1	30.05	35.05	15	60.1	60.1
60.3	60.1	60.1	20.1	25.1	13.37	60.1	60.1
60.3	60.1	60.1	15.08	20.08	11.29	60.1	60.1
333.1	60.1	45.1	29.5	34.5	15	60.1	45.1
333.1	45.1	30.1	24.5	29.58	14.51	45.1	30.59
299.8	30.1	15.1	25.83	30.83	14.71	30.59	15.88
302.2	15.1	0.1	22.9	27.9	14.16	15.88	1.72
302.2	0.1	345.1	20.36	25.36	13.45	1.72	348.27
30.7	345.1	0.1	27.17	32.17	14.87	348.27	3.14
36.2	0.1	15.1	25.2	30.2	14.62	3.14	17.76
36.2	15.1	30.1	19.74	24.74	13.24	17.76	17.76
80.3	30.1	45.1	23.75	28.75	14.35	17.76	32.1
80.3	45.1	60.1	23.73	28.73	14.34	32.1	46.45
137.3	60.1	75.1	28.91	33.91	14.98	46.45	61.43
137.3	75.1	90.1	20.98	25.98	13.64	61.43	75.07
140.9	90.1	105.1	21.31	26.31	13.74	75.07	88.81
140.9	105.1	120.1	18.28	23.28	12.71	88.81	101.53
134.2	120.1	134.2	18.67	23.67	12.86	101.53	114.39
132.5	134.2	134.2	18.83	23.83	12.92	114.39	114.39

Table 5.3: Algorithms - Sample Data (2/2)

	10°	360°	20°	30°	90°
DIRECT					
R.ANGLE					

Table 5.4: Differences Between Map Rotation Algorithms (1/3)

	10°	360°	20°	30°	90°
MIN
MAX

Table 5.5: Differences Between Map Rotation Algorithms (2/3)

	10°	360°	20°	30°	90°
MIN_MAX					
ADPT					

Table 5.6: Differences Between Map Rotation Algorithms (3/3)

5.3 Evaluation

Each algorithm offers something unique in terms of benefits and drawbacks. Although one might be tempted to judge rapidly the usefulness of each algorithm, our experience has shown that often our assumption can be wrong. Therefore there needs to be a formal evaluation, based on well-established techniques for measuring the usability of each rotation mode. Care has to be taken when doing such measurements due to impossibility to fully control the environment. There are also a number of factors that can only be measured in subjective manner, such as user preferences, for example. In order to provide for a better testing environment, and to take advantage of existing knowledge in performing these types of evaluation, we teamed-up with the Human Oriented Technology (HOT) lab at Carleton University.

Through numerous discussions between the various members of this joint committee, we have established a formal process to evaluate the map rotation features of *DynaMap*. We provide a summary of the procedure followed and of the results obtained in the following sections. However, we refer to the work of Narasimhan [76] for the complete report on the usability study.

5.3.1 Hypothesis

We expect the R.ANGLE algorithm to be the least preferred algorithm by the user due to size of each rotation: ninety degrees. We expect the ADPT rotation mode to be the best suited map rotation algorithm in a general setting. However, due to the number of special cases where a different algorithm might do better than ADPT, we suspect that a more encompassing solution should be proposed where the ADPT algorithm would be used by default, but the possibility of choosing another algorithm would be left to the user.

5.3.2 Lab Study

A lab environment allows the usability engineer to have more control over the experiment. We decided to first conduct a lab study of *DynaMap* to obtain an initial

understanding of the issues in evaluating such software. We wanted to use this opportunity to better prepare for a more in-depth field study. This strategy has also been confirmed by Goodman et al. in their report of field usability techniques for mobile guides [45].

Aim

The aim of the lab study was to see which algorithms can be individually distinguished by the user in an attempt to determine which ones would be more relevant to test in a field study setting. At the same time we hoped that we could get some indication of which algorithm seems to be preferred by the users from their limited exposure to the system.

Procedure

The lab experiment consisted of presenting to thirty-four participants a set of pre-recorded sequences of a user walking around the campus of Carleton University. The participants were placed in front of a desktop computer, inside one of the experiment's rooms in the HOTlab facilities. *DynaMap* was used, in its simulated environment, to play the sequences. In the simulated environment *DynaMap* runs on a PDA simulator for Win32 desktop machines, using GPS data previously recorded on text files to simulate a moving user.

The sequences were a cross product of eight simulated paths of about ten seconds each, with the six map rotation algorithms, for a total of forty-eight sequences. After each sequence the user was asked two questions:

- How well did the map rotation match the movements of the holder of the PocketPC?
- How confident are you in indicating the building or landmark you are now facing?

The user answered each question by setting the position of a sliding scale out of the ninety-nine available positions (see Figure 5.1).

Figure 5.1: The Lab-Study Questions

The eight pre-recorded paths were chosen based on two criteria: 1- the location on campus where the path was recorded, and 2- the pattern of movement of the path. We chose four locations: one surrounded by buildings, one near a river, one near a rail track and road, and one in the middle of a field. We feel that these four different locations provided a good coverage of the different artifacts a user might typically find around him/herself on a map, hence reducing the probability that the type of map displayed becomes a factor in the study. The two types of pattern of movement used were curved and zigzag. In a curved pattern, the path does not include sharp rotations. The zigzag pattern sits at the other end of the spectrum and therefore includes multiple sharp rotations. This allowed us to compare the impact on the user of the different map rotation algorithms based on the pattern of movement displayed. We recorded in each location two paths, one for each of the pattern of movement. Due to the difficulty of recording perfect path sequences to be used in a lab study, we developed a tool that allowed us to fine tune them once saved into a text file. This tool, which we called NMEAGen, can be seen in Figure 5.2.

Figure 5.2: NMEA 0183 Data Generator

Results of Lab Study by HOTlab in [76]

We present a brief summary of the results obtained. The lab study and its analysis were carried out by the HOTlab. For details of the analysis the reader is referred to [76]. Our contributions were the software preparation to carry out the study and participation in the design process for the study. In the results the DIRECT rotation mode is sometime referred to as “No Threshold”. The two terms are used intermittently throughout this Section and refer to the same rotation scheme.

The compiled results to the first question can be seen in Figure 5.3. We see that the curved pattern of movement produced more varied results than that of the zigzag pattern of movement. The 6X8 repeated measures ANOVA show that no significant main effect of algorithm was found. However, a significant interaction between algorithm and sequence $F(29, 956) = 1.64, P < .001$ was observed.

Figure 5.3: Means for Algorithm and Sequence for Question 1 [76]

We can see the results of four paired-related sample t-tests in Table 5.7. A Bonferroni correction was used, P critical = $.05/4 = .0125$. The Bonferroni correction is a conservative statistic method used to reduce the chances of spurious positives when multiple comparisons are being performed simultaneously. When looking at the outcome of multiple comparisons as done here, the chances of not finding something significant decreases ($.95 * .95 * \dots * .95$). Inversely the chances of finding anything significant therefore grow. The Bonferroni correction consists of lowering the critical significance level, normally .05 by dividing it by the number of comparisons performed, and to use the result as the new critical significance level [12].

The tests in Table 5.7 indicate that the right angle and the no threshold algorithms for the curve o-train, curve residence, and curve Loeb sequences were significantly different $t(33) = -3.236, p < .01, t(33) = -3.747, t(33) = -2.808$. The right angle algorithm was also significantly different from the zigzag Loeb sequence, $t(33) = -2.632, p < .01$.

The curved pattern of movement was then made one group and was compared to the zigzag pattern of movement. We can see the results in Figure 5.4. The 6X2 repeated measures ANOVA with Bonferroni correction ($p = .05/4 = .0125$) shows a

	Paired Differences			T	df	Sig.(2-tailed)
Paired Comparison	Mean	Std. Deviation	Std. Error Mean			
Right Angle, Curve O-train - No Threshold, Curve O-train	-16.5	29.731	5.0988	-3.23	33	.003
Right Angle, Curve Residence - No Threshold, Curve Residence	-18.5	28.880	4.9529	-3.74	33	.001
Right Angle, Curve Loeb - No Threshold, Curve Loeb	-12.1	25.159	4.3148	-2.80	33	.008
Right Angle, Zigzag Loeb - Max Only, Zigzag Loeb	-14.5	32.120	5.5085	-2.63	33	.013

Table 5.7: Paired Related Samples T-Tests for Sequences and Algorithm for Question 1 [76]

significant interaction between algorithm and sequence $F(5, 165) = 3.876, p < .002$.

The paired related t-tests shown in Figure 5.8 show that when curved sequences were displayed using the right angle algorithm, they were significantly rated lower compared to when a curved sequence was displayed using a no threshold algorithm, $t(33) = -4.48, p < .0001$, or min only algorithm, $t(33) = -3.23, p < .003$.

Figure 5.4: Mean Ratings for Curve and Zigzag Sequences and Algorithms for Question 1 [76]

		Paired Differences			T	df	Sig.(2-tailed)
Comparisons		Mean	Std. Deviation	Std. Error Mean			
Pair 1	Right Angle, Curve - No Threshold, Curve	-13.55	17.64	3.02	-4.48	33	.000
Pair 2	Right Angle, Curve - Min only, Curve	-10.03	18.08	3.10	-3.23	33	.003
Pair 3	Right Angle, Zigzag - Min Max, Zigzag	-4.44	16.97	2.91	-1.52	33	.137
Pair 4	Right Angle, Zigzag - Max Only, Zigzag	-4.67	19.25	3.30	-1.41	33	.166

Table 5.8: Pair Related Samples T-Tests Comparing Curve and Zigzag Sequences by Algorithm for Question 1 [76]

The results for question two were not as revealing. As we can see in Figure 5.5, no clear algorithm/sequence produced significantly different ratings. The same occurs when we combine all the curved pattern of movement sequences and all the zigzag pattern of movement sequences together. These results can be seen in Figure 5.6.

Figure 5.5: Means for Algorithm and Sequence for Question 2 [76]

Discussion

The lab study found that there are significant differences between the ratings of each algorithm between sequences of curved vs. zigzag pattern of movement. The no threshold performed better for curved sequences, and the max threshold only algorithm did best for zigzag sequences. This may be due to the fact that curved sequences tend to not generate large map rotations, hence do not confuse the user while always keeping the map accurately oriented with the user. In the zigzag case, the map tends to be rotated much more sharply. The max threshold only algorithm ensures that large map rotations are broken down into smaller ones, hence reducing the confusion induced to the user. It is important to note that for both types of patterns of movement, the adaptive algorithm achieved the third highest scores. This indicates that the adaptive algorithm may be a good compromise in the general case. We note that the sequences were about ten seconds in length, which may be too short

Figure 5.6: Mean Ratings for Curve and Zigzag Sequences for Question 2 [76]

for the adaptive algorithm to learn the pattern of movement, and therefore to show its true value. Such hypothesis should be confirmed or contradicted in a field study.

Finally, as per our hypothesis the results indicate the right angle algorithm scored significantly lower on both sequence types of pattern or movement. Therefore there would be little value-added in including the right angle map rotation in a field study.

5.3.3 Field Study

A field study is currently under preparation and therefore the results could unfortunately not be included in this document. In order to obtain more accurate results on the value of each map rotation algorithm in a real-life environment, we made the decision to run the field study as close as possible to the beginning of a new semester. This will allow us to find many participants (students) with little or no previous knowledge of the campus. These participants will therefore have a bigger incentive to rely on *DynaMap* to execute the requested tasks. The results of the field study will be presented as a separate publication.

5.4 Summary

We have shown the difficulty involved in displaying in a useful way to the user a map rotated in real-time so that it matches the user's orientation. We have shown that there exist multiple solutions to such problem by using different map rotation algorithms, namely *STATIC*, *DIRECT*, *R.ANGLE*, *MIN*, *MAX*, *MIN_MAX*, and *ADPT*. We described the algorithms in terms of their efficiency, memory usage, but more specifically in terms of their advantages and disadvantages from the point of view of the user's perception. Finally we have started a formal evaluation process of each algorithm used in different contexts, in partnership with usability experts of the HOT lab from Carleton University. We have seen from the lab study results that some algorithms, namely *DIRECT*, *MIN*, *MAX*, *MIN_MAX*, and *ADPT*, are more relevant than others in terms of their impact on user perceptions, depending on the type of user movement. We have decided to further study these algorithms in a more real life environment by executing a field study experiment.

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis we researched issues for the design and prototype implementation of a mobile and context-aware pedestrian navigation application: *DynaMap*. We designed and implemented an application that provides to its user personalized map visualization and navigation services. *DynaMap*, which has been used by a number of users, runs on a PocketPC^(TM) PDA with the help of a GPS, an electronic compass, and a spatial data server. We took into consideration multiple constraints imposed by the system such as the real-time nature of the application, the limited physical resources available (processing power, memory, battery, screen real estate), and the usability issues imposed by a mobile user. In particular we addressed the usability issues of real-time map rotation based on the user's orientation. We proposed different map rotation algorithms and explained their advantages and disadvantages in terms of user perception. Finally we provided in collaboration with the Human Oriented Technology (HOT) lab at Carleton University a lab usability study to find out empirically in which situation (user travel pattern, user's different surroundings, etc) each algorithm performs better.

We designed a Context Management System (CMS) that can support any type of user context that can easily be integrated into an application's logic, that can store and exchange context, and that can switch between the context of different users. We

showed how to use Dey's *Context Toolkit* [28] within the Context Manager component of *DynaMap* in order to reuse the context acquisition and context modeling solutions already available. Finally we provided the framework on how to achieve contextual maps and contextual shortest-path computations in *DynaMap*.

Based on generic requirements we proposed a system's design that favours architecture reuse. We relied on principles of code modularity and abstraction, as well as on a clear separation of concerns for each of the provided autonomous components. It should be possible to reapply *DynaMap*'s design in other mobile and context-aware GIS applications such as those to be developed for the IMA framework.

6.1 Future Work

More work needs to be spent o properly integrate more of the user's context in future applications. Each time we find new user context to be considered by an application we should evaluate and study the best way to do so from a usability point of view. One lesson learned with *DynaMap* was to rely as much as possible on the results of real-life usability experiments to properly understand the perceived impact on the user due to the context-aware behaviour of an application. A specific example of this is the contextual shortest path feature of *DynaMap*, which calculates the shortest path to a user's provided destination based on contextual data (user's preferred path, time of day, weather, etc). Such shortest path computation is undeniably something that users want, however it is less clear specifically how this should be achieved. It would be interesting to get answers to the following questions for example:

- How should we let the user specify which part of his/her context should be part of the shortest path computations (turn them on/off, use weights, global vs. one-time only settings, user provided settings or automatically learned, etc)?
- How to best provide the shortest path results (provide only one path, provide multiple paths based on different criteria's/weights, provide partial paths then recompute based on the new context or provide the entire path, etc)?

- How to visualize the shortest path results (graphically overlaid on top of the map with or without audio navigational instructions, using colour/boldness/movement to indicate emphasis/danger/alternatives, etc)?

Once the answers to the above questions are obtained one could integrate them into *DynaMap* and improve the usability of the current feature. In terms of the map rotation feature of *DynaMap*, it would be interesting to further study different formula for the ADPT rotation mode and see if we can achieve higher usability results in general. However we should first obtain the results of the field-study to see whether it confirms those of the lab experiment.

In terms of context integration we should further study the concept of one common Context Description Language (CDL) that can be used across applications. The one proposed in Chapter 4 can be used as a starting point and then one needs to see what is required to make it an all-encompassing solution. The issues here are to determine specifically for what purposes the CDL is used (user profiling, context data exchange, context distribution, context archiving/retrieving, etc) and how to use CDL to facilitate context usage across applications (a meta-language to publish the context to the external world, a query language to retrieve information about the available context, etc).

Finally, based on the lessons learned and on the solutions proposed throughout the development of *DynaMap* the IMA framework has to be brought into life. The framework will simplify the development of future applications by providing services that are common to all (e.g. finding out about available spatial data servers). Many of the solutions proposed in the design of *DynaMap* can be reused here and applied more widely.

Bibliography

- [1] *Energy-performance trade-offs for spatial access methods on memory-resident data*, November 2002.
- [2] Google maps. <http://maps.google.com/>, 2005.
- [3] Mapquest. <http://www.mapquest.com/>, 2005.
- [4] Pankaj K. Agarwal, Lars Arge, T. M. Murali, Kasturi R. Varadarajan, and Jeffrey Scott Vitter. I/o-efficient algorithms for contour-line extraction and planar graph blocking. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 117–126, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.
- [5] Daniel Ashbrook and Thad Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal Ubiquitous Comput.*, 7(5):275–286, 2003.
- [6] Yohan Baillot, Dennis Brown, and Simon Julier. Authoring of physical models using mobile computers. In *Proceedings of the 5th IEEE International Symposium on Wearable Computers*, page 39. IEEE Computer Society, 2001.
- [7] L. Barkhuus and A. K. Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. In *Proceedings of UbiComp 2003*, pages 149–156, Seattle, Washington, 2003. Springer.

- [8] Louise Barkhuus. Context information vs. sensor information: A model for categorizing context in context-aware mobile computing. In *Symposium on Collaborative Technologies and Systems*, pages 127–133, San Diego, CA, 2003.
- [9] Martin Bauer, Christian Becker, and Kurt Rothermel. Location models from the perspective of context-aware applications and mobile ad hoc networks. *Personal Ubiquitous Comput.*, 6(5–6):322–328, January 2002.
- [10] Jörg Baus, Antonio Krüger, and Wolfgang Wahlster. A resource-adaptive mobile navigation system. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 15–22, San Francisco, California, USA, 2002. ACM Press.
- [11] Michael Beigl, Tobias Zimmer, and Christian Decker. A location model for communicating and processing of context. *Personal Ubiquitous Comput.*, 6(5–6):341–357, 2002.
- [12] J. Martin Bland and Douglas G. Altman. Multiple significance tests: the bonferroni method. *British Medical Journal*, 310(6973):170, January 1995.
- [13] Bluetooth. The official bluetooth membership site. <https://www.bluetooth.org/>, 2004.
- [14] Stephen Brewster. Overcoming the lack of screen space on mobile computers. *Personal Ubiquitous Comput.*, 6(3):188–205, May 2002.
- [15] Jochen Burkhardt. *Pervasive Computing: Technology and Architecture of Mobile Internet Applications*. Addison-Wesley, November 2001.
- [16] R. Cáceres, J. Donham, B. Fitterman, D. Joerg, M. Smith, and T. Vetter. Mobile computing technology at vindigo. *IEEE Wireless Communications*, 9(1):50–53, February 2002.
- [17] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Dept. of Computer Science, Dartmouth College, Dept. of Computer Science, Dartmouth College, November 2000.

- [18] Keith Cheverst, Nigel Davies, Keith Mitchell, and Christos Efstratiou. Using context as a crystal ball: Rewards and pitfalls. *Personal Ubiquitous Comput.*, 5(1):8–11, 2001.
- [19] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Experiences of developing and deploying a context-aware tourist guide: the guide project. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 20–31, Boston, Massachusetts, United States, 2000. ACM Press.
- [20] Keith Cheverst, Keith Mitchell, and Nigel Davies. Exploring context-aware information push. *Personal Ubiquitous Comput.*, 6(4):276–281, 2002.
- [21] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *Symposium on Discrete Algorithms*, pages 139–149, 1995.
- [22] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *Symposium on Discrete Algorithms*, pages 139–149, 1995.
- [23] US Federal Communication Commission. Third generation wireless systems. <http://www.fcc.gov/3G/>, 2004.
- [24] OASIS UIML Technical Committee. Uiml. <http://www.uiml.org/index.php>, 2004.
- [25] Barbara Corona and Stephan Winter. Approaches to an ontology for pedestrian navigation services. Technical report, Institute for GeoInformation, Technical University of Vienna, Vienna, Austria, 2001.
- [26] Barbara Corona and Stephan Winter. Datasets for pedestrian navigation services. In J. Strobl, T. Blanschke, and G. Griesebner, editors, *Proceedings of the AGIT Symposium*, pages 84–89, Salzburg, Austria, 2001.

- [27] Nigel Davies and Hans-Werner Gellersen. Beyond prototypes: Challenges in deploying ubiquitous systems. *IEEE Pervasive Computing*, 1(1):26–35, January 2002.
- [28] Anind. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.
- [29] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.
- [30] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [31] Jerome E. Dobson and Peter F. Fisher. Geoslavery. *Technology and Society Magazine, IEEE*, 22(1):47–52, 2003.
- [32] Thomas Erickson. Some problems with the notion of context-aware computing. *Commun. ACM*, 45(2):102–104, 2002.
- [33] Shaojun Feng and Choi Look Law. Assisted gps and its impact on navigation in intelligent transportation systems. In *Proceedings of Intelligent Transportation Systems, 2002.*, pages 926–931, Singapore, Singapore, 2002. IEEE Press.
- [34] Sebastian Fischmeister. Mobile software agents for location-based systems. *Agents and Software Engineering*, 2592:226–239, 2002.
- [35] Sebastian Fischmeister, Guido Menkhaus, and Wolfgang Pree. Musa-shadow: concepts, implementation, and sample applications a location-based service supporting multiple devices. In *Proceedings of the Fortieth International Conference on Tools Pacific*, pages 71–79, Sydney, Australia, 2002. Australian Computer Society, Inc.
- [36] Adrian Friday, Nigel Davies, and Elaine Catterall. Supporting service discovery, querying and interaction in ubiquitous computing environments. In *Proceedings*

- of the 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 7–13, Santa Barbara, California, United States, 2001. ACM Press.
- [37] Georg Gartner. Location-based mobile pedestrian navigations services - the role of multimedia cartography. In *Proceedings of the International Joint Workshop on Ubiquitous, Pervasive and Internet Mapping*, Tokyo, Japan, 2004.
- [38] Georg Gartner. Private communication. 2004.
- [39] Vincent Geake. Web mapping: Get mobile, it's the place to be. <http://www.geoplace.com/ge/2000/0900/0900mob.asp>, 2000.
- [40] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7), 1997.
- [41] Geoworld. Gis world interview - roger tomlinson: The father of gis. <http://www.geoplace.com/gw/1996/0496/0496feat2.asp>, 2005.
- [42] Eric Gervais. The dynamicmap application. B.c.s. honour's project report, Carleton University, Ottawa, Canada, April 2002.
- [43] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165, Vancouver, British Columbia, 2005. Society for Industrial and Applied Mathematics.
- [44] Andrew V. Goldberg and Renato F. Werneck. Computing point-to-point shortest paths from external memory. In *Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, Vancouver, Canada, January 2005. SIAM.
- [45] Joy Goodman, Stephen Brewster, and Philip Gray. Using field experiments to evaluate mobile guides. In B. Schmidt-Belz and K. Cheverst, editors, *Proceedings of the 3rd Workshop on HCI in Mobile Guides*, Glasgow, Scotland, September 2004.

- [46] IEEE P802.11 Working Group. Ieee 802.11 wireless local area networks. <http://grouper.ieee.org/groups/802/11/>, 2004.
- [47] Mark Hampe and Birgit Elias. Integrating topographic information and landmarks for mobile navigation. In G. Gartner, editor, *Proceedings of the Symposium on Location Based Services & Telecartography*, pages 147–155, Vienna University of Technology, January 2004.
- [48] Lars Harrie, L. Tiina Sarjakoski, and Lassi Lehto. A variable-scale map for small-display cartography. In *Proceedings of the Joint International Symposium on GeoSpatial Theory, Processing and Applications*, page CDRom, Ottawa, Canada, July 2002.
- [49] P.F. Hartigan. Transport and processing of perishable information: Essential infrastructure for location based services. In *Third International Conference on 3G Mobile Communication Technologies*, pages 345–349, Kilrush, Ireland, May 2002. IEEE Press.
- [50] HAVi. Home audio video interoperability. <http://www.havi.org/>, 2004.
- [51] Jason I. Hong and James A. Landay. A context/communication information agent. *Personal Ubiquitous Comput.*, 5(1):78–81, 2001.
- [52] Martin Hope, Tom Chrisp, and Nigel Linge. Improving co-operative working in the utility industry through mobile context aware geographic information systems. In 135-140, editor, *Proceedings of the eighth ACM international symposium on Advances in geographic information systems*, Washington, D.C., United States, 2000. ACM Press.
- [53] Andy Hopper. Sentient computing. *The Royal Society Clifford Paterson Lecture*, 358(2000):2349–2358, 1999.
- [54] R. Hull, P. Neaves, and J. Bedfords-Roberts. Towards situated computing. In *First International Symposium on Wearable Computing*, pages 146 – 153, Cambridge, MA USA, 1997. IEEE.

- [55] David Hutchinson, Anil Maheshwari, and Norbert Zeh. An external memory data structure for shortest path queries (extended abstract). *Lecture Notes in Computer Science*, 1627:51–62, 1999.
- [56] iButton. Maxim ibutton. <http://www.ibutton.com>, 2004.
- [57] IETF. Service location protocol. <http://www.ietf.org/html.charters/svrloc-charter.html>, 2004.
- [58] Panasonic in-car technology. Website. http://www.panasonic-europe.com/car/index_frame.asp, 2004.
- [59] Jadwiga Indulska and Peter Sutton. Location management in pervasive systems. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 143–151, Adelaide, Australia, 2003. Australian Computer Society, Inc.
- [60] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
- [61] Lai Jin and Tatsuo Miyazawa. Mrm server: a context-aware and location-based mobile e-commerce server. In *Proceedings of the 2nd international workshop on Mobile commerce*, pages 33–39. ACM Press, 2002.
- [62] Gareth J.F. Jones and Peter J. Brown. Context-aware retrieval for ubiquitous computing environments. *Mobile and ubiquitous information access, Springer Lecture Notes in Computer Science*, 2954:227–243, 2002.
- [63] Eija Kaasinen. User needs for location-aware mobile services. *Personal Ubiquitous Comput.*, 7(1):70–79, 2003.
- [64] Alan Kay. Computers, networks, and education. *Scientific American*, pages 138–148, September 1991.
- [65] Panu Korpipää, Jani Mäntyjärvi, Juha Kela, Heikki Keränen, and Esko-Juhani Malm. Managing context information in mobile devices. *Pervasive Computing, IEEE*, 2(3):42–51, 2003.

- [66] Uwe Kubach and Kurt Rothermel. Exploiting location information for infostation-based hoarding. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 15–27, Rome, Italy, 2001. ACM Press.
- [67] Anil Maheshwari, Pat Morin, and Jörg-Rüdiger Sack. Progressive tins: Algorithms and applications. In *5th ACM international workshop on advances in GIS*, pages 24–29, Las Vegas, USA, November 1997. ACM-GIS.
- [68] Natalia Marmasse and Chris Schmandt. A user-centered location model. *Personal Ubiquitous Comput.*, 6(5–6):318–321, 2002.
- [69] Andrew J. May, Tracy Ross, Steven H. Bayer, and Mikko J. Tarkiainen. Pedestrian navigation aids: information requirements and design implications. *Personal Ubiquitous Comput.*, 7(6):331–338, December 2003.
- [70] Joseph F. McCarthy. Pocket restaurantfinder: A situated recommender system for groups. In *orkshop on Mobile Ad-Hoc Communication at the 2002 ACM Conference on Human Factors in Computer Systems*, Minneapolis, USA, April 2002. ACM Press.
- [71] Joseph F. McCarthy and Theodore D. Anagnost. Eventmanager: Support for the peripheral awareness of events. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 227–235, Bristol, UK, 2000. Springer-Verlag.
- [72] Joseph F. McCarthy, David H. Nguyen, Al Mamunur Rashid, and Suzanne Soroczak. Proactive displays & the experience ubicomp project. *SIGGROUP Bull.*, 23(3):38–41, 2002.
- [73] Microsoft. Universal plug and play. <http://www.havi.org/>, 2004.
- [74] Sun Microsystems. Jini network technology. <http://www.sun.com/software/jini/>, 2004.

- [75] Katashi Nagao and Yasuharu Katsuno. Agent augmented community: Human-to-human and human-to-environment interactions enhanced by situation-aware personalized mobile agents. In *Community Computing and Support Systems, Social Interaction in Networked Communities [the book is based on the Kyoto Meeting on Social Interaction and Communityware, held in Kyoto, Japan, in June 1998]*, pages 342–358. Springer-Verlag, June 1998.
- [76] Sheila Narasimhan. Perceptions and evaluations of digital map rotation schemes in pdas. Technical report, Carleton University, Ottawa, Canada, April 2005.
- [77] Laurence Nigay and Phil Gray. Architecture logicielle conceptuelle pour la capture de contexte. In *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l’Interaction Homme-Machine)*, pages 211–214, Poitiers, France, 2002. ACM Press.
- [78] Annu-Maaria Nivala. Interruptions in mobile map environments. Technical report, Helsinki Institute for Information Technology, Helsinki, Finland, Spring 2004.
- [79] Mark H. Nodine, Michael T. Goodrich, and Jeffrey Scott Vitter. Blocking for external graph searching. In *PODS ’93: Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 222–232, New York, NY, USA, 1993. ACM Press.
- [80] Donald A. Norman. *The Invisible Computer*. The MIT Press, 1998.
- [81] OMG. Uml: Unified modeling language specification, version 2.0. <http://www.uml.org>, 2005.
- [82] Sharon Oviatt. Multimodal interfaces for dynamic interactive maps. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 95–102, Vancouver, British Columbia, Canada, 1996. ACM Press.
- [83] Jade Agent Platform. Website. <http://jade.tilab.com/>, 2004.

- [84] Salil Pradhan. Semantic location. *Personal Ubiquitous Comput.*, 4(4):213–216, August 2000.
- [85] Cliff Randell and Henk Muller. The shopping jacket: Wearable computing for the consumer. *Personal Ubiquitous Comput.*, 4(4):241–244, August 2000.
- [86] Tumasch Reichenbacher. The world in your pocket - towards a mobile cartography. In *Proceedings of the 20th International Cartographic Conference*, Beijing, China, 2001.
- [87] Qun Ren and Margaret H. Dunham. Using semantic caching to manage location dependent data in mobile computing. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 210–221, Boston, Massachusetts, United States, 2000. ACM Press.
- [88] Thomas Rist and Patrick Brandmeier. Customizing graphics for tiny displays of mobile devices. *Personal Ubiquitous Comput.*, 6(4):260–268, September 2002.
- [89] Jörg Rüdiger Sack and Doron Nussbaum. The intelligent map agent research project. School of Computer Science, Carleton University, 2005.
- [90] Urgence Santé. Urgence santé. <http://www.urgences-sante.qc.ca/>, 2004.
- [91] Bill N. Schilit, Norman I. Adams, and Roy Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, December 1994. IEEE Press.
- [92] A. Schmidt, M. Beigl, and H.W. Gellersen. There is more to context than location. *Computers & Graphics Journal*, 23(6):893–902, December 1999.
- [93] Albrecht Schmidt. *Computing in Context*. PhD thesis, Lancaster University, England, U.K, November 2002.
- [94] Todd Simcock, Stephen Peter Hillenbrand, and Bruce H. Thomas. Developing a location based tourist guide application. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 177–183, Adelaide, Australia, 2003. Australian Computer Society, Inc.

- [95] Molly E. Sorrows and Stephen C. Hirtle. The nature of landmarks for real and electronic spaces. In *Proceedings of the International Conference on Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science*, pages 37–50. Springer-Verlag, 1999.
- [96] Dan Sperber and Deirdre Wilson. *Relevance, Communication and Cognition*. Blackwell, second edition, 1995.
- [97] Franz Steiner. *Multilingual Dictionary of Technical Terms in Cartography.*, page 568. ICA, International Cartographic Association, Wiesbaden, 1973.
- [98] Arunas Stockus, Alain Bouju, Frédéric Bertrand, and Patrice Boursier. Integrating gps data within embedded internet gis. In *Proceedings of the seventh ACM international symposium on Advances in geographic information systems*, pages 134–139, Kansas City, Missouri, United States, 1999. ACM Press.
- [99] Riku Suomela, Kimmo Roimela, and Juha Lehtikainen. The evolution of perspective view in walkmap. *Personal Ubiquitous Comput.*, 7(5):249–262, October 2003.
- [100] Mikko Tarkiainen, Kimmo Kauvo, Jari Kaikkonen, and Harri Heine. Simple turn-by-turn route guidance for pedestrians. In *8th World Congress on Intelligent Transport Systems*, Sydney, Australia, October 2001.
- [101] Bruce Thomas, Wayne Piekarski, David Hepworth, Bernard Gunther, and Victor Demczuk. A wearable computer system with augmented reality to support terrestrial navigation. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 168. IEEE Computer Society, 1998.
- [102] TinyStocks. The pocketpc companion to your gps. <http://www.tinystocks.com/gps.html>, 2004.
- [103] Norman G. Vinson. Design guidelines for landmarks to support navigation in virtual environments. In *Proceedings of the SIGCHI conference on Human*

- factors in computing systems*, pages 278–285, Pittsburgh, Pennsylvania, United States, 1999. ACM Press.
- [104] Kirsi Virrantaus, Jouni Markkula, Artem Garmash, Vagan Terziyan, Jari Veijalainen, Artem Katanosov, and Henry Tirri. Developing gis-supported location-based services. In *Proceedings of the Second International Conference on Web Information Systems Engineering.*, pages 66–75, Helsinki, Finland, December 2001. IEEE Press.
- [105] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.
- [106] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.
- [107] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(30):94–104, 1991.
- [108] Mark Weiser and John Seely Brown. The coming of age of calm technology. *Power Grid Journal*, 1(01), October 1996.
- [109] Jakub Weichert. The disappearing computer, 2000.
- [110] Terry Winograd. Architectures for context. *Human-Computer Interaction*, 16(2–4):401–429, December 2001.
- [111] Stephan Winter. *Route Specifications with a Linear Dual Graph*. Berlin, 2002.
- [112] Stephan Winter and Martin Tomko. Shifting the focus in mobile maps. In *International Joint Workshop on Ubiquitous, Pervasive and Internet Mapping*, Tokyo, Japan, September 2004.
- [113] Michael Wooldridge and Nicholas R. Jennings. *Intelligent agents: Theory and practice*, 1994.

- [114] Stephen S. Yau and Fariaz Karim. An adaptive middleware for context-sensitive communications for real-time applications in ubiquitous computing environments. *Real-Time Syst.*, 26(1):29–61, 2004.

APPENDIX A

LITERATURE BREAKDOWN

BY TOPICS OF INTERESTS

Acquisition process of context

- Methods used to acquire context
 - [28] [65] [93] [68] [114] [18] [51] [10] [19] [75] [17] [91] [54] [53] [71] [77]

Acquisition process of location

- Methods used to acquire the user's location
 - [5] [63] [61] [11] [9] [52] [85] [10] [94] [59] [35] [33] [34] [104] [84] [19] [98] [75] [86] [99] [101] [17] [105] [54] [53] [37] [106] [56]

Agent systems

- Systems that use fully or in part agents to accomplish their goal(s)
 - [61] [35] [34] [75] [83] [62] [40] [113] [96]

Context-aware project implementations

- Implementations of context-aware applications
 - [28] [5] [65] [93] [63] [68] [11] [18] [51] [6] [27] [10] [19] [75] [17] [91] [54] [53] [108] [109] [72] [70] [62] [71] [20]

Definition of Context-Awareness

- What defines context, and what defines an application that is context-aware?
 - [7] [28] [65] [93] [29] [114] [17] [91] [53] [108] [109] [8] [96]

Different learning, or pattern recognition projects attempted

- Context has a lot to do with interpretation. How does a system interpret different information correctly. What kind of learning, or pattern recognition algorithms were used to improve context understanding
 - [5] [65] [68] [75]

Discovering meaningful places or routes

- Using location as context, algorithms that try to discover meaningful locations or places for the user.
 - [5] [68] [47]

Energy preserving methods

- Mobile devices need to be self sufficient, and that includes from a power point of view. Can software be designed in such a way to reduce power consumption?
 - [93] [1]

Future vision of a ubiquitous world

- Mobile and ubiquitous computing was first introduced as a vision of the computing future. Since then, more and more people have shared their vision of such a world.
 - [93] [63] [18] [27] [9] [104] [86] [99] [17] [107] [105] [80] [15] [53] [108] [109] [78] [64]

Geographical Information Systems

- Deals with GIS. Anything that combines computing with geography.
 - [82] [6] [16] [52] [10] [33] [49] [104] [98] [86] [99] [111] [26] [37] [112] [47] [39]

Map Display

- Humans are quite sensitive to how maps are drawn. What tricks were used to draw maps the most efficiently, that is both feasible for the computer, and pleasing to the user.
 - [6] [16] [10] [94] [49] [104] [86] [99] [48] [78] [37] [112] [38]

Navigation

- Anything that touches the notion of something trying to get somewhere.
 - [6] [10] [86] [95] [103] [100] [69] [99] [101] [58] [111] [25] [26] [37] [112] [47]

Pattern Recognition

- This topic is about the classical pattern recognition field (pattern classification, or clustering techniques).
 - [5] [65] [68] [60] [75]

Previous work in location aware computing

- Implementation of location based systems
 - [28] [63] [61] [6] [16] [27] [9] [52] [85] [10] [94] [59] [35] [33] [49] [34] [104] [84] [19] [98] [75] [99] [101] [17] [105] [54] [53] [102] [72]

Psychology

- The cognitive side of computers. How do human behave, react.
 - [7] [28] [82] [95] [103] [100] [69] [25] [78]

Security/Privacy

- Collaboration between two mobile users, as well as the emergence of context-aware applications, brings many interesting security, privacy issues to the forefront.
 - [7] [63] [59] [31] [17] [105]

Service discovery

- Mobility is supposed to increase the number of services that we will take part of. How do one discover those services and then how do one maintain such discovery system in an important question.
 - [28] [34] [36] [50] [74] [57] [73]

Small display size

- Mobile devices are often small. This brings the issue of having to develop advanced applications, which are limited in the size of things they can display.
 - [63] [6] [16] [10] [94] [35] [104] [86] [99] [88] [14] [48] [78] [112]

Software engineering practice in the context of MUC or CA

- Systems that were developed keeping in mind maintainability, performance, flexibility, robustness, etc.
 - [114] [94] [34] [24]

Standards

- Anything that tries to bring a new standard.
 - [28] [93] [11] [9] [59] [104] [86] [25]

Usability of MUC, CA applications

- Overall and thorough studies
 - [7] [28] [93] [63] [18] [82] [6] [27] [19] [99] [14] [8] [32] [45] [78] [20]

Wireless data transmission optimization

- To be able to deliver information wirelessly is a great benefit for the everyday user. However wireless data transmissions are unique in their unreliability. How to deal with this problem is discussed here.
 - [11] [16] [49] [104] [98] [66] [87] [54] [53] [62] [23] [46] [13] [67]

APPENDIX B

GUI DESCRIPTION

LANGUAGE

Notation:

- (nothing) = Must have 1 and only 1 occurrence
- ? = Must have 0 or 1 occurrence
- + = Must have 1 or more occurrences
- * = Must have 0 or more occurrences

Macros:

- ANY_STR = Any stream of characters
- ID = a unique ANY_STR in the current context
- "..." = literal (any chars between quotes)

Document: DynamicMapFormData

Grammar:

- DynamicMapFormData(Title=ANY_STR) = Dialog+
- Dialog(DlgId=ID, Title=ANY_STR) = DlgEntry*

- DlgEntry(EntryId=ID, Label=ANY_STR, Type=("Text" or "Int" or "Double" or "Info" or "Bool" or "Senum" or "Menu" or "Date" or "Time") = (ElmMin?, ElmMax?, ElmChoice*, ANY_STR)
- ElmMin = ANY_STR
- ElmMax = ANY_STR
- ElmChoice(ChoiceId=ID) = ANY_STR

Semantic:

- Each xml Document describes one Form Data (The title makes up the dialog title)
- Each form Data can represents one or more dialogs, if there is more than one dialog, they will be shown on different tabs (the title is shown on the tab if there is a tab)
- Each dialog consist of 0 or more rows of controls
- each row of controls consist of a label and a control of one of the following types ("Text" or "Int" or "Double" or "Info" or "Bool" or "Senum" or "Menu" or "Date" or "Time")
- Each Row of Control can have text between the DlgEntry tags. This text represents a default value when the dialog is created and represents the returned value when the dialog is terminated
 - Text: The data type is any string of characters
 - Int: The data is an integer
 - Double: The data type is a floating point value
 - * If the control type is "Int" or "Double" min and max constraint on the values can optionally be passed through the ElmMin and ElmMax tags

- Info: The data type is a read-only string (for display purposed only)
- Bool: The data type is either "true" or "false"
- SEnum: The data type is a "S"ingle ID from one of the following ElmChoice tags (shown as a combo box)
- MEnum: The data type is a "M"ultiple string of IDs from the following ElmChoice tags (each ID separated by ";") (shown as a list box)
 - * If the control type is "Senum" of "Menum" then there must be at least one or more embedded ElmChoice tags. Each tags has a ChoiceId field that uniquely identifies it (the ';' character is forbidden), and the text enclosed by that tag is the text displayed to the user
- Date: The data type is a date of the following format "YYYY-MM-DD"
- Time: The data type is a time of the following format "HH:MM" when HH is a two digits integer between 00-23, and MM is a two digits integer between 00-59

Example:

```
<DynamicMapFormData Title="test">
<Dialog DlgId="Main" Title="Youhou">
<DlgEntry EntryId="aTextFld" Label="TextFldExample" Type="Text">
<ElmMin>This field is ignored</ElmMin>
<ElmMax>This field is ignored</ElmMax>
<ElmChoice ChoiceId="FirstChoice">This field is ignored</ElmChoice>
</DlgEntry>
<DlgEntry EntryId="aIntFld" Label="IntFldExample" Type="Int">
<ElmMin>0</ElmMin>
<ElmMax>100</ElmMax>
<ElmChoice ChoiceId="FirstChoice">This field is ignored</ElmChoice>
</DlgEntry>
<DlgEntry EntryId="aDoubleFld" Label="DoubleFldExample" Type="Double">
<ElmMin>0.0</ElmMin>
```

```

<ElmMax>100.0</ElmMax>
<ElmChoice ChoiceId="FirstChoice">This field is ignored</ElmChoice>
</DlgEntry>
<DlgEntry EntryId="anInfoFld" Label="InfoFldExample" Type="Info">
<ElmMin>This field is ignored</ElmMin>
<ElmMax>This field is ignored</ElmMax>
<ElmChoice ChoiceId="FirstChoice">This field is ignored</ElmChoice>
</DlgEntry>
<DlgEntry EntryId="aBoolFld" Label="BoolFldExample" Type="Bool">
<ElmMin>This field is ignored</ElmMin>
<ElmMax>This field is ignored</ElmMax>
<ElmChoice ChoiceId="FirstChoice">This field is ignored</ElmChoice>
</DlgEntry>
<DlgEntry EntryId="aSenumFld" Label="SenumFldExample" Type="Senum">
<ElmMin>This field is ignored</ElmMin>
<ElmMax>This field is ignored</ElmMax>
<ElmChoice ChoiceId="FirstChoice">First Choice</ElmChoice>
<ElmChoice ChoiceId="SecondChoice">Second Choice</ElmChoice>
FirstChoice
</DlgEntry>
<DlgEntry EntryId="aMenuFld" Label="MenuFldExample" Type="Menu">
<ElmMin>This field is ignored</ElmMin>
<ElmMax>This field is ignored</ElmMax>
<ElmChoice ChoiceId="FirstChoice">First Choice</ElmChoice>
<ElmChoice ChoiceId="SecondChoice">Second Choice</ElmChoice>
FirstChoice;SecondChoice
</DlgEntry>
<DlgEntry EntryId="aDateFld" Label="DateFldExample" Type="Date">
<ElmMin>This field is ignored</ElmMin>
<ElmMax>This field is ignored</ElmMax>
<ElmChoice ChoiceId="FirstChoice">This field is ignored</ElmChoice>

```

```
</DlgEntry>
<DlgEntry EntryId="aTimeFld" Label="TimeFldExample" Type="Time">
<ElmMin>This field is ignored</ElmMin>
<ElmMax>This field is ignored</ElmMax>
<ElmChoice ChoiceId="FirstChoice">This field is ignored</ElmChoice>
</DlgEntry>
</Dialog>
</DynamicMapFormData>
```

APPENDIX C

CONTEXT DESCRIPTION LANGUAGE

Notation:

- (nothing) = Must have 1 and only 1 occurrence
- ? = Must have 0 or 1 occurrence
- + = Must have 1 or more occurrences
- * = Must have 0 or more occurrences

Macros:

- ANY_STR = Any stream of characters
- ID = a unique ANY_STR in the current context
- "... " = literal (any chars between quotes)

Document: ContextDescriptionData

Grammar:

- ContextDescriptionData(Comment=ANY_STR) = ContextData+
- ContextData(ContextId=ID, Name=ANY_STR) = ContextUnit*

- ContextUnit(ContextSubId=ID, Name=ANY_STR, Type=("Text" or "Int" or "Double" or "Bool" or "Senum" or "Menum" or "Date" or "Time" or "ContextUnit")) = (ContextUnit+, ElmChoice, ANY_STR)
- ElmChoice(ChoiceId=ID) = ANY_STR

Semantic:

- Each xml Document describes a set of contextual information.
- Each piece of contextual information is represented by a ContextData element
- Each Context data contains 0 or more context units
- A context unit has a name, and describes one value of type ("Text" or "Int" or "Double" or "Bool" or "Senum" or "Menum" or "Date" or "Time" or "ContextUnit")
- Except for type "ContextUnit", where the child is another context unit element, the value of the context unit is stored as an ANY_STR in the ContextUnit element
 - Text: The data type is any string of characters
 - Int: The data is an integer
 - Double: The data type is a floating point value
 - Bool: The data type is either "true" or "false"
 - SEnum: The data type is a "S"ingle ID from one of the following ElmChoice tags
 - MEnum: The data type is a "M"ultiple string of IDs from the following ElmChoice tags (each ID separated by ";")
 - * If the control type is "Senum" or "Menum" then there must be at least one or more embedded ElmChoice tags. Each tags has a ChoiceId field that uniquely identifies it (the ';' character is forbidden)

- Date: The data type is a date of the following format "YYYY-MM-DD"
- Time: The data type is a time of the following format "HH:MM" when HH is a two digits integer between 00-23, and MM is a two digits integer between 00-59

Example:

```
<ContextDescriptionData Comment="test">
<ContextData ContextId="CTTID" Title="MyDog">
<ContextUnit ContextSubId="NameID" Name="DogName" Type="Text">
Fido
</ContextUnit>
<ContextUnit ContextSubId="AgeID" Name="DogAge" Type="Int">
2
</ContextUnit>
<ContextUnit ContextSubId="WeightID" Name="DogWeight" Type="Double">
20.3
</ContextUnit>
<ContextUnit ContextSubId="IsMaleID" Name="DogGender" Type="Bool">
true
</ContextUnit>
<ContextUnit ContextSubId="FoodID" Name="DogFood" Type="Senum">
<ElmChoice ChoiceId="foodchoicetwo">soso</ElmChoice>
yummy
</ContextUnit>
<ContextUnit ContextSubId="FriendsID" Name="DogFriends" Type="Menu">
<ElmChoice ChoiceId="friendone">Daisy</ElmChoice>
<ElmChoice ChoiceId="friendtwo">Jack</ElmChoice>
Daisy;Jack
</ContextUnit>
<ContextUnit ContextSubId="BirthDateID" Name="DogBirthDate" Type="Date">
```

```

2003-04-12
</ContextUnit>
<ContextUnit ContextSubId="TimeID" Name="DogSupperTime" Type="Time">
17:00
</ContextUnit>
<ContextUnit ContextSubId="CompetitionsID" Name="DogCompetitions" Type="Conte:
<ContextUnit ContextSubId="CompetitionDateID" Name="Date" Type="Date">
2004-23-10
</ContextUnit>
<ContextUnit ContextSubId="CompetitionScoreID" Name="Score" Type="int">
94.3
</ContextUnit>
</ContextUnit>
</ContextData>
</ContextDescriptionData>

```