

# An Application Framework for Mobile, Context-Aware Trails

**Cormac Driver**

A thesis submitted to the University of Dublin, Trinity College  
in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy (Computer Science)

April 2007

# Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work. I agree that Trinity College Library may lend or copy this thesis upon request.

---

Cormac Driver

Dated: April 26, 2007

# Acknowledgements

Firstly I would like to thank my supervisor Dr. Siobhán Clarke for her guidance, encouragement and support over the years. I have learned a great deal from working with her.

I also had the pleasure of working alongside Éamonn Linehan, Mike Spence and Shiu Lun Tsang on the Hermes project; their insightful views on my work were and are much appreciated.

I would like to thank my academic colleagues and friends in the Distributed Systems Group for making it a pleasure to be part of this unique research environment.

Thank you to my parents, brothers and other family members for their constant support with all of my academic endeavours over the years.

Finally, thanks to my wife for everything she has done to help me with this, she knows exactly what it is and I appreciate it.

**Cormac Driver**

*University of Dublin, Trinity College*

*April 2007*

# Abstract

Time management strategies for planning and scheduling activities increase the effectiveness of either personal or corporate time use. Supporting techniques are commonly based around the use of prioritised to-do lists. While the use of to-do lists for time management is beneficial, their static nature reduces their effectiveness in dynamic environments where users are mobile and activity properties can change over time. The prioritised ordering of a carefully considered, predefined to-do list can quickly become obsolete as its owner begins addressing activities and unforeseen events occur.

Mobile, context-aware computing is a computing paradigm in which small, portable devices such as Personal Digital Assistants (PDAs) and smart phones have access to information, known as context, about the situation in which they are being used and dynamically adapt application behaviour as appropriate. This paradigm facilitates automatic adaptation of a mobile user's schedule so that it accurately reflects the reality in which the user exists and maintains utility despite the occurrence of unforeseen events. Automatic, context-based schedule adaptation is at the core of a wide range of applications for the mobile user who has a set of activities that may or should be carried out throughout the day at different locations.

Implementing a mobile, context-aware activity scheduling application requires addressing two common challenges. Firstly, an application must be capable of automatically ordering a list of activities in an effective manner with respect to relevant context. Existing approaches to mobile, context-based activity ordering are constrained in the number of activities they can cope with or are server-based and subject to wireless network disconnection. Secondly, an application must be capable of identifying when it is necessary to reorder a list of activities to ensure that the list order maintains utility in the face of context change. Techniques for identifying when it is necessary to reorder a list of activi-

ties are generally based on periodically assessing the ordering, resulting in the possibility of an activity ordering becoming temporarily out of sync with the user's reality. To date, mobile, context-based activity scheduling applications have typically been designed and implemented in an application-specific manner - mostly as research prototypes. Consequently, developers have had to repeatedly tackle the challenges inherent to this class of application.

This thesis describes an application framework for the development of mobile, context-aware *trails-based* applications. A trail is a contextually scheduled collection of activities and represents a generic model that can be used to satisfy the activity management requirements of a wide range of context-based activity scheduling applications. The framework supports developers by providing a generic, extensible implementation of the trails model. Structure and behaviour common to mobile, context-aware trails-based applications is provided, supporting context-based activity schedule composition (trail generation), identification of whether or not schedule reordering is required following context change (reconfiguration point identification) and subsequent automatic schedule reordering as appropriate (trail reconfiguration).

The framework is evaluated through the development of three case study applications. The case studies illustrate how the framework can be reused and extended to support the development of a range of mobile, context-aware trails-based applications with differing requirements. In addition, results of empirical experiments conducted to assess the responsiveness of the trail generation implementation, the accuracy of the reconfiguration point identification mechanism and human satisfaction with computer-generated trails are presented.

## Publications Related to this Ph.D.

**Cormac Driver**, Éamonn Linehan, Mike Spence, Shiu Lun Tsang, Laura Chan and Siobhán Clarke. Facilitating Dynamic Schedules for Healthcare Professionals. In *Proceedings of the 1st International Conference on Pervasive Computing Technologies for Healthcare*, Innsbruck, Austria, 2006. IEEE.

**Cormac Driver**, Éamonn Linehan and Siobhán Clarke. A Framework for Mobile, Context-Aware Trails-based Applications: Experiences with an Application-led Approach. In *Workshop 1 - "What Makes for Good Application-led Research in Ubiquitous Computing?"*, 3rd International Conference on Pervasive Computing (*PERVASIVE 2005*), Munich, Germany, 2005.

Mike Spence, **Cormac Driver** and Siobhán Clarke. Sharing Context History in Mobile, Context-Aware Trails-based Applications. In *1st International Workshop on Exploiting Context Histories in Smart Environments*, 3rd International Conference on Pervasive Computing (*PERVASIVE 2005*), Munich, Germany, 2005.

Éamonn Linehan, **Cormac Driver** and Siobhán Clarke. Route Generation for Adaptable Trails-based Applications. In *3rd UK-UbiNet Workshop*, University of Bath, UK, 2005.

Mike Spence, **Cormac Driver** and Siobhán Clarke. Collaborative Context in Mobile, Context-Aware Trails-based Applications. In *3rd UK-UbiNet Workshop*, University of Bath, UK, 2005.

**Cormac Driver** and Siobhán Clarke. Hermes: Generic Designs for Mobile, Context-Aware Trails-based Applications. In *Workshop on Context Awareness*, 2nd International Conference on Mobile Systems, Applications, and Services (*MobiSys 2004*), Boston, USA, 2004.

**Cormac Driver** and Siobhán Clarke. Hermes: A Software Framework for Mobile, Context-Aware Trails. In *1st International Workshop on Computer Support for Human Tasks and Activities, 2nd International Conference on Pervasive Computing (PERVASIVE 2004)*, Vienna, Austria, 2004.

Siobhán Clarke and **Cormac Driver**. Context-Aware Trails. *IEEE Computer*, 37(8):97-99, August, 2004.

# Contents

<b>Acknowledgements</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>Publications Related to this Ph.D.</b>	<b>vi</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xvii</b>
<b>List of Listings</b>	<b>xviii</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Time Management . . . . .	1
1.2 Mobile Context-Aware Computing . . . . .	3
1.3 Trails . . . . .	3
1.3.1 The Hermes Project . . . . .	5
1.4 Challenges . . . . .	6
1.4.1 Trail Generation . . . . .	6
1.4.2 Trail Reconfiguration Point Identification . . . . .	8
1.4.3 Lack of Reusable Software . . . . .	9
1.5 The Application Framework . . . . .	9
1.6 Contribution . . . . .	11
1.7 Roadmap . . . . .	12



<b>Chapter 2</b>	<b>State of the Art</b>	<b>14</b>
2.1	Mobile, Context-Aware Tourist Guides . . . . .	14
2.1.1	GUIDE . . . . .	15
2.1.1.1	Trail Generation . . . . .	16
2.1.1.2	Reconfiguration Point Identification . . . . .	16
2.1.1.3	Developer Support . . . . .	16
2.1.2	P-Tour . . . . .	17
2.1.2.1	Trail Generation . . . . .	18
2.1.2.2	Reconfiguration Point Identification . . . . .	18
2.1.2.3	Developer Support . . . . .	21
2.1.3	Dynamic Tourist Guide . . . . .	21
2.1.3.1	Trail Generation . . . . .	22
2.1.3.2	Reconfiguration Point Identification . . . . .	22
2.1.3.3	Developer Support . . . . .	23
2.1.4	Cyberguide . . . . .	24
2.1.5	LoL@ . . . . .	25
2.1.6	CRUMPET . . . . .	25
2.1.7	m-ToGuide . . . . .	26
2.1.8	Hypermedia Tour Guide . . . . .	27
2.1.9	Summary . . . . .	27
2.2	Context-Aware To-do Lists . . . . .	30
2.2.1	TaskMinder . . . . .	30
2.2.2	CybreMinder . . . . .	32
2.2.3	<i>comMotion</i> . . . . .	32
2.2.4	PlaceMail . . . . .	33
2.2.5	Place-Its . . . . .	34
2.2.6	Castaway . . . . .	35
2.2.7	Summary . . . . .	36
2.3	Mobile, Context-Aware Application Frameworks . . . . .	36
2.3.1	Mobile Bristol . . . . .	37
2.3.2	Stick-e Note . . . . .	39

2.3.3	BerlinTainment . . . . .	40
2.3.4	PEACH . . . . .	43
2.3.5	HyCon . . . . .	45
2.3.6	iCAP . . . . .	47
2.3.7	Summary . . . . .	48
2.4	Context-Awareness Frameworks . . . . .	49
2.5	Chapter Summary . . . . .	51
<b>Chapter 3 Design</b>		<b>53</b>
3.1	Design Approach . . . . .	54
3.1.1	Initial High-Level Framework Design . . . . .	55
3.1.2	Context Acquisition and Modelling in Hermes . . . . .	57
3.1.2.1	Communication and Service Discovery . . . . .	58
3.1.2.2	Message Types and Message Processing . . . . .	60
3.1.2.3	Context Modelling . . . . .	61
3.1.2.4	Summary . . . . .	62
3.1.3	Application 1: Oisín Goes to Trinity . . . . .	63
3.1.3.1	Trail Generation in Oisín . . . . .	65
3.1.3.2	Reconfiguration Point Identification in Oisín . . . . .	67
3.1.3.3	Summary . . . . .	69
3.1.4	Application 2: RiddleHunt . . . . .	69
3.1.4.1	Trail Generation in RiddleHunt . . . . .	71
3.1.4.2	Reconfiguration Point Identification in RiddleHunt . . . . .	75
3.1.4.3	Summary . . . . .	76
3.2	Application Framework . . . . .	77
3.2.1	Response Time . . . . .	78
3.2.2	Multi-Attribute Utility Theory . . . . .	80
3.2.3	Trail Generation . . . . .	83
3.2.3.1	Completed Activities . . . . .	84
3.2.3.2	Impossible Activities . . . . .	84
3.2.3.3	Clashing Activities . . . . .	85

3.2.3.4	The Relevant and Irrelevant Sets . . . . .	87
3.2.3.5	Trail Ordering . . . . .	89
3.2.3.6	Reusability and Extensibility . . . . .	91
3.2.3.7	Summary . . . . .	93
3.2.4	Reconfiguration Point Identification . . . . .	93
3.2.4.1	Differences in Set Membership . . . . .	94
3.2.4.2	Differences in Relevance Rankings . . . . .	95
3.2.4.3	Reusability and Extensibility . . . . .	97
3.2.4.4	Summary . . . . .	98
3.3	Chapter Summary . . . . .	98
<b>Chapter 4 Implementation</b>		<b>99</b>
4.1	Application Framework Overview . . . . .	100
4.2	Trail/Activity Specification . . . . .	103
4.2.1	Trail Persistence . . . . .	105
4.2.2	Extensibility . . . . .	107
4.3	Trail Generation . . . . .	108
4.3.1	The <code>reconfigure()</code> method . . . . .	109
4.3.2	Extensibility . . . . .	114
4.4	Reconfiguration Point Identification . . . . .	116
4.4.1	Extensibility . . . . .	120
4.5	Configuration files . . . . .	121
4.5.1	Extensibility . . . . .	125
4.6	Chapter Summary . . . . .	126
<b>Chapter 5 Evaluation</b>		<b>127</b>
5.1	Framework Reusability and Extensibility . . . . .	128
5.1.1	Day Planner . . . . .	130
5.1.1.1	Implementation . . . . .	130
5.1.1.2	Analysis . . . . .	135
5.1.2	Music Festival Trail . . . . .	136
5.1.2.1	Implementation . . . . .	136

5.1.2.2	Adding a new context source . . . . .	137
5.1.2.3	Adding a new activity attribute . . . . .	138
5.1.2.4	Using the new behaviour . . . . .	138
5.1.2.5	User Interface . . . . .	139
5.1.2.6	Analysis . . . . .	139
5.1.3	Theme Park Trail . . . . .	140
5.1.3.1	Implementation . . . . .	141
5.1.3.2	Adding a new context source . . . . .	142
5.1.3.3	Adding a new activity attribute . . . . .	143
5.1.3.4	Adding new activity comparators . . . . .	143
5.1.3.5	Adding a new evaluation function . . . . .	144
5.1.3.6	Using the new behaviour . . . . .	144
5.1.3.7	User Interface . . . . .	144
5.1.3.8	Analysis . . . . .	145
5.1.4	Summary . . . . .	146
5.2	Trail Generation and Reconfiguration . . . . .	147
5.2.1	Trail Generation - Activity Scheduling . . . . .	149
5.2.1.1	Brute Force . . . . .	150
5.2.1.2	Genetic Algorithm . . . . .	150
5.2.1.3	Simulated Annealing . . . . .	151
5.2.1.4	Analysis . . . . .	152
5.2.2	Trail Generation - Activity Consideration . . . . .	154
5.2.2.1	Results . . . . .	154
5.2.2.2	Analysis . . . . .	154
5.2.3	Reconfiguration Point Identification Accuracy . . . . .	155
5.2.3.1	Day Planner Results . . . . .	157
5.2.3.2	Music Festival Results . . . . .	157
5.2.3.3	Theme Park Results . . . . .	161
5.2.3.4	Analysis . . . . .	161
5.3	Trail Quality . . . . .	163
5.3.1	Trail Quality Experiment Results . . . . .	166

5.3.2	Analysis . . . . .	167
5.4	Chapter Summary . . . . .	169
<b>Chapter 6 Conclusions and Future Work</b>		<b>171</b>
6.1	Achievements . . . . .	171
6.2	Perspective . . . . .	174
6.3	Future Work . . . . .	176
6.3.1	Distributed Trail Generation . . . . .	177
6.3.2	Trail Robustness . . . . .	177
6.3.3	Activity Dependencies and Constraints . . . . .	178
6.4	Chapter Summary . . . . .	179
<b>Appendix A User Study Results</b>		<b>180</b>
A.1	Oisín goes to Trinity - User Study Results . . . . .	180
<b>Appendix B Further Implementation Detail</b>		<b>182</b>
B.1	GPS Location Context . . . . .	182
<b>Appendix C Trail Quality Experiment Materials</b>		<b>187</b>
C.1	Information Sheets . . . . .	187
C.1.1	Group 1 . . . . .	187
C.1.2	Group 2 . . . . .	188
C.2	Activity Scheduling Problems . . . . .	190
C.2.1	Activity Scheduling Problem Legend . . . . .	190
C.2.2	Example Problem . . . . .	191
C.2.3	Example Solution . . . . .	192
C.2.4	Group 1: Problem 1 . . . . .	193
C.2.5	Group 1: Problem 2 . . . . .	194
C.2.6	Group 1: Solution 1 . . . . .	195
C.2.7	Group 1: Solution 2 . . . . .	196
C.2.8	Group 2: Solution 1 . . . . .	197
C.2.9	Group 2: Solution 2 . . . . .	198
C.2.10	Group 2: Problem 1 . . . . .	199

C.2.11 Group 2: Problem 2 . . . . .	200
C.3 Questionnaire . . . . .	201
<b>Bibliography</b>	<b>202</b>

# List of Figures

1.1	The Hermes Architecture . . . . .	5
2.1	Reconfiguration point identification in P-Tour [117] . . . . .	20
2.2	Separation between authoring tool and device-specific implementation [105]	38
2.3	BerlinTainment framework architecture [135] . . . . .	41
2.4	The HyCon framework architecture [17] . . . . .	46
2.5	The conceptual framework for context-aware systems [9] . . . . .	50
3.1	Ad hoc communication and service discovery in Hermes . . . . .	59
3.2	Hermes context model top level hierarchy and examples . . . . .	62
3.3	Screen shots of the Oisín graphical user interface on the Zaurus . . . . .	64
3.4	High-level design of trails behaviour in Oisín . . . . .	66
3.5	Screen shots of the reconfiguration menu and edit screen in Oisín . . . . .	68
3.6	The activities in the set $X$ and the player location . . . . .	72
3.7	Identification of the members of set $Y$ . . . . .	73
3.8	High-level design of trails behaviour in RiddleHunt . . . . .	74
3.9	Context-based activity set reduction . . . . .	85
3.10	The trail generation process . . . . .	86
4.1	High-level application framework class diagram . . . . .	100
4.2	The <code>Trail</code> and <code>Activity</code> classes . . . . .	104
4.3	High-level sequence of actions in trail generation . . . . .	108
4.4	Interactions between classes in the trail generation implementation . . . . .	110
5.1	The text-based display produced for the day planner application . . . . .	134

5.2	The framework extensions facilitating the music festival application . . . .	137
5.3	The text-based display produced for the music festival application . . . .	139
5.4	The framework extensions facilitating the theme park application . . . .	142
5.5	The text-based display produced for the theme park application . . . .	145
5.6	Brute force trail generation response times . . . . .	150
5.7	Simulated annealing trail generation response times . . . . .	151
5.8	Results of the activity consideration experiment . . . . .	155
5.9	Day planner results with $\tau = 0.95$ (top) and $\tau = 0.85$ (bottom) . . . .	158
5.10	Music festival results with $\tau = 0.95$ (top) and $\tau = 0.85$ (bottom) . . . .	159
5.11	Theme park results with $\tau = 0.95$ (top) and $\tau = 0.85$ (bottom) . . . .	160
5.12	An activity scheduling problem given to trails experiment subjects . . . .	165
B.1	GPS location context class diagram . . . . .	183
C.1	Activity scheduling problem legend . . . . .	190
C.2	Example activity scheduling problem . . . . .	191
C.3	Example activity scheduling problem solution . . . . .	192
C.4	Group 1: Activity scheduling problem 1 . . . . .	193
C.5	Group 1: Activity scheduling problem 2 . . . . .	194
C.6	Group 1: Activity scheduling problem solution 1 . . . . .	195
C.7	Group 1: Activity scheduling problem solution 2 . . . . .	196
C.8	Group 2: Activity scheduling problem solution 1 . . . . .	197
C.9	Group 2: Activity scheduling problem problem 2 . . . . .	198
C.10	Group 2: Activity scheduling problem 1 . . . . .	199
C.11	Group 1: Activity scheduling problem 2 . . . . .	200
C.12	The activity scheduling problem solution validation questionnaire . . . .	201



# List of Tables

2.1	Summary of mobile, context-aware activity scheduling support . . . . .	28
3.1	Trail evaluation dimensions (on a scale from 0-100) and sample data . . .	82
3.2	Activity evaluation dimensions for clash resolution and sample data . . .	87
3.3	Activity evaluation dimensions for relevance and sample data . . . . .	88
3.4	The relevant set and the existing trail ranked by relevance . . . . .	95
5.1	Explanation of extensions necessitated by the music festival application .	140
5.2	Explanation of extensions necessitated by the theme park application . .	146
5.3	Results of the reconfiguration point identification experiment . . . . .	161
5.4	Further investigation of the trials with $\tau = 0.95$ . . . . .	162
5.5	Trail quality experiment timing results . . . . .	167
5.6	Trail quality experiment solution validation results . . . . .	168
5.7	Trail quality experiment solution quality results . . . . .	169

# List of Listings

4.1	An example activity specification . . . . .	105
4.2	The <code>update()</code> method in the <code>ReconfigurationEngine</code> class . . . . .	109
4.3	The <code>reconfigure()</code> method in the <code>ReconfigurationEngine</code> class . . . . .	109
4.4	Calculating activity priority value in the <code>MAUTRelevanceComparator</code> class	112
4.5	The evaluation function in the <code>Trail</code> class . . . . .	113
4.6	Excerpt from the <code>reconfigurationRequired()</code> method . . . . .	117
4.7	2nd excerpt from the <code>reconfigurationRequired()</code> method . . . . .	118
4.8	The <code>getKendallValueForReconfig()</code> method . . . . .	119
4.9	The <code>trail.properties</code> file . . . . .	122
4.10	The <code>userPreferences.properties</code> file . . . . .	123
4.11	The <code>normalization.properties</code> file . . . . .	125
5.1	Excerpt from the day planner <code>trail.properties</code> file . . . . .	131
5.2	The <code>userPreferences.properties</code> file . . . . .	132
5.3	The <code>normalization.properties</code> file . . . . .	133
B.1	The <code>doLocationChange()</code> method in the <code>LocationGenerator</code> class . . . . .	182
B.2	The <code>convertToXY()</code> method in the <code>GPSCConversion</code> class . . . . .	185

# Chapter 1

## Introduction

There is currently no reusable, extensible software designed to support developers to implement mobile, context-aware activity scheduling applications. This thesis describes an application framework for mobile, context-aware *trails*-based applications that eliminates the need for developers to readdress common challenges each time they implement an application of this type. This introductory chapter presents the motivation for investigating context-based time management and introduces mobile, context-aware computing. The trails concept is presented, along with a brief introduction to the Hermes project - an umbrella project under which this work was completed. The common challenges in mobile, context-aware activity scheduling are explained along with an overview of the application framework and the contribution of this thesis. Finally, a roadmap for the remainder of this document is presented.

### 1.1 Time Management

Time is a special resource in that it cannot be stored and saved for later use - time that is not used wisely cannot be retrieved. Unfortunately, people are not naturally skilled at managing their time. Many are good at keeping busy and appearing productive as opposed to actually using their time effectively [95], with procrastination being the biggest waster of time [83]. For this reason, tools and techniques for how best to manage time have been actively studied for many years [13]. Time management concerns managing time as effectively as possible, where effective means that the time available is used to complete

as many activities as possible without procrastinating, with important activities taking precedence over those of lesser importance. Time management strategies for planning and scheduling time are primarily based around the use of the prioritised to-do list [31]. A to-do list is a collection of activities that must be completed during a period of time, with activities crossed off as they are completed. The activities in the list are generally scheduled in order of importance but may simply be listed arbitrarily. The benefits of using to-do lists to manage time include [116]:

- Less likelihood of forgetting even minor tasks.
- Less likelihood of procrastination because there is a realistic idea of the work that needs to be done, and the time available to do it.
- Increased flexibility when deciding what to do and when to do it because high priority tasks are identified.
- A short and long-range view of the work coming up.

To-do lists are traditionally created using paper and pen. Numerous software equivalents are available on a range of hardware platforms, notably desktop computers, laptops, Personal Digital Assistants (PDAs) and mobile phones [5, 34, 97, 119]. Many e-mail clients include task list applications [55, 94, 125, 57] and several web-based task list applications have been developed [1, 29, 47, 66, 129].

While the use of to-do lists for time management is beneficial, their static nature reduces their effectiveness in dynamic environments where users are mobile and activity properties can change over time. People typically plan their to-do list at a base (typically home or work) and take their list with them to refer to at the places where activities are performed [77]. The prioritised ordering of a carefully considered to-do list can quickly become obsolete as its owner leaves their base and begins addressing activities and unforeseen events occur. To remain useful, a task system must allow adaptation, in the form of rescheduling, in the face of unexpected problems. Adaptation also enables opportunities to save time spent on irrelevant or less than optimal tasks [76].

## 1.2 Mobile Context-Aware Computing

Mobile context-aware computing is a computing paradigm in which applications can discover and take advantage of contextual information (such as user location, time of day, nearby people and computing devices, and user activity) [112]. Mobile computers are generally small, portable devices that allow the user to move away from the traditional desktop environment while retaining the ability to undertake computing tasks. Mobile computing devices include laptops, mobile phones and PDAs. Mobile, context-aware computing is based on the ability of mobile computing devices to recognise aspects of the situation in which they are being used. This situational information is referred to as *context* and has been defined by Dey as “any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the applications themselves” [36]. Mobile, context-aware applications automatically adapt to discovered context by changing their behaviour as appropriate to better suit the user. Example applications of mobile, context-aware computing include location-aware telephone call forwarding [130], situation-aware self-managing mobile phones [118], context-aware medication monitors [4] and weather-aware clothes hanger-based information displays [82]. Mobile, context-aware computing exhibits the key properties necessary to facilitate the implementation of an activity scheduling application for the mobile user that automatically adapts so that it accurately reflects the reality in which the user exists and maintains a relevant schedule of activities in the face of emergent, unforeseen events.

## 1.3 Trails

A trail is a contextually scheduled collection of activities and represents a generic model that can be used to satisfy the activity management requirements of a wide range of context-based activity scheduling applications for the mobile user [28]. Applications that use knowledge of the user’s situation to maintain a list of activities that the user must or should undertake can use the trails model, regardless of the nature of the application presented to the end user. For example, the trails model can support the context-aware

activity scheduling behaviour in an application for delivery couriers in the same manner as it can in applications used by schoolchildren on a field trip to a zoo, tourists visiting a city or museum, doctors in a hospital or music fans at a music festival.

Trails are composed of activity specifications in a process known as *trail generation*. An activity includes properties such as its opening hours, whether it is mandatory or optional, a priority value, a description, a location, and an estimated duration. Trail generation involves ordering activities in the most effective manner possible based on the activity properties and context. For example, a person using a trails-based day planner application may start the day by selecting a number of activities to undertake from their database of common activities. Based on activity properties and the context being considered by the application e.g., location, time and user preferences, the application will generate a trail representing the most effective activity ordering for the user at that moment in time.

Trail order is dynamically affected by significant changes in the context that is relevant to the trail. This process is known as *trail reconfiguration* and involves re-executing the trail generation mechanism with the new context data as input in order to find the most effective trail for the user in the current situation. For example, in a trails-based application for a healthcare professional operating in a hospital environment, receipt of context from a patient heart rate monitor that is deemed to be irregular would cause the user's trail to be reconfigured so that the patient receives attention sooner than he was scheduled to. Trail reconfiguration is triggered following the identification of a trail reconfiguration point i.e., a point at which it is necessary to reconfigure the trail in order to ensure that it reflects the user's reality and maintains utility in the face of context change. For example, in the hospital application described, the reading from the patient heart rate monitoring equipment combined with the knowledge that the other patients on the healthcare professional's trail were stable, caused the trail to be reconfigured. The process of recognising when a trail needs to be reconfigured is known as *trail reconfiguration point identification*.

### 1.3.1 The Hermes Project

The Hermes project<sup>1</sup> is investigating extensible, generic components for mobile, context-aware applications. The project is concerned with developing components for context acquisition (infrastructure for obtaining context from sensor devices), collaborative context (context information that is obtained via collaboration between a number of sensors and higher-level devices), context reasoning (deducing new and relevant information from various sources of context-data), context modelling (representing context in a manner that makes it accessible to applications) and trail management (dynamically maintaining a schedule of activities based on context).

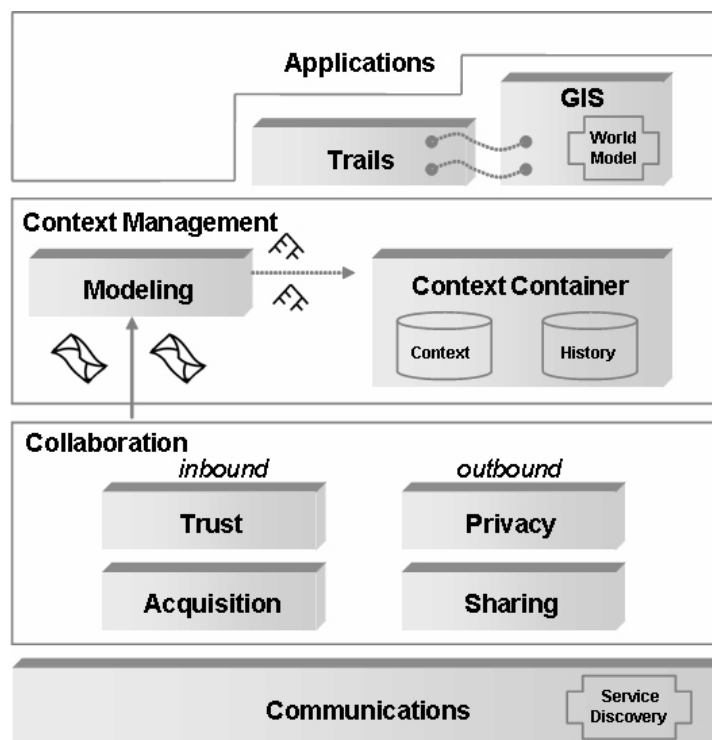


Figure 1.1: The Hermes Architecture

This thesis is concerned specifically with the trails management component that provides Hermes-based applications with the ability to support the implementation of applications containing mobile, context-aware activity scheduling behaviour. The **Trails** component resides beneath the **Applications** layer in the Hermes architecture illustrated in Figure 1.1 and assumes the availability of the context service below. The context ser-

<sup>1</sup><http://www.dsg.cs.tcd.ie/hermes>

vice provides the `Trails` component with context information used for trail generation and reconfiguration point identification in an application-usable format.

## 1.4 Challenges

The implementation of a mobile, context-aware activity scheduling application involves addressing two challenges common to this type of application. Firstly, an application must be capable of automatically ordering a list of activities in an effective manner with respect to relevant context (trail generation). Existing approaches to mobile, context-based activity ordering are either constrained in the number of activities they can cope with because of device limitations, or are server-based and subject to wireless network disconnection. Secondly, an application must be capable of identifying when it is necessary to reorder a list of activities to ensure that the list order maintains utility in the face of context change. A balance is required between avoiding the execution of unnecessary reordering processing (a resource-intensive task) on a resource-constrained mobile device and ensuring the list order is always the ‘best’ one (trail reconfiguration point identification). Existing techniques for identifying when it is necessary to reorder a list of activities are generally based on periodically assessing the ordering. This approach raises the possibility of an activity ordering becoming temporarily out of sync with the user’s reality.

To date, mobile, context-based activity scheduling applications have typically been designed and implemented in an application-specific manner. Consequently, developers have had to repeatedly tackle the challenges inherent to this class of application, hindering progress in areas that researchers are typically interested in such as application deployment and user evaluation. These challenges are discussed in more detail below.

### 1.4.1 Trail Generation

Trail ordering is a combinatorial optimisation problem similar to the classic Travelling Salesman Problem (TSP) [74]. Combinatorial optimisation problems are those where the set of feasible solutions is discrete and the goal is to find the best possible solution by exploring the usually large solution spaces of these problems [32]. The TSP is described



as follows: Given a collection of cities and the costs of travelling from any city to any other city, what is the cheapest route that visits all of the cities and returns to the starting point? The trail generation problem differs from the TSP in that the number of activities is variable (based on context) and therefore the solution space can dynamically increase and decrease in size. Additionally, in the general case, it is not necessary to return to the starting point to complete a trail. The most obvious solution to combinatorial optimisation problems is to generate all permutations of the elements in question and rate each permutation against a predefined notion of optimality e.g., shortest round-trip distance in the case of the TSP. However, the number of permutations is  $n!$ , where  $n$  is the number of activities. From a response time perspective this brute force solution becomes impractical as the number of elements in the set under consideration increases. Experience with this approach in a mobile, context-aware activity scheduling application has shown that it becomes infeasible as the number of activities increases [25]. Therefore, it is necessary to find a satisfactory solution by trading solution quality against application responsiveness. Using approximation algorithms (e.g., heuristic and random number-based approaches) it is possible to achieve solutions with a high probability of being within 2-3% of optimal in a practical amount of time. However, determining what exactly a practical time is on a per-application basis and achieving this level of efficiency on a resource-constrained mobile device are non-trivial issues.

The trail generation problem also compounds the TSP by necessitating a more complex *evaluation function* [109]. An evaluation function quantifies the optimality of a solution, essentially encoding a human notion of optimality within the trail generation algorithm. The TSP typically uses the total distance between all cities to assess the worth of a particular permutation of cities. However, when evaluating a trail composed of, for example, activities on a campus, a wider range of factors must be considered. Good general examples of such factors are activity properties (activity opening hours, whether the activity is mandatory or optional, the category of the activity, crowding levels at the activity location), the user's current location and user preferences e.g., the activity priority relative to other activities on the trail. The implementation of the evaluation function therefore becomes a multi-attribute utility estimation problem [128], where weights representing relative importance are assigned to the various attributes considered

in the evaluation function and a single value for each permutation of activities is produced. This single value can be used in the same manner as the round-trip route distance in the TSP to compare candidate solutions. The additional processing required by the evaluation function affects the overall efficiency of the trail generation algorithm.

### 1.4.2 Trail Reconfiguration Point Identification

The ability of a mobile, context-aware activity schedule to reflect the state of the user's environment through the presentation of a consistently effective trail is central to its acceptance by end-users. Trail reconfiguration point identification involves identifying *when* a trail must be reconfigured so that it maintains utility following context change. When a reconfiguration point is identified, the trail generation mechanism is re-executed to determine the trail that best serves the user in the new contextual situation. Reconfiguring the trail every time a new context event occurs ensures that the trail always reflects the user's reality. However, trail generation is a resource-intensive process and reconfiguring the trail every time context changes can critically impact on application performance when operating on a resource-constrained mobile device. The ideal situation is one in which the trail is reconfigured *only* and *always* when necessary. Unfortunately, it is not possible to fully understand the effects a context change will have on a trail without reconfiguring the trail with the new context data as input. Therefore, a balance must be realised between avoiding unnecessary reconfiguration and ensuring that the trail consistently reflects the user's reality.

A common approach is to avoid identifying when a trail needs to be reconfigured by simply reconfiguring it periodically [24]. In this technique, an application is programmed to invoke the trail generation mechanism every time a predefined period of time passes. When the trail generation mechanism executes, the trail is reconfigured. The trail will then represent the current state of the user's environment. This avoids repeatedly reconfiguring the trail and negatively impacting application performance. However, there are two drawbacks to this approach. Firstly, it gives rise to the possibility that, during the intervals between periodic reconfigurations, context events can occur that cause the trail to become obsolete e.g., an activity that a user is en route to could close unexpectedly.

Therefore this approach is not suitable for use in applications where significant context events may occur more frequently than periodic trail reconfiguration. Secondly, periodic reconfiguration occurs regardless of the contextual situation and therefore will execute unnecessarily if the context is the same as, or similar to, the context considered during the last reconfiguration.

### **1.4.3 Lack of Reusable Software**

It is often the case that the primary objective of research efforts involving mobile, context-aware activity scheduling applications is not specifically related to investigating trail generation and reconfiguration point identification mechanisms. Evaluating human reaction to mobile, context-aware computing applications [24, 12], investigating new models of service delivery in specific areas (e.g., tourism) [98, 63, 8], investigating specific techniques for user preference elicitation [70] and the capabilities of mobile, context-aware hardware [24, 3] are examples of motivating factors that have led to researchers to implement mobile, context-aware activity scheduling applications. Such applications have been implemented from the ground up without the aid of reusable software to support developers in tackling the challenges posed by trail generation and reconfiguration point identification. The main drawbacks of developing applications without framework support are increased development timescales and increased costs [99].

Implementing reusable software requires developers to ensure that the components they create are generic and extensible [58]. This requires significantly more development effort than implementing the required components in an application-specific manner with as little effort as is necessary to support the primary research objectives. For this reason, developers of mobile, context-aware activity scheduling applications have not tackled the challenge of developing reusable software and have therefore had to repeatedly tackle the common challenges.

## **1.5 The Application Framework**

This thesis describes an application framework for mobile, context-aware trails-based application development. The application framework supports developers by providing

generic structure and behaviour that addresses the common challenges in mobile, context-aware activity scheduling.

A software framework is a reusable implementation of all or part of a software system expressed as a set of classes (some abstract) with behaviours defining the way in which instances of those classes collaborate [106]. The term ‘application framework’ is used to describe a software framework that constitutes a generic application for a specific domain area [100]. In answer to the challenges presented in Section 1.4, the framework provides reusable and extensible mechanisms for trail generation and reconfiguration point identification that execute on a mobile device.

The framework supports trail generation through context-based activity set reduction and an extensible collection of activity permutation generation mechanisms. Context is used to prune an application’s set of theoretically possible activities by removing *impossible*, *complete* and *clashing* activities. Clashes are resolved based on weighted user preferences for clash resolution. If a pruned set of activities is too large to be reasoned over in a reasonable response time (defined in Chapter 3) the activity set is divided. The most *relevant* activities are grouped into a set known as the *relevant set* (the size of which is defined by the developer depending on the desired response time). The remaining activities are stored separately and migrate to the relevant set following significant context change. Permutations of the activities in the relevant set are evaluated using a user-preference-based evaluation function and a single permutation is composed into a trail. The trail generation mechanism is capable of generating trails that humans consider to be ‘reasonable’.

Trail reconfiguration and reconfiguration point identification are supported through the identification of significant context events and subsequent trail reconfiguration that invokes the trail generation mechanism. Context event significance is measured by calculating the effect that a context event has on the correlation between two sets of activities, where both sets are ranked by relevance. The activity set is ranked by relevance following a context event and the activities in the current trail are ranked by relevance. The correlation between the two ranked sets is calculated and the context event is classified as significant if the similarity value calculated is below a developer or user defined threshold.

The framework facilitates developers in implementing mobile, context-aware trails-

based applications by providing generic, extensible solutions to common trails-related issues. This makes the development of these applications more accessible to software developers.

## 1.6 Contribution

The application framework described in this thesis contributes to the state of the art in the area of mobile, context-aware activity scheduling by addressing the following issues:

- Current approaches to trail generation in mobile, context-aware activity scheduling applications are limited in the number of activities that they can consider due to response time requirements. This thesis describes a mobile-device-based trail generation mechanism that uses context-based activity pruning to reduce the number of activities considered during permutation evaluation. This allows applications to include a large number of activities, with activities being considered during the scheduling process based on how relevant they are to the user's current situation.
- Current permutation generation and evaluation mechanisms are implemented in an application-specific manner, hindering their reuse in subsequent development efforts. The application framework described in this thesis provides three approaches to permutation generation as well as an architecture purposely designed to facilitate the extension of the framework through the addition of new permutation generation approaches that integrate with the remainder of the framework. Activity permutations are evaluated by a user preference-driven evaluation function.
- Current approaches to trail reconfiguration point identification are based on periodic trail reordering. This thesis describes a preference-driven trail reconfiguration point identification mechanism that address the limitations with periodic reconfiguration by identifying significant context events that necessitate trail reconfiguration as they occur.
- The trail generation and reconfiguration point identification approaches in the application framework are based on an extensible range of user preferences, allowing

the algorithms to produce user-specific results without requiring source code modifications. Trail generation is extensible through the consideration of additional contexts and preferences for activity pruning and activity permutation evaluation, and the specification of additional permutation generation mechanisms. The reconfiguration point identification mechanism is extensible through the consideration of additional contexts. Both the trail generation and reconfiguration point identification behaviour can be reused regardless of whether or not it has been extended.

This thesis describes three main evaluation approaches. First, the contribution as regards the reusability and extensibility of the framework is evaluated through the development of three case study applications. The case studies illustrate how the framework can be reused and extended to support the development of a range of mobile, context-aware trails-based applications. Second, results of empirical experiments conducted to assess the responsiveness of the trail generation implementation and the accuracy of the reconfiguration point identification mechanism are presented. Third, details of an experiment concerning human satisfaction with the trails generated by the framework are presented.

## 1.7 Roadmap

The remainder of this thesis is organised as follows. Chapter 2 presents an overview of the state of the art in the areas of mobile, context-aware tourist guides, context-aware to-do lists and application frameworks for mobile, context-aware applications. The discussion of research into mobile, context-aware activity scheduling focuses on how the related research deals with trail generation and trail reconfiguration point identification. Chapter 3 describes the design of the application framework for mobile, context-aware trails-based applications and Chapter 4 presents implementation detail. In Chapter 5, the framework is evaluated by applying it to the development of three mobile, context-aware activity scheduling applications with differing requirements, illustrating reuse of, and extension to, the base application framework. The results of empirical experiments conducted to assess the responsiveness of the trail generation mechanism and the accuracy of the trail reconfiguration point identification mechanism are also presented, along with the results

of the trail quality experiment. Finally, Chapter 6 concludes and discusses potential areas for further research.

# Chapter 2

## State of the Art

This chapter assesses research that investigates, to varying degrees, mobile, context-aware activity scheduling behaviour. The extent to which each of the related projects addresses the challenges identified in Section 1.4 is discussed. The state of the art in the following areas is presented:

- Mobile, context-aware tourist guides.
- Context-aware to-do lists.
- Application frameworks for mobile, context-aware computing.

A brief overview of context-awareness frameworks in general is also presented.

### 2.1 Mobile, Context-Aware Tourist Guides

To date, most research into mobile, context-aware activity scheduling has focused on development of tourist guide applications. However, many of these applications do not support dynamic, context-based activity management. Some present the user with a static tour on a mobile device and track their location, while others use context to generate a tour but offer no subsequent tour adaptation. In terms of trail generation and reconfiguration point identification, GUIDE [24], P-Tour [81] and the Dynamic Tourist Guide (DTG) [70] are the most sophisticated. In addition, Cyberguide [3], LoL@ [8], Crumpet [98], m-ToGuide [63] and the Hypermedia Tour Guide [12] include trails-related



behaviour such as activity selection based on user preferences, tour provision (albeit static), location-based activities and user location tracking. The remainder of this section reviews these applications and analyses the extent to which they support mobile, context-aware activity scheduling.

### **2.1.1 GUIDE**

GUIDE is a mobile, context-aware tourist guide application for tourists in the city of Lancaster, UK [24]. GUIDE users can view web-based information related to their current location, access interactive services e.g., accommodation booking systems, and send and receive messages, enabling groups of tourists to keep in touch while exploring the city. The system also allows users to request a structured city tour by selecting attractions from a predefined, categorised set of popular Lancaster attractions. The system, deployed on a mobile device, generates a recommended sequence for visiting the chosen attractions based on the user's location (derived from the WiFi base station to which the user is connected) and the opening and closing times of the attractions. The ordering of the tour can dynamically change based on the user's location and the current time. To avoid seeming overly authoritarian, the system allows users to manually change the tour ordering by selecting the activity they wish to do next, regardless of its place in the recommended tour and the effect the decision has on the feasibility of the other activities. The GUIDE developers carried out an extensive user study to validate the concept of using mobile, context-aware technology to assist users with completing a collection of activities. This user study illustrates that the GUIDE system was successfully deployed, with all users considering the location-aware navigation and information retrieval mechanisms provided by the system to be both useful and reassuring. Additionally, the majority of the user study subjects said that they were prepared to trust the information presented by the system, including the navigation instructions. The published lessons learned and experiences in terms of location context acquisition, information modelling, context-aware user interface design and context-aware information presentation are valuable to those who wish to develop and deploy interactive context-aware systems.

### **2.1.1.1 Trail Generation**

GUIDE uses a brute force algorithm to generate the optimal ordering for visiting a set of attractions. This brute force approach systematically enumerates all the possible attraction orderings and retains the ordering that best satisfies the user’s requirements in terms of visiting the maximum amount of the activities selected for the least cost (in terms of time and walking distance). The resource-intensive nature of this approach limits GUIDE to a maximum of nine activities if a reasonable response time is required [25]. The developers of the GUIDE system do not specify the response time value that they consider to be reasonable.

### **2.1.1.2 Reconfiguration Point Identification**

The GUIDE system “regularly” calculates whether or not the tour that the user is following is appropriate given the current context. A tour can become inappropriate if the user stays longer than anticipated at an attraction or an attraction announces that it is closing early. The implementation details of this periodic approach to reconfiguration point identification are not presented by the GUIDE developers.

Periodic reconfiguration techniques expose tours to the possibility of becoming out of sync with the user’s reality as described in Section 1.4.2. This issue is exacerbated by the fact that only positional and temporal context are sensed automatically in GUIDE, meaning that other context events such as changes to activity opening and closing hours must be uploaded manually if they are to be considered by the system.

### **2.1.1.3 Developer Support**

The initial objective of the GUIDE project was to investigate the use of wireless broadcast schedules. GUIDE uses a cell-based wireless communications infrastructure to download the attraction information that is displayed to tourists and also to generate tourist location context (a tourist’s location is the WiFi cell server they are currently communicating with). This objective evolved once the project began and the focus shifted to exploring the human factors issues associated with developing a mobile, context-aware tour guide application [26]. Although the information model, cell-based wireless communications

infrastructure and the high-level software architecture have been discussed in GUIDE publications [33], the provision of application framework support was not a research objective. The trail reconfiguration and reconfiguration point identification mechanisms were designed in an application-specific manner to meet the requirements of the GUIDE system, resulting in limited potential for reuse in future development efforts.

### 2.1.2 P-Tour

P-Tour is a personal navigation system that allows tourists in Nara, Japan to compose a sub-optimal (within 2% of optimal) multi-destination schedule [81]. P-Tour takes the following factors into account when generating a tourist schedule:

- User preferences:
  - The user-assigned priority for each destination.
  - The time at which the user wishes to visit each destination.
- The monetary outlay required to visit each destination.
- The user's current location and the location of each destination.
- The destination time restrictions (opening and closing hours).

Similar to the GUIDE system, tour destinations in P-Tour are selected from a predefined list. Users can also add their own destinations to the collection of available destinations. The end-user application resides on a mobile device and communicates via WiFi with a remote server that is responsible for tour generation. Routes between the destinations on the tour are illustrated on a map-based user interface. The schedule of destinations is dynamically adapted based on deviation from the route between two tour destinations. The P-Tour system successfully achieves its stated goals of providing multi-destination schedules based on multiple criteria, navigating the user from destination to destination and modifying the schedule automatically based on context changes. The use of a heuristic approach to tour generation illustrates the feasibility of non-optimal tour generation

techniques, while the reconfiguration point identification technique illustrates how context can be used to move beyond simple periodic reconfiguration by identifying when a tour actually needs to be reconfigured.

### **2.1.2.1 Trail Generation**

The P-Tour client application (residing on a mobile device) communicates, via WiFi, with a remote server that executes a genetic algorithm [109] written in Java. The algorithm is capable of computing tours of 14 activities to within 2% of optimal in 15.5 seconds. The application limits the number of activities considered so that the application responds to the user in a timely manner. A genetic algorithm is a guided random search technique used to find approximate solutions to combinatorial optimisation problems through application of the principles of evolutionary biology to computer science. Genetic algorithms use biologically-derived techniques such as inheritance, mutation, natural selection, and recombination (or crossover). Candidate solutions are evaluated by means of a fitness function. A fitness function quantifies the optimality of a solution in a genetic algorithm so that each individual solution may be ranked against all the other candidate solutions. The optimal tour for the 14 destinations was also calculated once using an exact algorithm e.g., brute force, with no time constraints so that the results of the genetic algorithm could be compared to the known optimal solution.

The P-Tour client application residing on the mobile device does not contain any behaviour to dynamically adapt the user's schedule. In the case of disconnection from the wireless network the application ceases to function as it should and the user experiences loss of tour maintenance service. The P-Tour algorithm is designed to work on a powerful server platform and its performance in terms of response time would be significantly degraded by migration to a resource-constrained platform such as a PDA or smart phone.

### **2.1.2.2 Reconfiguration Point Identification**

The P-Tour reconfiguration point identification mechanism, introduced as an extension to the initial application, is based on identifying the geographical area that the user should be in at any point along the tour [117]. The application calculates expected locations for a user by assuming that the user moves at a predefined constant speed. P-tour tracks

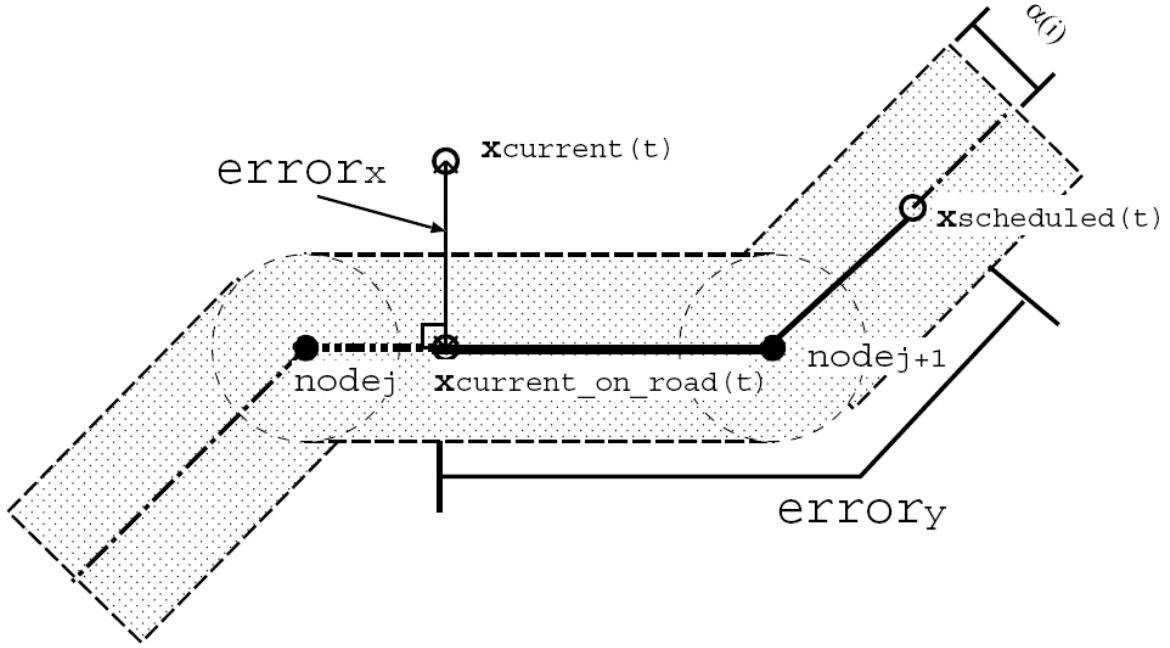
the user’s location at “regular intervals (for example, every 60 seconds)” and records their current situation. If the user has deviated from the suggested route between two destinations the negative impact of the deviation is calculated. This value is calculated as the difference between the tour evaluation value for the current tour and the tour evaluation value for the new schedule computed with the delay. If the negative impact is significant a reconfigured tour is displayed to the user (otherwise a warning is displayed). The details of the approach to determining route deviation are illustrated in Figure 2.1 and listed below:

1. Calculate the expected location of the user  $X_{\text{scheduled}}(t)$  at time  $t$ , assuming that the user moves at a constant speed.
2. Measure the user’s actual location  $X_{\text{current}}(t)$  at time  $t$ , using the client’s location system.
3. Find the location  $X_{\text{current\_on\_road}}(t)$  on the route that is nearest to the user’s current location  $X_{\text{current}}(t)$ .
4. Let  $error_X$  be the distance between  $X_{\text{current}}(t)$  and  $X_{\text{current\_on\_road}}(t)$ .
5. Let  $error_Y$  be the distance on the route between  $X_{\text{scheduled}}(t)$  and  $X_{\text{current\_on\_road}}(t)$ . If the user is going behind schedule  $error_Y$  is multiplied by -1.

The user’s situation is defined by the values of  $error_X$  and  $error_Y$  as follows (here  $\alpha(i)$  is determined by the width of the road  $i$ .  $\beta$  and  $\gamma$  are constants determined by how tight the current schedule is):

- Wrong route:  $\alpha(i) < error_X$
- A little behind scheduled:  $\gamma \leq error_Y \leq \beta$
- Severely behind schedule:  $error_Y < \gamma$
- Ahead of schedule:  $\delta < error_Y$

In order for this approach to work, the server periodically (at the same interval as it acquires the user’s location) recalculates the tour to assess the impact of the deviation.



**Figure 2.1:** Reconfiguration point identification in P-Tour [117]

If there is no delay then the cost of generating the value of the tour with the current delay value is incurred unnecessarily. The approach has been evaluated over a distance of 1.5km, comparing the  $error_y$  value with and without the warning facility. The  $error_y$  value was shown to be -20 with the warning facility in place and -60 without the warning facility. This result shows that the approach is effective at keeping users on schedule.

While the reconfiguration point identification mechanism employed by P-Tour improves on the basic periodic approach introduced by the GUIDE system, it has some limitations. The time-based periodic element of the mechanism (location is tracked periodically) raises the possibility that an important context event could occur between one context assessment and the next. This means that the tour can be in a state inconsistent with the user's reality, without an alert being issued or a reconfiguration occurring. It is unlikely that this would actually be a problem in the P-Tour application where only location context is considered during reconfiguration and users on foot cannot move too far between reconfigurations. However, if this approach was used by applications that consider more dynamic contexts (not just user location) it is likely that inconsistencies between the real world and the computer representation would occur. Additionally, the P-Tour reconfiguration point identification approach is based on changes in location context only and the model does not cater for extension through the addition of non-spatial

contexts such as changes to activity opening hours, queuing times at attractions and changes in user preferences.

### **2.1.2.3 Developer Support**

The objective of the P-Tour project was to build on existing applications used in Japan for single destination navigation by providing a mobile, context-aware system for tourists, who typically want to visit multiple destinations. The application uses an application-specific approach to trail generation and reconfiguration point identification and does not aim to support developers who wish to reuse the algorithms.

### **2.1.3 Dynamic Tourist Guide**

The Dynamic Tourist Guide (DTG) is a mobile agent that selects tourist attractions in Goerlitz, Germany from a predefined database (based on elicited user preferences) and plans a tour of these attractions [70]. The user's interest level in the attractions, the time available for the tour, the attraction opening hours and the user's location are used to generate a schedule of attractions. The DTG can generate a schedule of 16 activities to within 5% of optimal in 5.5 seconds. The DTG provides users with schedules that serve their needs by being sensitive to their personal interests. The schedules created by the DTG aid tourist destination management organisations by distributing tourists evenly across a set of geographically dispersed tourist attractions, therefore minimising queuing and maximising throughput. The DTG end-user application is deployed on a mobile device that communicates via WiFi with a remote tour-generation server. Tours are dynamically reconfigured based a user's progress on a tour, similar to the approach employed in P-Tour. The DTG developers have designed and evaluated three preference elicitation techniques and have illustrated some success with automatically predicting user preference for a collection of destinations based on elicited preferences. These preferences can be used to automatically select destinations for tourists to visit, serving the tourist by exposing them to destinations they were not aware of and the tourist industry by spreading tourists among attractions, hence reducing crowding at popular attractions.

### 2.1.3.1 Trail Generation

The DTG is capable of computing a tour composed of 16 activities to within 5% of optimal in a response time of 5.5 seconds [123]. The DTG limits the number of activities on a tour so that the response time requirement of 5 seconds can be met. The server-based agent uses a directed depth-first algorithm [109] that incorporates a number of heuristics. Depth-first search is an algorithm for traversing a tree (representing the solution space) that starts at the root and explores each branch as far as possible before backtracking. The DTG search algorithm is augmented with a number of heuristics for choosing the branches that look the most likely to yield a good solution. The heuristics include using an average duration estimate for all attractions and trading off attraction relevance against the cost (in time) of traveling to and exploring the attraction. These heuristics, while increasing the efficiency of the algorithm, can result in tours that maximise the number of attractions, hence achieving a higher overall score (based on the DTG tour evaluation function). However, such tours can have a lower average score per attraction, meaning that tourists may not be that interested in many of the attractions and their favourites may be absent.

The DTG developers have acknowledged the problem of wireless network disconnection and take advantage of periods of connectivity by using a preloader to cache information about the forthcoming tour activities. However, there is no tour management behaviour on the mobile device that can be used while disconnected. As with the P-Tour tour generation algorithm, the DTG tour generation algorithm is implemented to work on a powerful server platform and cannot be easily migrated to the mobile device-based client application.

### 2.1.3.2 Reconfiguration Point Identification

The DTG reconfiguration point identification mechanism is based on periodically tracking user progress on a tour. The tour can be adapted by adding activities if the initial activities are completed ahead of time and by removing activities when the user spends longer than anticipated at an activity. The authors state that “the permanent tracking of the tour progress considers external influences to adapt the tour” [124]. They also



mention “permanent supervision” of the tour [124] and that the application contains behaviour to “consistently supervise the ongoing tour” [69]. However, the DTG actually checks if reconfiguration is required every 3 minutes, similar to the periodic approach used in GUIDE. However, the behaviour that executes periodically is not simply trail reordering as in GUIDE but a method to assess if reconfiguration is required. The DTG assesses if the user’s current tour progress has caused them to deviate from the initial plan by more than 30 minutes. If this is the case the tour is reconfigured e.g., activities are added if the user is ahead of schedule or removed if they are behind schedule.

The reconfiguration point identification approach improves on that used in GUIDE by adding intelligence to the behaviour that executes periodically, consequently avoiding unnecessary reconfiguration. However, by tracking only user progress on the tour (similar to the approach taken by P-Tour) the DTG does not consider other contexts that can affect a tour such as changes to activity opening hours or resource expiration e.g., all tickets for a play selling out while a user is en route to the theatre. Additionally, by executing the tour comparison behaviour periodically the DTG is subject to the same issues as GUIDE and P-Tour in terms of tours becoming temporarily out of sync with the user’s reality.

The evaluation of the DTG reconfiguration point identification mechanism was conducted using tours that had durations of between 1 hour and 1.5 hours. The threshold of 30 minutes proved to be too long relative to the tour durations, resulting in very few tour adaptations (only 9% of all tours were adapted). This resulted in users following tours that did not accurately reflect their reality.

### **2.1.3.3 Developer Support**

The primary objectives of the DTG project concern the investigation of user preference elicitation techniques, and the use of profile information to compose schedules of activities for users. Developing the trail generation algorithm in a generic and extensible manner for use by other developers was not a design concern. This is evidenced by the use of heuristics for search algorithm optimisation. The heuristics used are tailored to the tourism domain and do not generalise to mobile, context-aware activity scheduling in general. For example, it may not be possible to make generalisations about a physician’s

tasks in the way that the DTG makes generalisations about tourist attractions e.g., using a single activity duration regardless of whether the attraction is a church spire or an art museum.

#### 2.1.4 Cyberguide

The Cyberguide system developed at Georgia Institute of Technology, Atlanta represents the earliest work on developing a mobile, context-aware system for tourists [3]. Cyberguide was a major inspiration to the related systems that followed, notably GUIDE. The system does not support activity scheduling but lets the user discover attractions at different locations by providing a map annotated with the user's location. The Cyberguide system executes on a mobile device and uses an infrared beacon-based location service to position a tourist on a map. The map displays the locations of various demos on display at the Graphics, Visualization, and Usability Center at Georgia Tech. Users can use the map to guide themselves to the various attractions, and attraction icons are updated to show that the user has visited them. The outdoor version of the system (CyBARguide) utilises the Global Positioning System (GPS) to obtain information data and allows users to view the locations of various bars near the Georgia Tech campus. Users can make notes about the bars they visit and can also add establishments not already present in the database.

The Cyberguide system did not focus on providing trails to users and therefore it does not contain trail generation or reconfiguration point identification behaviour in the same manner as GUIDE, P-Tour and the DTG. However, it did introduce the concept of allowing a user to navigate through a collection of activities using a mobile device, location tracking system and map-based user interface. Subsequent mobile, context-aware activity scheduling applications, including those that can be developed using the application framework described in this thesis, have built on the concepts introduced in the Cyberguide system.

### 2.1.5 LoL@

LoL@, the Local Location Assistant, is a mobile, context-aware tour guide for visitors to the first district of Vienna, Austria [8]. The LoL@ application executes on a mobile device and uses a UMTS<sup>1</sup>-based [43] location service to guide tourists on a static tour, providing them with multimedia information related to the sights they encounter. The location information is used to position users on the map-based interface and guide them to the next point of interest.

The application was developed as a prototype UMTS application in an effort to provide a UMTS-based location service and therefore the primary focus is not on activity scheduling. The tours provided are predefined by the application developers and therefore the application has no requirement for context-based trail generation or dynamic context-based trail adaptation. LoL@ extends the concepts introduced by the Cyberguide system by ordering the activities that users can undertake. Additionally, LoL@ further validates the concept of using mobile, context-aware technology to assist users in completing a set of activities.

### 2.1.6 CRUMPET

The CRUMPET (CREation of User-friendly Mobile services PERSONALISED for Tourism) system provides a tourist attraction recommendation service to users based on their personal interests and current location [113]. The objectives of the CRUMPET project are to implement and trial tourism-related, value-added services to nomadic users across fixed and mobile networks, and to evaluate agent technology as a suitable approach for creating nomadic services. The end-user application executes on a mobile device that communicates via a wireless network (WiFi or GSM) connection with the remote CRUMPET services. The user can select an attraction from the list provided by CRUMPET and request a tour to this attraction i.e., a route from their current location to the selected attraction.

CRUMPET does not support the creation of multi-attraction tours (trail generation) and therefore does not consider the concept of reconfiguration point identification in the

---

<sup>1</sup>Universal Mobile Telecommunications System

same manner as the applications discussed at the beginning of this section. However, CRUMPET illustrates how contexts such as location and user preferences can be used to determine the set of activities that are relevant to a particular user in a specific contextual situation.

### 2.1.7 m-ToGuide

m-ToGuide is a tourist guide application designed for city travellers [63]. The prototype application was developed as part of an industry-led project targeted at the European tourism market. The primary objective was to replicate the services of a human tour guide on a mobile device that can be rented to tourists. The application runs on a mobile device and communicates via WiFi and GSM/GPRS with the remote m-ToGuide server. The application shows the tourist's location on a map-based interface. Predefined points of interest are automatically selected from a database based on stereotyped default user profiles e.g., family, business traveller. The user's profile is dynamically updated and refined by tracking user behaviour (and subsequently adapting) and through manual user input. The recommended attractions are then dynamically adapted based on refinement of the user's profile. The application guides users through the points of interest, providing routes between the attractions. The application also provides a ticketing facility that allows users to purchase tickets for attractions on their tour.

m-ToGuide does not support context-based activity schedule generation and reconfiguration point identification. The adaptation in m-ToGuide concerns updating the user's preference model based on context events (user input and user actions i.e., what attractions are visited). Updated preference models are used to improve the recommendations that the application makes to users. This type of behaviour is particularly useful in tourist applications when users do not necessarily know exactly what attractions they want to see. The DTG and CRUMPET also recommend attractions to users based on elicited preferences, however only m-ToGuide learns about preferences automatically and refines its recommendations accordingly.

### 2.1.8 Hypermedia Tour Guide

A multimedia tourist guide has been developed for Genoa’s Costa Aquarium [12]. The guide, executing on a mobile device, assists users throughout their tour of the aquarium by providing information about each exhibit (fish tank). Tour order is predefined (as in LoL@) and is not sensitive to changes in context. The tour on the mobile device is arranged in the same sequence as the physical layout of the tanks in the museum, and users select the “next tank” option when they want to view the multimedia information for the next exhibit. Users can create thematic paths through the aquarium by using a “link” tool to link related tanks together e.g., a tour of all crustacean tanks. Through these tools, users can manually create trails and manually reorder them.

The Hypermedia Tour Guide does not contain trail generation and reconfiguration point identification behaviour, nor does it provide activity recommendations like CRUM-PET and m-ToGuide. It can provide static tours like those provided by LoL@. By conducting a comprehensive user study of the application, the Hypermedia Tour Guide project validates the concept of mobile device-based guide applications. The user study illustrated strong public interest and acceptance of computer-based personal guides that provide context-based information e.g., information about the tank at which the user is situated.

### 2.1.9 Summary

Table 2.1 provides a summary of the features provided by state of the art mobile, context-aware tourists guides. None of the projects surveyed state the provision of generic support for trail generation and reconfiguration point identification as a research objective. GUIDE (Section 2.1.1), P-Tour (Section 2.1.2) and the DTG (Section 2.1.3) are the most sophisticated applications but they do not provide any level of support for software reuse in terms of their trail generation and trail reconfiguration implementations. All applications that support trail generation impose a limit on the number of activities considered by the application in order to meet response time requirements. The server-based approaches of P-Tour and the DTG are not appropriate for deployment on a mobile platform due to their dependence on the considerable resources of their host servers. Addition-

ally, these applications provide no support for disconnected operation. The assumptions made by the DTG trail generation algorithm (in the form of heuristics) make it unsuitable for general use outside of the tourism domain (as does the fact that tour attractions are selected based on preferences and not by the user directly). The periodic nature of the reconfiguration point identification mechanisms employed by GUIDE, and to a lesser extent P-Tour and the DTG, mean that the tours provided by these applications are susceptible to becoming temporarily out of sync with the user’s reality. Additionally, P-Tour and the DTG monitor user progress on a trail for deciding if reconfiguration should occur, ignoring contexts related to activity properties.

Feature	GUIDE	P-Tour	DTG	Cyberguide	LoL@	m-ToGuide	CRUMPET	Hypermedia Guide
<b>Trail Generation</b>								
Mobile device-based	√	○	○	○	○	○	○	○
Server-based	○	√	√	○	○	○	○	○
Unlimited activities	○	○	○	○	○	○	○	○
<b>Reconfiguration Point Identification</b>								
Periodic	√	√	√	○	○	○	○	○
Context-based	○	√	√	○	○	○	○	○
<b>Developer Support</b>	○	○	○	○	○	○	○	○

√ = full support ○ = no support

**Table 2.1:** Summary of mobile, context-aware activity scheduling support

A survey of the state of the art in context-aware, mobile tourism guides has been conducted at the Vienna University of Technology, Austria [115]. Of the tour guides included in the survey only GUIDE provides trail generation and dynamic reconfiguration behaviour (P-Tour and the DTG are not discussed). In the ‘Lessons Learned’ section the authors point out the unaddressed issues that they identified in the state of the art during their survey. The issues identified are as follows (issues addressed in this thesis are listed in italics):

- Tourism is not considered as a social activity.

- *The balance between thin and thick clients is problematic.*
- *Potential of incorporating external context not exploited.*
- OGC standards for exchanging geospatial information widely ignored.
- *Time and network context are seldom used.*
- *The potential of combining context properties is not exploited.*
- Context chronology not widely supported.
- Proprietary representation of context data.
- Context availability partly regarded.
- Varying automation of context acquisition.
- *Push-based context access not widely supported.*
- *Dynamic adaptation of guided tours is not generally provided.*
- *Extensibility of adaptation operations is not commonly recognised.*

The application framework presented in this thesis answers the issues listed in italics through the provision of reusable and extensible trail generation and reconfiguration point identification components for mobile devices that consider multiple context sources (the relevance of which can be set by the user) to make trail generation and reconfiguration point identification decisions. Of those issues not addressed by this thesis, all but the first issue listed are addressed by other components in the Hermes framework (notably the `Context Management` component and the `Geographical Information System` component). The first issue (‘tourism is not considered as a social activity’) is partially addressed in `GUIDE` through the provision of a messaging client that assists groups of tourists with keeping in touch as they explore the city separately or in subgroups.

## 2.2 Context-Aware To-do Lists

This section presents research projects involving the design and implementation of context-aware to-do lists and discusses the behaviour they provide that addresses the challenges described in Section 1.4.

### 2.2.1 TaskMinder

The TaskMinder [73] application developed at Georgia Institute of Technology uses context information (location, user activity, task history) and machine learning techniques to provide a to-do list management tool that recommends tasks to the user based on context. The primary objective is to improve on static computerised to-do lists such as Outlook Task [5], Daily To-Do List [34], Tree To Do List [97] and Task-O-Matic [119]. The end-user application is deployed on a laptop and communicates via WiFi with a remote TaskMinder server. The server is responsible for suggesting which tasks the user should undertake when the user makes a task query (tasks are initially entered by the user), similar to the task suggestion behaviour in m-ToGuide, CRUMPET and the DTG. The server suggests tasks to users based on the following factors:

- The user's current level of activity - derived from the number of application windows the user has open on their laptop.
- The user's location - derived from WiFi information.
- The current time of day.
- The due date of the task (if previously specified by the user).
- Historical user requests:
  - The type of task requests the user has made in the past.
  - The times at which the requests were made.
  - The feedback rating the user assigned to the task suggestions previously received.



Task suggestions are made based on user input that consists of a preferred task duration. The system estimates how long certain tasks will take based on task history (it is not possible for the user to specify task durations when entering a task to the system) and returns the appropriate tasks, in order of importance, to the user. The server uses a weighted syntactic matching technique to map existing tasks to historical tasks to ascertain if the existing tasks are similar to those that have been completed in the past. The words in the task description are weighted so that certain words have a greater effect on finding a match. For example, if searching for matches to a task called ‘Lunch with Jeff’, the word ‘Lunch’ has a higher weight than the word ‘with’. If a match is found, the situation that the historical task was completed in is compared to the user’s current context. If the situations match then the task is considered to be appropriate for the user. If no matches are found (or if the user has no task history) the system assigns default values to tasks so that some tasks, regardless of relevance, are returned. System recommendations improve incrementally as a task history database is compiled.

TaskMinder does not feature the concept of generating a trail through a set of tasks defined by the user, nor does it let users directly specify situations in which they want to be reminded about tasks. The task specification is rather course-grained e.g., the user is unable to specify how long they estimate a task will take, inhibiting schedule generation behaviour. The application is not proactive in the manner of a mobile, context-aware activity scheduling application. The list of tasks recommended to the user does not automatically update based on context change. Finally, the tasks recommended to the user are recommended in order of importance and there is no consideration of the list as an entity or the relationships between activities. Without considering the list as an entity it is not possible to compare candidate lists and provide the best list ordering to the user i.e., to ensure that lists don’t contain activities that clash. Therefore, it is possible that the activities on the list provided by TaskMinder may clash with each other, forcing the user to make clash resolution decisions.

### 2.2.2 CybreMinder

The CybreMinder system addresses the need for context-awareness in to-do lists by providing context-aware reminder behaviour [35]. The tool has two main features - reminder creation and reminder delivery. Reminders can be created by the user for himself or for third parties. A collection of reminders serves as a to-do list. Each reminder has a subject, a priority level (from lowest to highest), a description and an expiration date. Users can also associate a situation with each reminder e.g., the user must be in a certain building at a certain time for the reminder to be valid. Relationships other than ‘=’ can be used. For example, the user could set a trigger time for after 9pm by using the ‘>’ relation. Other supported relations are ‘≤’, ‘≥’ and ‘<’.

Reminders are delivered when the situation specified in a task reminder exists or when the task expiration date has been reached. Users can be notified by email, SMS on a mobile phone, the display on a nearby computer or by local printer (to emulate paper to-do lists).

CybreMinder, by allowing users to schedule reminders for specific situations, gives them more control over the activities they are reminded about than TaskMinder. However, like TaskMinder, the application does not consider the notion of grouping sets of reminders effectively based on the user’s context. The application is primarily focused on supporting the specification and triggering of individual tasks. If multiple tasks are triggered simultaneously there is no facility to reason about the collection of tasks and suggest how the user should address them. Therefore CybreMinder cannot be used to support trail generation or reconfiguration point identification, only static tour behaviour such as that featured in LoL@ and the Hypermedia Tour Guide.

### 2.2.3 *comMotion*

*comMotion* is a location-aware reminder system that maintains to-do lists for users at the locations that the lists are associated with [80]. *comMotion* introduces a location-learning agent that observes the user’s frequent locations over time and allows them to be labelled e.g., ‘Work’, ‘Home’. To-do lists can then be associated with these labelled locations. A to-do list in *comMotion* is composed of text items or audio recordings. When users enter

a location associated with a to-do list they will be notified by an audio alert that they have tasks associated with their current location. Tasks are ticked off by the user as they are completed.

*comMotion* also provides a reminder system based on the 3M Post-It<sup>2</sup> note metaphor. This behaviour is similar to that provided by CybreMinder. Context-aware reminders can be sent via email, specifying the location name in the subject line. Reminders can be constrained to a certain date or date and time range and the user can specify the frequency of repetition for the reminder e.g., daily, weekly, monthly or none. When the user enters a context specified by a reminder the relevant reminder will be displayed. Users can also subscribe to information services such as headline news, weather forecasts and current movie listings on a per-location and time basis e.g., the user could request to receive a movie schedule on a Friday after leaving their place of work.

Similar to TaskMinder and CybreMinder, *comMotion* does not support reasoning about how best to undertake the tasks specified in a to-do list when a to-do list is displayed to the user. There is no reasoning beyond identification of when a reminder should be triggered. This prevents *comMotion* from being able to facilitate the implementation of mobile, context-aware activity scheduling applications apart from those in which the user or programmer is responsible for manually scheduling all tasks.

#### 2.2.4 PlaceMail

The PlaceMail system [77] developed at the University of Minnesota provides location-aware reminders in a manner similar to CybreMinder and *comMotion*. Users of PlaceMail can use the system to specify messages to be delivered to them when they are in a specific predefined location or set of locations. Users can also specify a delivery date and time instead of, or in addition to, delivery locations. The PlaceMail system executes on a mobile phone platform, differing from TaskMinder, CybreMinder and *comMotion* which are deployed on laptop, desktop to multiple delivery platforms and mobile PC respectively. While there is a task creation interface on the phone there is also a web interface, with the same functionality, for ease of task entry. Messages are entered by providing a delivery

---

<sup>2</sup><http://www.postit.com>

location (or set of delivery locations), a message body (text or audio) and an optional date and time. The user is notified of message delivery by a short audio alert and delivered messages can be rescheduled for future redelivery.

PlaceMail uses a client-server architecture. User data and locations are stored in a database residing on a remote server. The user's mobile phone receives the user's tasks and locations via a wireless network connection when the user logs into PlaceMail. The user's mobile phone (equipped with GPS) transmits its location to the server every 60 seconds and the server determines whether there are any messages relevant to the current location. Messages deemed relevant are delivered.

Context-aware delivery of individual messages is the primary focus of the PlaceMail application, a focus it shares with all of the context-aware to-do list applications reviewed so far. As a result, PlaceMail does not support reasoning about groups of reminders to determine the most effective way to undertake the tasks described in the reminders. PlaceMail can be used to develop applications with static activity schedules but not context-aware schedules such as those offered by GUIDE, P-Tour and the DTG.

### **2.2.5 Place-Its**

A Place-It is a virtual post-it note that can be posted by its author to remote places and displayed to the author when he enters that location in the future [120]. This behaviour is similar to that provided by CybreMinder, *comMotion* and PlaceMail. Like PlaceMail, the Place-Its application executes on a mobile phone and considers user location when deciding whether or not a note should be shown to the user. Three main components collaborate to provide the Place-Its reminder functionality:

1. Trigger - identifies whether the reminder should be signalled upon arrival at or departure from the associated location.
2. Text - the message associated with the note.
3. Place - the location with which the note is associated.

Once a note is triggered it is automatically placed in a Removed Place-Its list, unlike notes in *comMotion* (Section 2.2.3) where the user manually specifies that notes have

been addressed. Once a note has been removed it can be edited and re-posted to the same or a different location. This functionality prevents loss of information in the case where a note is displayed to the user but not actually acted upon in the real world.

The Place-Its system follows the same single activity-centric model of task management as TaskMinder, CybreMinder, *comMotion* and PlaceMail. The Place-Its system does not support any form of context-aware reasoning about collections of notes in terms of recommending a schedule of activities based on the contents of the Place-Its. It introduces the notion of automatically designating a task as having been completed.

### 2.2.6 Castaway

The Castaway project is investigating the development of a context-aware task management system [65]. The vision of the Castaway project consists of three parts:

1. To support the fast and convenient input of tasks the instant they are conceived.
2. To provide a lightweight, flexible tool to view and manage these tasks.
3. To provide a system for reminding users of their tasks at precisely the right place and/or time.

The Castaway developers are currently investigating the application user interface, which is a major focus of their work. Castaway has prototyped various task views including a list view, a map view and a calendar view and has evaluated these views by means of a user study. This user study highlighted user preferences for map-based task management interfaces. The Castaway developers have also investigated techniques for managing information display when clusters of task icons appear in one location and obscure each other. In terms of user interface design the Castaway project is significantly more advanced than the other context-aware to-do list applications described in this section.

The task management behaviour envisioned in Castaway is described in a similar manner to that of TaskMinder, CybreMinder, PlaceMail and Place-Its, where the focus is on managing individual tasks as opposed to providing the user with an effective context-aware ordering for a collection of tasks. This will prevent Castaway from providing context-aware activity scheduling.

### 2.2.7 Summary

This section has presented the state of the art in mobile, context-aware to-do lists. In general, the applications allow users to specify contextual situations e.g., a location and time pair, with which reminders are associated. The user is notified of a reminder when its associated contextual situation exists.

The applications surveyed are informative from the perspective of learning about the type of context information users typically need to schedule reminders. Location and time are the two contexts mostly widely used by the applications discussed. These contexts are supported in the application framework presented in this thesis.

While context-aware to-do list applications improve on static to-do lists by prompting the user about specific tasks when appropriate, the applications do not facilitate context-aware activity scheduling. Tasks are reasoned about individually as opposed to collectively, meaning that the concept of one specific task grouping being more valuable to the user than another is not considered. When a collection of tasks or reminders is displayed to the user there is no advice regarding how to go about undertaking the various activities i.e., there is no activity ordering. Tasks/reminders are presented when the contextual situation associated with the task exists, regardless of whether the user is actually in a position to act on the task or not. Additionally, tasks presented to the user simultaneously may clash.

Apart from TaskMinder, all of the applications reviewed can be used to develop mobile, context-aware applications that provide static activity schedules to user in the same manner as the LoL@ tourist guide application. Dynamic context-aware activity scheduling behaviour is not supported.

## 2.3 Mobile, Context-Aware Application Frameworks

This section describes projects investigating application frameworks for mobile, context-aware computing. The types of applications supported by these application frameworks include mediascapes, context-aware museum and city guides, hypermedia applications and generic rules-based context-aware applications. Although none of the application frameworks specifically address mobile, context-aware activity scheduling in the same

manner as GUIDE, P-Tour and the DTG, they can be used to develop applications that provide some support for activity scheduling for the mobile user. The basic functionality of each application framework is presented, along with a discussion of the support for context-aware activity scheduling provided and the application framework design. The projects reviewed are informative in terms of illustrating how application frameworks can support developers in implementing mobile, context-aware applications.

### 2.3.1 Mobile Bristol

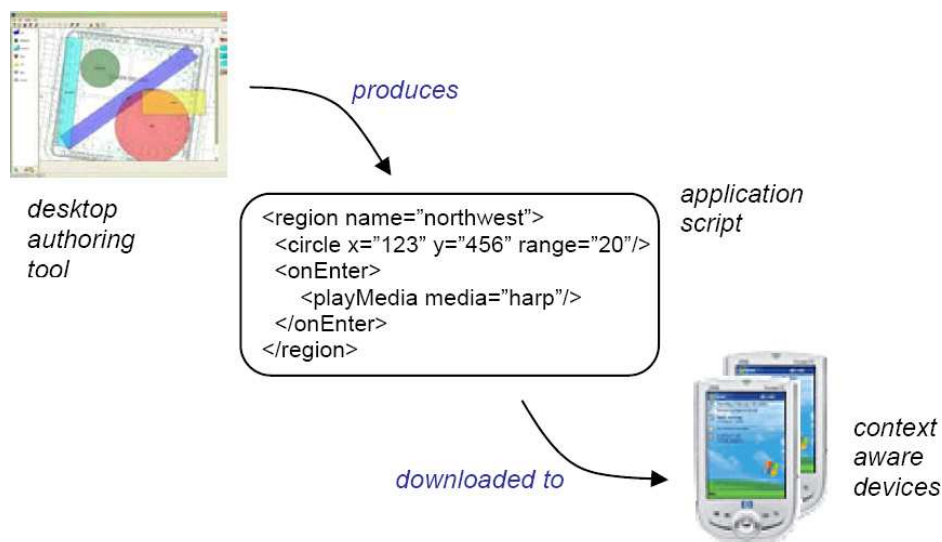
The Mobile Bristol toolkit is an application framework that supports rapid authoring of mobile, context-aware applications [105]. The application area supported by the framework is that of ‘mediascapes’ - applications that are concerned with delivering digital media e.g., video, audio, still images, plain text or HTML, when predefined contextual situations occur e.g., when the user enters a specific geographic area. The application allows users to specify the following properties of their application:

- The type of digital content the user will encounter.
- Where the content is encountered (e.g., inside a specific area, outside a set of areas).
- How the digital media is triggered (e.g., automatically, when a button is pressed, at a certain time).
- How interactions are presented to the user (audio or screen-based).

The Mobile Bristol developers were inspired by the democratisation of publishing enabled by the Web where almost anyone can make and deploy a simple web site. The toolkit has been used to build applications by diverse user groups (aided by developers) including school children, educationalists, artists and television programme makers. Applications developed include soundscapes for a piece of open ground near a school [133], a location-sensitive heritage guide for a ferry boat company [30] and mediascape-based art installations for an urban square [105].

In line with the project objective of enabling non-programmers to develop mobile, context-aware applications, the toolkit provides an authoring environment that has a

graphical user interface (GUI). A programmer's editor is provided for users who wish to specify behaviour that is more complex than that supported by the GUI. Each new function that is programmatically defined is added to the default set available via the GUI, allowing toolkit extensions defined by experienced developers be used by developers who only use the GUI. New functions are described in a scripting language called MBML (Mobile Bristol Markup Language). This XML-based language contains constructs for conditional logic, state variables and functions and makes the authoring environment independent from the deployment platform, as illustrated in Figure 2.2. MBML scripts must be interpreted by client devices and to date an implementation for the iPAQ<sup>3</sup> PDA has been developed. The separation of the behavioural specification from the device-specific interpretation has the desirable property of facilitating the deployment of the same application on many platforms. This benefits application developers but requires a significant amount of work on the part of a third party to write the device-specific MBML translations.



**Figure 2.2:** Separation between authoring tool and device-specific implementation [105]

The Mobile Bristol toolkit can be used to develop static tour guide applications similar to the LoL@ application described in Section 2.1.5 and location-aware reminder systems like those discussed in Section 2.2. However, the toolkit does not support the notion of dynamically changing the manner in which users of mediascape applications are directed

<sup>3</sup><http://www.ipaq.com>



through an active space and therefore does not automatically support context-aware activity scheduling. The framework could be extended to facilitate context-based activity scheduling through the addition of new MBML functions, although this would be a significant undertaking for an application developer.

### 2.3.2 Stick-e Note

The Stick-e Note architecture developed at the University of Kent, UK was one of the first attempts at supporting the development of mobile, context-aware applications [18]. The conceptual architecture proposes the use of the electronic equivalent of a post-It note to allow authors to develop mobile, context-aware applications in much the same manner as they would web pages. Stick-e notes reside on a mobile device and contain information that is displayed to users when the contextual situation described in the note exists. Notes are created using Standard Generalized Markup Language (SGML) and displayed in a web browser. The following types of context can be considered by Stick-e Note applications:

- Location - a note is triggered when a user is in the location specified by the note.
- Adjacency - a note is triggered when the user is near other objects e.g., user is in the presence of John (or any person, animal, object carrying a suitable identity transponder).
- Critical states - a note is triggered when a critical state is reached e.g., a share price or the air temperature rises above a specified threshold.
- Computer states - a note is triggered when a user accesses a certain directory.
- Imaginary companions - notes are triggered based on users stating that they are in the presence of imaginary companions e.g., architects, cartographers. Notes relevant to the imaginary companion's specialist topic are triggered when appropriate to simulate a guided tour by an expert.
- Time - a note is triggered at a specified time.

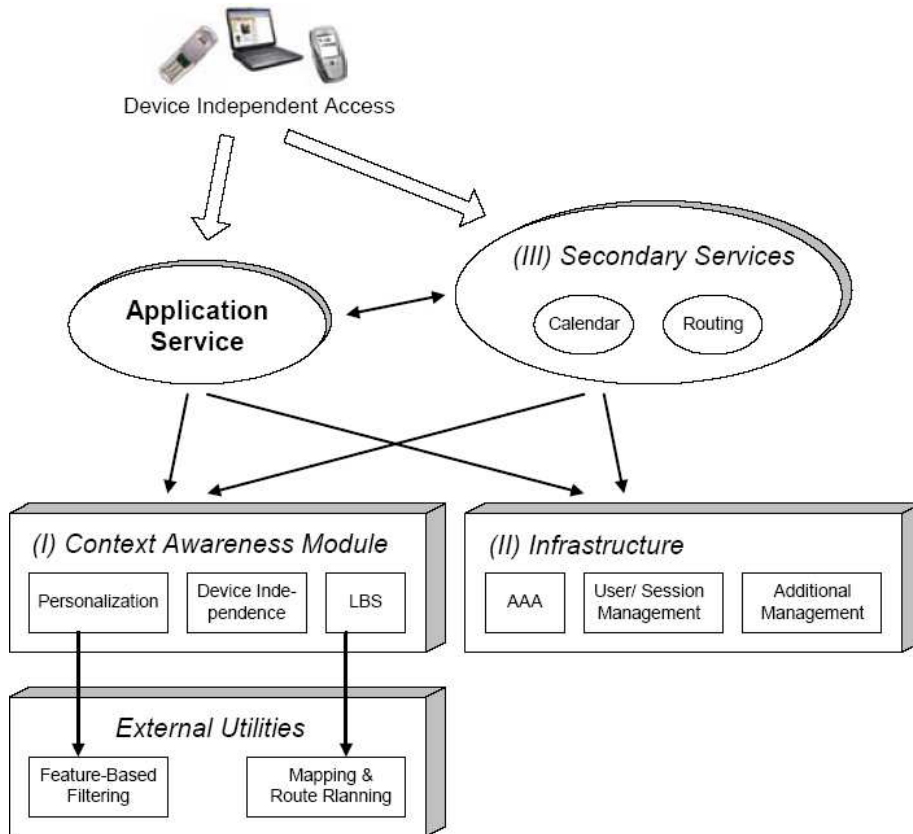
The software to support the use of SGML Stick-e documents consists of four components. SEPREPARE allows authors to prepare notes and documents. SEMANAGE deals with the management of primed documents (a subset of all the notes on the device that user has marked as being eligible for triggering). SETRIGGER runs in the background and causes any of the currently primed notes to be triggered if its context becomes satisfied. SESHOW stores the triggered notes and presents them to the user. Several instances of this component may run at once, each one representing a different application e.g., tourist guide, personal pager etc. An actual implementation of the software is not discussed. The software components are not accessible to Stick-e note authors, who define application behaviour through SGML note specifications in a similar manner to authors of Mobile Bristol applications.

The Stick-e architecture can be used to create guided tour applications that work by triggering notes based on user location. The notes could guide the user by containing directions to the next note location, as well as the information about the location the user is currently in. The architecture can also be used to develop task reminder applications that notify the user of tasks that can be completed when they are in specific situations, similar to the context-aware to-do list applications discussed in Section 2.2. Like the context-aware to-do list applications, Stick-e notes are not reasoned about collectively and therefore the Stick-e architecture does not automatically support the concept of a dynamic activity schedule. The framework could be extended to provide dynamic note content based on current context. Such notes could be used to implement mobile, context-aware activity scheduling applications. The proposed extension would require significant changes to all framework components.

### **2.3.3 BerlinTainment**

The BerlinTainment serviceware framework is a multi-agent system that supports the development of mobile, context-aware services [135]. The framework supports the connection of heterogeneous end-user mobile devices to a remote application server's services, with a focus on the provision of entertainment services. The framework, illustrated in Figure 2.3 consists of several modules, each of which contains several services:

- The context-awareness module provides services for:
  - Personalisation (filtering information based on user context).
  - Location-based services (provides knowledge of the user's location and co-located points of interest).
  - Device and network independence (generates user interfaces for different devices and intelligent session management for network migration).
- The infrastructure module provides services for management of users, sessions and services.
- The external utilities module provides services for mapping, route planning and data filtering (used by the context-awareness component).



**Figure 2.3:** BerlinTainment framework architecture [135]

The framework implementation is based on a FIPA<sup>4</sup>-compliant Multi-Agent System architecture called JIAC (Java Intelligent Agent Componentware) [7]. JIAC is a component-

<sup>4</sup>The Foundation for Intelligent Physical Agents - <http://www.fipa.org>

based architecture and provides services such as mobility, communication, security and a general methodology for agent-oriented application development. As illustrated in Figure 2.3, the architecture is server-based, with client devices communicating wirelessly with application services hosted on remote servers. JIAC provides behaviour to support device-independent access to services by determining the device type and transforming the user interfaces as appropriate e.g., into HTML/WML for browser-based devices or VXML for voice-based interfaces. The component-oriented nature of the BerlinTainment framework facilitates component replacement without affecting the behaviour of client components. Additionally, applications can choose which components are necessary and limit the application to only those components.

The service framework has been used to build an entertainment planning application for the city of Berlin, Germany [134]. The application assists users in finding restaurants, plays, movies and concerts based on their location, and provides a routing service to selected attractions. The application also provides an intelligent day planner service. The day planner allows the user to schedule multiple high-level activities for a given day and to receive personalised, location-based recommendations for each activity e.g., specific film recommendations if the user selected a cinema activity. The user uses the system to connect to the ticketing services of relevant attractions and to plan routes between the various attraction locations. The service selects and schedules attractions based on their properties, most notably time constraints and location, to minimise the temporal and spatial distances between the recommended activities. The schedules presented to users can be changed by removing attractions or searching for new recommendations. The BerlinTainment-related publications have not discussed the details of the schedule generation algorithm used by the remote server when composing schedules.

The BerlinTainment service framework does not support dynamic reordering of a suggested schedule based on changes in user context and there is no concept of changes to activity properties. Additionally, the client/server architecture employed indicates that the schedule generation algorithm is designed to use the resources of a powerful server platform like P-Tour and the DTG and would require significant re-engineering if it was to be deployed on a mobile client device. Like CRUMPET, BerlinTainment illustrates how context can be used to determine the activities that are relevant to a particular user

given his current context.

### 2.3.4 PEACH

The Personal Experience with Active Cultural Heritage Project (PEACH) is investigating the concept of an ‘Active Museum’ [71]. Active museums are characterised by:

- Multiple users in a single place.
- The size of the set of users using the active museum system changing dynamically.
- Services being provided by a collection of components that can join and leave the environment and can operate anywhere. This behaviour is hidden from the user who interacts with the system as if it is a standalone system.
- Overlapping services that require components with overlapping behaviour to coordinate in order to decide which component should provide a specific service and how.

The objective of the project is to provide an agent-based architecture to support the development of museum guide systems that integrate with the visitor’s physical museum experience and do not compete with the actual exhibit items for the visitor’s attention. Through the development of two museum guide systems for museums in Trento, Italy and Haifa, Israel the PEACH developers composed a set of generic agents that can be used to provide context-based guide systems for museums. The agent framework contains the following components:

- *Spatial Information Mediator* - responsible for providing the user’s location to the other agents that use this information. Location data is provided periodically or can be requested. The agent models positioning relationships between users and museum exhibits.
- *User Modellers* - responsible for providing a collection of ways in which to model system users.

- *Presentation Composer* - responsible for providing audio, text, slide or video presentations about exhibits.
- *Information Brokers* - responsible for providing the information that is displayed to the users (multimedia artefacts that can be stored locally or remotely).
- *Presentation Clients* - responsible for managing the presentation of information in a device-specific manner.

The Presentation Clients reside on the user's mobile device while the remainder of the components are deployed remotely. The components are relatively application-specific e.g., a component is dedicated to location sensing and dissemination as opposed to context management in general. However, within the application area supported by the framework there is scope for component extension without affecting the system as a whole e.g., the specifics of the location system can be changed without any effect on clients.

Using these components, developers can implement active museum applications that, based on user location and mobile device characteristics, will display multimedia information about the exhibit the user is near. The PEACH framework facilitates the development of applications like CybreMinder, LoL@ and, to a lesser extent, applications that can be built using the Mobile Bristol toolkit and the Stick-e Note architecture. PEACH does not have the expressive power of Mobile Bristol or Stick-e Note in terms of describing contextual situations. Although the framework supports tour guide applications it does so only in the same manner as the Stick-e Note architecture and the Mobile Bristol toolkit, where the tours are specified by developers before the user uses the application, meaning that there is no scope for flexibility. The concept of reasoning about collections of activities is not considered, as all logic supported is at the individual activity level. Therefore PEACH does not support museum guide applications featuring context-based trail generation and reconfiguration point identification.

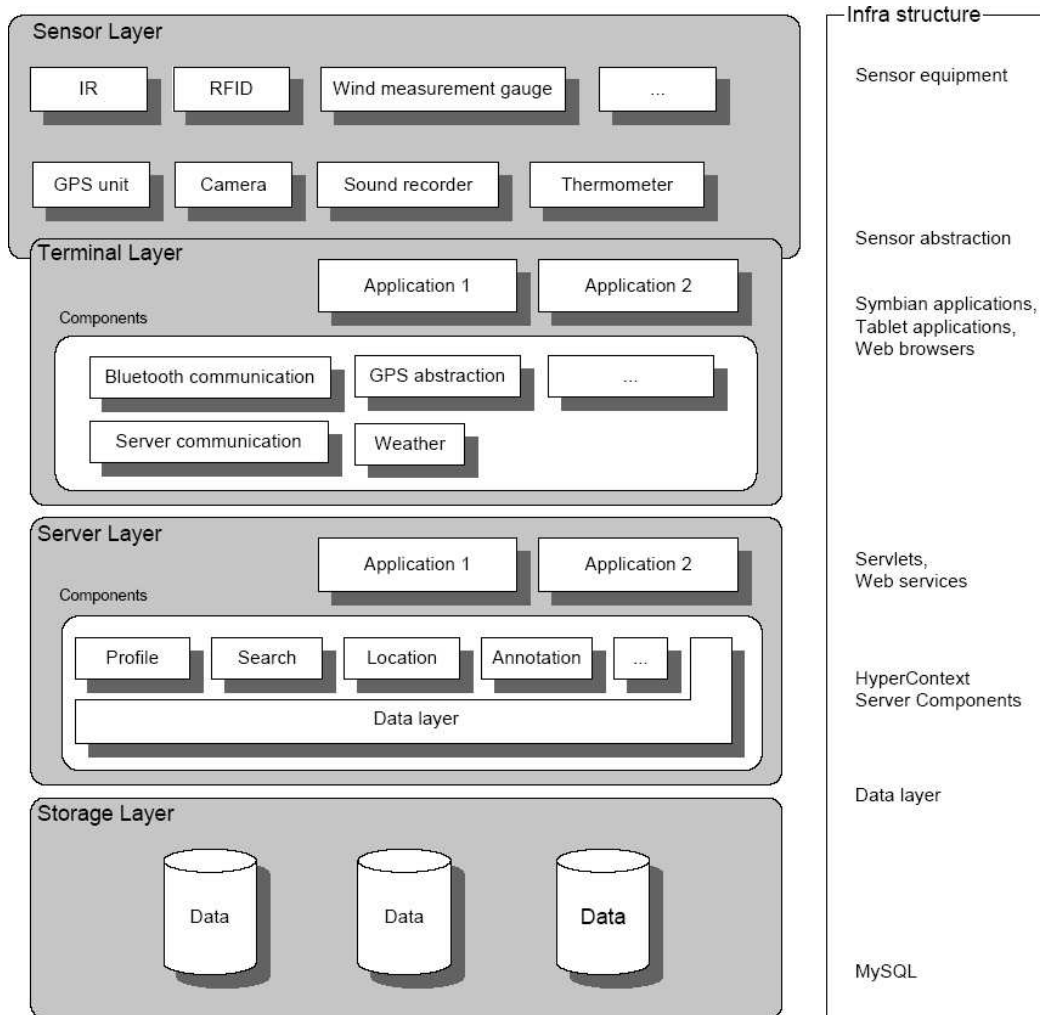
### 2.3.5 HyCon

HyCon is an application framework for context-aware hypermedia systems [17]. HyCon supports the implementation of ‘authoring in the field’-type applications, allowing users annotate locations on a map-based interface with information (hyperlinks and other multimedia information) while exploring the actual location in the physical world.

The HyCon framework is divided into four layers as illustrated in Figure 2.4. The storage layer (at the bottom of the diagram) is responsible for persistent storage and resides on a remote server. The server layer forms the basic functionality of the framework, providing data layer, location, annotation and subscription sub-components. The terminal layer resides on end-user mobile devices that have access to sensor information, and provides mechanisms for communicating with the remote server housing the storage and server layers. The sensor layer provides mechanisms for parsing and reasoning about raw sensor information. The components at each layer are used to create service applications that implement interfaces to the components’ functionality. The component-based approach is similar to that used in BerlinTainment, PEACH and Hermes (with which HyCon shares common components), minimising inter-component dependencies to facilitate component reuse and extension.

The HyCon framework has been used to implement a prototype application called HyConExplorer. The application is deployed on a mobile device and communicates with the server components via WiFi or GPRS. The map-based application provides implementations of several context-aware hypermedia techniques:

- Context-aware browsing - allows users to see hypermedia artefacts based on their current location and the current time of day. The user’s map-based interface is annotated with the hypermedia artefacts that exist for the location the user is currently in.
- Context-aware searching - allows users to search for specific artefacts e.g., shops, and bases the results on the user’s location, the time of day and the opening hours of the shops.
- Context-aware annotation - allows users to associate hypermedia content with map



**Figure 2.4:** The HyCon framework architecture [17]

locations.

- Context-aware linking - allows users to link geographically distinct hypermedia artefacts together.

The HyConExplorer application can use its context-aware linking functionality to express links or paths through a collection of hypermedia objects. Each link represents directions from one location to another, with the linked objects representing the destinations. Using this functionality the tool can be used to create static tours like those provided by LoL@. Tour guide applications created by the HyConExplorer work in the same manner as those created using the Mobile Bristol toolkit, the Stick-e Note architecture and PEACH, where digital artefacts are associated with specific locations and contain information about the



current location as well as directions to the next location on the tour. HyConExplorer does not consider the concept of dynamically adapting link relationships between hypermedia objects and therefore cannot support the dynamic generation and reconfiguration of trails through hypermedia objects based on context information.

### 2.3.6 iCAP

iCAP (Context-aware Application Prototyper) is an application framework that facilitates the authoring of context-aware applications by the end-user (similar to the Mobile Bristol toolkit) [37]. iCAP allows users to describe a situation and an action to associate with it. The desktop-based system facilitates prototyping via a visual, rule-based system that supports three types of context-aware behaviour:

1. Simple if-then rules - rules where an action is triggered when a condition is satisfied e.g., “If I am home alone, then play my favourite music at high volume”.
2. Relationship-based actions - supports behaviours involving personal, spatial and temporal relations e.g., “If my housemate is in the next room while I am in the house, then remind me to ask him about the grocery shopping”.
3. Environment personalisation - supports the personalisation of environments based on the different preferences of its inhabitants e.g., adjust room light level depending on the preferences of its current occupant(s).

Users interact with iCAP by first creating elements relevant to their rules (if they do not already exist in the system repository). The elements included in the system repository by default are: objects, activities, locations, people and time. Second, the elements created/selected are composed to create rules e.g., associating objects with locations. Finally, the rule created can be simulated or connected to live context sensing infrastructure.

iCAP contains two main components: a visual rule-building interface and a rules engine that stores the user-specified rules and evaluates them when an application using the rule is running. The software components themselves are not designed with the express purpose of being extensible by iCAP users. The iCAP user interface allows non-technical users to specify rules for context-aware applications through the manipulation

of graphics. Users are not required to edit rule source code. Therefore any extension of application default behaviour is done through the specification of new rules and the elements relevant to rules. This is a similar approach to that taken in the Mobile Bristol toolkit.

iCAP can be used to build guide applications by creating a collection of rules that present content e.g., user-defined activity descriptions, when someone enters a particular location. Reminder systems can be built in much the same way. In this way, iCAP facilitates the creation of systems like the context-aware to-do lists described in Section 2.2 and context-aware tour-guides such as LoL@, as opposed to dynamic activity scheduling applications like GUIDE, P-Tour and the DTG. The expressiveness of iCAP in terms of defining situations is similar to that of the Mobile Bristol toolkit and the Stick-e Note architecture as opposed to more domain specific frameworks such as PEACH.

### **2.3.7 Summary**

This section has presented the state of the art in application frameworks for mobile, context-aware applications. Some of the frameworks facilitate the creation of context-aware applications through the use of a graphical user interface and extensible rule-base (Mobile Bristol toolkit, Stick-e Note and iCAP), while the remainder provide generic application components that can be reused and extended by software developers.

Although none of the application frameworks explicitly support mobile, context-aware activity scheduling, they can be used to provide some level of related behaviour. All of the application frameworks reviewed can be used to create both static tour guide applications and context-aware to-do lists applications. iCAP, the Stick-e Note architecture, BerlinTainment and HyCon list these applications in their publications as examples of applications that can be built using their respective application frameworks. However, none of the application frameworks target the mobile, context-aware activity scheduling domain. In the same manner as the context-aware to-do lists discussed in Section 2.2, the application frameworks do not support the development of applications that can reason about the relationships between collections of tasks or activities - they support only reasoning about individual activities. For this reason the application frameworks can not

automatically provide generic, reusable and extensible support for mobile, context-aware trails-based applications. A significant amount of software development effort is required in order to extend the relevant application frameworks so that they can be used to develop trails applications.

## 2.4 Context-Awareness Frameworks

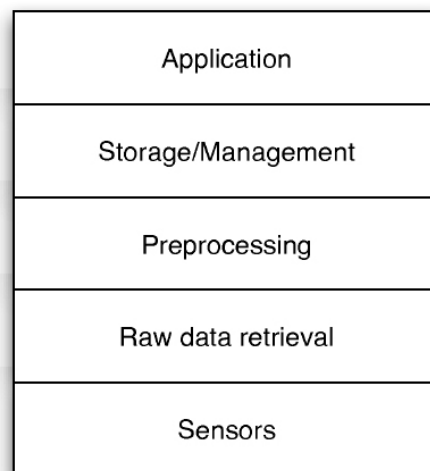
Context-awareness frameworks provide developers with some or all of the basic tools required to create applications that adapt their behaviour based on changes in relevant context. Numerous frameworks have been proposed in recent years such as JCAF [10], the Sentient Object Model [14], AURA [22], Hydrogen [51], ContextFabric [52], Context Shadow [59], ContextStudio [68], ContextPhone [103], Gaia [107] and the Context Toolkit [110]. The primary goal of a context-awareness framework is to make context information available to application developers so that they can build applications without concerning themselves with the specifics of context acquisition and management. Context-awareness frameworks therefore address challenges such as context retrieval (acquisition of raw context data from heterogeneous sensors), context data transformation (converting raw context data to a higher-level format), context reasoning (deducing new context information by combining context from various sources) and context modelling (providing application developers with access to context).

Baldauf et al. at the Vienna University of Technology, Austria have published a survey of context-awareness frameworks that explains the different elements common to context-aware systems by describing a conceptual context-awareness framework [9]. The framework, illustrated in Figure 2.5, consists of the following layers (from bottom to top):

- *Sensor Layer* - this layer consists of physical sensors (e.g., light, audio, location, acceleration, touch, temperature), virtual sensors (e.g., activity sensing by monitoring mouse movement and keystrokes) and logical sensors - combining physical and virtual sensors (e.g., determining user location by recording PC logins and looking up a database that maps PCs to locations).
- *Raw Data Retrieval* - uses appropriate drivers for physical sensors and APIs for

virtual and logical sensors to provide abstract methods for sensor access, making it possible to exchange underlying sensors e.g., a GPS location system could be replaced by an RFID system without major modification to the current and upper layers.

- *Preprocessing* - responsible for reasoning and interpreting contextual information, raising the results of the Raw Data Retrieval layer to a higher level of abstraction for use by application developers.
- *Storage Management* - this layer organises the context data gathered by the lower layers and offers them to the client (Application layer) via public interfaces. This layer supports both synchronous (client polls server) and asynchronous (client subscribes to server) context data access.
- *Application* - the behaviour that reacts to context events is implemented in this layer e.g., a smart device display that uses the lower layers to detect bad environment illumination and reacts by displaying text in higher colour contrast.



**Figure 2.5:** The conceptual framework for context-aware systems [9]

The survey also discusses existing context-awareness frameworks and illustrates the services they provide to application developers. These frameworks address common challenges in context management such as context acquisition, fusion and reasoning but due

to their application-independent nature they do not explicitly support context-aware activity scheduling. The concepts put forward in these works are informing related areas of the Hermes framework architecture, as discussed briefly in Chapter 3.

## 2.5 Chapter Summary

This chapter has presented the state of the art in mobile, context-aware activity scheduling for tourists, context-aware to-do lists and application frameworks for mobile, context-aware computing. A brief overview of context-awareness frameworks in general was also presented. While the most sophisticated approaches to activity scheduling for mobile tourists (GUIDE, P-Tour and the DTG) provide mechanisms for trail generation and reconfiguration point identification that are suitable for meeting their specific objectives, the techniques are not sufficiently generic and therefore cannot be used to support application developers implementing mobile, context-aware activity scheduling applications outside the tourism domain. In addition, the trail generation mechanism used by GUIDE limits the number of activities that can be considered to nine and considers a limited range of contexts. This situation is improved by P-Tour and the DTG but at the cost of introducing a remote tour calculation server. P-Tour and the DTG do not provide tour management behaviour on the mobile device to support disconnected operation. The reconfiguration point identification mechanisms provided by all three approaches contain a periodic element, leading to potential for tours becoming out of date between periodic reconfigurations. Although P-Tour and the DTG improve on the GUIDE approach by only reconfiguring the tour if necessary, their reconfiguration point identification decision making techniques are based on user location and cannot be easily extended to cater for other types of non-spatial context.

The state of the art projects in context-aware to-do list management allow users to associate reminders (in various forms e.g., text and audio) with contextual situations. The reminders are triggered when appropriate. These applications cannot make suggestions, based on context, about how users should go about undertaking the activities referred to in their reminders in the case where users have multiple reminders in a contextual situation. The applications reason about single activities as opposed to the relationships

between multiple activities and therefore do not support context-aware activity scheduling.

Application frameworks for numerous mobile, context-aware application areas exist but none support the implementation of mobile, context-aware activity scheduling in a way that meets the challenges presented in Section 1.4. Many of the frameworks facilitate the implementation of tour guides that provide static tours and context-aware to-do list management applications. However, for the same reason that the context-aware to-do list management systems do not support trail generation and reconfiguration, the mobile, context-aware application frameworks cannot, without significant extension, be used to build mobile, context-aware trails-based applications. Context-awareness frameworks do not support the implementation of specific application types.

The next chapter describes the design of an application framework that addresses these limitations and provides components for trail generation and reconfiguration point identification to support developers in implementing mobile, context-aware trails-based applications.

# Chapter 3

## Design

The state of the art approaches to mobile, context-aware activity management presented in the previous chapter illustrate that the challenges discussed in Section 1.4 are not addressed by existing tools and applications.

A trail generation mechanism should execute on a mobile platform to avoid loss of service due to wireless network disconnection, and should not constrain the number of activities that an application can consider. State of the art approaches to trail generation are server-based, and provide no disconnected operation in relation to trail generation. These approaches also limit the maximum number of activities they can consider based on their response time requirements.

A reconfiguration point identification mechanism should identify when trail reconfiguration is actually necessary in order to avoid needlessly consuming resources through unnecessary speculative trail reconfiguration. Additionally, the mechanism should maximise the amount of time that the trail accurately represents the state of the user's environment. State of the art approaches to reconfiguration point identification are based on periodic trail reconfiguration and the consideration of a limited and non-extensible set of contexts, giving rise to the possibility that the trail will become out of sync with the user's environment. Additionally, none of the state of the art approaches to trail generation and reconfiguration point identification are designed to be reused or extended by third party developers.

This chapter describes the design of an application framework for mobile, context-aware trails-based applications that addresses trail generation and reconfiguration point

identification in a reusable, extensible, application-independent manner. The chapter begins with a discussion of the application development-led design approach followed during the development of the framework. The initial high-level design of the Hermes framework is present next, along with an overview of how Hermes supports the acquisition and modelling of context for use in trails applications. This is followed by a discussion of the trails applications developed during the design process, and details of the design of the trail generation and trail reconfiguration point identification mechanisms included in the application framework.

### 3.1 Design Approach

The application framework for mobile, context-aware trails was developed based on the ‘Three Examples’ approach to framework development [106]. This approach involves building three example applications (prototype applications) of the same type e.g., trails applications, and composing a framework from the elements common to the three applications. This approach is suitable for developing application frameworks for a specific problem domain. The rationale behind the approach is that “no one is smart enough” to develop the correct abstractions for a particular class of application on paper alone. Initial designs may be acceptable for single applications but the ability to generalise for many applications can only be acquired by building applications and determining which abstractions are being reused across the applications. The more example applications that are considered during this process, the more general the resultant framework will be. However, designing and implementing applications is a non-trivial undertaking and therefore limiting the number of applications developed is necessary to arrive at a completed framework within a reasonable time frame.

The ‘Three Examples’ approach was adapted for the implementation of the application framework described in this thesis. The development of the first example application was substituted with an investigation into the design of four mobile, context-aware trails-based applications in order to learn about behaviour that different mobile, context-aware trails-based applications potentially have in common without going to the expense of full application development. Two example applications were developed subsequently -



informed by the design work already completed.

The remainder of this subsection discusses the initial design phase and the trail generation and reconfiguration point identification behaviour included in the two prototype applications developed as part of the application framework development process. The context acquisition and modelling behaviour available in the Hermes framework is also presented in this section to illustrate how trails applications can acquire and manage context.

### 3.1.1 Initial High-Level Framework Design

Prior to the implementation of the first example application, requirements for four mobile, context-aware trails-based applications were specified. UML use case, component, class and sequence diagrams were subsequently composed for the four applications. The goal of this process was to identify the high-level responsibilities of the components required to implement a mobile, context-aware trails-based application, hence defining the core research areas on the Hermes project. The four applications were:

1. A City Route Planner. This application supports city visitors or inhabitants generating routes from their current location to another location or collection of locations. The focus is more on context-aware route planning than multiple activity scheduling, where context such as live road traffic data, mode of transport and location of friends can dynamically affect a user's route. The key characteristic of this application from a research perspective is the consideration of context data in determining the most effective order in which to visit city locations, necessitating the design of behaviour to acquire context and generate and evaluate candidate trail solutions.
2. A Delivery Courier Support System. This application supports the operation of mobile delivery couriers by effectively scheduling delivery jobs assigned to them from a centralised head office. A courier's trail is composed of delivery jobs and is generated and reconfigured based on contexts such as job priority, package type, courier location, delivery location, delivery deadline, and courier shift start/end times. The key characteristics of this application, besides the requirement for trail generation, are the consideration of contexts not considered in the route planner

application, notably the restrictions on primary actor availability i.e., the courier's shift times, and the application of the concept of a trail in a business application.

3. A Treasure Hunt Game. This application generates trails for participants in a mobile, context-aware treasure hunt game. Trails are used to navigate between clue locations and are generated based on player preference for factors such as clue type and difficulty level. The key characteristic of this application is the potential necessity for dynamic trail reconfiguration that arises as a result of changes in the difficulty level of treasure hunt clues. The difficulty level of a clue increases when solved, making it more difficult for each subsequent player that views it. The application of the trails concept to mobile gaming is also notable as it further illustrates the applicability of the concept.
4. A Campus-based Student Support System. This application supports students in completing a set of compulsory and optional activities on their first day at college. The trails are affected by user location and activity opening hours, activity type, activity obligation (whether the activity is mandatory or not) and user-specified priority level. The key characteristic of this application is the introduction of the notion of activity obligation, facilitating the specification of activities that must be completed and are therefore treated preferentially in a situation where all activities cannot be completed.

During the design of these four applications, the following trails-related requirements were identified:

1. A trail generation component should generate effective activity schedules based on an extensible range of context sources and an extensible user preference model, without constraining the number of activities an application can consider. The component should support code reuse and extension to facilitate the development of a range of trails applications.
2. The trail generation component must be capable of scheduling a non-trivial number of activities.

3. The trail generation component should generate trails that are considered reasonable by humans.
4. Trails applications must contain behaviour to identify when an activity schedule needs to be reconfigured so that the schedule accurately reflects the state of the user's environment following context change. This behaviour should be reusable and extensible so as to facilitate the development of a range of trails applications.

Requirements for acquiring and modelling the context information required by the trail generation and reconfiguration point identification behaviour were also identified, and these requirements are being addressed in the Hermes framework as described in Section 3.1.2. The initial framework design phase was followed by the development of two prototype mobile, context-aware trails-based applications. A campus-guide application for Trinity College is described in Section 3.1.3 and RiddleHunt, a trails-based riddle solving game, is described in Section 3.1.4.

### **3.1.2 Context Acquisition and Modelling in Hermes**

The Hermes framework contains reusable context acquisition and modelling behaviour, in addition to the trails-based behaviour proposed in this thesis, that supports applications in acquiring and modelling context from a range of sources e.g., remote mobile devices and sensors. Although not a contribution of this thesis, the author worked on a team that implemented the context acquisition and modelling behaviour in Hermes that is described in this section to illustrate how context can be acquired by trails-based applications in practice. Both the trail generation and reconfiguration point identification behaviour in the application framework assume the availability of context data as input in order to produce results i.e., to generate a trail that best serves the user given their current situation as described by the available context data, and to identify, based on context change, when the trail should be reconfigured. Certain contexts, such as user location and current time, can be acquired via local sensor access e.g., a directly connected GPS device for location and the system clock for current time. However, remote context sources such as the location of other players (a context used in RiddleHunt) must be acquired through collaboration with the producers of the context e.g., the other game players in the case of

remote player location context. Hermes provides components to facilitate the acquisition of both local and remote context. Peer-to-peer ad hoc service discovery is used to discover and communicate with remote devices that provide a context service. Hermes provides an object-oriented context model with XML mapping so that context information can be queried by application logic and shared between devices.

It is important to note that trails applications built using the application framework do not have to use the Hermes context acquisition and modelling behaviour in order obtain context. Developers can implement the classes responsible for context generation by using the Hermes framework or they can implement their own custom context management behaviour.

### **3.1.2.1 Communication and Service Discovery**

The Communication component in the Hermes framework illustrated in Figure 1.1 on page 5 is responsible for peer-to-peer ad hoc device discovery. It facilitates the discovery of remote devices and the transfer of context e.g., player game state and location in RiddleHunt, between application users. Remote devices within proximity are discovered via an ad hoc service discovery protocol and devices then connect directly to share context. Every device broadcasts and listens for remote devices on a well known port. The broadcast device periodically sends the IP address and port that remote devices can use to establish a connection. When a remote device receives this broadcast, it first verifies that the broadcasting device is not already connected. The remote device then connects and sends its service discovery information to the broadcasting device. On receiving the remote device's service discovery information, the broadcasting device sends its own service discovery information. The two devices are now connected and can exchange context data.

In order to eliminate reliance on a fixed network or third party, it must be possible to locally determine when a connection to a device is no longer valid. Inactive connections must be removed in order to reclaim resources, such as ports or memory, allocated for service discovery information. A connection remains valid until no messages are exchanged between devices for a specified interval of time. This implies the devices are no longer in communication range or the communication component has been disabled, as

the broadcast messages should be received within the time interval.

The Service Discovery component allows a device to advertise, discover and invoke services on remote devices. Context-aware service discovery is important for applications executing on mobile devices, as not all devices encountered will offer the same services. Additionally, the invocation methods may be different for functionally equivalent services. Hermes supports the use of service discovery to discover and retrieve remote context in a generic manner. To explore what context is available, an application first decides which context types it is interested in receiving. To be useful to a specific application, this should be a subset of the types that the application can handle. The context acquisition process is illustrated in Figure 3.1. When a remote device advertises a context type an application is interested in, the Service Discovery component of the local device sends a context type request through the Communication component. The remote device then handles the request and responds with the desired context value.



**Figure 3.1:** Ad hoc communication and service discovery in Hermes

Service Discovery also handles advertisement of context services offered by the local

device. An application specifies the context types it is interested in sharing, taking the privacy preferences of the user into account. A service description message describing each type of sharable context on the device is then created. For example, the service description message illustrated in the lower part of Figure 3.1 is advertising the `LatLonLocation` and the `Game` types in the RiddleHunt application. This message is sent to remote devices every time one of the relevant context types is updated locally e.g., every time the user solves a riddle or changes location. The time each context type was last modified is also included in the description. This prevents remote devices from requesting context values they already have.

### 3.1.2.2 Message Types and Message Processing

The context acquisition and modelling behaviour in Hermes supports the following types of incoming and outgoing messages:

- Context data from various sources:
  - Application derived contexts e.g., context input from users.
  - Device contexts e.g., battery life.
  - Context obtained from devices connected directly to the device e.g., information from a GPS device or motion sensor.
  - Contexts supplied by third parties e.g., wireless sensors or other user devices.
- Context requests to remote devices.
- Service description messages to remote devices.
- Application messages e.g., start and join game messages for RiddleHunt.
- Broadcast messages advertising the port and IP address used to communicate with the user's device.

XML is used as the message format in Hermes, meaning that applications can use the portions of the message they understand, while ignoring parts that they do not. XML also

facilitates standardisation of communication between different platforms and programming languages. Leveraging the extensibility and standardisation of XML provides the opportunity for different versions of an application, or completely different applications on diverse device types, to communicate.

Message processing components on mobile devices should be able to handle multiple messages at once. At the same time, they should not allow the number of messages processed to hinder application responsiveness. Hermes accomplishes this by employing a message-handling thread pool with a limited number of threads. When a message is received, regardless of the source, it is placed in the queue. When all preceding messages have been removed from the queue, the XML message description is converted to a message object and a thread from the pool is assigned to that message. This thread owns the message and carries it through the appropriate components until it comes to its final destination. For example, if the message is a piece of context that is deemed consistent with the context already stored by the application, it is marshalled through the modelling component and into the context model. Similarly, if it is an application message, it is delivered directly to the application

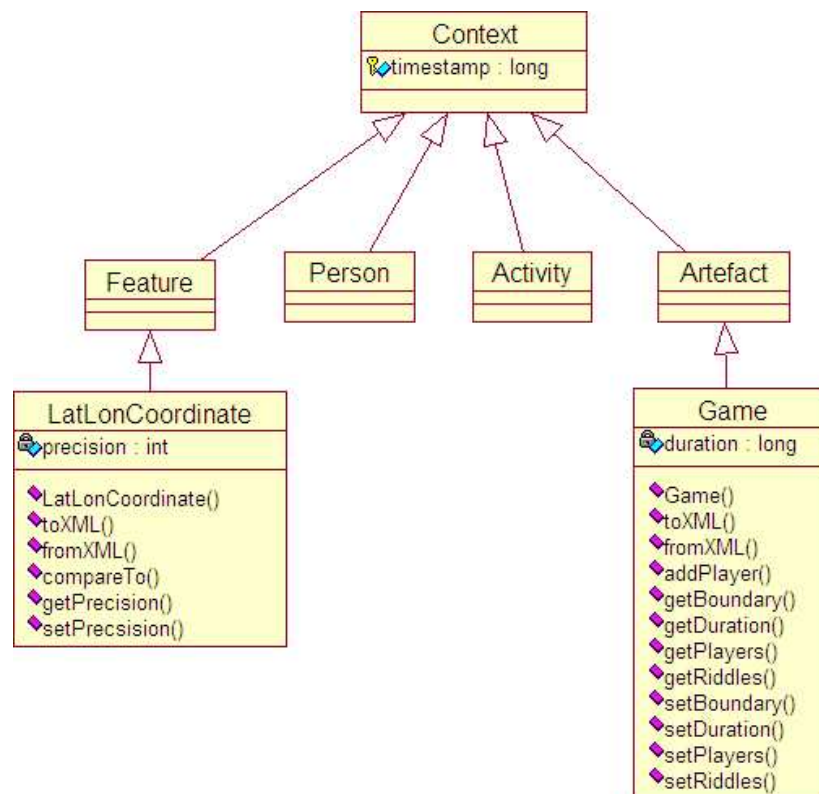
### **3.1.2.3 Context Modelling**

The way in which context is modelled determines not only how context will be stored but also the formats in which it can be exchanged and the possible reasoning over it. Hermes uses a hierarchical, object-oriented context model. This model lends itself to modelling real-world objects and their various relationships. Object orientation provides for a world model that is extensible, and enables the use of XML for context storage and exchange via an object XML mapping.

The structure of the context model impacts on services such as the context query processor and the context fusion service, as it determines the complexity and responsiveness of operations performed by these components. The context model's structure also facilitates extension through the addition of new types of context while minimising the impact of such changes on clients. The Hermes context model places the following four types of context at the root of the hierarchy:

- Activities - real world tasks that can form part of a trail.
- People - information about people.
- Artefact - virtual objects or services as well as mobile objects.
- Features - information about the structure of the environment.

These context types, along with example subclasses, are illustrated in Figure 3.2. An example XML context description is illustrated in Figure 3.1. The two context types, `LatLonCoordinate` and `Game`, are subtypes of `Feature` and `Artefact` respectively. The context model makes the context data acquired through Hermes's context acquisition behaviour accessible to applications in a manner that facilitates ease of use in trails and other context-aware applications.



**Figure 3.2:** Hermes context model top level hierarchy and examples

### 3.1.2.4 Summary

In addition to the trails-related behaviour described in this thesis, Hermes facilitates peer-to-peer service discovery and object-oriented context modelling with XML mapping.



The components encapsulating this behaviour support the acquisition and modelling of context information from both local and remote sources that is required as input to the trail generation and reconfiguration point identification behaviour described in this thesis. The RiddleHunt application (discussed in Section 3.1.4) uses Hermes’s context support to access to information regarding the location and game state of other players that is used during trail generation and reconfiguration point identification.

### 3.1.3 Application 1: Oisín Goes to Trinity

The ‘Oisín goes to Trinity’ application (abbreviated to Oisín) provides campus-wide trails for the Trinity College Dublin campus. As Oisín was the first prototype application developed, the primary research goal was the validation of the trails concept. This was achieved through the development of a mobile, context-aware trails-based application (Oisín) and its subsequent evaluation via user trial. In terms of trail generation and reconfiguration point identification, the focus was on assessing the practicality of brute force trail generation and minimising the amount of time that the user’s trail is out of sync with their physical environment.

Oisín was implemented using the Java 2 Micro Edition (Personal Basis Profile) [84] that supports the implementation of Java applications with graphical user interfaces on resource-constrained mobile devices. The application was deployed on a Sharp Zaurus SL-5600<sup>1</sup> PDA and used an external serial Magellan SporTrakPRO<sup>2</sup> GPS device to determine user location.

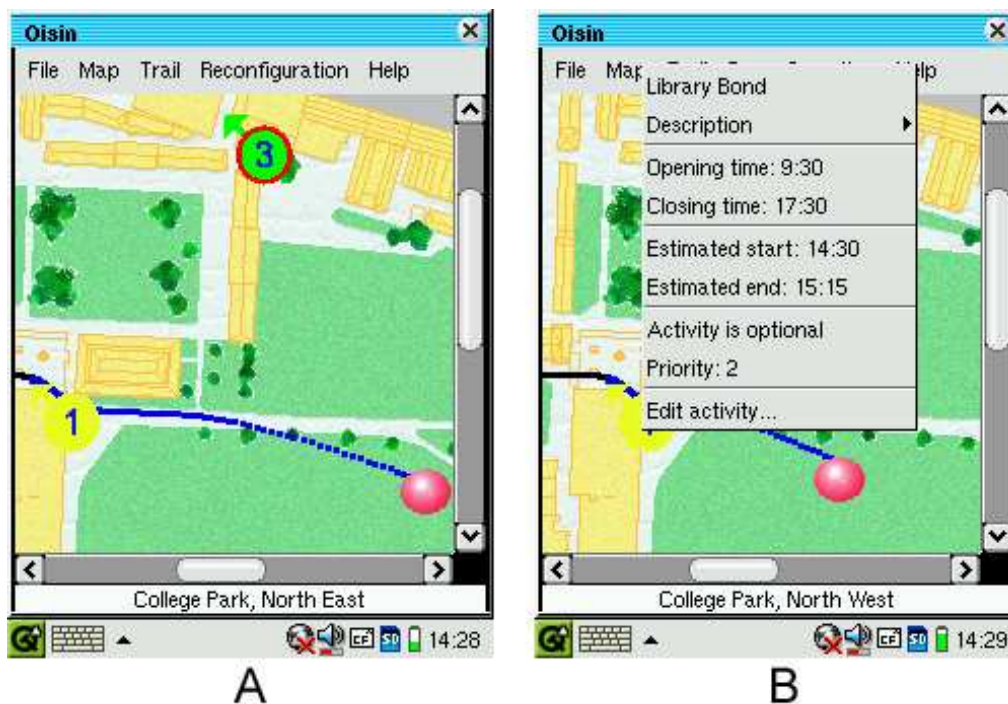
Oisín can be used to provide a range of campus-based mobile, context-aware trails applications including a tourist guide application for visitors to the many attractions on the 400-year-old campus and the student support application described in Section 3.1.1. Figure 3.3 contains two screen shots of Oisín running the student support system on the Zaurus. Screen shot A shows the user’s location (indicated by a red dot icon as well as the textual description ‘College Park, North East’ in the status bar). A path from the user’s location to the location of the first activity and the location of the third activity are also shown. The third activity is currently deemed to be impossible, indicated by a

---

<sup>1</sup><http://www.zaurus.com>

<sup>2</sup><http://www.magellangps.com>

red outline on the activity icon. Screen shot B illustrates the context-sensitive activity menu that is accessed by clicking (screen tapping) on any activity icon. The menu shows information specific to the activity selected (in this case Activity #1 - submission of a library bond), including the estimated activity start and end times based on its position in the trail.



**Figure 3.3:** Screen shots of the Oisín graphical user interface on the Zaurus

A user study involving 21 subjects was conducted to assess various aspects of Oisín, including the user interface, the hardware form factor, the trail generation and reconfiguration behaviour and user acceptance of mobile, context-aware trails-based applications. Subjects were familiarised with the concept of mobile, context-aware activity scheduling and then used a campus activity guide application to complete a number of activities on the Trinity College campus such as visiting an art gallery, a natural history museum and the Book of Kells. The unexpected closure of a high priority activity was simulated to force a significant trail reconfiguration while the user was en route to the activity in question. Trail reconfiguration also occurred based on changes in user location. Subjects completed a questionnaire and informal interview following the application trial. The results of the trails-related user study questions are included in Appendix A.1. These

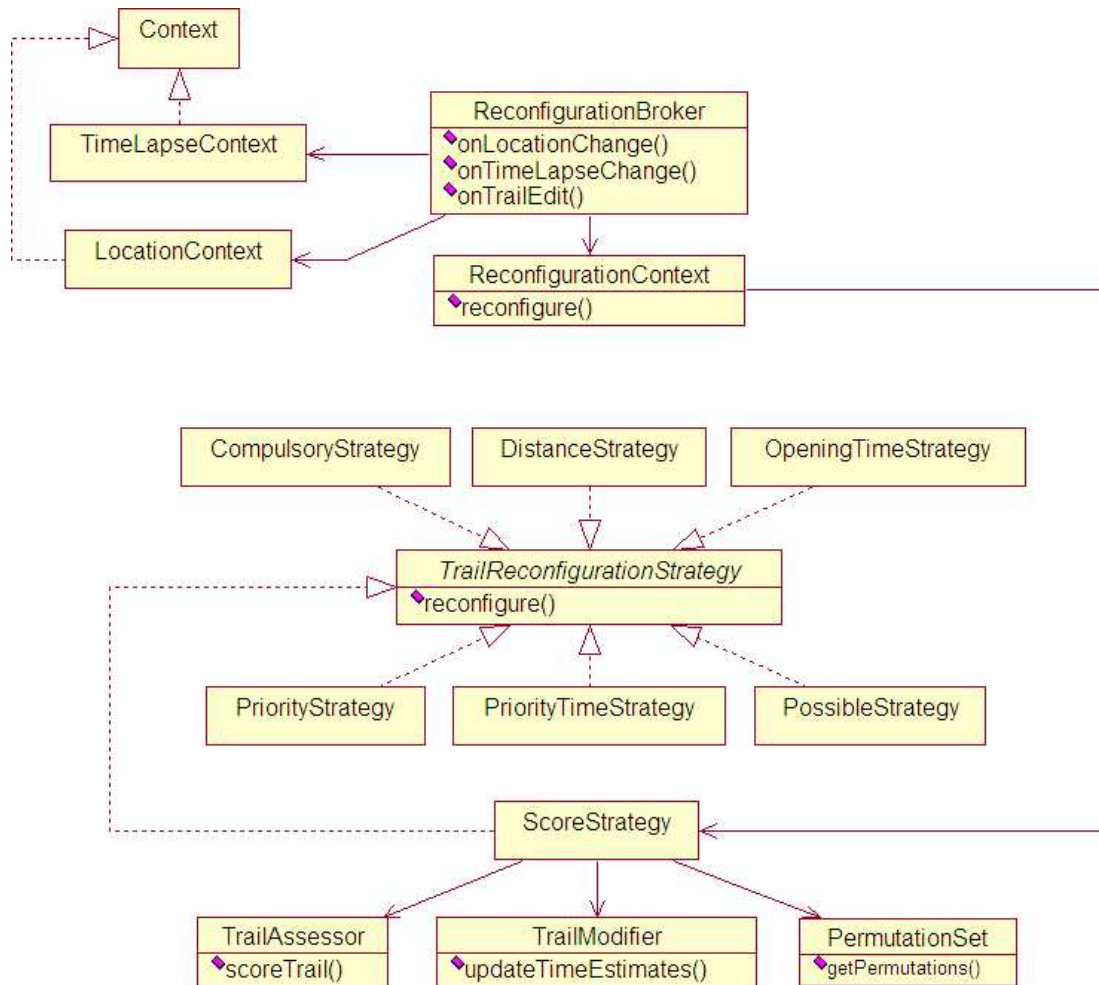
results indicate a general acceptance of the trails concept and an appreciation of its utility. The majority of the subjects noticed that the order of the activities on their trail changed when the trail was reconfigured. These changes were anticipated and deemed positive by the majority of subjects. As regards the negative aspects of the application, the hardware form factor, which required subjects to hold two separate devices connected via cable, was deemed to be awkward to use by the majority of subjects. Additionally, subjects commented in post-trial interviews that the application was “slow” and subject to pauses during execution. This was the result of a high degree of unnecessary trail reconfiguration (discussed in Section 3.2.4).

The following subsections discuss the design of the trail generation and reconfiguration point identification techniques used in Oisín.

### 3.1.3.1 Trail Generation in Oisín

Trails are generated in Oisín using a brute force algorithm, the execution of which is triggered by the receipt of location and time lapse contexts. The brute force algorithm exhaustively generates and evaluates all permutations of the user’s current trail and returns the trail deemed optimal by the evaluation function. Oisín’s high-level software design, containing only key classes and methods, is illustrated in Figure 3.4. The `ReconfigurationBroker` class is responsible for coordinating trail generation and receives location and time context from the `LocationContext` and `TimeLapseContext` classes. The receipt of this context triggers the execution of the trail generation mechanism.

The brute force trail generation mechanism uses the `PermutationSet` class to generate all permutations of the trail. These permutations are assessed by an evaluation function that assigns a value to candidate trails based on specific trail properties or combinations of properties. Numerous trail evaluation strategies, subclasses of `TrailReconfigurationStrategy`, are included in Oisín. `ScoreStrategy` is the evaluation function used in Oisín because it incorporates all the trail properties considered by the other strategy classes. `ScoreStrategy` assigns relative importance weights to the trail properties it considers so that, for example, a candidate trail containing a greater number of compulsory activities is assigned a higher score than a trail with activities that are not compulsory but are nearer to the user. In Oisín, the user’s obligation to



**Figure 3.4:** High-level design of trails behaviour in Oisín

undertake an activity (whether an activity is compulsory or not) is more important than the proximity of an activity to the user.

Once all permutations of the user’s trail have been generated, `ScoreStrategy` calculates the estimated start time and end time for each activity using the `TrailModifier` class. These values are calculated based on both the position of the activity in the candidate trail and the estimated activity duration. The estimated start and end times for each activity are used to determine whether it will be possible for the user to undertake all activities if following the candidate trail in question. Activities are marked as possible or impossible based on their estimated start and end times.

The `reconfigure()` method in `ScoreStrategy` assigns a single value numerical score to each permutation of activities. The score value for each candidate solution is generated by the `scoreTrail()` method in `TrailAssessor`. This method scores each trail permu-

tation based on six trail properties. The trail properties considered in the evaluation function are as follows:

- Trail length (in metres).
- The number of activities possible.
- The efficiency of the time usage.
- The user priorities satisfied.
- The number of compulsory activities possible.
- Whether the user's chosen first activity is scheduled in the first position on the trail.

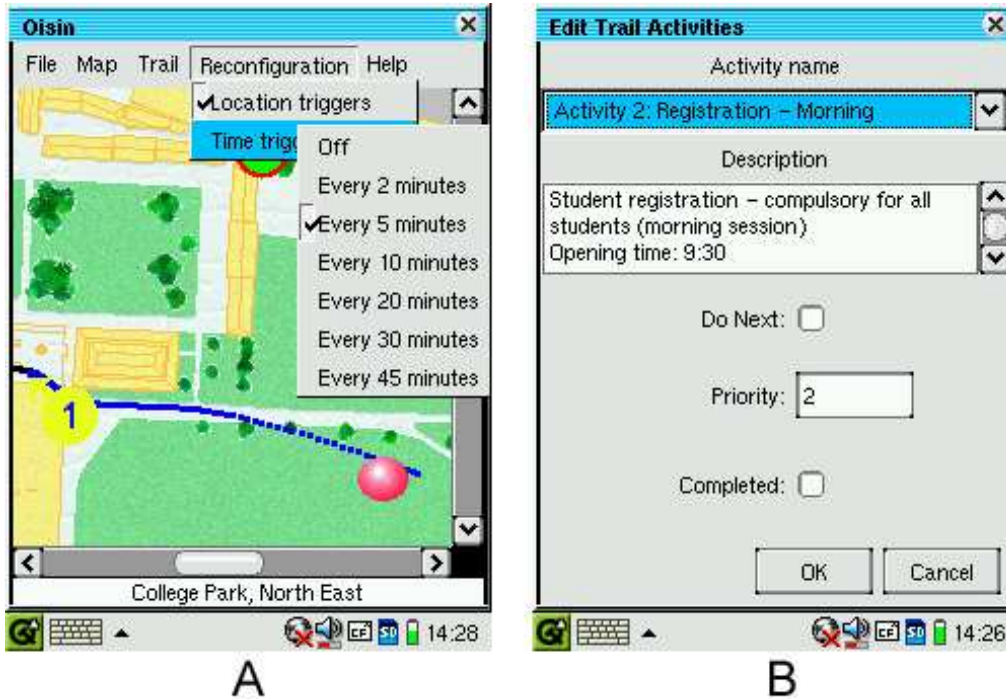
The user study questions presented in Appendix A.1 indicate that this technique is capable of generating trails that users agree with. However, similar to the brute force trail generation technique implemented in the GUIDE system, the responsiveness of trail generation mechanism in Oisín deteriorated significantly as the number of activities increased. In post-trial interviews, user study subjects expressed frustration with the trail reconfiguration response time of 30 seconds, during which time the application was 'busy', denying service to users. This frustration was exacerbated by the fact that trail was reconfigured far more frequently than was actually necessary in an effort to ensure that the user's trail consistently reflected their physical environment. The trail reconfiguration point identification mechanism is explained in the following subsection.

### **3.1.3.2 Reconfiguration Point Identification in Oisín**

There are five events that trigger trail reconfiguration in Oisín.

1. Location change - when the user changes symbolic location e.g., from Front Square west to Front Square east.
2. Time lapse - every time a specified period of time elapses.
3. Selection of the 'Do Next' option - when the user specifies that a certain activity should be done next, overriding trail order.

4. Activity priority change - when the user changes the priority of an activity via the user interface.
5. Activity completion - when the user specifies via the user interface that an activity has been completed.



**Figure 3.5:** Screen shots of the reconfiguration menu and edit screen in Oisín

Figure 3.5 contains screen shots of Oisín’s reconfiguration menu (screen shot A) and activity edit screen (screen shot B). The reconfiguration menu allows users to control events 1 and 2. Location triggers can either be on or off. Time lapse triggers can be turned off or set to one of a range of time lapse intervals. Location and time triggers can run simultaneously. The `ReconfigurationBroker` class illustrated in Figure 3.4 contains methods to invoke its `reconfigure()` method when it receives location change and time lapse context events. The activity edit screen allows users to control events 3-5. The selection of any of these options invokes the `onTrailEdit()` method in the `ReconfigurationBroker`, resulting in trail reconfiguration.

In the case of events 3 and 5, reconfiguration is always done only when necessary i.e., activity completion and ‘Do Next’ always cause the trail ordering to change. However,

while events 1, 2 and 4 cause reconfiguration to occur, the trail ordering does not always change following reconfiguration triggered by these events. Reconfiguration that does not cause trail order to change results in the limited resources of the Zaurus being needlessly consumed. Unnecessary reconfiguration occurred most notably in relation to location change events. These events occurred most frequently but rarely changed the ordering of the trail due to the relatively low importance of activity proximity in the candidate trail evaluation function.

### **3.1.3.3 Summary**

The Oisín goes to Trinity application provides developers with the ability to deploy campus-based trails applications. However, it is limited by the approaches to trail generation and reconfiguration point identification used. The basic brute force approach limits the number of activities that can be considered by the application and the hard-coded weights in the candidate trail evaluation function make the customisation of application behaviour difficult. The reconfiguration point identification technique used has the potential to reconfigure the trail unnecessarily and frequently did so in practice.

The next application implemented was RiddleHunt, a mobile, city-based riddle solving game. The development of RiddleHunt, in terms of trail generation and reconfiguration focused on finding solutions to the problems encountered in the development of Oisín goes to Trinity.

### **3.1.4 Application 2: RiddleHunt**

After the implementation and evaluation of Oisín, the first version of the application framework existed. The goal of the trails-related work in the second application was to address the problems encountered in relation to trail generation and reconfiguration point identification in Oisín. In terms of trail generation, the focus was on extending the number of activities an application can consider, thereby increasing the range of applications that can be built using the application framework. This is achieved by scheduling a subset of the total number of activities each time the trail is reconfigured. In order to select the activities to be included in the trail each time it is reconfigured, a measure of activity

‘relevance’ is introduced. The relevance value of an activity represents how deserving the activity is of the user’s attention relative to the other activities i.e., how relevant the activity is to the user. Activity relevance is calculated based on the user’s preferences and the current state of the context being considered by the application. In terms of reconfiguration point identification, the goal was to reduce the number of unnecessary trail reconfigurations by monitoring the subset of activities considered relevant. Trail reconfiguration is identified as necessary when a change in relevant set membership occurs. Improving the trail generation and reconfiguration point identification mechanisms in the manner proposed allows for more realistic user schedules to be modelled, and reduces unnecessary resource usage on mobile devices.

RiddleHunt is a mobile, context-aware riddle solving game designed to be played in Dublin City. The game is played by multiple players who have to solve riddles at various locations around the city. The application was implemented using the Java 2 Micro Edition (Personal Basis Profile) for deployment on a Hewlett-Packard iPAQ Pocket PC (h6300 series) and uses an external Bluetooth Tom Tom Navigator<sup>3</sup> GPS device to determine user location.

RiddleHunt allows application designers to distribute virtual riddles throughout the city, and players are required to answer as many riddles as possible during the allotted game time. Trails are generated to aid users in getting to as many suitable riddle locations as possible. Riddle suitability depends on user preferences for trail generation e.g., preference for riddle type and riddle value. Riddles have a type e.g., ‘Maths’, ‘Trivia’ and ‘Word’, and a value that is based on the difficulty level of the riddle e.g., novice, intermediate or expert. RiddleHunt uses the context acquisition and modelling behaviour in the Hermes framework to share game state context between mobile players. When players are within proximity of each other they can connect directly to share context. The ability to share game state information allows trails to be generated based on up to date information about riddles i.e., has another player already solved a particular riddle (in this case the riddle is not worth as much to other players). Bonus points are added to a player’s score if they solve a certain riddle before another player.

In terms of game design rationale, the decision to reduce the value of a riddle each

---

<sup>3</sup><http://www.tomtom.com>



time it is solved was taken for two reasons. First, it adds a new source of context that was not considered in Oisín i.e., activity property change. By making a greater number of context sources available to the trail generation algorithm, it can generate trails that more accurately reflect the user’s environment. Second, it generates context information that players cannot easily obtain themselves, illustrating the benefit of following a context-aware trail when playing a mobile game.

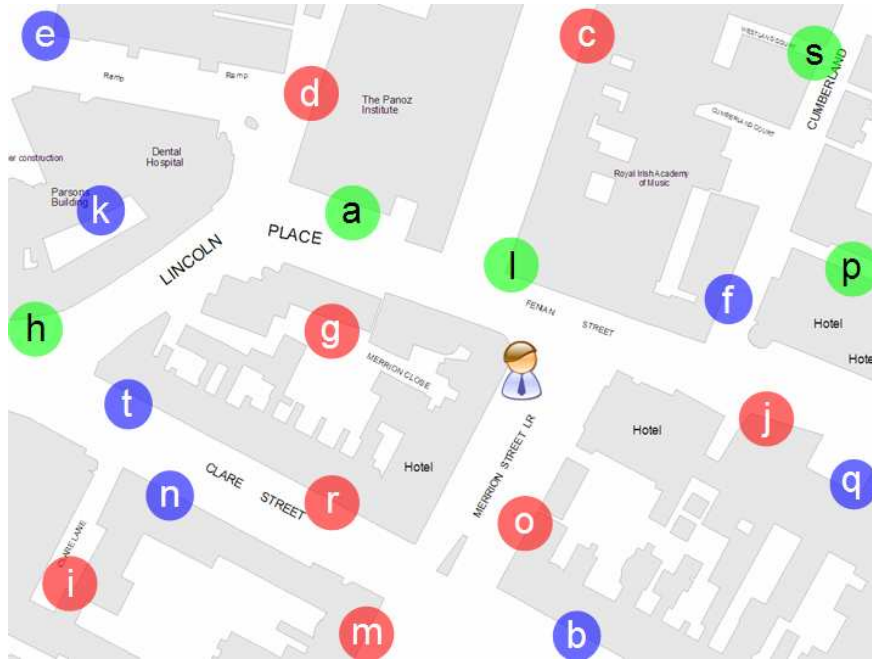
The trail generation and reconfiguration point identification mechanisms used in RiddleHunt are discussed below.

### 3.1.4.1 Trail Generation in RiddleHunt

Trail generation in RiddleHunt extends the basic approach used in Oisín by increasing the number of activities that the application can consider. However, because brute force trail generation has time complexity  $O(n!)$ , it is not possible from an application responsiveness perspective to simply reuse the brute force algorithm from Oisín with a bigger  $n$  value. Therefore, RiddleHunt takes advantage of the context information available to reduce the number of activities (riddles) considered during each execution of the trail generation algorithm. For example, a RiddleHunt application contains a predefined set of 20 riddles,  $X$ , where  $X = \{a, b, c, d, \dots, t\}$  and the riddles are geographically dispersed around Dublin City. This situation is illustrated in Figure 3.6, which shows player and riddle locations. Red riddles i.e.,  $\{c, d, g, i, j, m, o, r\}$ , are the most valuable while green riddles i.e.,  $\{a, h, l, p, s\}$ , are the least valuable. In order to generate trails for RiddleHunt players within a reasonable response time<sup>4</sup> the application considers only a subset of the riddles in  $X$ . Based on the current context (player location, player preferences and riddle properties), the trail generation behaviour in RiddleHunt composes a set of activities,  $Y$ , where  $Y \subset X$  and the cardinality of  $Y$  is the maximum number of riddles that can be reasoned about using a brute force approach in a reasonable response time. Using this approach, RiddleHunt can provide trails to users that consist of the activities that are most relevant to the user at the time of trail generation. The riddles identified as relevant to the player in the example situation are illustrated in Figure 3.7, with the irrelevant

---

<sup>4</sup>The meaning of the word ‘reasonable’ in relation to trail generation response time is discussed in Section 3.2.1.

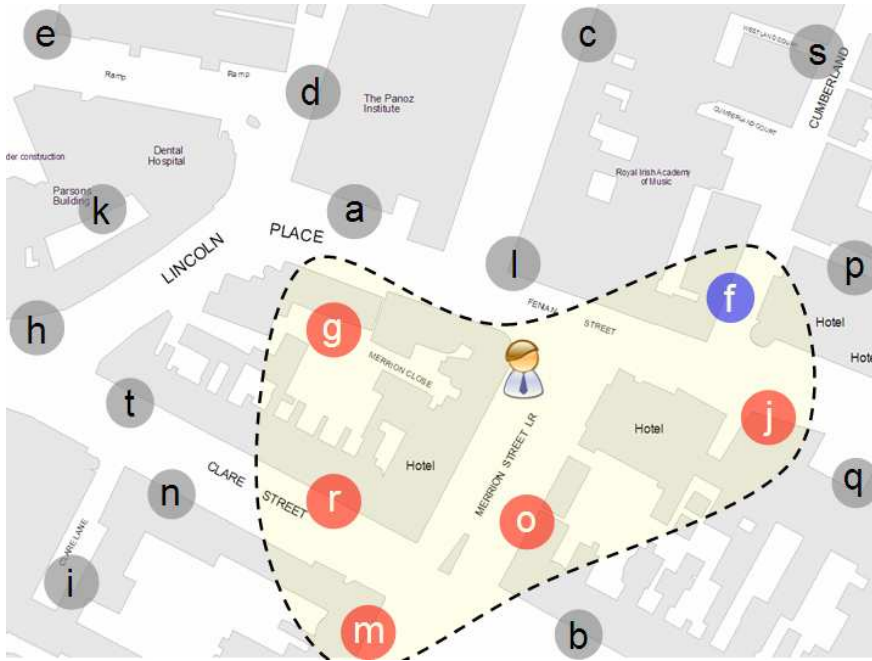


**Figure 3.6:** The activities in the set  $X$  and the player location

activities greyed out. As riddles are completed or context changes, membership of the set of relevant activities ( $Y$ ) is updated and trail reconfiguration occurs.

Figure 3.8 illustrates the high-level class design of the trails behaviour in RiddleHunt. Only key classes and methods are shown in the diagram. The `TrailsLogic` class, responsible for coordinating trail generation, receives `Player` and `Game` context updates that provide player location and riddle state context respectively. When these context events are received, the collection of activities considered by the application is first pruned and then sorted by relevance.

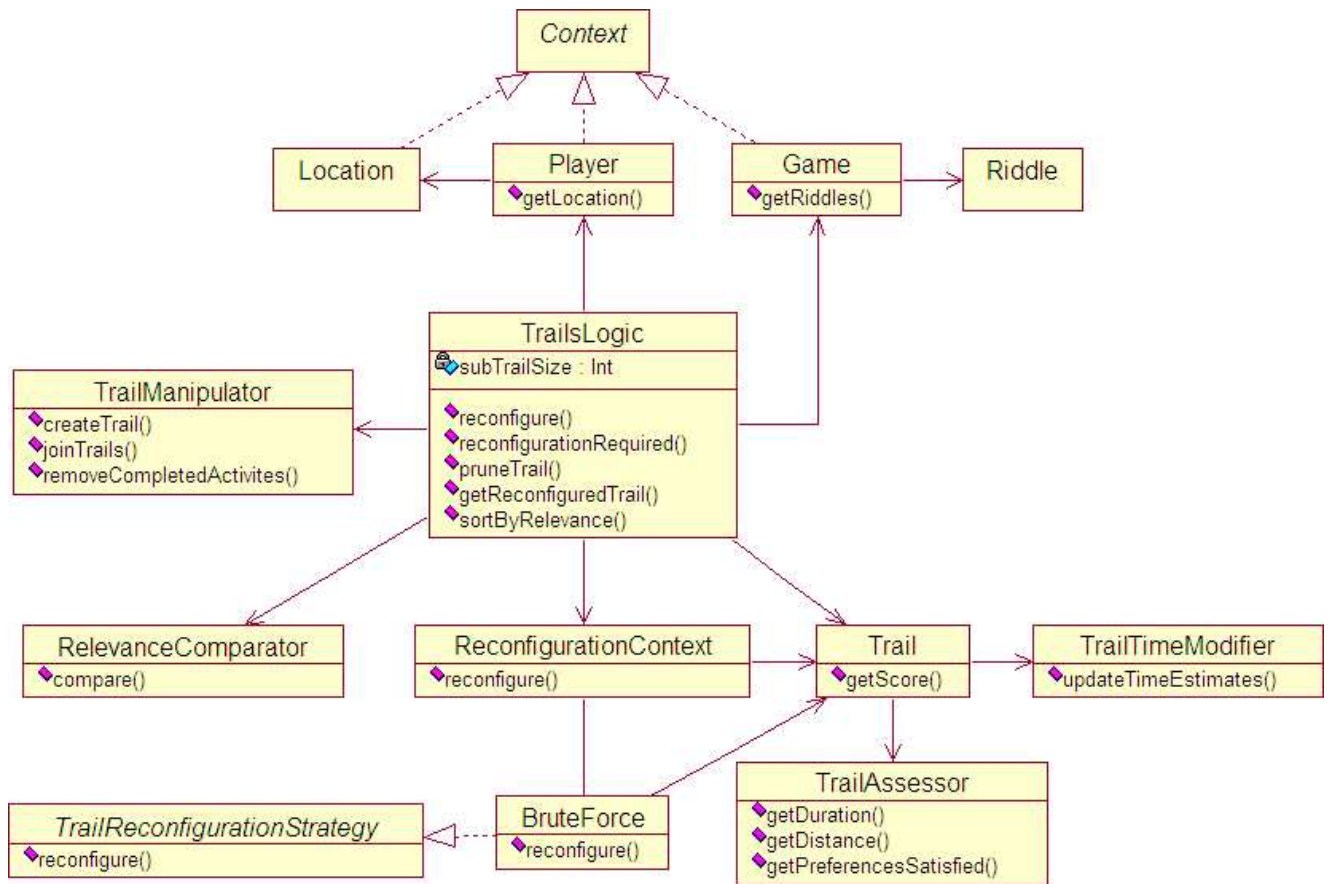
`TrailsLogic` contains methods to prune completed activities/riddles from the trail, reducing the number of trail permutations and consequently making trail generation more efficient. `TrailsLogic` also contains behaviour to select the most relevant activities for consideration during trail generation if necessary i.e., when the number of activities is greater than that which can be considered in a reasonable response time. The `sortByRelevance()` method sorts the collection of activities by relevance. The relevance of an activity in RiddleHunt is defined by the proximity of the activity to the user, the value of the riddle and the relationship between the activity type and the user's preference for activity type (riddle category).



**Figure 3.7:** Identification of the members of set  $Y$

When the collection of activities has been pruned of any completed activities and sorted by relevance, it can be assessed to see whether or not it should be reconfigured. This behaviour is encapsulated in the `reconfigurationRequired()` method (discussed in Section 3.1.4.2). If reconfiguration is necessary, the `getReconfiguredTrail()` method is invoked. This method assesses the number of activities in the trail. If the number of activities in the trail is greater than the number of activities that can be reasoned about in a reasonable response time using the brute force method, the `createTrail()` method in `TrailManipulator` is used to create a subtrail of the most relevant activities. The `subTrailSize` attribute stores the size of the subtrail to create. The `createTrail()` method represents the implementation of the identification of the relevant set  $Y$  as illustrated in Figure 3.7.

When a subtrail has been created, the optimal ordering of the activities in the subtrail is calculated. A significant difference from the Oisín implementation is that the trail generation strategy classes no longer provide different methods of scoring candidate solutions. RiddleHunt considers trail generation strategies to be different techniques for generating candidate solutions to be evaluated by a single evaluation function stored in the `Trail` class. This decision was made because all trail properties should be consid-



**Figure 3.8:** High-level design of trails behaviour in RiddleHunt

ered during candidate solution evaluation, while different candidate solution generation techniques e.g., brute force and genetic algorithm, may be appropriate depending on application requirements and device capabilities. Brute force is the trail generation strategy used in RiddleHunt. `BruteForce` generates candidate trails to represent all permutations of the subtrail and uses the `getScore()` method in each candidate solution to return a numerical value.

The `getScore()` method in `Trail`, using `TrailTimeModifier` and `TrailAssessor` to perform the same function as they did in Oisín (Figure 3.4), is the evaluation function used in RiddleHunt. The following factors are considered when calculating the score of a trail:

- Trail length (in metres).
- Trail duration (in minutes).

- Riddle values.
- The extent to which user preferences for riddle type are satisfied.

The trail with the highest score is returned to `TrailsLogic`. The activities not considered during trail generation are appended to the subtrail using the `joinTrails()` method in `TrailManipulator`, and marked as unscheduled. The trail, containing both scheduled and unscheduled activities, is returned to the client. Annotating activities with information regarding whether or not they are currently scheduled allows the user interface logic to represent currently scheduled and unscheduled activities differently. For example, scheduled activities can be represented by coloured icons and overlaid with sequence numbers representing their position in the trail, whereas unscheduled activities can be represented by greyed out icons with no sequencing information.

#### 3.1.4.2 Reconfiguration Point Identification in RiddleHunt

Reconfiguration point identification in RiddleHunt is based on changes in membership of the set of activities that constitute the subtrail (set  $Y$ ). Therefore, only context events that cause a significant difference in the relevance values of the activities will cause trail reconfiguration to occur. The decision to move away from the reconfiguration point identification mechanism used in Oisín, where all context events trigger reconfiguration, was taken in a bid to minimise the number of unnecessary trail reconfigurations. The reconfiguration point identification mechanism used in RiddleHunt triggers reconfiguration only when it will result in the generation of a trail different to the one currently being followed by the user.

The trail activities (set  $X$ ) are sorted by relevance each time a context event occurs and the top  $z$  activities, where  $z = \text{subTrailSize}$ , are isolated, producing set  $X_o$ . If the activities in  $X_o$  are equal to those in the existing subtrail  $Y$  ( $X_o = Y$ ) then reconfiguration does not occur. There are three context events that cause membership differences between  $X_o$  and  $Y$ :

1. Riddle completion. When a riddle is completed it is pruned from the trail and no longer considered during reconfiguration. The completed riddle,  $r$ , cannot be

a member of  $X$  ( $r \notin X$ ) and therefore cannot be a member of  $X_o$  ( $r \notin X_o$ ). Therefore it is no longer possible for  $X_o$  to be equal to  $Y$  ( $X_o \neq Y$ ), necessitating, and consequently triggering, trail reconfiguration.

2. Player location change. When players move around the game space their proximity to riddles changes, causing relevant activities to become irrelevant and previously irrelevant activities to become relevant, affecting the membership of  $X_o$ . In RiddleHunt, a location change occurs when the user's GPS position causes a change in the user's X, Y position on the user interface. Any effect to the membership of  $X_o$  results in  $X_o \neq Y$ , triggering reconfiguration.
3. Riddle value change. When a riddle is solved by a player its value to other players who have not yet solved it is reduced. A reduction in riddle value makes a riddle less relevant to a player looking for riddles to solve. The reduction of a riddle's value may be great enough to cause it to be removed it from  $X_o$ , triggering reconfiguration.

By monitoring changes in the set  $X_o$  from which the subtrail is composed it is possible to identify when trail reconfiguration should occur. Context events that do not result in  $X_o \neq Y$  are ignored, meaning that no unnecessary reconfiguration occurs. However, this approach does not recognise the need for reconfiguration caused by fluctuations in the relevance values of the activities deemed relevant (those currently in the subtrail). These fluctuations may not be significant enough to cause a change in set membership, but they can be significant enough to cause the trail being followed to be suboptimal. For example, if the user moves towards a riddle,  $f$ , away from another activity,  $j$ , then, all things being equal,  $f$  will become more relevant than  $j$  based on proximity. This change in the relevance values of the two activities may not be great enough to cause  $j$  to be removed from  $X_o$  but it should cause the trail to be reconfigured so that the user is instructed to focus his attentions on  $f$ . This issue is addressed in the application framework.

### 3.1.4.3 Summary

RiddleHunt introduces a trail generation approach that allows an application to include a large number of activities by using context to reduce the number of activities considered

during each trail reconfiguration. The set of activities that are considered are those deemed most relevant at the time of reconfiguration.

The reconfiguration point identification mechanism is based on changes to the set of activities with the highest relevance values. This method eradicates unnecessary reconfiguration. However, due to an inability to reason about the internal structure of the set of relevant activities, the trail that the user is following can become suboptimal without reconfiguration being triggered. This issue is addressed in the application framework, which is described in the next section.

## 3.2 Application Framework

RiddleHunt introduced the notion of removing completed activities and calculating a relevance value for each activity to improve on the limited trail generation and reconfiguration point identification behaviour in Oisín. While these advances allowed applications to include more activities by considering only the most relevant during trail generation, areas for improvement remained. First, in terms of removing irrelevant activities from the activity set before the trail generation process begins, activities that are not possible due to the current context and activities involved in clashes can also be removed. Second, in terms of trail reconfiguration, significant differences between the relevance values of activities should be acted on, even if they do not cause a membership change in the set of relevant activities used to generate a trail. Third, both Oisín and RiddleHunt assume a fixed set of application-specific user preferences, precluding the generation of trails based on the preferences of individual users. Finally, although RiddleHunt introduced the ability to utilise many candidate trail generation techniques, both applications were restricted to using brute force. These issues are addressed in the application framework.

The remainder of this section is as follows. Section 3.2.1 discusses application response time as it relates to trail generation and proposes a reasonable response time for the trail generation algorithm in the application framework. Section 3.2.2 introduces Multi-Attribute Utility Theory - a preference-based object evaluation technique used to facilitate the consideration of non-static user preferences in the trail generation algorithm without requiring source code modification. This background information is followed by

the design of the trail generation and reconfiguration point identification techniques in Section 3.2.3 and Section 3.2.4 respectively.

### 3.2.1 Response Time

The “Holy Grail” of system response time engineering was defined by Miller in a theoretical paper that proposes a set of guidelines for application developers [85]. Miller suggests a maximum delay of 2 seconds following a request, with an optimum response time of 0.5 seconds so as to maintain the conversational nature of the interaction between humans and machines. According to Galletta et al., this theory was upheld as a “gold standard” in web-design well into the 90s [44]. However, the goal has not been widely achieved.

Application response time literature suggests that there is a timescale within which it is optimal to deliver a result to the user. Returning an answer past the wrong end of this timescale has the potential to frustrate the user and can discourage them from using the application in the future [88]. In the most extreme case, the result will not be returned at all due to the user terminating the operation after an amount of time spent waiting. This second phenomenon is described in a discussion of website response time where it is advised that applications should not take too long to return a result due to the tendency of users to abandon tasks that take too long [44]. Galletta et al. discuss a generally accepted maximum response time of between 8-12 seconds and states that anywhere between 0-9 seconds is acceptable for websites. Hoxmeier and DiCesare have shown that satisfaction with website response time is constant from 0-9 seconds and begins to diminish from 12 seconds onwards [53].

Myers proposes that the range of an acceptable response time, 0-9 seconds, can be extended by presenting feedback to the user during the waiting period [87]. Nah and Kim have demonstrated this effect, showing that subjects given a progress indicator e.g., a progress bar, will wait on average 38 seconds for a hyperlink that does not return a result [88]. Subjects given the same link without any progress indication terminated the action after 13 seconds. However, in subsequent trials, the group without progress indication only waited for 3 seconds and the wait time for those with progress indication dropped to 7 seconds. The conclusion is that patience does not last forever and appropriate wait



times are determined dynamically, evolving based on user experience with a system.

On the question of what exactly is the ideal response time for a website, Galletta et al. state that a website requires a maximum response time of 8 seconds to promote a positive reaction. However, he goes on to say that users are more likely to tolerate more lengthy delays from familiar sites, presumably as they are aware of what they are waiting for and place a certain value on it. In other words, it is worth waiting 20 seconds to view a particular piece of content that is known to be of a certain quality.

The question that arises is whether or not the findings of researchers involved in website response time research can be generalised and applied to the question of trail generation response time. T.W. Butler begins his study into computer response time and its effect on user performance by stating that although there has been a lack of work in this area at the time of writing (1983), it was generally accepted that different user tasks have different response time requirements for optimal user performance [19]. Testing these assertions he concluded that degradation of user performance in response to increased response time appears to be similar for tasks that are cognitively different. This throws some doubt on the generally accepted assertion that different tasks have different optimal response times.

The fundamental advice on response time has basically been the same for about forty years according to Nielsen [89]. He presents the following guidelines based on Miller's early work and later work by Card et al. [20].

- 0.1 seconds is the limit for having the user believe that the system is reacting instantaneously.
- 1 second is the limit for the user's flow of thought to stay uninterrupted, but even then they will notice delay.
- 10 seconds is about the limit for keeping the user's attention focused on the dialogue.
- For longer delays, users will want to perform other tasks while waiting and so should be given feedback indicating when the computer expects to be done. Feedback is especially important if the delay is variable, as the user doesn't know what to expect.

Nielsen has also stated, in a non-peer reviewed addendum to his writing on response time (published on his personal website<sup>5</sup>), that the guidelines he presents are applicable to all applications and that the guidelines for web-based applications are the same as those for all applications. He goes on to say that given that the guidelines have been in place for so long that they are unlikely to change any time soon, regardless of what technology comes next [90].

For the purposes of this thesis, a ‘reasonable’ response time for the trail generation algorithm is considered to be between 0-12 seconds. The upper-bound can be increased through the provision of suitable progress indication to the user. The approach to trail generation in the application framework is designed to allow a variable number of activities to be considered during trail generation so that this reasonable response time can be adhered to. The number of activities considered in a specific application depends on the capabilities of the mobile device hosting the trails application and the manner in which candidate trails are generated.

### 3.2.2 Multi-Attribute Utility Theory

The evaluation functions in both Oisín and RiddleHunt contain hard-coded values that specify how much bearing particular trail properties have on the score for each candidate solution. This approach to multi-attribute decision making is not suitable for a generic trail generation algorithm due to its lack of flexibility, and therefore the approach in the application framework has been implemented in a manner that facilitates user specification of weights for trail properties considered during trail generation, without requiring source code modifications. This approach is based on Multi-Attribute Utility Theory (MAUT) [128].

Multi-Attribute Utility Theory is a technique for evaluating objects based on user interest in various aspects of the objects. The overall evaluation  $v(x)$  of an object  $x$  is defined as the weighted addition of its evaluation with respect to its relevant value dimensions. For example, a trail can be evaluated on dimensions such as number of activities possible, duration and length. The object evaluation is defined by the overall

---

<sup>5</sup>[www.useit.com](http://www.useit.com)

value function:

$$v(x) = \sum_{i=1}^n w_i v_i(x)$$

In this function,  $v_i(x)$  is the evaluation of the object on the  $i$ -th value dimension  $d_i$ , and  $w_i$  is the weight determining the impact of the  $i$ -th value dimension on the overall function, also known as the relative importance of the dimension.  $n$  is the number of different value dimensions and  $\sum_{i=1}^n w_i = 1$ , meaning that the sum of the weights for the different value dimensions equals 1.

For each value dimension  $d_i$  the evaluation  $v_i(x)$  is defined as the evaluation of the attributes composing the dimension:

$$v_i(x) = \sum_{a \in A_i} w_{ai} v_{ai}(l(a))$$

Here  $A_i$  is the set of all attributes relevant for  $d_i$  and  $v_{ai}(l(a))$  is the evaluation of the actual level  $l(a)$  of attribute  $a$  on  $d_i$ .  $w_{ai}$  is the weight determining the impact of the evaluation of attribute  $a$  on value dimension  $d_i$ .  $w_{ai}$  is also referred to as the relative importance of attribute  $a$  for  $d_i$ . For all  $d_i (i = 1, \dots, n)$  holds  $\sum_{a \in A_i} w_{ai} = 1$ . For example, the scenic value of a tourist trail can be calculated by considering attributes such as the type of structure housing each activity and the existence of proximate public parks and spaces.

Attributes are evaluated on a scale representing the levels of an attribute e.g., on a scale of 0 to 100 a very scenic trail might have the value 95. All values must be normalised to this scale for comparison.

Table 3.1 illustrates the evaluation of four candidate trail solutions along five value dimensions. The user-specified relative weights are included in the right-most column and the total score calculated for each trail is included in the bottom row. A higher score indicates trail superiority. The values for trail length, duration and idle time are represented using negative values as these are negative trail characteristics and should decrease the trail score. The score for Trail 1 is calculated as follows:

$$(80 * 0.55) + (50 * 0.15) + (-55 * 0.0) + (-38 * 0.1) + (0 * 0.2) = 47.7$$

In this manner, the worth of a collection of candidate trails can be calculated based on

user preferences to determine best trail for a particular user. The user weights can be changed to produce different trail evaluations without changing the underlying approach.

Dimension	Trail 1	Trail 2	Trail 3	Trail 4	User Weight
Activities Possible	80	40	70	80	0.55
Mandatory Activities	50	10	40	40	0.15
Length	-55	-64	-72	-83	0.0
Duration	-38	-85	-84	-92	0.1
Idle Time	0	0	-27	-12	0.2
<b>Score</b>	47.7	23.65	30.7	38.4	1

**Table 3.1:** Trail evaluation dimensions (on a scale from 0-100) and sample data

Apart from MAUT, a number of alternative approaches to multi-attribute decision making were considered. Constraint satisfaction problems [127] were investigated as a technique for representing variable user preference values for multiple trail properties, but the concept of using constraints to model user preference, as illustrated by Zhang and Pu [136], did not map to the problem specification as naturally as MAUT. Conditional preference networks (CP-Networks) [16] can be used to solve multi-attribute decision problems where user preferences are unknown or incompletely specified. CP-Networks are therefore suitable for use in situations where user preferences can only be elicited in an incremental manner during the execution of the application. This is not the case in trails applications, where value dimensions represent basic, comprehensible trail or activity concepts and related preferences can be set either by the developer or user prior to initial trail generation. As user preferences evolve based on experience using the application, the preference values for all value dimensions can be updated repeatedly, facilitating incremental revelation of preferences. Therefore, the use of CP-Networks was deemed unnecessary. Various heuristic strategies for preference-based object evaluation such as the equal weight heuristic, the elimination-by-aspects heuristic and the satisficing strategy [96] were also considered. These heuristic approaches are modelled on techniques used by humans, both individually and in groups, to solve multi-attribute problems [2]. The possibility of using heuristics was discarded as they are not guaranteed to find the solution that maximises each of the value dimensions relative to the user preferences. [60].

### 3.2.3 Trail Generation

The major challenge in trail generation concerns the production an effective trail within a reasonable amount of time without restricting the number of activities an application can consider. Additionally, the evaluation function used to evaluate candidate trail solutions should recognise the preferences of individual users. Oisín limits the number of activities it can consider based on the behaviour of the brute force algorithm used to generate the optimal trail. The evaluation function used to evaluate candidate solutions contains hard coded relative importance weights, meaning that all users receive the same trail when in the same contextual situations, regardless of their personal preferences. RiddleHunt improves on Oisín by introducing the notions of activity relevance and subtrails, facilitating the generation of trails composed of the activities that are most relevant to the user and allowing RiddleHunt to consider a large number of activities. RiddleHunt also introduces the concept of pruning activities that no longer need to be considered when generating trails, reducing the number of candidate solutions that must be evaluated. However, like Oisín, RiddleHunt does not provide a preference elicitation mechanism and consequently, the relative importance weights in both the evaluation function and the activity relevance function are hard coded.

The generic trail generation algorithm in the application framework extends the context-based activity set reduction and partial trail generation techniques introduced in RiddleHunt. As well as removing completed activities, activities that are not possible based on the current context e.g., those that cannot be done due to time restrictions or those that clash with other, more preferable activities, are also removed. This reduces the number of activities (and permutations thereof) that must be considered by the trail generation algorithm, increasing algorithm efficiency. If the number of activities remaining after activities have been removed is greater than that which can be reasoned about in a reasonable amount of time, the activities are assigned a relevance value and the most relevant activities (based on user preferences and current context) are considered during trail generation. This allows a trail containing the most relevant activities to be generated for the user even when the application contains more activities than can be reasoned about efficiently. This approach is explained in detail throughout the remainder

of this section.

An activity set contains all of the activities that a user can theoretically do while using a trails application. Reducing the number of prospective activities reduces the number of trail permutations that must be evaluated when finding the best activity ordering. The trail generation mechanism in the application framework uses context in four main ways: first to prune the activity set of completed activities, second to prune the activity set of impossible activities, third to resolve activity clashes and finally, if necessary, to divide the remaining activities into two sets, the relevant set and the irrelevant set. This process is illustrated in Figure 3.9. Line 1 in Figure 3.9 represents a complete activity set,  $X$ , containing ten activities named with letters of the alphabet  $a$  through  $j$ :  $X = \{a, b, c, d, e, f, g, h, i, j\}$ . A high-level overview of the decisions and actions involved in the process of generating a trail from a set of activities is illustrated in the activity diagram in Figure 3.10.

### 3.2.3.1 Completed Activities

An activity is *completed* when the activity status is manually changed to ‘complete’ via an application user interface. Line 2 in Figure 3.9 illustrates the identification of two completed activities in  $X$ ,  $c$  and  $i$ . These activities are removed from  $X$ , reducing the set cardinality by 2. In the activity diagram in Figure 3.10, completed activities are removed in step (A).

### 3.2.3.2 Impossible Activities

An activity is considered to be *impossible* if it is not available for the user to undertake even though it has not been completed. There are two ways in which an activity can become impossible. First, when it is no longer possible for the user to get to the activity location and complete it before the activity closing time is reached. Second, when a non-temporal activity limit is reached. For example, it will eventually become impossible to attend a movie when the theatre reaches capacity attendance. Line 3 in Figure 3.9 illustrates the identification of an impossible activity,  $a$ , based on the current location context which is represented by the globe icon.  $a$  is removed from  $X$ . Figure 3.10 illustrates that impossible activities are removed at step (A) following the removal of

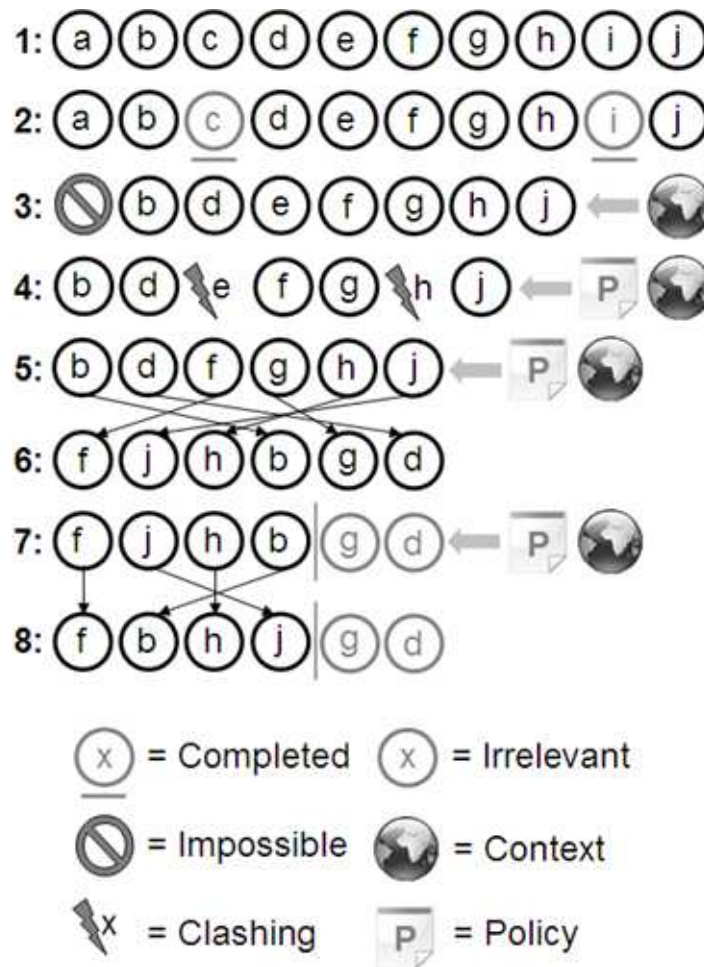


Figure 3.9: Context-based activity set reduction

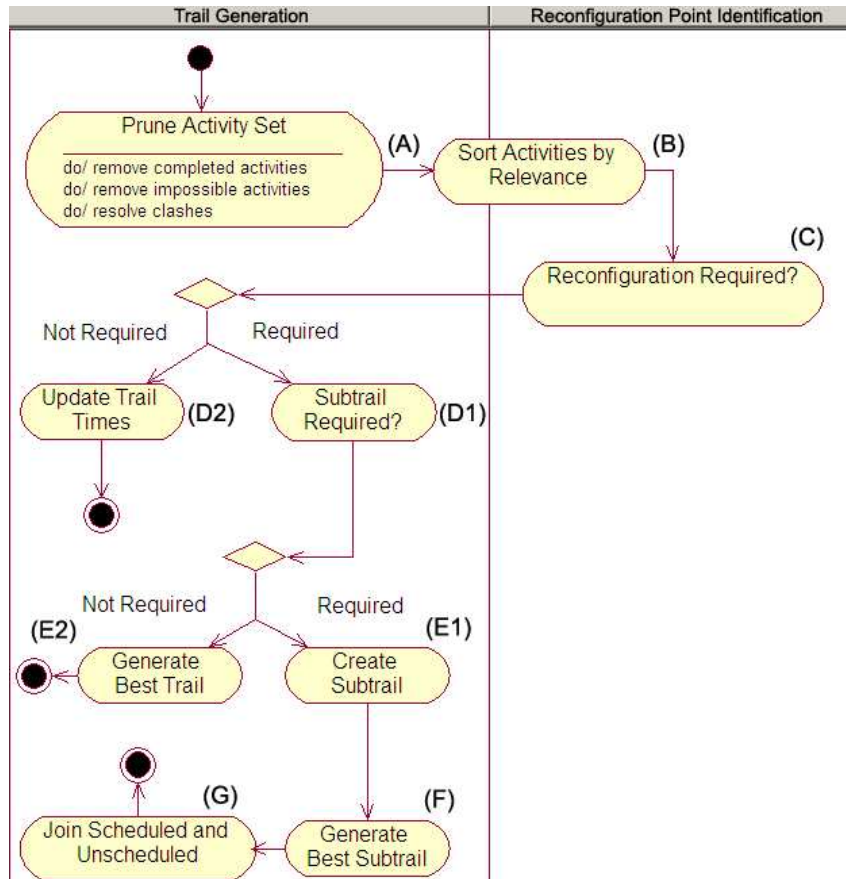
completed activities.

### 3.2.3.3 Clashing Activities

Two activities clash when, although independently possible, their opening hours and estimated durations conflict, leading to a situation in which the completion of one activity renders the other impossible. For example, two movies clash if they are showing at the same time and have similar running times. When clashes are identified they must be resolved, resulting in the rejection of one activity and its removal from the activity set.

Clash resolution is achieved via a technique based on MAUT that considers user-specified weights when comparing the following activity properties:

- Proximity - the distance from the user's current location to the activity location.



**Figure 3.10:** The trail generation process

- Priority - the user-specified numerical priority level for the activity e.g., 5 for high, 1 for low.
- Obligation - whether the activity is mandatory or optional.

The clash resolution technique produces a single value for each activity by aggregating values for the set of weighted trail properties listed above. A numerical value for each relevant activity property is calculated, normalised and multiplied by the user-specified weight for that property. The values for each of the properties are summed and the activity with the lowest overall value is rejected and removed from the activity set. In the unlikely event that the scores are equal, a single activity is randomly selected for rejection. Table 3.2 illustrates the comparison of two clashing activities. Activity 2 is rejected because it has a lower score than Activity 1. Activities that have been removed from the activity set as a result of clash resolution can be reinstated in the event of significant context change.



Dimension	Activity 1	Activity 2	User Weight
Proximity	-80	-40	0.6
Priority	60	10	0.1
Obligation	100	0	0.3
<b>Score</b>	-12	-23	1

**Table 3.2:** Activity evaluation dimensions for clash resolution and sample data

Line 4 in Figure 3.9 illustrates the identification of a clash between activities  $e$  and  $h$  based on knowledge of activity properties and user location. The clash is resolved using the policy described above which is represented by the file icon.  $e$  is rejected and removed from  $X$ . Step (A) in Figure 3.10 shows that activities rejected as a result of a clash are removed following the removal of completed and impossible activities.

### 3.2.3.4 The Relevant and Irrelevant Sets

After the context-based pruning of the activity set  $X$ , a number of candidate activities remain from which a trail is composed. Depending on the number of activities in  $X$ , the activity set may be split into a relevant set ( $X_o$ ) and an irrelevant set ( $X_a$ ). This division occurs if the cardinality of the activity set  $X$  is greater than the number of activities that can be reasoned over in a reasonable application response time (between 0-12 seconds). The relevant set  $X_o$  comprises activities that, based on context, are considered most relevant for the user at a given point in time.

Activities are sorted by relevance (shown as part of both the trail generation and reconfiguration point identification processes in step (B), Figure 3.10) and a decision regarding whether or not reconfiguration is required (discussed in Section 3.2.4) is made in step (C). If reconfiguration is not required, the estimated activity start and end time of each activity scheduled on the trail is updated based on the current context and the process terminates. If reconfiguration is required, a check occurs to assess whether a subtrail is necessary. If so, the top  $z$  activities (where  $z$  is the cardinality of  $X_o$  defined by the application developer based on a desire to achieve a reasonable response time) are stored in  $X_o$ . This process is illustrated by step (E1) in Figure 3.10. The remaining activities are stored in the irrelevant set  $X_a$ . Step (E2) represents the situation in which there is no necessity to create a subtrail during trail generation.

The relevance value for an activity is calculated using a user preference-driven MAUT approach similar to that used for clash resolution. The relevance value is the result of summing normalised weighted values for the following activity properties:

- Proximity (defined in Section 3.2.3.3).
- Priority (defined in Section 3.2.3.3).
- Obligation (defined in Section 3.2.3.3).
- Urgency - how urgently an activity must be addressed based on its opening hours.

Table 3.3 illustrates the comparison of two activities based on user-defined weights for the four value dimensions. Activity 2 is rejected because it has the lower score. Lines 5 and 6

<b>Dimension</b>	<b>Activity 1</b>	<b>Activity 2</b>	<b>User Weight</b>
Proximity	-45	-57	0.6
Priority	20	80	0.15
Obligation	100	0	0.05
Urgency	82	64	0.2
<b>Score</b>	-2.6	-9.4	1

**Table 3.3:** Activity evaluation dimensions for relevance and sample data

in Figure 3.9 illustrate sorting an activity set by relevance based on current context and a user-defined policy. Line 7 illustrates the division of the activity set into the relevant and irrelevant sets,  $X_o$  and  $X_a$ .

When the relevant set has been populated with activities, the trail that best satisfies the user's preferences can be generated (step (F) in Figure 3.10). This trail represents a specific ordering of the activities in  $X_o$ . The activities in the irrelevant set  $X_a$  are appended to the generated trail ( $X_o \cup X_a$ ) and marked as unscheduled, illustrated in step (G). This allows all of the activities to be displayed to the user, but only those scheduled on the trail have scheduling information associated with them e.g., a position in the trail and an estimated start/end time. The completion of a trail activity results in a member of the irrelevant set  $X_a$  being drawn into the relevant set  $X_o$  and scheduled on the trail. At any point, a context event can cause a revision of the relevant set that results in activity migration from the irrelevant set to the relevant set and vice versa.

### 3.2.3.5 Trail Ordering

The trail ordering mechanism assesses relevant set orderings and returns the best fit to the user's preferences. A MAUT-based evaluation function considers user-specified weights for various trail properties and returns a single numerical value for each permutation. The following trail properties are considered when evaluating a candidate trail:

- The number of activities possible.
- The number of mandatory activities possible.
- The total trail length/travel distance required.
- The trail duration.
- The amount of idle time.

Table 3.1 in Section 3.2.2 illustrates the evaluation of four candidate trails using the value dimensions listed above. Three approaches for generating candidate trails are included in the framework:

- Brute force.
- Simulated annealing.
- Genetic algorithm.

Brute force trail generation finds the optimal trail by exhaustively evaluating all permutations of a set of activities and returning the best trail. Brute force trail generation has time complexity  $O(n!)$ . Therefore it is not advisable to allow a trail generation algorithm using brute force to execute without controlling the number of activities considered as this would have a detrimental effect on application responsiveness.

Simulated annealing is a probabilistic approach to combinatorial optimisation problem solving [109]. In metallurgy, annealing is the process used to harden metals and glass by heating the materials to a high temperature and cooling them gradually in a controlled manner to increase the size of the crystals and reduce defects. The heat causes atoms to become unstuck from their initial positions and wander randomly through states of higher

energy, with the slow cooling providing more chance of finding configurations with lower internal energy (greater hardness) than the initial one. Mapping this technique to trails, activities are analogous to atoms and the trails are analogous to the material, composed of atoms, that is being modified. The simulated annealing algorithm replaces the current solution with a random ‘nearby’ solution i.e., the order of the activities in the trail is modified to some degree at each step. The nearby solution is chosen based on the value of a temperature parameter that controls the cooling of the material. The temperature parameter represents a numeric parameter that controls the magnitude of change to the trail being modified at each step in the algorithm. The solution changes almost randomly when the temperature is high but changes become increasingly less dramatic as the temperature moves towards zero. Therefore, when the control (temperature) parameter is high, the order of the activities in the trail will fluctuate almost randomly in a bid to find a better configuration. As the control parameter moves towards zero, the changes will become less erratic until the order stabilises and the final trail ordering is returned. In simulated annealing, the solution produced is not guaranteed to be optimal, but solution quality depends on the length of the annealing schedule i.e., how long the algorithm is programmed to run for (the starting value of the temperature parameter and how fast it moves towards zero) and how many trials the algorithm carries out at each step.

A Genetic algorithm (GA) is a guided random search technique that simulates biological evolution by combining two parent candidate trails to produce what should be a better child trail [109]. A genetic algorithm begins with a set of randomly generated candidate trails (chromosomes in GA terminology) known as the initial population. The next generation of states is produced by first selecting the best trails from the current generation as judged by a fitness function (i.e., the evaluation function used to generate scores for candidate trails). Pairs of trails are then selected to reproduce using a crossover method that selects the number of activities (genes in GA terminology) from each trail that make it into the new child trail. Finally, newly created trails are subject to random mutation with a small independent probability, swapping the position of two activities in the trail. Evolution continues for the number of generations specified. Genetic algorithm solution quality depends on the size of the population and the number of generations the algorithm runs for. In turn, these factors affect the execution time.

Brute force trail generation is appropriate when the number of activities to be considered during trail generation can be limited to a number that can be reasoned about in a reasonable response time. This is suitable for many applications in which users prefer to have a subset of their activities scheduled optimally as opposed to have all their activities scheduled suboptimally. However, as response times with brute force increase exponentially as activities are added, going beyond the consideration of a small number of activities during trail generation is not feasible. Response times with simulated annealing and genetic algorithms do not increase exponentially, the increase in response time following the addition of an activity is far less significant than with the brute force algorithm. Simulated annealing has been shown to perform better (in terms of solution quality) than genetic algorithms when both algorithms are terminated after the same time period [72]. Therefore, simulated annealing is preferable when application execution time must be constrained. Achieving a reasonable response time with simulated annealing or a genetic algorithm requires making concessions in terms of solution quality, depending on the number of activities being considered. If the number of activities is relatively large then solution quality must be sacrificed if the application is to respond in a timely fashion. This illustrates the general trade-off between solution quality and application responsiveness in combinatorial optimisation problem solving.

Lines 7 and 8 in Figure 3.9 illustrate the trail ordering process, using current context and a user-weighted policy to order the relevant set to best serve the user.

### **3.2.3.6 Reusability and Extensibility**

The trail generation mechanism in the application framework provides a generic approach to trail generation that can be reused when implementing mobile, context-aware trails-based applications. The proposed approach has also been designed to be customised and extended by developers. The following aspects of the trail generation mechanism can be customised by developers without requiring invasive source code modifications:

- Clash resolution policy. The weights of the properties considered during clash resolution can be altered on a per-application or per-user basis to achieve different results with the same clash resolution implementation.

- Definition of relevance. The weights of the properties considered in the definition of activity relevance can be set in many different configurations to produce different ‘most relevant’ orderings from the same set of activities.
- Relevant set size. The number of activities in the relevant set can be specified on a per-application or per-user basis.
- Evaluation function. The weights assigned to the trail properties used in the evaluation function are configurable in the same manner as those used in the clash resolution and relevance policies, facilitating the production of many different ‘best’ trails without source code modification.
- Candidate solution generation algorithm. The candidate solution generation algorithm used at runtime can be selected on a per-application or per-user basis without requiring source code modification. Additionally, the properties of the genetic and simulated annealing algorithms can be configured by the developer without requiring source code modification.

In terms of extensibility, the approach is designed to facilitate the implementation of multiple approaches to candidate solution generation and evaluation e.g., brute force and genetic algorithm. The specifics of whether an exact or heuristic approach is used are hidden from the client, allowing developers to extend the application framework by plugging in new implementations. This provides the opportunity for experimentation with specific implementations of combinatorial optimisation problem solving techniques.

The application framework can also be extended to add new contexts and trail/activity properties by subclassing the relevant classes in the default implementation. A generic context source specification is extended to provide location and time lapse context. This generic specification can be extended to provide further concrete context types. The default trail and activity specifications containing generic attributes and behaviour that are provided in the framework can be extended and customised through inheritance to meet the requirements of specific applications.

### 3.2.3.7 Summary

The trail generation mechanism included in the application framework facilitates the development of applications that can include a large number of activities by using context to prune the activity set of completed, impossible and clashing activities, and to determine the relevance of each activity. Clashes between activities are resolved based on user preferences for clash resolution. The notion of activity relevance is also determined by user preference values. Based on a maximum response time of 12 seconds an application using this approach will include as many of the relevant activities as possible when generating and reconfiguring trails. The remaining activities are marked as unscheduled. The user's trail is composed of the activities that are most relevant to the user given the current situation.

### 3.2.4 Reconfiguration Point Identification

The major challenge in reconfiguration point identification concerns minimising unnecessary trail reconfiguration (which needlessly consumes mobile device resources) while maximising the amount of time that the trail reflects the reality in which the user exists. Reconfiguration point identification in Oisín is based on the occurrence of context events, namely location change and time lapse events. Trail reconfiguration is triggered each time a context event occurs. This approach ensures that the trail always reflects events in the user's physical environment (in terms of location), however it also results in the majority of reconfigurations having no effect on the ordering of the existing trail. Reconfiguration point identification in RiddleHunt is based on changes in the set of activities judged to be most relevant to the user following the receipt of a context event. This approach avoids unnecessary trail generation by triggering reconfiguration only when the set of activities to be included on the trail has changed, meaning that the resultant trail will contain at least one activity not in the current trail. However, this technique considers only a shallow, surface-level comparison of the activity sets, reacting only to relevant set membership change as opposed to changes in the internal composition of the relevant set. Consequently, context events that cause significant changes to the internal structure of the relevant set do not trigger reconfiguration, meaning that the user can potentially be

following a trail that does not reflect recent context events.

The reconfiguration point identification technique in the application framework is based on the concept of activity relevance introduced by the trail generation mechanism and extends the basic approach developed for RiddleHunt. A trail is reconfigured if there is a 'significant' difference between the set of relevant activities from which the existing trail is composed and the new set of relevant activities generated following notification of a new context event e.g., each time the user changes location or an activity property changes. This decision point is illustrated by step (B) in Figure 3.10. Periodic reconfiguration is also supported to cater for situations when no other context events are generated. Both the time interval for periodic reconfiguration and the significance threshold used for identifying differences between the two sets of relevant activities are configurable by the application developer or user. There are two ways in which a significant difference between activity sets can arise - differences in set membership and differences in activity relevance rankings.

#### **3.2.4.1 Differences in Set Membership**

If the new relevant set generated following notification of a context event contains one or more activities that are not on the current trail then the trail is reconfigured automatically. It is not possible in this case for trail reconfiguration to occur unnecessarily i.e., without changing the composition of the trail, as the two activity sets are not equal and therefore the new trail will contain at least one activity that was not scheduled on the previous trail. This behaviour caters for the following situations:

1. Activity migration between the relevant set and the irrelevant set.
2. Activity completion.
3. An activity becoming impossible.
4. The dynamic addition of new activities to the application that have a high relevance value.

If the two relevant sets contain the same activities then none of the situations listed above exist. In the case of relevant set equality an examination of the activity rankings



generated based on activity relevance is required.

### 3.2.4.2 Differences in Relevance Rankings

A trail is not automatically reconfigured if the new relevant set contains the same activities as the relevant set that the current trail is based on. In this case a comparison of the two sets of activities is required to ascertain if there are differences in the internal composition of the two sets that warrant reconfiguration. This is the behaviour that is not included in the RiddleHunt implementation of reconfiguration point identification (Section 3.1.4.2).

Kendall’s rank correlation coefficient, known as Kendall’s  $\tau$ , calculates the correspondence between two rankings [64]. This approach is used to calculate the degree of correspondence between the new relevant set and the relevant set used to generate the user’s current trail. Kendall’s  $\tau$  coefficient is defined as follows:

$$\tau = \frac{2P}{\frac{1}{2}n(n-1)} - 1 = \frac{4P}{n(n-1)} - 1$$

where  $n$  is the number of items and  $P$  is the sum, over all the items, of items ranked after the given item by both rankings. The  $\tau$  coefficient has the following properties:

- If the agreement between two rankings is perfect (i.e., the two rankings are the same) the coefficient has value 1.
- If the disagreement between two rankings is perfect (i.e., the rankings are opposites of each other) the coefficient has value -1.
- For all other arrangements the value lies between -1 and 1, with higher values indicating stronger agreement between rankings. If the rankings are completely independent the coefficient has value 0.

Activity	a	b	c	d	e	f	g	h
New relevant set	1	2	3	4	5	6	7	8
Previous relevant set	3	4	1	2	5	7	8	6

**Table 3.4:** The relevant set and the existing trail ranked by relevance

The following example (adapted from [122]) illustrates how Kendall’s  $\tau$  can be used to calculate the degree of agreement between two sets of relevance-ranked trail activities. Table

3.4 illustrates the positions of eight activities in the relevant set ( $\{a, b, c, d, e, f, g, h\}$ ) generated following notification of a context event and the relevant set used to generate the current trail ( $\{c, d, a, b, e, h, f, g\}$ ). There is some correlation between the two rankings, however the correlation is far from exemplifying perfect agreement. Kendall's  $\tau$  can be used to calculate the degree of agreement between the two sets. The first entry in the previous relevant set ranking has five higher ranks to the right of it, therefore contributing 5 to the  $P$  value for this entry. The second entry in the existing trail ranking has four higher ranks to the right of it making 4 the contribution to  $P$ . Moving through the set in this way results in the following:

$$P = 5 + 4 + 5 + 4 + 3 + 1 + 0 + 0 = 22$$

Following the calculation of  $P$  it is possible to calculate  $\tau$ :

$$\tau = \frac{44}{28} - 1 = 0.57$$

The resultant  $\tau$  value, 0.57, indicates a moderate level of agreement between the two rankings.

In the case of relevant set membership equality, a trail is reconfigured in the application framework if the  $\tau$  value for correspondence between the new and previous relevant sets is below a threshold defined by the application developer or user. For example, if a threshold of 0.8 was in place for the example above then the trail would be reconfigured. This approach allows an extensible range of contexts to be considered during reconfiguration point identification as the receipt of any context event can trigger the generation of a new relevant set and the comparison of this set to the set used to form the existing trail. The contexts that trigger reconfiguration must be considered in the MAUT approach to activity relevance calculation, otherwise they can have no effect on the relevant set generated following the notification of the new context event. The cost (in terms of time) of comparing relevant sets each time a context event occurs is far less than the cost incurred by unnecessary trail reconfiguration, making reconfiguration point identification a preferable alternative to reconfiguring the trail each time a context event is received. This is illustrated by example in Section 5.2.3.4 during an analysis of the accuracy of the reconfiguration point identification technique.

Apart from Kendall's  $\tau$ , a number of alternative rank correlation techniques were considered for use in the design of the reconfiguration point identification mechanism. Pearson's product-moment correlation coefficient (PMCC) [23] is a measure of correlation between two variables measured on the same object. PMCC measures the tendency of the variables to increase or decrease together e.g., does human body weight increase with height? This measure requires the assumption that the relationship between the variables is linear, which isn't the case in the comparison of two ranked sets of activities. Additionally, PMCC is not well suited for use with small sample sets such as the subtrails used in trail generation. Spearman's  $\rho$  [121], a rank correlation coefficient that performs a similar function to Kendall's  $\tau$ , was also considered for use in measuring the difference between relevant sets. Like Kendall's  $\tau$ ,  $\rho$  does not require the assumption that the relationship between variables is linear. Spearman's  $\rho$  is considered to be comparable to Kendall's  $\tau$  in terms of statistical power [15, 50] and is noted as being the most commonly used method of calculating rank correlation [46]. However, as discussed by Noether [91], the interpretation of Kendall's coefficient is intuitively simple whereas assigning an interpretation to Spearman's coefficient is a non-trivial undertaking. Additionally, the algebraic structure of Spearman's coefficient is far more complex than that of Kendall's. Consequently, Kendall's  $\tau$  was deemed the most appropriate approach to adopt.

### 3.2.4.3 Reusability and Extensibility

The application framework provides a generic solution to reconfiguration point identification. The approach can be reused as is or can be customised and extended to meet the needs of specific applications. In terms of customisation, the  $\tau$  value that specifies when a difference between two sets of activities ranked by relevance is significant can be modified without source code alteration to change the sensitivity of the reconfiguration point identification approach. The approach is based on the concept of activity relevance and therefore when the framework is extended through the introduction of new contexts or trail properties into the activity relevance calculation, the reconfiguration point identification mechanism will automatically consider the new information. It is not apparent that developers will need to extend the rank correlation method used to compare relevant sets. However, irrespective of this, it is possible to extend the existing implementation

and override the default rank correlation behaviour.

#### **3.2.4.4 Summary**

The reconfiguration point identification mechanism included in the application framework improves on the approach used in RiddleHunt by supporting not only reconfiguration triggered by changes in relevant activity set membership, but also by significant differences in the relevance values of individual activities when relevant set membership is the same. The difference between the relevance rankings of two sets containing the same activities is measured using Kendall's rank correlation coefficient. The definition of what is a significant difference between two sets is configurable by the developer or user. This approach facilitates the minimisation of the occurrence of unnecessary trail reconfigurations, i.e., minimising mobile device resource wastage, while maintaining consistency between the user's physical environment and the representation of that environment provided by the trails application on their mobile device.

### **3.3 Chapter Summary**

This chapter has described the design methodology with which the application framework proposed in this thesis was developed. The chapter illustrates how the application framework evolved through early requirements gathering work and how Hermes supports context acquisition and modelling. The implementation of two prototype mobile, context-aware trails-based applications, Oisín and RiddleHunt, was described in terms of the role the applications played in the application framework development process. The design of the trail generation and reconfiguration point identification mechanisms included in the application framework were then described. The following chapter presents the implementation detail of the application framework features described in this chapter.

# Chapter 4

## Implementation

The previous chapter describes the design of the trail generation and reconfiguration point identification techniques and illustrates how the application framework supports the generation and reconfiguration of trails composed of the activities that are considered, based on context, to be most relevant to the user. This chapter describes in detail how these techniques are implemented. The application framework, implemented in Java<sup>1</sup>, is composed of generic implementations of the trail generation and reconfiguration point identification techniques that can be reused by applications that require trails behaviour. The default implementations are developed in a manner that facilitates extension by developers who wish to specify application-specific trails behaviour.

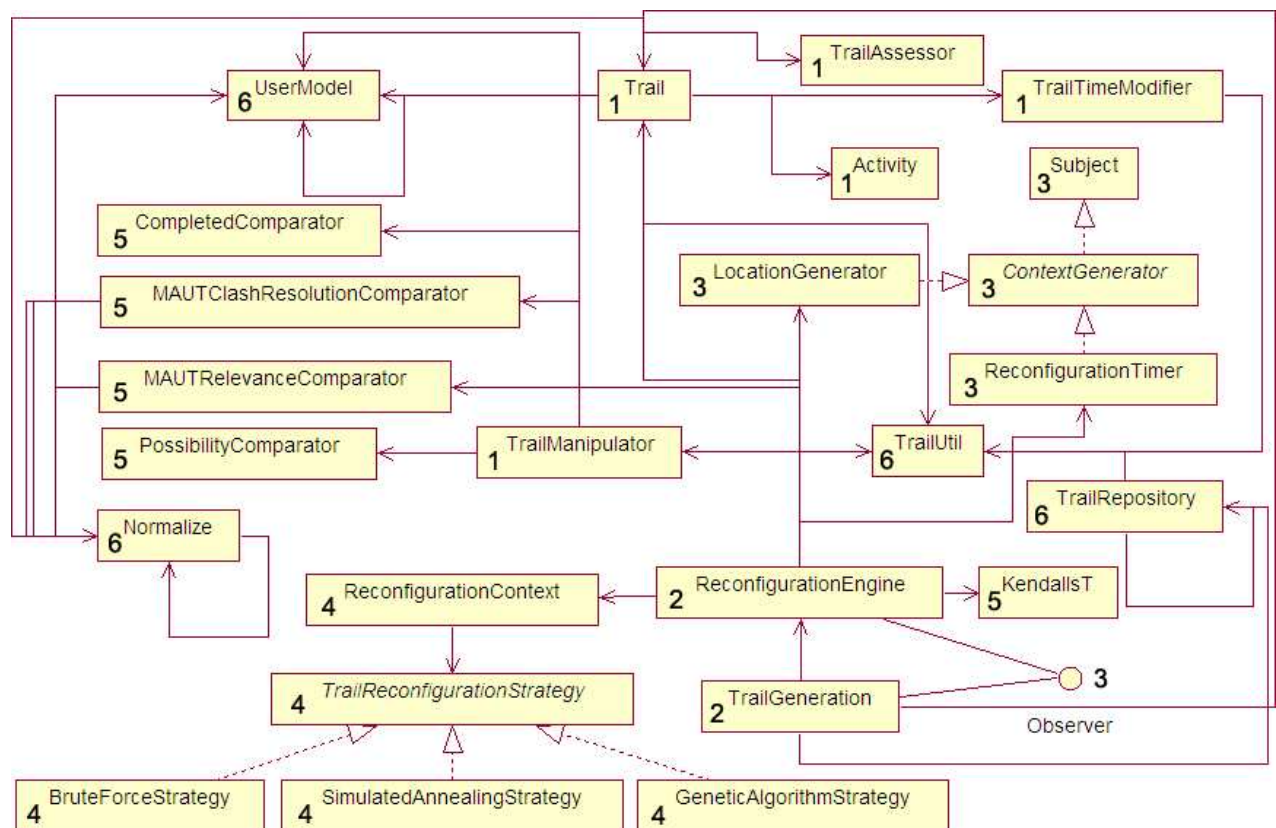
The chapter begins with a high-level overview of the classes that compose the application framework. This is followed by a presentation of the attributes and behaviour in the generic trail and activity specifications, and a discussion regarding the manner in which the default implementations can be extended to support specialised activity and trail attributes and behaviour. The implementation details for both the trail generation and reconfiguration point identification techniques are presented next, illustrating how the classes in the application framework collaborate to provide trail generation and reconfiguration behaviour. A discussion of the extension points in both implementations is also presented. Finally, the configuration files that facilitate customisation of application framework behaviour without requiring source code modification are discussed.

---

<sup>1</sup>Java 2 Micro Edition (J2ME) Personal Basis Profile [84]

## 4.1 Application Framework Overview

The application framework consists of 34 classes and 5 properties files that provide structure and behaviour for mobile, context-aware trails-based applications. This section describes the implementation of the application framework at a high-level, illustrating the core classes<sup>2</sup> and outlining their responsibilities. Figure 4.1 illustrates ‘uses’ relationships between the classes in the application framework. The classes are logically grouped into 7 groups and each class in the diagram is annotated with the number of the group it belongs to. The class groups that compose the application framework are as follows:



**Figure 4.1:** High-level application framework class diagram

1. *Trails*. This collection of classes has four primary responsibilities:

- (a) To provide a representation of the user’s trail. Section 4.2 discusses the classes that define the behaviour and attributes of a trail and an activity.

<sup>2</sup>The classes omitted from the diagram are those related to the implementation of the concrete trail reconfiguration strategies.

- (b) To provide behaviour to manipulate both the contents of the trail and the attributes of the activities. The number of activities on a trail is modified when subtrails are created and when activities are completed, become impossible or are removed as a result of a clash. This behaviour is contained in `TrailManipulator`. Additionally, activity properties must be updated following context events e.g., the activity start and end time estimates must be updated based on changes in user location or the passage of time. This behaviour is contained in `TrailTimeModifier`.
- (c) To provide various types of information about candidate trail solutions based on their activity ordering and the current context e.g., the number of possible activities and the total time required to complete the activities on the trail. This behaviour is encoded in `TrailAssessor`.
- (d) To provide a trail evaluation function. The class that represents the user's trail (`Trail`) is self-describing in that it contains behaviour to evaluate the activity ordering it contains. The evaluation function uses `TrailAssessor` to assess candidate trail solutions along the value dimensions discussed in Section 3.2.3.5.

2. *Controllers*. This group of classes is responsible for using the behaviour defined in the other class groups to coordinate the trail generation and reconfiguration process. `TrailGeneration` acts as a gateway to the services of the application framework which are coordinated by `ReconfigurationEngine`. `ReconfigurationEngine` receives context events, assesses if reconfiguration is required and reconfigures the trail as appropriate. `ReconfigurationEngine` informs `TrailGeneration` of changes to the user's trail following reconfiguration.

3. *Context Generators*. The classes in this group are responsible for generating context information and making it available to the class that is responsible for coordinating trail reconfiguration (`ReconfigurationEngine` in the *Controllers* group). The `Subject` class, along with the `Observer` interface, represents the implementation of the Observer design pattern that defines a one-to-many relationship between a subject object and any number of observers so that when a subject changes state,

all its observer objects are notified and updated automatically [45]. Clients e.g., `ReconfigurationEngine`, subscribe for notification about context events generated by the classes that implement the abstract `ContextGenerator` class. The application framework provides location and time lapse contexts by default. A detailed description of the GPS-based implementation of the `LocationGenerator` class is included in Appendix B.1.

4. *Trail Reconfiguration Strategies.* This group of classes is responsible for shielding the primary application controller (`ReconfigurationEngine`) from the specifics of how candidate trail solutions are generated during trail reconfiguration. A number of different strategies can be used to generate and evaluate candidate trails. `ReconfigurationContext` forms part of the Strategy design pattern [45]. The Strategy pattern allows a family of algorithms to be defined, encapsulated and used interchangeably. This facilitates multiple variants of the candidate trail generation behaviour without requiring the client, `ReconfigurationEngine`, to change how it invokes the behaviour or uses the returned value. `TrailReconfigurationStrategy` specifies the methods that must be implemented by the concrete strategies (`BruteForce`, `SimulatedAnnealing` and `GeneticAlgorithm`).
5. *Activity Comparators.* Comparator classes, which implement the `Comparator` interface from the standard `java.util` package, provide a comparison function that imposes a total ordering on some collection of objects. The activity comparators in the application framework are responsible for providing various ways in which to compare activities. Activity comparators are generally used to sort collections of activities based on some criteria e.g., activities are compared by relevance in `MAUTRelevanceComparator` which compares activities based on multiple value dimensions. Comparators can also be used to compare two activities at a time. This usage model is employed when `MAUTClashResolutionComparator` is invoked to resolve a clash between two activities. The implementation of Kendall's rank correlation coefficient (`KendallsT`) is also included in this group.
6. *Utilities.* The utilities group contains classes that do not provide core application framework behaviour but are necessary nonetheless. `Normalize` provides methods



to transform trail and activity assessment values e.g., the number of activities possible and the trail length, so that they are relative to each other on a specified scale, enabling comparison. `UserModel` provides access to the weights that are used in both the MAUT-based techniques for activity comparison and the trail evaluation function, both of which evaluate objects based on multiple value dimensions for which preference weights are specified. `TrailRepository` and `TrailUtil` provide behaviour to load trail specifications from disk and perform miscellaneous behaviour e.g., data format conversion and distance estimation, respectively.

The remainder of this chapter discusses how the groups of classes in the application framework collaborate to provide reusable, extensible trails behaviour. Section 4.2 discusses the attributes and behaviour in the trail and activity implementations and illustrates how the default implementations can be extended. Section 4.3 illustrates how the class groups collaborate to provide trail generation behaviour and discusses how the generic trail generation behaviour can be extended. Section 4.4 presents the implementation of reconfiguration point identification technique based on Kendall's rank correlation coefficient and discusses how it can be extended. Section 4.5 discusses the configuration files used to specify application properties and customise behaviour without source code modification.

## 4.2 Trail/Activity Specification

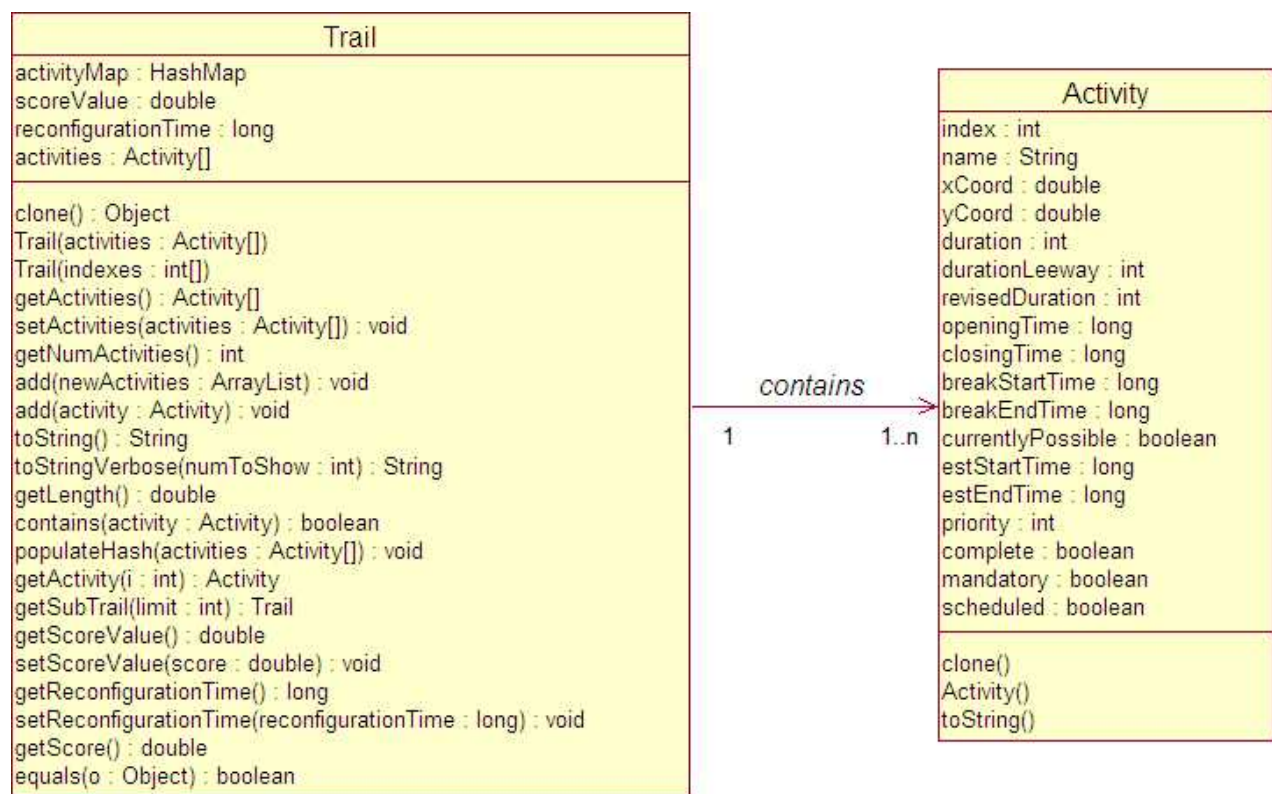
In order to implement the concrete classes that represent a trail and an activity, the attributes and behaviour of both objects had to be stated explicitly. The primary goal during of the implementation of the trail and activity classes was to specify enough attributes and behaviour to make the classes suitable for reuse without extension, without precluding or discouraging applications from reusing the default implementations<sup>3</sup>. The attributes selected for inclusion in the generic implementation of a trail are those that have been observed, during the design process, to be common to all the trails applications considered. The same design rationale was applied to the selection of activity

---

<sup>3</sup>It is reasoned that the presence of a significant amount of irrelevant attributes and behaviour relative to the application under consideration discourages framework use.

attributes. However, although certain applications e.g., RiddleHunt, do not consider the notion of activity time constraints, a decision was made to include support for activity time constraints by default as time is generally an important factor in scheduling activities. Activity properties that do not apply to the activities in a specific application can remain set to default values so that they are essentially ignored e.g., if an activity does not have time constraints then the opening and closing time are ignored. The behaviour in the trail implementation represents generic trail operations e.g., add activity and get activity, as well as the evaluation function.

Figure 4.2 illustrates the behaviour and properties of the trail and activity representations in the application framework. The class diagram shows all the attributes and methods in `Trail` and all the attributes and selected<sup>4</sup> methods in `Activity`. A trail can be composed of one or more activities.



**Figure 4.2:** The Trail and Activity classes

`Activity` contains the attributes that describe a generic activity. Some attributes are assigned a value at application start-up whereas other attributes are only assigned a value

<sup>4</sup>The only methods not shown are the accessor methods for the attributes listed.

at runtime. For example, activity attributes such as name, duration and opening time are included in the persistent specification of an activity on disk (discussed in Section 4.2.1), whereas the estimated activity start time and whether the activity is currently possible can only be determined during the trail generation process, as the values are based on the activity's position in the trail.

The `Trail` class contains a collection of activities called `activities`. The contents of the `activities` collection is dynamic, with the order of the activities in the collection representing the order of the activities on the trail. Activities can be added and removed, change position and can have their attribute values updated. The `Trail` class contains a `scoreValue` attribute that is set by the `getScore()` method which represents the implementation of the trail evaluation function. The majority of the methods in `Trail` are concerned with the collection of activities it stores, either returning information about it or modifying the contents.

### 4.2.1 Trail Persistence

Trail information i.e., the static specification of information regarding the activities on a trail, resides on disk and is read into the application at application startup. The application framework contains a utility class (`TrailRepository`) that is responsible for reading trail information from disk and creating `Activity` and `Trail` objects based on textual activity specifications. Trails are described as a collection of activity specifications in a properties file<sup>5</sup> that is read by the `TrailRepository` class. An example activity specification is illustrated in Listing 4.1. The activity specification contains the unique activity identifier (the number between the word *activity* and the specific property name e.g., *description* at line 1). It also contains a value for each activity attribute that is known prior to runtime. These attributes are described below:

---

Listing 4.1: An example activity specification

---

```
1 activity.1.description=Data Structures Lecture
```

---

<sup>5</sup>A relational-database management system (RDBMS) was used to manage trail/activity information in Oisín, but a more generic, flat file-based approach needed to be adopted based on the lack of widespread RDBMS support on mobile platforms.

```
2 activity .1.x=435
3 activity .1.y=365
4 activity .1.openingTime=09:30
5 activity .1.closingTime=10:30
6 activity .1.closingTime.leeway=0
7 activity .1.breakStart=0
8 activity .1.breakEnd=0
9 activity .1.duration=60
10 activity .1.duration.leeway=0
11 activity .1.priority=5
12 activity .1.mandatory=false
```

---

- **description** - a short text description or name for the activity e.g., ‘Data Structures Lecture’ as illustrated on line 1 in Listing 4.1.
- **x** - the x activity location coordinate (activities are located on a 2D grid that corresponds to their position on a map-based interface<sup>6</sup>).
- **y** - the y activity location coordinate.
- **openingTime** - the time from which the activity is available to be undertaken.
- **closingTime** - the time at which the activity ceases to be available to the user.
- **closingTime.leeway** - the amount of leeway in the activity closing time e.g., a shop may stay open 10 minutes later than advertised to facilitate customers in finishing their shopping. This property is set to zero by default.
- **breakStart** - the time at which the activity becomes temporarily unavailable e.g., if the activity is not available during lunch hours. By default, activities do not have breaks.

---

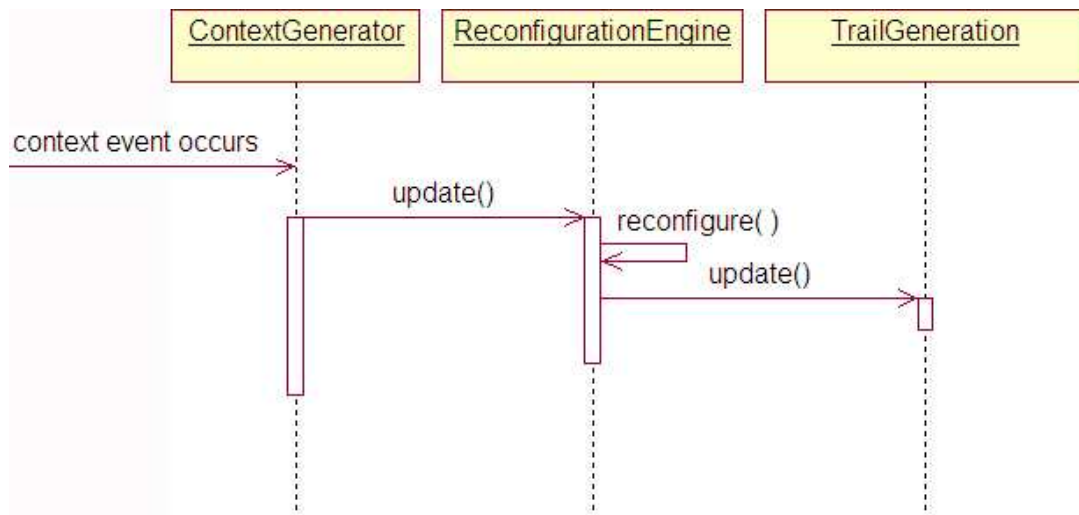
<sup>6</sup>X, Y coordinates were chosen as the format to represent activity location as it is envisaged that trails applications will have map-based user interfaces and it is easier for developers to acquire X,Y coordinates for a map location than, for example, GPS coordinates. Therefore, developers can specify locations as X,Y coordinates and user location data e.g., GPS information, can be translated to X,Y to calculate the user’s location from the activity locations. GPS to X, Y translation is described in Appendix B.1. Text names cannot be used to represent locations as it is not possible to calculate distances using text-based location data.

- `breakEnd` - the time at which the activity resumes availability.
- `duration` - the estimated amount of time (in minutes) that the user needs to spend to complete the activity.
- `duration.leeway` - the amount of leeway in the duration i.e., how much quicker can the activity be completed if necessary.
- `priority` - the priority of this activity relative to other activities. Higher numerical values indicate higher priority.
- `mandatory` - a Boolean value indicating whether or not the activity must be undertaken.

The `loadTrail()` method in `TrailRepository` iterates through the activity specifications contained in the properties file and creates an `Activity` object for each. A `Trail` is then instantiated with the collection of activities passed as an argument to the constructor. The initial trail order reflects the order in which the activities are listed in the properties file and is not based on the context information considered by the application. It is this trail that is manipulated by the trail generation behaviour. Once a trail has been loaded from disk it is possible to generate a trail order that reflects the current context.

### 4.2.2 Extensibility

The application framework provides generic, reusable implementations of a trail and an activity. If the default trail and activity specifications are not expressive enough they can be extended through standard Java inheritance. For example, to cater for the inclusion of an application-specific activity property a new class is defined that extends `Activity`. In the case that the new property is static and known prior to runtime e.g., a property that provides further descriptive information about an activity, the new property is included in the persistent activity specification stored on disk. Consequently, the utility class that loads activity specifications from disk and creates an initial trail containing these activities must be extended. The new utility class extends `TrailRepository` and



**Figure 4.3:** High-level sequence of actions in trail generation

overrides the `loadTrail()` method so that it considers the new activity property when instantiating `Activity` objects based on activity specifications. An example of the application framework being extended in this manner is discussed in Chapter 5 during the evaluation of the reusability and extensibility of the application framework.

### 4.3 Trail Generation

The trail generation process that produces the best trail for the user based on the current context, including user preferences, is triggered by the occurrence of context events. Figure 4.3 contains a sequence diagram illustrating the interaction between the classes involved in invoking trail generation behaviour. Concrete instances of `ContextGenerator` e.g., `LocationGenerator` and `ReconfigurationTimer`, generate context events and subsequently invoke the `update()` method in `ReconfigurationEngine`, which is an observer of context events. The `update()` method, illustrated in Listing 4.2, evaluates the origin of the call and acts based on the type of context event that is received. If a location event is received (line 2), the `reconfigure()` method (discussed in Section 4.3.1) is invoked (line 3). The `reconfigure()` method checks if reconfiguration is necessary before reconfiguring the trail. If a periodic reconfiguration context event is received (line 4) the trail is reconfigured using the `forceReconfigure()` method (line 5). This method reconfigures the trail without consideration for whether it is necessary or not, ensuring that the trail

is periodically reconfigured in the absence of other context events. `TrailGeneration`, which makes the trail available to non-application framework code e.g., the user interface code, is notified when the trail is reconfigured.

---

Listing 4.2: The `update()` method in the `ReconfigurationEngine` class

---

```
1 public void update(Subject s) {
2     if (s instanceof LocationGenerator) {
3         reconfigure();
4     } else if (s instanceof ReconfigurationTimer) {
5         forceReconfigure();
6     }
7 }
```

---

### 4.3.1 The `reconfigure()` method

The `reconfigure()` method in `ReconfigurationEngine`, illustrated in Listing 4.3, contains behaviour to reconfigure a trail following notification of a context event. Figure 4.4 illustrates the interactions between the primary classes involved in the implementation of the trail generation behaviour.

---

Listing 4.3: The `reconfigure()` method in the `ReconfigurationEngine` class

---

```
1 public synchronized Trail reconfigure() {
2     setTrail(this.pruneTrail(trail));
3     setTrail(this.sortByRelevance(trail));
4
5     if (reconfigurationRequired()) {
6         this.setTrail(this.getReconfiguredTrail(trail));
7         if (currentTrail != null)
8             this.setCurrentTrail((Trail) trail.clone());
9         notifyObservers();
10    } else {
11        TrailTimeModifier.updateTrailTimes
```

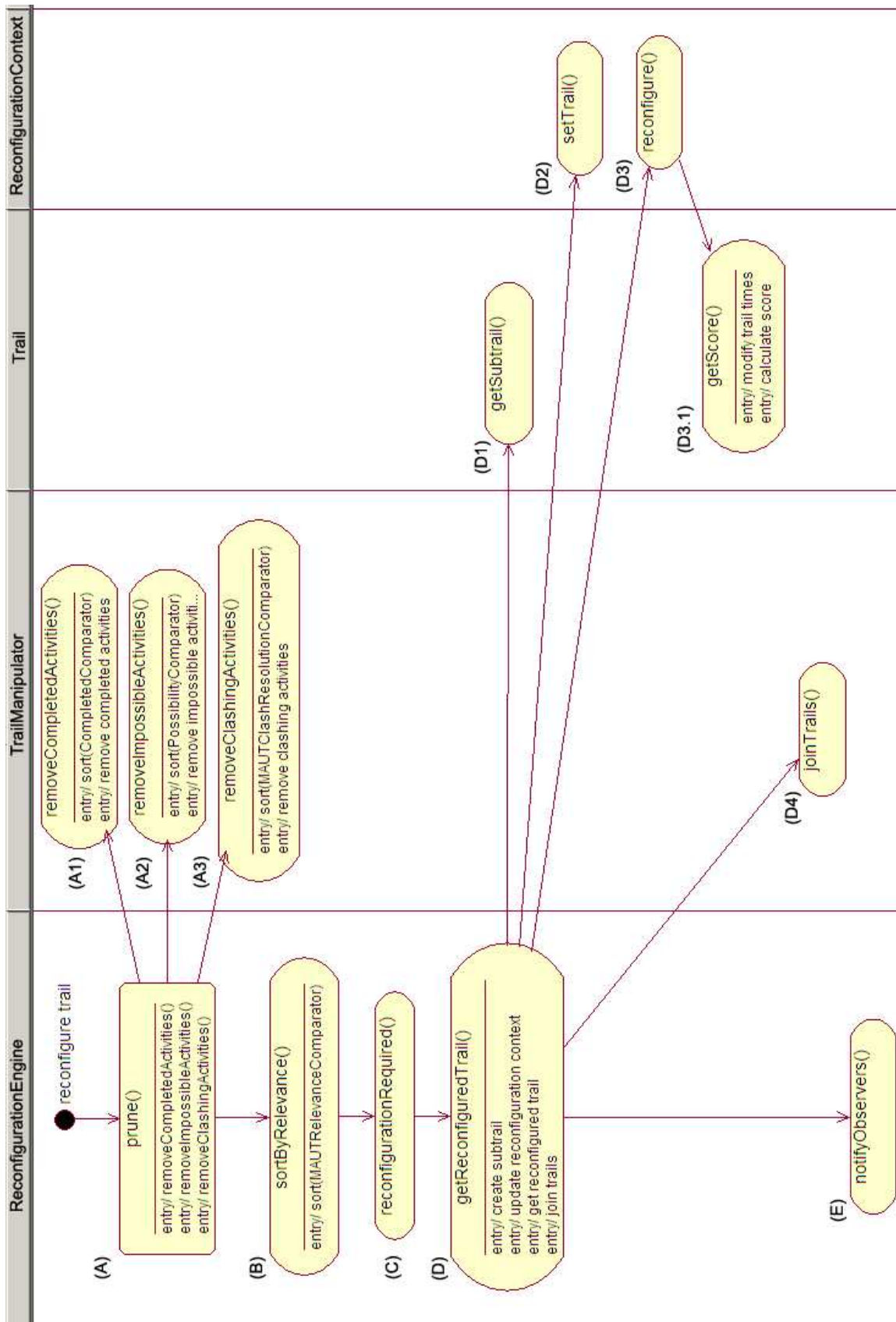


Figure 4.4: Interactions between classes in the trail generation implementation



```

12         (this.getCurrentTrail());
13         trail = (Trail) this.getCurrentTrail().clone();
14         notifyObservers();
15     }
16     return trail;
17 }

```

---

The `reconfigure()` method begins by removing unnecessary activities from the trail using the `prune()` method in `ReconfigurationEngine` (step A in Figure 4.4). `prune()` (line 2 in Listing 4.3) is responsible for pruning the trail of completed, impossible and clashing activities. Three individual methods in `TrailManipulator` implement the pruning behaviour. The method responsible for removing completed activities (`removeCompletedActivities()` at step A1) from the trail uses `CompletedComparator` to first sort the activities in the trail by whether they are completed or not. The completed activities are then removed from the trail. `removeImpossibleActivities()` (step A2) works in the same manner, using the `PossibilityComparator` to sort the activities in the trail based on whether they are currently possible or not. Activities that are not possible are removed from the trail. The `removeClashingActivities()` method (step A3) uses `MAUTClashResolutionComparator` to resolve clashes between activities that are identified as clashing. The clash resolution comparator uses `Normalize` to convert activity assessment values so that they are relative to each other, and `UserModel` is used to access the user-specified weights for comparing clashing activities. When the trail has been pruned of unnecessary and impossible activities it is sorted by relevance.

The `sortByRelevance()` method (step B) in `ReconfigurationEngine` is invoked after `prune()` returns (line 3). `MAUTRelevanceComparator` sorts activities by comparing them by how relevant they are to the user. In order to calculate the relevance value of an activity, `MAUTRelevanceComparator` calculates values for the following activity properties:

- Proximity - how near is the activity to the user.
- Priority - what is the priority of the activity.

- Obligation - is the activity mandatory or not.
- Urgency - how soon is the activity closing time.

The values for these properties are first normalised and then multiplied by the user-specified preference weights acquired from `UserModel`. The normalised activity properties are multiplied by their associated weights and added to, or subtracted from, the total activity score as appropriate. The score is increased in the case of the priority value and the obligation value, and reduced in the case of proximity and urgency as a greater distance from the user and an activity closing time further in the future makes an activity less relevant. If the score for the first activity is greater than that for the second activity then the first activity is considered more relevant. Listing 4.4 illustrates how the contribution of activity priority towards the overall activity score is calculated in the `compare()` method in `MAUTRelevanceComparator`. Lines 1 and 2 illustrate how the priority value of the first activity is retrieved, normalised and multiplied by the user-specified weight for priority. Lines 4 and 5 calculate the value for the second activity.

---

Listing 4.4: Calculating activity priority value in the `MAUTRelevanceComparator` class

---

```

1 scoreA += ( Normalize . getNormalizedValue ( actA . getPriority ( ) ,
2 Normalize . getInstance ( ) . getPriorityRange ( ) ) * priorityWeight ) ;
3
4 scoreB += ( Normalize . getNormalizedValue ( actB . getPriority ( ) ,
5 Normalize . getInstance ( ) . getPriorityRange ( ) ) * priorityWeight ) ;

```

---

When the trail has been sorted by relevance it is ready to be reconfigured if necessary. The `reconfigurationRequired()` method (step C and line 5 in Listing 4.3) in `ReconfigurationEngine` is invoked next. This method, discussed in Section 4.4, identifies whether reconfiguration is required (Figure 4.4 assumes that reconfiguration is required). If reconfiguration is required the trail is reconfigured using `getReconfiguredTrail()` (line 6 in Listing 4.3). If reconfiguration is not required then the estimated activity start and end times are updated (lines 11 and 12) based on the current time and the user's location. This behaviour is encapsulated in `TrailTimeModifier`.

The `getReconfiguredTrail()` method (step D) first assesses if a subtrail is necessary. This is achieved by comparing the number of activities in the trail against the maximum subtrail size set by the developer in a properties file. A subtrail is created if the trail is larger than the maximum subtrail size. Figure 4.4 assumes that a subtrail is necessary (step D1). The subtrail is passed to `ReconfigurationContext` (step D2) and the `reconfigure()` method (in `ReconfigurationContext`) is invoked (step D3).

The `reconfigure()` method in `ReconfigurationContext` invokes one of the concrete implementations of `TrailReconfigurationStrategy` (not shown in the diagram). These classes use exact or heuristic approaches to generate candidate trail solutions and, as illustrated in step D3.1 in Figure 4.4, they use the `getScore()` method in `Trail` to evaluate the worth of each candidate solution.

---

Listing 4.5: The evaluation function in the `Trail` class

---

```

1 public double getScore () {
2     TrailTimeModifier.updateTrailTimes (this);
3     double trailScore = 0;
4
5     double activitiesPossibleWeight = UserModel.getInstance () .
6         getScoreActivitiesPossibleWeight ();
7     ...
8     int activitiesPossible = TrailAssessor .
9         getNumberOfActivitesPossible (this);
10
11     if (!TrailAssessor.isTrailFullyImpossible (this)) {
12         trailScore += Normalize.getNormalizedValue
13             (activitiesPossible , this.getNumActivities ())
14             * activitiesPossibleWeight;
15         ...
16     } else {
17         trailScore = 0;
18     }
19     this.setScoreValue (trailScore);

```

```
20     this.setReconfigurationTime(System.currentTimeMillis());
21     return trailScore;
22 }
```

---

The `getScore()` method is the implementation of the trail evaluation function. The method first updates the trail time estimates based on the user's current location, the current time and the position of each activity in the trail. The estimated start and end time of each activity in the trail is updated by `TrailTimeModifier`. Line 2 in Listing 4.5 illustrates the invocation of this behaviour. Lines 5 and 6 illustrate the setting of the weight for the activities possible value dimension. The other value dimensions - obligation, length, duration and idle time - are not illustrated in Listing 4.5 but are also set (represented by the ellipses at line 7). Next, `TrailAssessor` is used to determine the number of activities on the trail that are possible. If the candidate trail has any activities possible then the value for each value dimension, normalised and multiplied by the user-specified weight for the dimension, is added or subtracted to/from the `trailScore` attribute as appropriate. Lines 12-14 illustrate the addition of the value for the activities possible value dimension to the `trailScore` attribute. When all value dimensions have been assessed, the score attribute of the trail is set to be equal to the trail score just calculated and the time that the reconfiguration occurred at is recorded (shown on lines 19-20). The trail score is then returned to the client.

When the best trail has been returned to the `getReconfiguredTrail()` method in `ReconfigurationEngine`, `TrailManipulator` is used to join the scheduled activities with those not considered relevant enough for inclusion in the set of activities ordered by the trail reconfiguration strategy (step D4). The subtrail is joined with the activities of lower relevance and a trail consisting of scheduled and unscheduled activities is returned. Any observers of trail reconfiguration, in this case `TrailGeneration`, are notified (step E and line 9 in Listing 4.3).

### 4.3.2 Extensibility

The generic trail generation behaviour in the application framework has been designed to facilitate extension through the specification of application-specific behaviour. The

default trail generation implementation is extensible in the following ways:

- Implementation of new strategies for generating candidate trail solutions.
- Implementation of new context sources.
- Extension/redefinition of the trail evaluation function.
- Extension/redefinition of the activity comparators.

The Strategy pattern is used to facilitate the addition of trail reconfiguration strategies to the application framework. This is achieved by defining new behaviour that extends the parent class of the trail reconfiguration strategies that are provided by default (brute force, genetic algorithm and simulated annealing). `TrailReconfigurationStrategy` is an abstract class that contains one method - `reconfigure()`. This method accepts a `ReconfigurationContext` object as a parameter and returns a reconfigured trail. The `reconfigure()` method must be implemented by all concrete trail reconfiguration strategies. How this method is implemented does not concern the client, as long as a reconfigured trail is returned. This facilitates the creation of many trail reconfiguration strategies that take different approaches (e.g., exact, heuristic) to finding the best trail for the user based on the current context. Extensions to the framework in this manner are hidden from `ReconfigurationEngine`, requiring only `ReconfigurationContext` to be extended so that it makes the new trail reconfiguration strategy available for use during reconfiguration.

The application framework can also be extended to consider new types of context during reconfiguration by defining new context sources that extend `ContextGenerator`. `ContextGenerator` extends `Subject`, a class that provides methods for adding, removing and notifying observers of events in subjects. The `Observer` interface provides an `update()` method that is called in concrete implementations of the `notifyObservers()` method in subclasses of `Subject` i.e., `ContextGenerator`. When a subject changes state, all its observer objects are notified and updated automatically. Once a new context source has been defined, `ReconfigurationEngine` must be extended to take the new information into account. The constructor is extended to add `ReconfigurationEngine` as an observer of the new context source. The `update()` method in `ReconfigurationEngine`,

invoked by context generators following context events, is overridden with an implementation that takes the appropriate behaviour based on the type/value of the received context event. In the general case, the appropriate behaviour will involve the invocation of `reconfigure()` as illustrated on line 3 of Listing 4.2, where `reconfigure()` is invoked following the receipt of a location context event.

The trail evaluation function can be extended/redefined by implementing a new trail class that extends `Trail` and overrides the `getScore()` method. The evaluation function is typically extended when a new activity attribute and associated context source are implemented. The evaluation function is extended to consider the value of the new activity attribute during trail evaluation.

In terms of activity comparison techniques, the framework can be extended by either defining a new class that implements the `java.util.Comparator` interface or by extending the appropriate existing comparator class e.g., `MAUTRelevanceComparator` and overriding the sole method of that class, `compare()`. In the case that the application framework is extended through the addition of a new context type, the relevant comparator classes should, if appropriate, be extended so that their `compare()` methods consider the new context value when comparing activities e.g., if the new context type impacts on activity relevance.

## 4.4 Reconfiguration Point Identification

Reconfiguration point identification is implemented in `ReconfigurationEngine` by the `reconfigurationRequired()` method. The Boolean value returned by this method indicates whether or not the trail should be reconfigured following the receipt of a context event. As illustrated at step C in Figure 4.4 and line 5 in Listing 4.3, the reconfiguration point identification behaviour is invoked as part of the trail generation behaviour.

The first part of the `reconfigurationRequired()` method is illustrated in Listing 4.6. This behaviour assesses changes in the membership of the relevant set following receipt of a context event. Line 7 caters for the first trail reconfiguration, a situation in which the user's current trail is null and therefore reconfiguration of the initial trail read in from disk is always required. The `lastRelevanceSort` attribute (that represents the

current trail sorted by relevance) is assigned a value and `true`, indicating reconfiguration is required, is returned. Lines 11 and 12 check if the trail recently sorted by relevance and the current trail being followed by the user have the same amount of activities. If the number of activities in both trails is not equal e.g., if an activity has become impossible due to time constraints, then reconfiguration is required and `true` is returned. If the trails do contain the same amount of activities then their contents must be compared. Lines 17-23 calculate how many activities need to be compared (the maximum being the subtrail size) and lines 25-30 compare the contents of the current trail and the trail sorted by relevance. If the content differs i.e., if one trail has an activity that the other doesn't, then reconfiguration is required.

---

Listing 4.6: Excerpt from the `reconfigurationRequired()` method

---

```
1 private boolean reconfigurationRequired () {
2     boolean reconfigRequired = false;
3     // the trail sorted by relevance
4     Trail trail = this.getTrail();
5     Trail current = this.getCurrentTrail();
6
7     if (current == null) {
8         lastRelevanceSort = (Trail) this.getTrail().clone();
9         return true;
10    }
11    else if (trail.getNumActivities () !=
12            current.getNumActivities ()) {
13        lastRelevanceSort = (Trail) this.getTrail().clone();
14        return true;
15    }
16
17    int numActivities = 0;
18    if (trail.getNumActivities () > subTrailSize) {
19        numActivities = subTrailSize;
20    }
```

```

21     else {
22         numActivities = trail.getNumActivities();
23     }
24
25     for (int i = 0; i < numActivities; i++) {
26         Activity activityB = current.getActivities()[i];
27         if (!trail.contains(activityB)) {
28             reconfigRequired = true;
29         }
30     }
31     ...

```

---

The second part of the `reconfigurationRequired()` method, illustrated in Listing 4.7 (which follows on from Listing 4.6), caters for the situation in which the contents of the two sets of activities being compared are equal. Line 4 checks if the value returned by assessing the correlation between the ordering of the trail sorted by relevance and the last time the trail was sorted by relevance (which is the current trail sorted by relevance) is less than the variance threshold set in the properties file (shown in Listing 4.9 in Section 4.5). If so, then reconfiguration is required (line 6).

---

Listing 4.7: 2nd excerpt from the `reconfigurationRequired()` method

---

```

1 ...
2 if (!reconfigRequired) {
3     if (!trail.equals(lastRelevanceSort)) {
4         if (KendallsT.getKendallValueForReconfig(lastRelevanceSort,
5             trail) < relevantSetVarianceThreshold) {
6             reconfigRequired = true;
7         }
8     }
9 }
10 lastRelevanceSort = (Trail) this.getTrail().clone();
11 return reconfigRequired;

```

---



`KendallsT` is used to produce the correlation value ( $\tau$ ). This class contains a method called `getKendallValueForReconfig()`, illustrated in Listing 4.8. This method implements Kendall's  $\tau$  rank correlation coefficient as described in Section 3.2.4.2. The method first creates an `int` array that represents the positions of the activities in the activity set sorted by relevance (the second argument) relative to the ordering of the activities in the current trail sorted by relevance. This process is illustrated by the following example. An application considers five activities - Act #1...Act #5. Trail 1 is the existing trail sorted by relevance and Trail 2 is the activity set sorted by relevance following the receipt of a context event. An ordering is produced for the activities in Trail 2 based on their positions relative to the ordering in Trail 1:

Trail 1: Act #4, Act #5, Act #1, Act #3, Act #2  $\rightarrow$  1, 2, 3, 4, 5

Trail 2: Act #3, Act #5, Act #4, Act #1, Act #2  $\rightarrow$  4, 2, 1, 3, 5

This behaviour is implemented between lines 4-9. With the relative ordering in place, the  $P$  value is calculated (lines 11-20). The  $P$  value is used to complete the calculation of the  $\tau$  value. This behaviour is contained on lines 22-24. The  $\tau$  value is returned and compared to the reconfiguration threshold value in the `ReconfigurationEngine` class where a decision is made regarding whether reconfiguration is required or not.

---

Listing 4.8: The `getKendallValueForReconfig()` method

---

```

1 public static double getKendallValueForReconfig(Trail
2     existingTrail, Trail newlyGenerated) {
3     double kendall = 0.0;
4     int [] activities = new int[newlyGenerated.getNumActivities()];
5     for (int i = 0; i < newlyGenerated.getNumActivities(); i++) {
6         Activity activity = newlyGenerated.getActivities()[i];
7         int index = getPosition(existingTrail, activity.getIndex());
8         activities[index] = i;
9     }
10

```

```

11     int p = 0;
12     for (int i = 0; i < activities.length; i++) {
13         int index = activities[i];
14         for (int j = i + 1; j < activities.length; j++) {
15             int anotherIndex = activities[j];
16             if (index < anotherIndex) {
17                 p++;
18             }
19         }
20     }
21
22     kendall = (((p * 2) / ((new Double(existingTrail.
23         getNumActivities()).doubleValue() / 2) * (existingTrail.
24         getNumActivities() - 1))) - 1);
25     return kendall;
26 }

```

---

#### 4.4.1 Extensibility

The reconfiguration point identification mechanism is based on measuring differences between two sets of activities sorted by relevance. Therefore, extending or redefining the behaviour of the mechanism involves modifying the code that calculates either the relevance of an activity or the difference between two sets of activities that contain the same activities.

Extending or redefining the measure of activity relevance involves creating a new comparator class that implements `java.util.Comparator` and implements the `compare()` method<sup>7</sup>. By default, relevance is calculated based on user specified weights for four activity properties (proximity, priority, obligation and urgency). If a new property is added, a value dimension weight must be specified for the new property in the properties file (discussed in Section 4.5) that stores user preferences for activity comparison. As a result, `UserModel` must also be extended to take the new user context into account.

---

<sup>7</sup>`MAUTRelevanceComparator` can also be extended. In this case `compare()` must be overridden.

The behaviour of the default relevance comparator class can also be modified without extension by setting the value dimension weights in the relevant properties file.

The implementation of Kendall's rank correlation coefficient can be extended by specifying a new class derived from `KendallsT` and overriding `getKendallValueForReconfig()`. However, due to the lack of obvious advantages associated with using alternative techniques such as Spearman's  $\rho$  [121] (as discussed in Section 3.2.4.2) it is not envisaged that developers will be interested in extending the activity comparison behaviour. It is more likely that the threshold value that determines whether a significant difference exists between two equal (in terms of membership) sets of activities will need to be modified. The value of the threshold can be modified external to the application framework source code in a properties file. The configuration files used by the application framework source code are discussed in the next section.

## 4.5 Configuration files

The `java.util.Properties` class loads and stores key/value pairs from a file and manages them in memory, thereby facilitating the use of persistent application variables. The application framework uses properties files to store application properties that are used to customise the behaviour of the application framework without requiring source code modification. Properties files are used by the following classes:

- `UserModel` - reads user preferences (MAUT value dimension weights) from disk.
- `ReconfigurationEngine` - reads the relevant set size and the  $\tau$  threshold for reconfiguration point identification (discussed in Section 4.4).
- `TrailRepository` - reads the activity definitions when creating the initial trail at application start time.
- `ReconfigurationTimer` - reads the periodic reconfiguration interval.
- `ReconfigurationContext` - reads which concrete `TrailReconfigurationStrategy` class to use.

- **Normalization** - reads trail and activity ranges used for producing normalised values.
- **GeneticAlgorithmStrategy** - reads genetic algorithm algorithm parameters.
- **SimulatedAnnealingStrategy** - reads simulated annealing algorithm parameters.

The `trail.properties` file, populated with sample data, is illustrated in Listing 4.9. Line 3 contains the property that sets the value for the `subTrailSize` attribute in `ReconfigurationEngine`. Line 5 contains the property (in milliseconds) read by the `ReconfigurationTimer` class to determine the time intervals between periodic reconfiguration. Lines 7-9 contain three properties, only one of which is active at any one time. These properties specify the concrete trail reconfiguration strategy that is used to find the best trail for the user. This property is read by the `ReconfigurationContext` class. Line 15 specifies the number of activities that should be read from the list of activity specifications that follow below (previously discussed in relation to Listing 4.1). This property is read by the `TrailRepository` class, as are the activity specifications.

---

Listing 4.9: The `trail.properties` file

---

```

1 # TRAIL GENERATION PROPERTIES
2 #####
3 trail.generation.subtrail=5
4
5 trail.reconfiguration.timeInterval=10000
6
7 trail.reconfiguration.strategy=brute
8 #trail.reconfiguration.strategy=genetic
9 #trail.reconfiguration.strategy=annealing
10
11 trail.reconfiguration.relevantSet.varianceThreshold=0.8
12
13 # ACTIVITY SPECIFICATION
14 #####

```

```

15 activities.number=7
16
17 activity.1.description=Attend Lecture 1
18 activity.1.x=435
19 activity.1.y=365
20 activity.1.openingTime=09:30
21 activity.1.closingTime=10:30
22 activity.1.closingTime.leeway=0
23 activity.1.breakStart=0
24 activity.1.breakEnd=0
25 activity.1.duration=60
26 activity.1.duration.leeway=0
27 activity.1.priority=5
28 activity.1.mandatory=false
29
30 ...

```

---

The `userPreferences.properties` file is illustrated in Listing 4.10. Lines 8-11 contain the user-specified weights for each of the value dimensions considered during activity relevance determination. Lines 17-19 contain the weights for clash resolution and lines 25-29 contain the weights used in the trail evaluation function. All properties in the `userPreferences.properties` file are read by the `UserModel` class that makes them available to the `MAUTRelevanceComparator`, `MAUTClashResolutionComparator` and `Trail` classes.

---

Listing 4.10: The `userPreferences.properties` file

---

```

1 # USER PREFERENCE PROPERTIES
2 #####
3
4 ##### RELEVANCE PREFERENCES #####
5 # weights for sorting activities by relevance
6 # values should add up to 1

```

```

7
8 relevance . proximity . weight = 0.5
9 relevance . priority . weight = 0.1
10 relevance . mandatory . weight = 0.1
11 relevance . urgency . weight = 0.3
12
13 ##### CLASH RESOLUTION #####
14 # weights to use when resolving activity clashes
15 # preference values should add up to 1
16
17 clash . proximity . weight = 0.3
18 clash . priority . weight = 0.2
19 clash . mandatory . weight = 0.5
20
21 ##### TRAIL GENERATION/RECONFIGURATION #####
22 # weight for trail scoring (trail generation/reconfig)
23 # preference values should add up to 1
24
25 score . activitiesPossible . weight = 0.6
26 score . mandatorySupported . weight = 0.00
27 score . length . weight = 0.2
28 score . duration . weight = 0.00
29 score . idleTime . weight = 0.2

```

---

The `normalization.properties` file contains the upper limits for the trail and activity properties that are normalised during activity and trail comparison and evaluation. Listing 4.11 contains a `normalization.properties` file with sample contents. Line 6 contains the value for the `proximityRange` property, meaning that any activity that is 1000 metres or more from the user will have the maximum value of 100 when normalised. Line 9 contains the value for the length of a trail. Line 12 contains the value that specifies what constitutes the highest priority value. The value considered to be the maximum trail duration is specified on line 15 and line 18 specifies the daily time range, which in

this case is 24 hours (86400000 milliseconds).

---

Listing 4.11: The `normalization.properties` file

---

```
1 # TRAIL NORMALIZATION PROPERTIES
2 #####
3
4 #the distance range for proximity i.e., zero to the value
5 #specified below – in metres.
6 trail.normalize.proximityRange=1000
7
8 #the length range for a whole trail – in metres.
9 trail.normalize.lengthRange=100000
10
11 #the range for priority – between 1 and 5, where 5 is the highest
12 trail.normalize.priorityRange=5
13
14 #the range for trail duration – in minutes
15 trail.normalize.durationRange=600
16
17 #the range of time within a day in milliseconds
18 trail.normalize.dailyTimeRange=86400000
```

---

### 4.5.1 Extensibility

All of the properties files in the application framework are extensible by augmenting the existing files with new property specifications (no notion of extension via inheritance exists in relation to properties files). For example, defining a new property to represent the weight of a new value dimension for activity relevance calculation involves adding the new property to the `userPreferences.properties` file using the same notation as the existing properties. `UserModel`, the class that makes user preferences available to the behaviour that uses MAUT to evaluate objects, must be extended to take the new property into account i.e., it must read the property from disk and provide accessor methods. The

properties files that express activity specifications and application/algorithm properties can all be extended in the same manner as the user preferences properties file.

## 4.6 Chapter Summary

This chapter has described the implementation of the trail generation and reconfiguration point identification techniques described in Chapter 3. The application framework provides a generic trail and activity specification that facilitates reuse while also supporting extension. The approaches to trail generation and reconfiguration are implemented in a similar manner and can be reused as is or can be extended through the development of new candidate solution generation techniques, a new evaluation function, new context sources and new activity comparison measures. The behaviour of the evaluation function, candidate solution generation techniques, reconfiguration point identification technique and the activity comparison techniques can be customised external to the source code through the use of configuration files.

The following chapter evaluates the application framework through the development of three case study applications that illustrate how it can be reused and extended. The chapter also describes the evaluation of the responsiveness of the trail generation implementation and the accuracy of the reconfiguration point identification implementation. Details of an experiment concerning human satisfaction with trails generated by the framework are also discussed.



# Chapter 5

## Evaluation

The previous chapter describes how the approaches to trail generation and reconfiguration point identification proposed in Chapter 3 are implemented to realise the application framework. This chapter discusses its evaluation in relation to the stated requirements in Section 3.1.1, which was conducted with three objectives:

1. To determine the extent to which the application framework can be reused and extended to facilitate the development of a range of mobile, context-aware trails-based applications.
2. To quantify both the responsiveness of the trail generation implementation and the accuracy of the reconfiguration point identification implementation.
3. To evaluate human opinion on the quality of the trails generated by the application framework.

Section 5.1 discusses how the first evaluation objective was achieved through the examination of a number of case study applications built using the application framework. Section 5.2 discusses how the second evaluation objective was achieved by means of lab experiments. Section 5.3 discusses the study conducted in order to meet the third evaluation objective. The chapter concludes in Section 5.4 with a summary of the evaluation findings.

## 5.1 Framework Reusability and Extensibility

Object-oriented application frameworks are intended to reduce the cost and improve the quality of software by making reusable software available to developers [40]. Accordingly, application frameworks consist of ready-to-use and partially complete classes that compose an overall application architecture that specifies the composition and interaction of classes. The production of concrete applications using an application framework typically involves customising existing behaviour by overriding methods in newly created subclasses, hence saving developer time and ensuring adherence to a proven design [100]. Therefore, in order to be considered useful, an application framework must be capable of serving as the basis to a range of applications within a specific domain i.e., it must be reusable. In addition, a useful application framework must also be capable of supporting the specification of behaviour that it does not cater for by default i.e., it must be extensible.

Although proven techniques for evaluating specific applications exist, techniques for designing and evaluating the infrastructure intended to aid the development of these applications are much less well formed [39]. For example, an evaluation framework for mobile, context-aware applications has been developed by Scholtz and Consolvo [114]. However, the evaluation areas, which include attention, adoption and appeal, are only applicable to reasoning about applications as opposed to the infrastructure used to develop them. According to Edwards et. al [39] and Johnson [61], user-centered infrastructure design demands applications to demonstrate the power of the infrastructure. That is, in order to illustrate the capabilities of an application framework it is necessary to develop applications that use the framework as their basis. By developing and studying applications developed with a software framework it is possible to illustrate the capabilities of the framework in terms of reusability and extensibility.

A case study is an in-depth examination of a single instance or event [41] and can provide sufficient information to help judge if specific computing technologies are of benefit when applied in a specific domain [67]. Case studies can be used to illustrate the capabilities of an application framework by considering each application in a set of example applications as an individual case study, and evaluating the degree of reuse and the ease

of extensibility afforded by the application framework in each case. The consideration of multiple case studies serves to evaluate the suitability of an application framework for developing a range of applications within a specific domain. Of the projects reviewed in Chapter 2, all but one<sup>1</sup> of the mobile, context-awareness application frameworks discussed in Section 2.3 used application case studies as a means of framework evaluation, indicating that the use of case studies is an accepted form of framework evaluation and that framework evaluation is generally of a qualitative nature and rather than a quantitative one.

The use of a single case study for conducting an evaluation of a software method or tool has been criticised because single case study results are difficult to generalise [67]. It is preferable to conduct multiple case studies or a survey of a large group of development projects using a software framework (although such surveys can be prohibitively expensive [67]). However, Flyvbjerg has illustrated that it is a common misconception that it is not possible to generalise from a single case. He states that it is often possible to generalise on the basis of a single case, and that the case study may be “central to scientific development via generalisation as supplement or alternative to other methods” [41]. Regardless of this argument, the application framework evaluation contained in this section errs on the side of caution and discusses three application case studies that illustrate how the application framework can be reused and extended to implement applications with different requirements. All three applications reuse the base framework behaviour, with two of the applications requiring specific framework extensions. The case study applications were selected to illustrate how the application framework can be used by developers to implement applications of varying complexity. The first case study illustrates how the application framework can be reused without extension to develop a basic trails application. Application specifics are defined in the framework’s properties files and no source code extensions are necessary. The second case study explores the development of an application that requires framework extensions to support a context source and activity attribute that are not supported by default in the application framework. The third case study investigates the implementation of an application with requirements that

---

<sup>1</sup>Stick-e Note [18] is a conceptual framework and was not implemented, precluding evaluation by application development case study.

necessitate extensions to fundamental framework behaviour such as the trail evaluation function and the activity relevance measurement mechanism. The three case studies are discussed throughout the remainder of this section.

### 5.1.1 Day Planner

The easiest way to use a framework is to use existing classes only i.e., without implementing any concrete subclasses [61]. The first case study explores how the application framework can be used to develop a day planner application. Its key characteristic is that it represents the set of trails applications that can be built using the application framework without extension (e.g., tourist guide applications like those discussed in Section 2.1 or a campus activity planner like the Oisín goes to Trinity application discussed in Section 3.1.3). The day planner application is required to help the user undertake a set of activities by generating and managing a trail based on the properties of the activities specified, the user's location, the current time and the user's preferences. The default behaviour in the application framework can be reused, without extension, to implement the day planner application. The developer is only required to provide application-specific information, such as the details of each activity, via the properties files.

#### 5.1.1.1 Implementation

In order to implement the day planner application using the application framework the developer is required to edit the `trail.properties` file (introduced in Section 4.5) to specify activity details and configure the following application properties:

- The maximum number of activities that can be scheduled during trail generation i.e., the subtrail/relevant set size.
- The periodic reconfiguration time interval.
- The  $\tau$  threshold for reconfiguration point identification i.e., the minimum level of similarity that must exist between the activities in the current trail and the activity set sorted by relevance following a context event in order for reconfiguration to be deemed unnecessary.

Listing 5.1 contains an excerpt from the `trail.properties` file for the day planner application. A subtrail size of 5 is set on line 1, specifying that the user's five most relevant activities will be selected for scheduling at times when the user has five or more activities to complete. The periodic reconfiguration interval is set on line 2 to be five minutes (30000 milliseconds) so that, in the absence of location change events, the user's trail will be reconfigured automatically every five minutes. The brute force strategy for trail generation is selected on line 3, meaning that the user's trail will be managed through the process of exhaustively evaluating all activity permutations each time reconfiguration is deemed necessary. The  $\tau$  value of 0.8 (line 4) defines the effect a location or time lapse context event must have on the correspondence between the user's trail and the activity set (sorted by relevance following the context event) in order for reconfiguration to be deemed necessary. Line 6 specifies the number of activities to be included in the application and the first activity specification is contained between lines 6-17, with the remainder following below in the same fashion (as indicated by the ellipses on line 20).

---

Listing 5.1: Excerpt from the day planner `trail.properties` file

---

```
1 trail.generation.subtrail=5
2 reconfiguration.timeInterval=30000
3 trail.reconfiguration.strategy=brute
4 relevantSet.varianceThreshold=0.8
5
6 activities.number=7
7
8 activity.1.description=Tennis w/ John
9 activity.1.x=435
10 activity.1.y=365
11 activity.1.openingTime=09:30
12 activity.1.closingTime=10:30
13 activity.1.closingTime.leeway=0
14 activity.1.breakStart=0
15 activity.1.breakEnd=0
16 activity.1.duration=60
```

```

17 activity .1.duration.leeway=0
18 activity .1.priority=5
19 activity .1.mandatory=true
20 ...

```

---

The developer can also edit user preferences, although defaults can be used. The contents of the `userPreferences.properties` file (introduced in Section 4.5) for the day planner application are illustrated in Listing 5.2. The user preferences for determining activity relevance are specified between lines 9 and 12. The preferences for clash resolution are contained between lines 19 and 21 and the trail evaluation function preferences are specified on lines 28-32. The modification of these values affects which activities are selected for scheduling and the ordering of the user's trail.

---

Listing 5.2: The `userPreferences.properties` file

---

```

1 # USER PREFERENCE PROPERTIES
2 #####
3
4 ##### RELEVANCE PREFERENCES #####
5
6 # preferences to use when sorting activities by relevance
7 # preference values should add up to 1
8
9 relevance .proximity.weight=0.5
10 relevance .priority.weight=0.1
11 relevance .mandatory.weight=0.1
12 relevance .urgency.weight=0.3
13
14 ##### CLASH RESOLUTION #####
15
16 # preferences to use when resolving activity clashes
17 # preference values should add up to 1
18

```

```

19 clash.proximity.weight=0.3
20 clash.priority.weight=0.2
21 clash.mandatory.weight=0.5
22
23 ##### TRAIL GENERATION/RECONFIGURATION #####
24
25 # preferences for trail generation and reconfiguration
26 # preference values should add up to 1
27
28 score.activitiesPossible.weight=0.7
29 score.mandatorySupported.weight=0.00
30 score.length.weight=0.3
31 score.duration.weight=0.00
32 score.idleTime.weight=0.00

```

---

The properties in the `normalization.properties` file (introduced in Section 4.5) can also be set. Listing 5.3 illustrates the values of the trail and activity ranges used for normalisation in the day planner application.

---

Listing 5.3: The `normalization.properties` file

---

```

1 # TRAIL PROPERTY NORMALIZATION PROPERTIES
2 #####
3
4 #the distance range for proximity i.e., zero to the value
5 #specified below – in metres.
6 trail.normalize.proximityRange=1000
7
8 #the length range for a whole trail – in metres.
9 trail.normalize.lengthRange=3200
10
11 #the range for priority – between 1 and 5, where 5 is the highest
12 trail.normalize.priorityRange=5
13

```

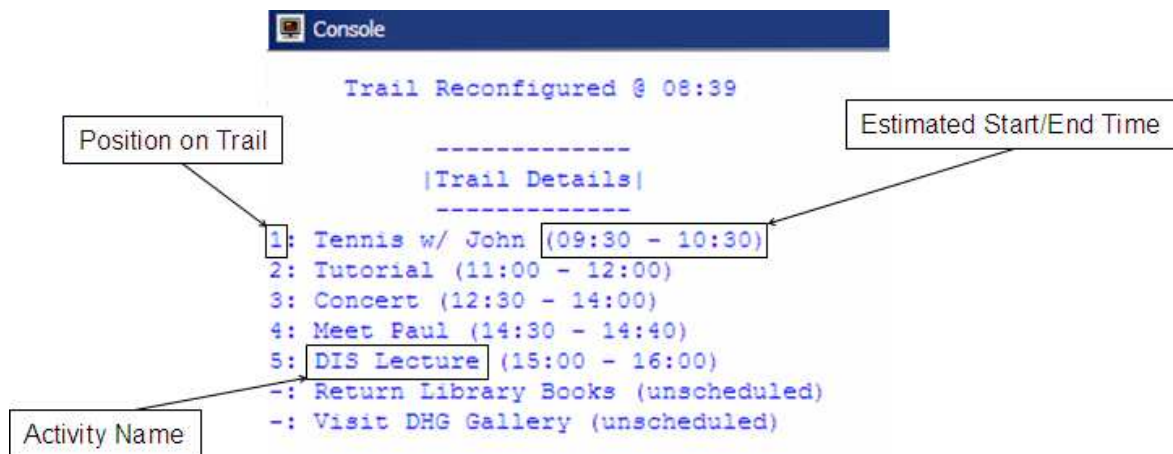
```

14 #the range for trail duration – in minutes
15 trail.normalize.durationRange=600
16
17 #the range of time within a day in milliseconds
18 trail.normalize.dailyTimeRange=86400000

```

---

Figure 5.1 shows an annotated screen shot of the default text-based user interface that is produced by the application framework for the day planner application. A message is displayed each time that the trail is reconfigured that includes the time that the reconfiguration occurred. Information about the user’s trail is displayed below and includes the following details:



**Figure 5.1:** The text-based display produced for the day planner application

- The position of the activity on the trail. The activity either has a number associated with it to indicate its position or a ‘-’ character to indicate that it is currently unscheduled.
- The activity name. This information is displayed regardless of whether the activity is scheduled or not.
- The estimated start and end time of each activity given its position on the trail. If an activity is currently unscheduled then the word ‘unscheduled’ is displayed in place of the time information.



### 5.1.1.2 Analysis

Reuse level is the standard metric for measuring the amount of software reuse in an application [99] and is generally expressed as a percent of the total source lines for the application. The metric measures the ratio of external items to total items used in an application, where external items are those whose implementation is necessitated due to lack of infrastructure support.

Given concrete definitions of a mobile user's set of activities and specific values for trail generation, user preference and trail property normalisation properties, 100% of the application framework's code base can be reused without extension to provide trail management behaviour for the day planner application. In order to use the application framework in this manner, software developers are not required to fully learn the framework. This is an advantage to developers as learning an application framework is more difficult than learning, for example, a regular class library. Class libraries can be learned one class at a time whereas frameworks, the classes of which are designed to work together, must be learned all at once [61]. The developer of the day planner and similar applications is required to understand only the meaning of each property in the properties files and how the value of each property affects the behaviour (in terms of trails produced) of the application framework. The manner in which in the behaviour that produces the results is implemented does not need to be understood as no subclassing is necessary.

Figure 5.1 illustrates the text-based user interface that is produced by default when the application framework is used to implement an application. The provision of graphical interfaces does not fall within the scope of the framework discussed in this thesis. However, the Hermes framework (discussed in Section 1.3), of which the application framework described in this thesis is a component, is investigating map-based user interfaces for trails application. Therefore, support for trails-based applications with graphical user interfaces will be available to developers using the completed Hermes framework. Alternatively, developers can implement their own application-specific graphical interfaces based on the trail information produced by the application framework.

### 5.1.2 Music Festival Trail

Not all trails applications can be implemented in the same manner as the day planner application i.e., by reusing the application framework without extension. Another way to use an application framework is to define new concrete subclasses of framework classes and use them to implement an application [61]. The second case study explores how the application framework can be extended to cater for applications that require the following behaviour:

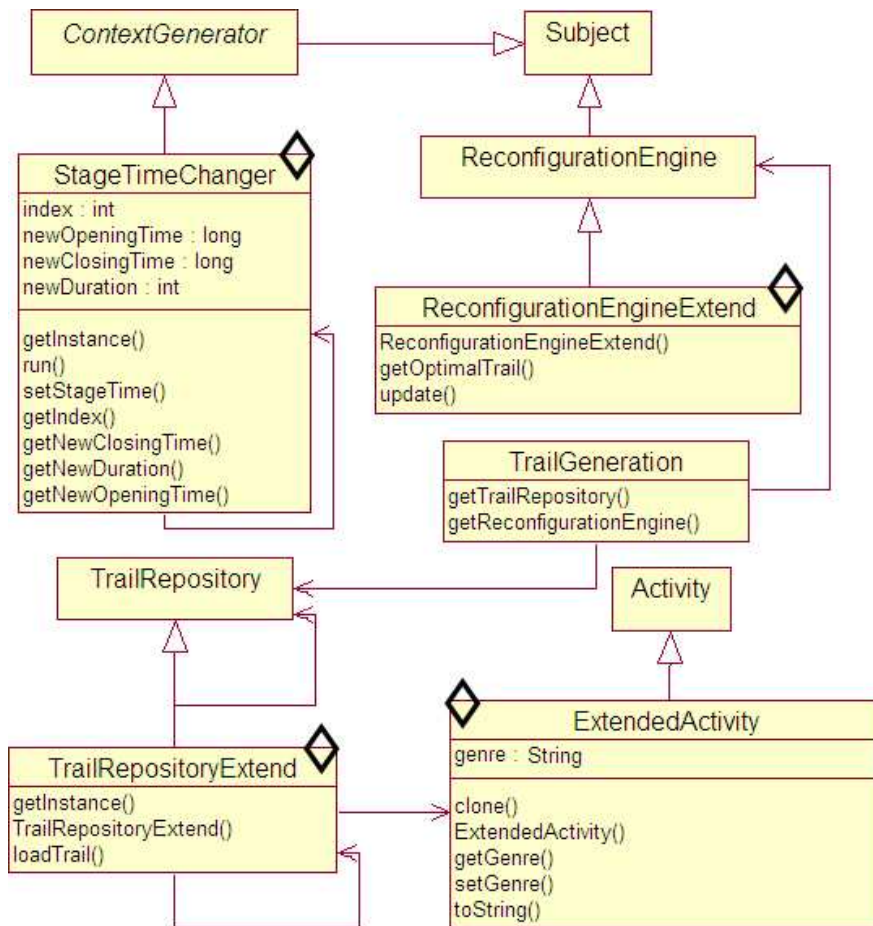
- The use of a context type not provided by the application framework.
- The use of activities with an attribute not provided by the application framework.

These two framework extensions are not interdependent and therefore the case study also serves to illustrate how applications that only require one of these extensions can be implemented.

Music festivals present numerous musical performances, usually related by genre or theme, across multiple stages. Music fans must therefore choose which performances to see on which stages, inevitably having to resolve clashes between favourite artists. Fans generally make a plan from the published running order, but scheduled stage times are often deviated from. This means that music fans are left waiting for an artist to appear while missing an artist they would have liked to see on another stage. The music festival application is required to manage a user's schedule of selected musical performances by generating and reconfiguring trails based on dynamic stage time information, the user's location, the current time and their preferences. In order to implement a trails application for music festival goes the application framework must be extended by adding a stage time context generator and related logic. A new activity attribute, genre, is also added to aid activity descriptions.

#### 5.1.2.1 Implementation

Figure 5.2 illustrates the classes (and their respective parent classes) that are added to the application framework in order to implement the music festival application. The application-specific classes are annotated with diamond shapes to distinguish them from framework classes.



**Figure 5.2:** The framework extensions facilitating the music festival application

### 5.1.2.2 Adding a new context source

The new stage time change context source is catered for through the addition of a class that extends the generic context source in the application framework. `StageTimeChanger` is responsible for acquiring and sending stage time context events to the class coordinating trail reconfiguration (a child of `ReconfigurationEngine`), triggering trail reconfiguration if necessary. `StageTimeChanger` extends the abstract `ContextGenerator` class in the same manner as the application framework’s default context sources, location and time lapse (as discussed in Section 4.1), and includes code to acquire stage time context information from a remote context generation server.

`ReconfigurationEngine`, the class responsible for receiving context events and consequently invoking trail reconfiguration, is extended to take the new context source into account. The constructor in `ReconfigurationEngineExtended` adds itself as an ob-

server of context events generated by the new stage time context source. The `update()` method, responsible for taking action based on context events, is overridden so that stage time context changes are handled i.e., the `reconfigure()` method is invoked and trail reconfiguration occurs as necessary.

### 5.1.2.3 Adding a new activity attribute

A new attribute, `genre`, is added to the default activity description so that musical performances can be better described<sup>2</sup>. This necessitates the extension of the standard activity specification. `Activity` is extended by adding the `genre` attribute and extending the constructor, `toString()` and `clone()` methods so that they include the new attribute in their behaviour. Accessor methods for the new attribute are also added.

As a result of the introduction of a new activity attribute, the activity specification stored in the `trail.properties` file (illustrated in its default form in Listing 4.9) is extended. Consequently, the behaviour that reads in activity specifications from disk and creates the initial trail must be modified to read the new activity attribute as well as the default ones. `TrailRepository` is extended, producing `TrailRepositoryExtend` that overrides the `loadTrail()` method from the parent class. The new method creates an instance of the extended activity for each activity listed in the `trail.properties` file and returns a trail containing these activities.

### 5.1.2.4 Using the new behaviour

`TrailGeneration` is responsible for both invoking the application framework's trail management behaviour and for making the trail available to the user interface and any other application logic that developers wish to implement. `TrailGeneration` invokes the behaviour that loads an initial trail from disk, starts the context service and instantiates the class that coordinates trail reconfiguration based on context change. The `getTrailRepository()` and `getReconfigurationEngine()` methods are redefined so that the framework uses the extended versions of the classes referenced by these methods. The `startContextService()` method is augmented to initialise the stage time context source.

---

<sup>2</sup>A band's name tends to reveal little about the type of music they perform.

Finally, as with the day planner application described in Section 5.1.1, user preferences and application properties can be set in the relevant properties files.

#### 5.1.2.5 User Interface



```
Console

Trail Reconfigured @ 16:06

-----
|Trail Details|
-----
1: Outkast - Main Stage (17:00 - 18:00)
2: Buck 65 - Rising Stage (19:00 - 19:45)
3: Delorentos - Main Stage (19:48 - 20:29)
4: Berkeley - New Stage (20:33 - 21:14)
5: Bill Coleman - Treatment Zone (21:17 - 22:02)

Completed Activities
-----
Dr. Octagon - Main Stage
Joanna Newsom - Acoustic Tent
```

**Figure 5.3:** The text-based display produced for the music festival application

Figure 5.3 contains a screen shot of the default text-based user interface produced by the application framework, without extension, for the music festival application. The user interface contains the same type of information about the trail as is illustrated in the annotated day planner user interface in Figure 5.1. The screen shot of the music festival interface also contains information regarding the activities that the user has already completed. Similar information was not shown in relation to the day planner application as the user had not completed any activities.

#### 5.1.2.6 Analysis

Through the addition of a source of stage time context and the extension of the activity specification, the application framework is equipped to support the management of a set of disparately located musical performances. The application is responsive to changes in user location, performance stage times and current time. 483 lines of code spread across

4 classes were added to the 5776 lines of code in the base application framework. This represents a code reuse percentage of 91.6%.

Table 5.1 lists the extensions made to the application framework during the implementation of the music festival application and identifies the reason for each extension.

Extension	New Context Source	New Activity Property
New stage time context	✓	
Extended reconfiguration engine	✓	
Extended activity		✓
Extended trail repository		✓
Redefined initialisation class	✓	✓

**Table 5.1:** Explanation of extensions necessitated by the music festival application

Both of the new behaviours added (the new context source and the new activity attribute) required the extension of two framework classes, while the class that invokes the application framework’s behaviour was redefined to consider the framework extensions.

The music festival case study illustrates how the use of design patterns in the implementation provides hooks to ease the extension of the application framework in relation to the new context source (Observer pattern as discussed in Section 4.1). However, while the framework is designed to facilitate extension, using the application framework in the manner described in this case study is naturally more complicated than using it without extension. However, the case study illustrates that, in the hands of a developer with a moderate understanding of the application framework, the framework can be used to express a much wider range of applications than those that use location and time lapse context only.

### 5.1.3 Theme Park Trail

The music festival case study illustrates how the application framework can be extended to support applications that require additional context sources and activity attributes. However, it does not represent the situation in which a developer wishes to implement a new context source that affects the behaviour of the trail evaluation function and the activity comparison techniques in the application framework. The context added in the music festival case study resulted in changes to the activity opening and closing times.

These attributes are considered, by default, during both the evaluation of a trail and activity comparison. The third case study explores how the application framework can be extended through the addition of a context source that necessitates extension to the default activity specification and the consideration of the new activity attribute during trail evaluation and activity comparison.

Theme parks are notorious for the amount of time that patrons spend queuing for rides. This case study illustrates the manner in which the application framework can be extended to consider the notion of activity queuing time, and support an application for theme park visitors. The theme park application is required to consider ride queuing time, along with user location, preferences and the current time, when generating a trail for a theme park visitor. Implementing the theme park application involves extending the default activity specification so that each activity has an associated queuing time, and providing a source of ride queuing time context. The addition of the new context source and related activity attribute impacts on the following areas of the framework:

- The trail evaluation function. The evaluation function is required to consider ride queuing time when scheduling the user's chosen activities.
- The clash resolution mechanism. The queuing time associated with each activity must be considered when assessing whether any of the user's chosen activities clash with each other.
- The activity relevance measurement mechanism. Activity queuing time must be considered as a value dimension in the calculation of the relevance of each activity as users are likely to consider activities with shorter queuing times to be more relevant than those with longer queuing times.

#### **5.1.3.1 Implementation**

Figure 5.4 illustrates the extensions and new behaviours that are necessary in order to support the theme park application. Due to space limitations the diagram generally depicts the subclasses added to the application framework and not the classes that have been extended.





so that queuing time context events trigger the invocation of the `reconfigure()` method.

### 5.1.3.3 Adding a new activity attribute

A new activity attribute, `queuingTime`, is added to the default activity specification so that the queuing time of each theme park activity can be represented. This behaviour is contained in `ExtendedActivity`, a subclass of `Activity`. The extended activity also adds accessor methods for the new attribute and overrides the default behaviour of `getDuration()` from `Activity` so that the activity queuing time is added to the estimated duration before the estimated duration is returned. Similar to the extension of the activity specification in the music festival application (described in Section 5.1.2.3), `TrailRepository` is extended to initialise a trail composed of extended activities at application startup.

### 5.1.3.4 Adding new activity comparators

Unlike the addition of stage time change context in the music festival case study, the addition of queuing time context affects an activity attribute that is not already considered when activities are being compared<sup>3</sup>. Therefore, the behaviour used to compare activities during both clash resolution and relevance sorting must be redefined so that it considers the queuing time of each activity when making activity comparison decisions.

Two new comparator classes are implemented by extending `java.util.Comparator`. `MAUTClashResolutionComparatorExtend` is used to resolve clashes between two activities and `MAUTRelevanceComparatorExtend` is used to sort a collection of activities by relevance. The `compare()` method in each comparator is augmented so that it now considers queuing time as a value dimension. The `userPreferences.properties` file is extended to add queuing time weights for clash resolution, activity relevance and trail evaluation (the extended evaluation function is discussed in Section 5.1.3.5) so that developers/users can specify how much of an impact activity queuing time should have on the respective behaviours. `UserModel` is extended to make the value dimension weights available to the activity and trail comparison behaviour. The `normalization.properties`

---

<sup>3</sup>The stage time context in the music festival application affects the opening and closing time of each activity. These attributes are considered by default during trail evaluation and activity comparison.

files and `Normalize` are also extended to facilitate the comparison of an activity's queuing time to other activity properties.

#### **5.1.3.5 Adding a new evaluation function**

The trail evaluation function must also take the queuing time attribute into account when generating trail scores so that the trail produced reflects the user's preference as regards queuing time e.g., the user may want the trail that minimises queuing time. `Trail` is extended to produce `ExtendedTrail`, and the `getScore()` method is overridden to provide a new implementation of the evaluation function that factors in the total queuing time for the trail. This behaviour is supported by the implementation of a method in the extended version of `TrailAssessor` that compares how similar the trail being evaluated is to the trail activities ordered by least queuing time.

#### **5.1.3.6 Using the new behaviour**

The new behaviour implemented for the theme park application is used in the same manner as the new behaviour implemented for the music festival application (described in Section 5.1.2.4). The `getTrailRepository()` and `getReconfigurationEngine()` methods in `TrailGeneration` are redefined to provide access to the extended classes, and the method that starts the context services is augmented to include the context source added in the theme park application.

Finally, as in the day planner and music festival case studies, user preferences and application properties (notably the user-specified weights for queuing time in relation to activity clash resolution, relevance calculation and trail evaluation) can be set in the relevant properties files.

#### **5.1.3.7 User Interface**

Figure 5.5 contains a screen shot of the text-based user interface that is produced without framework extension for the theme park application. This display contains the same information as the user interface screen shot discussed in relation to the day planner case study (Section 5.1.1.1).

```
Console

Trail Reconfigured @ 10:15

-----
|Trail Details|
-----
1: Tower of Terror (10:18 - 10:48)
2: Indiana Jones (10:51 - 11:26)
3: Peter Pan (11:27 - 11:52)
4: Donald's Boat (11:55 - 12:10)
5: California Screaming (12:12 - 12:32)
-: Enchanted Tiki Room (unscheduled)
-: Pirates of the Caribbean (unscheduled)
-: Big Mountain Railroad (unscheduled)
```

**Figure 5.5:** The text-based display produced for the theme park application

### 5.1.3.8 Analysis

By extending the application framework so that it can cater for the concept of activity queuing time, it is possible to implement a trails application to aid theme park visitors in reducing queuing time. The theme park application is responsive to changes in user location, ride queuing time, user preferences and current time. 903 lines of code spread across 11 classes were added to the 5776 lines of code in the base application framework. This represents a code reuse percentage of 84.3%.

Table 5.2 lists the extensions made to the application framework during the implementation of the theme park application and identifies the reason for each extension. Comparing Table 5.2 to Table 5.1 on page 140 illustrates that 7 of the 12 extensions (those marked with an asterisk) are exclusive to the theme park case study and support the addition of a new concept (queuing time) to the framework as opposed to just a context source relating to existing activity attributes and a descriptive activity attribute. The difference in the code reuse percentage between the two case studies (91.6% in the music festival case study, 84.3% in the theme park case study) illustrates the impact of the extra extensions necessitated by the theme park application.

It is clear from this case study that reusing the application framework to implement the theme park application requires a good understanding of the framework. Extending the classes that form the core of the framework is the most difficult way to reuse a

Extension	New Context Source	New Activity Property
Queuing time context	✓	
Extended reconfiguration engine	✓	
Extended activity		✓
Extended trail repository		✓
Redefined initialisation class	✓	✓
Extended evaluation function*	✓	✓
New relevance measure*	✓	✓
New clash resolution measure*	✓	✓
Extended trail assessor*	✓	✓
Extended trail manipulator*	✓	✓
Extended user model*	✓	✓
Extended normalize*	✓	✓

**Table 5.2:** Explanation of extensions necessitated by the theme park application

software framework, however it is also the most powerful [61]. Therefore, by spending the time required to learn the application framework, developers will be able to add new concepts to the default base, greatly altering the default behaviour while retaining a high level of code reuse. This case study has illustrated that the application framework provides both structure to encourage extension (in relation to the context and comparator superclasses) and behaviour that can be customised (e.g., the evaluation function) to foster the development of trails applications that consider an unbounded set of contexts and behaviours.

#### 5.1.4 Summary

The case studies presented in this section illustrate how the application framework can either be reused without modification to develop specific trails applications based on location and time lapse context, or extended through the addition of new context sources, activity properties and new concepts that affect behaviour such as the evaluation function. As the amount of knowledge that developers have about the application framework increases, their ability to reuse and extend the framework to produce applications based on unforeseen context sources that consider new trail concepts increases in tandem. The code reuse level was shown to decrease as application framework usage became more advanced and the amount of extensions increased. However, the code reuse level was above 84% in all three case studies.

While the case studies are concerned with the details of specific applications, they also serve to illustrate the scope of the applications that can be developed using the framework. Without extension the framework is capable of being reused to produce applications based on user location, activity time constraints and user preferences. Examples of such applications include field study support applications for school children (similar to those described in relation to the HyCon Framework [17] but with dynamic trail adaptation), trails applications for tourists such as those described in Section 2.1 and to-do list/day planner applications similar to those discussed in Section 2.2 but with automatic context-based schedule reordering. Extending the framework to consider additional context sources greatly widens the scope of the applications that can be developed. By considering contexts that affect default activity properties such as time constraints and activity location, the highly dynamic environment in which mobile delivery couriers operate can be modelled. In this situation, activities represent parcel delivery and collection jobs, and priority can be used to represent the importance of each job. Reconfiguration can be triggered by changes in delivery urgency and collection location. The scope of the applications produced by the framework can be expanded even further when extensions are made that facilitate the consideration of new concepts that affect both the definition of an activity and the way in which activities and trails are evaluated e.g., the concept of queuing time that was added in the theme park case study. This opens up a range of possibilities including an application to dynamically schedule activities for doctors working in hospitals [38], where activities are used to represent both medical and administrative duties, and context generated by patient monitoring sensors is considered alongside standard framework contexts such as activity time constraints and user location when evaluating trails and activities.

The following section presents the quantitative evaluation of two core aspects of application framework behaviour - the trail generation and reconfiguration point identification mechanisms.

## 5.2 Trail Generation and Reconfiguration

This section presents the results of lab experiments conducted to assess the following:

- The number of activities that can be scheduled during trail generation while adhering to a response time of 12 seconds.
- The number of activities that can be considered (but not scheduled) during trail generation. A response time of 2 seconds was imposed on the behaviour being assessed during this experiment. 2 seconds was considered to be a reasonable proportion of the total response time (12 seconds) to dedicate to activity set pruning and activity relevance calculation.
- The accuracy of the trail reconfiguration point identification mechanism.

The trail generation evaluation had two objectives. The first was to quantify the capabilities of each of the three concrete trail generation techniques that are included in the application framework (brute force, genetic algorithm and simulated annealing). This involved measuring how many activities each technique can schedule on a trail within 12 seconds, where the trail produced is the best fit to the user's preferences for trail generation. The specifics of how the results were calculated are explained further in Section 5.2.1. The results of this experiment illustrate that the application framework is capable of generating trails on a resource-constrained mobile platform within a reasonable response time that contain a non-trivial number of scheduled activities. Therefore, applications developed using the framework are capable of producing trails that are of higher quality than non-computer-generated trails (this statement is supported by the results of the trail quality study discussed in Section 5.3). The results of the trail generation experiment can also be used by framework developers to a) inform their decision regarding which trail generation approach to use and b) reason about the capabilities of their own framework extensions in the area of trail generation strategies. The second objective of the trail generation evaluation was to quantify how many activities can be considered during trail generation without spending more than 2 seconds on activity set pruning and sorting of the activity set by relevance. This involved measuring the time taken to prune the activity set and sort activities by relevance following the receipt of a context event. Details of precisely how the results were calculated are contained in Section 5.2.2.

The objective of the reconfiguration point identification experiment was to quantify the accuracy of the technique in determining whether or not trail reconfiguration is necessary following the occurrence of a context event. This involved assessing the decisions made by the reconfiguration point identification technique following the receipt of context events, where the decision that should be made in each case is known. Full details regarding how the results were calculated are provided in Section 5.2.3.

All experiments required context events to trigger the execution of the behaviour being measured - either trail generation or reconfiguration point identification. These context events were simulated in the lab as opposed to generated during real world deployment. The simulation of context data, an active research area within mobile, context-aware computing [54, 11, 93, 86], allows researchers to conduct evaluations of context-based technology without undergoing the cost of a full application deployment. The use of simulations provides control over environmental parameters, facilitating evaluations that may prove difficult to conduct in the real world, given its volatile nature. While there is no substitute for using deployment and subsequent user studies to evaluate the effectiveness of a system in terms of criteria such as those proposed by Scholtz [114] (e.g., application appeal and command of user attention), simulation is an appropriate technique for certain types of evaluation. The experiments discussed in this section do not require user input, nor are the results user-specific or subjective. Therefore they are suitable candidates for the use of simulation.

The remainder of this section presents and discusses the results produced during lab experiments that measured the behaviour of the trail generation and reconfiguration point identification techniques.

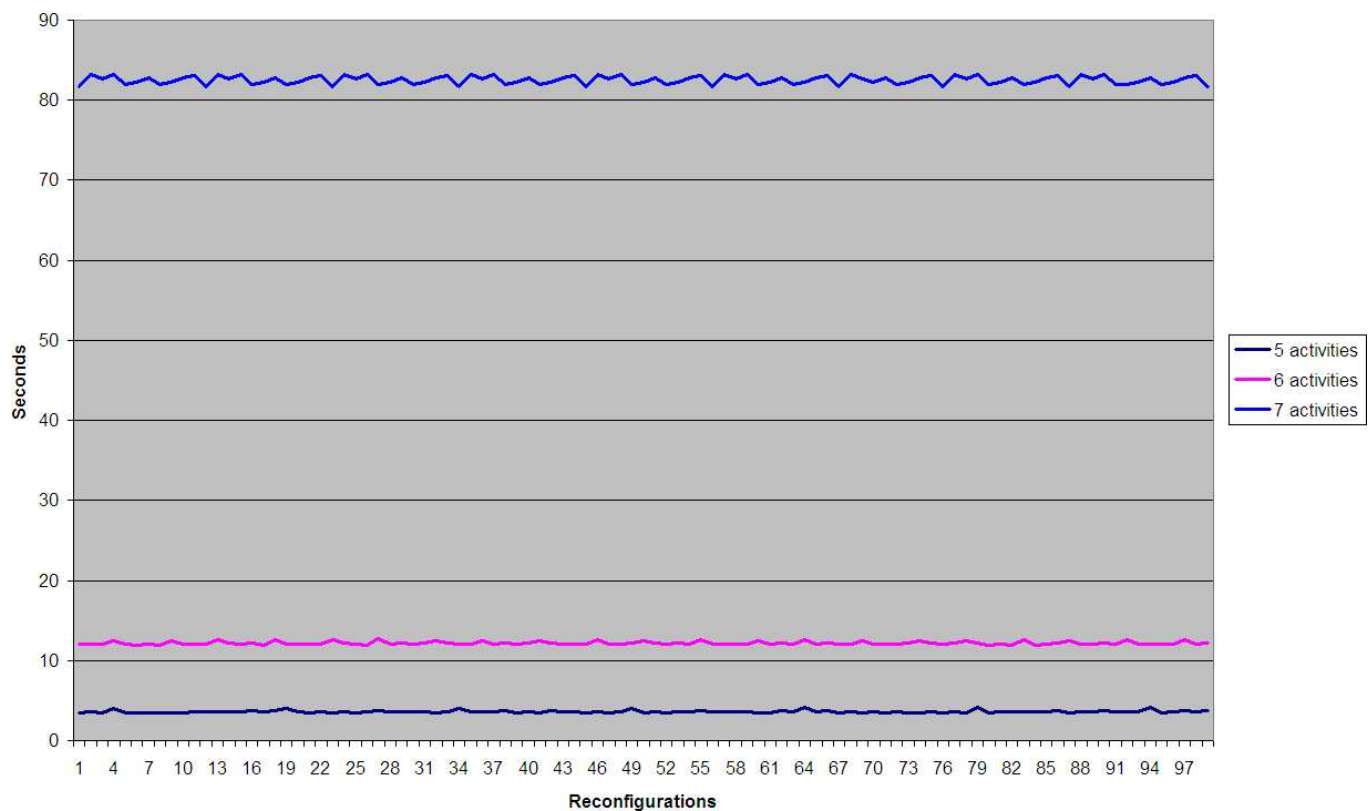
### 5.2.1 Trail Generation - Activity Scheduling

The responsiveness of the trail generation mechanism was calculated by recording the duration between the instant before the invocation of the `reconfigure()` method in `ReconfigurationEngine` and the instant after `reconfigure()` returns. All three candidate solution generation techniques were evaluated and the results are discussed throughout the remainder of this subsection. The mobile device used for the experiments was a

HP iPAQ h6300 series with a Texas Instruments OMAP1510 168 MHz processor and 64 MB of RAM.

### 5.2.1.1 Brute Force

Figure 5.6 illustrates the response times of the brute force trail generation algorithm for 5, 6 and 7 activities over the course of 100 trail reconfigurations. The average response time of the trail generation algorithm with a subtrail size of 5 activities is 3.16 seconds, 12.16 seconds for 6 activities and 82.5 seconds for 7 seconds. Therefore, the trail generation approach in the application framework can generate an optimal subtrail of 5 activities within a reasonable response time (or 6 activities by marginally exceeding the response time limit) on the mobile device used in the experiment.



**Figure 5.6:** Brute force trail generation response times

### 5.2.1.2 Genetic Algorithm

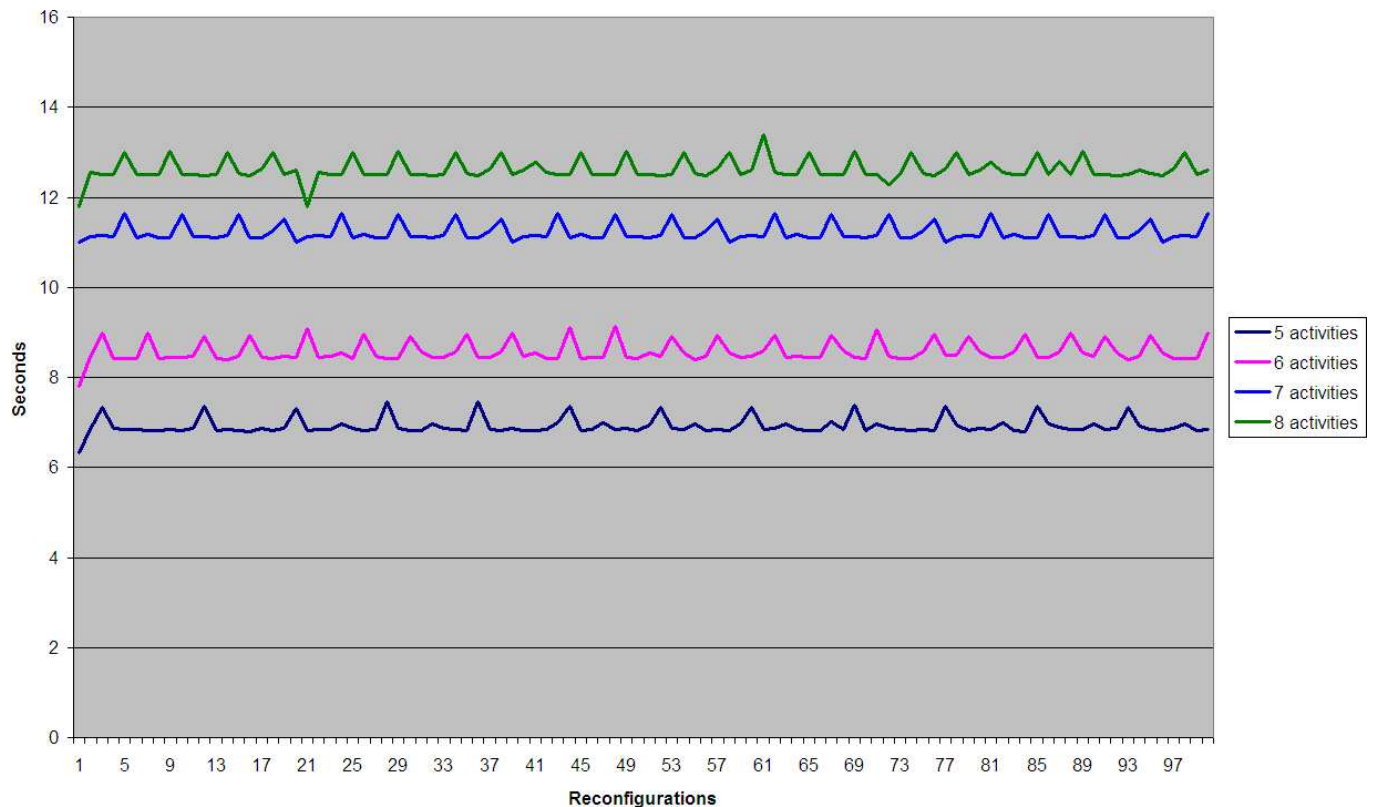
The genetic algorithm proves unable to generate an optimal subtrail of 5 activities within a reasonable response time. Executing the genetic algorithm on the mobile device with a



subtrail size of 5, and parameters that result in the generation of the optimal trail, takes 158.1 seconds on average.

### 5.2.1.3 Simulated Annealing

Figure 5.7 illustrates the response times for four subtrail sizes (5, 6, 7 and 8) using the simulated annealing trail generation algorithm over 100 trail reconfigurations. The algorithm is supplied with parameters that result in the consistent generation of the optimal trail. The average response time with a subtrail of 5 activities is 6.9 seconds, 8.6 seconds for 6 activities, 11.2 seconds for 7 activities and 12.6 seconds for 8 activities. Using the simulated annealing algorithm therefore facilitates the generation of trails with a subtrail of 7 activities within a reasonable response time on the mobile device used in the experiment. It is possible to consider 8 activities in the subtrail if the response time upper limit is marginally exceeded.



**Figure 5.7:** Simulated annealing trail generation response times

#### 5.2.1.4 Analysis

The response time experiments show that the application framework is capable of scheduling between 5 and 7 activities within a reasonable response time using brute force and simulated annealing, while the genetic algorithm proves to be too resource intensive for deployment on the mobile device used in the experiments.

The brute force results in Figure 5.6 illustrate the exponential nature of the algorithm and the infeasibility of its usage as the number of activities in the subtrail is increased. Improvements in mobile device technology will increase the number of activities that can be considered using brute force within a reasonable response time. For example, the desktop machine used to develop the application framework<sup>4</sup> can generate an optimal subtrail of 10 activities in 9.3 seconds using the brute force algorithm.

The genetic algorithm, which creates many objects during execution to simulate the process of biological evolution, proves to be unsuitable for deployment on a resource-constrained mobile device. The response of time 158.1 seconds for a subtrail of 5 activities shows that it is infeasible to use this approach on the mobile device used in the experiment. The same algorithm executing on the desktop machine has an average response time of 1.25 seconds for 5 activities. Subtrails of 6 and 7 activities respond in 3.5 and 6 seconds respectively, indicating the linear nature of the algorithm and its potential for use on more sophisticated mobile devices.

Simulated annealing works by modifying a single solution (as opposed to the genetic algorithm which generates many candidate solutions and evolves them until a single solution is chosen). It has been shown previously that genetic algorithms typically take 10-24 times longer than simulated annealing to achieve similar results [79]. Additionally, it has also been shown that simulated annealing algorithms perform better than genetic algorithms when both algorithms are given the same amount of time within which to produce a result [72]. The simulated annealing algorithm outperforms both the brute force algorithm and the genetic algorithm, with the results in Figure 5.7 illustrating the linear nature of the algorithm and the contrast between the cost of adding an activity when using simulated annealing and the cost of the same operation with brute force.

---

<sup>4</sup>A Dell Optiplex GX260 with an Intel Pentium 4 2.2GHz processor and 512 MB of RAM.

Simulated annealing can generate an optimal subtrail of 8 activities within 12.6 seconds. This response time, along with that of brute force at 6 activities, is marginally above what is considered to be reasonable for the purpose of this thesis. The reduction of the subtrail size by one in both cases ensures that the response times adhere to the acceptable boundary.

The results of this experiment indicate that simulated annealing is the best algorithm to select when developing an application because it can consider the most activities within a reasonable response time. However, as discussed in Section 3.2.3.5, using this algorithm requires an understanding of how the parameters in the `simulatedAnnealing.properties` file affect the behaviour of the algorithm in terms of execution time and solution quality. Brute force is guaranteed to produce the best trail and can consider a significant number of activities within a reasonable response time. Therefore, brute force it is a better choice in terms of lessening the cognitive burden on the developer.

In summary, the results of the trail generation experiments demonstrate that the application framework is capable of generating non-trivial trails within a reasonable response time. The trail quality experiment discussed in Section 5.3 illustrates that activity scheduling problems involving seven activities pose a significant challenge to humans who typically spend close to two minutes (or more) composing a solution that may not be the best fit to their preferences given the current context.

Finally, it is important to note that the experiments described in this section do not measure response time as the end-to-end response time defined by Macabee [78]. In the words of Macabee, end-to-end response time is: “The time between the start of users request (indicated by depressing a key or a button) and the time when the user can use the data supplied in response to the request”. The measurements used in the trail generation experiments do not consider the time required to render the result of the trail generation process to the user. The amount of time required for this operation will vary depending on the user interface employed. The subtrail sizes in both the brute force algorithm and simulated annealing algorithm can, if necessary, be reduced to compensate for the result rendering overhead, therefore facilitating an end-to-end response time within the reasonable bounds.

## 5.2.2 Trail Generation - Activity Consideration

Pruning the activity set and sorting the activity set by relevance (shortened to ‘activity set preparation’) are the first steps in the trail generation process. The amount of time taken by these operations dictates how much time is left for the generation of the best trail for the user. Therefore, if too many activities are included in an application, the time required for activity set preparation will have the effect of reducing the number of activities that can be scheduled on the trail. This experiment quantifies how many activities can be included in an application so that the time spent on preparing the activity set for activity scheduling does not exceed 2 seconds. The responsiveness of the activity set preparation behaviour was calculated by executing the day planner application on the same mobile device used in activity scheduling experiment<sup>5</sup> and recording the time taken for activity set preparation i.e., the combined execution time of the `pruneTrail()` and `sortByRelevance()` methods in `ReconfigurationEngine`. The day planner application was executed with multiples of 10 activities until the response time limit of 2 seconds was reached. 100 context events were generated per execution of the application.

### 5.2.2.1 Results

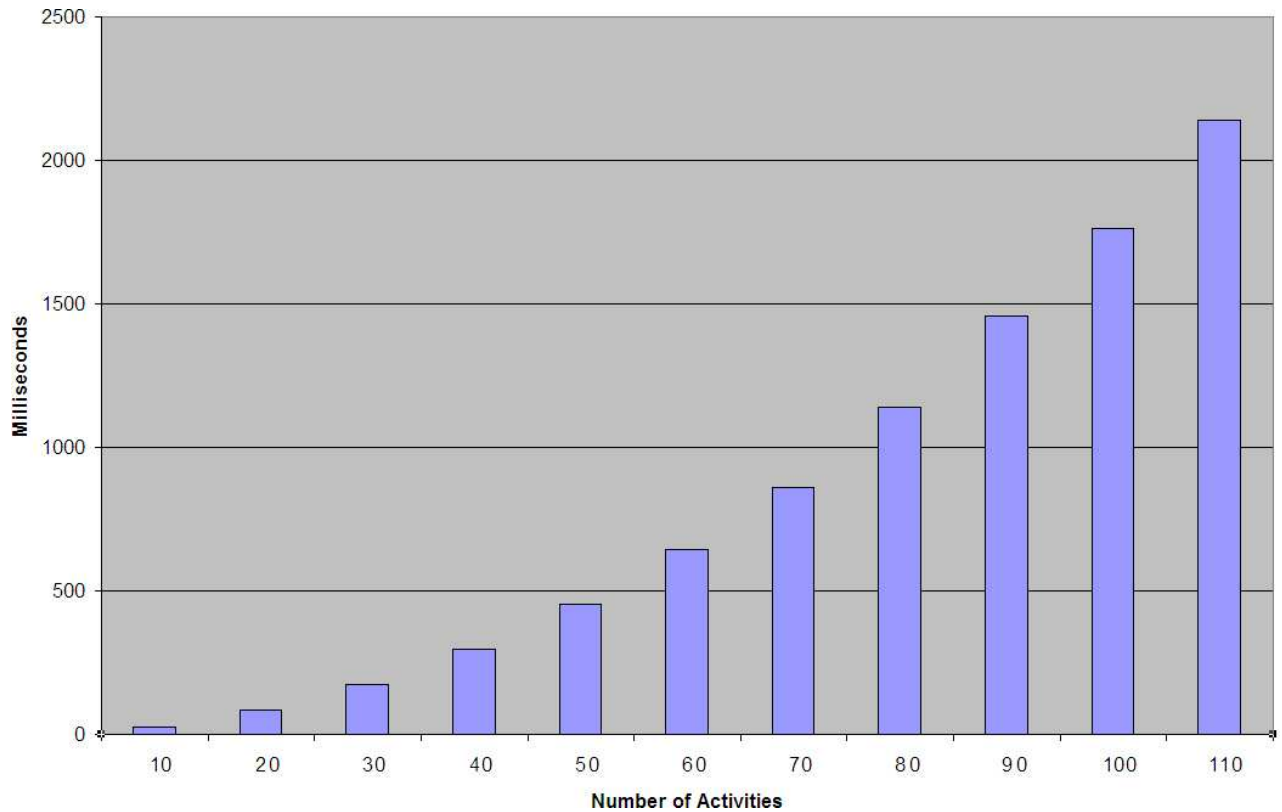
Figure 5.8 illustrates the results of the activity consideration experiment. The results show that the process of activity set preparation for an activity set of size 10 took 26.18 milliseconds on average. As activities are added to the activity set, the average response time for activity set preparation increases in a linear fashion to the point where an activity set size of 110 takes 2140.69 milliseconds (just over 2 seconds).

### 5.2.2.2 Analysis

The results of the activity consideration experiment illustrate that between 100 and 110 activities can be considered during trail generation without spending more than 2 seconds on activity set preparation. This contrasts with applications like GUIDE, P-Tour and the DTG (discussed in Section 2.1), where the maximum number of activities that can be considered within a reasonable response time are 9, 14 and 16 respectively. The

---

<sup>5</sup>HP iPAQ h6300 series with a Texas Instruments OMAP1510 168 MHz processor and 64 MB of RAM



**Figure 5.8:** Results of the activity consideration experiment

number of activities that can be considered during trail generation and reconfiguration in applications built using the application framework depends on the capabilities of the device executing the application e.g., the desktop machine used to develop the application framework can consider 1000 activities in 2 seconds. The ability of the application framework to include a large number of activities and schedule them as they become relevant to the user gives developers the power to design and implement applications such as RiddleHunt (discussed in Section 3.1.4) in which it is necessary to have a relatively large amount of activities to make the game interesting for players.

### 5.2.3 Reconfiguration Point Identification Accuracy

The accuracy of the trail reconfiguration point identification mechanism was assessed by simulating context events in each of the case study applications described in Section 5.1 and identifying if each context event was correctly handled by the reconfiguration point identification mechanism. Each application was executed twice, once with no reconfigura-

tion point identification (reconfiguration was triggered each time a context event occurs) and once with the reconfiguration point identification mechanism in place. During the first execution of the application, a file was created that noted whether or not each reconfiguration was necessary or unnecessary. This was measured by comparing the trail produced by the reconfiguration to the trail prior to reconfiguration - if they were the same then the reconfiguration was unnecessary. During the second execution, in which smart reconfiguration was used, the decisions made by the reconfiguration point identification mechanism (required or not required) were recorded. The decisions made in the first and second trials were then compared.

The same one hundred context events were generated during the execution of each version of each case study application, and the reconfiguration point identification trials were carried out using two  $\tau$  values - 0.95 and 0.85. These  $\tau$  values were chosen because  $\tau$  values in and around that region (0.8 - 0.95) were shown to produce good results during application development and testing. When comparing the findings of the first trial for each application against the trials with reconfiguration point identification there are four possible outcomes. Each trail reconfiguration is classed as one of the following:

1. Unnecessary as Not Required. Reconfiguration proven to be unnecessary by the first trial is identified as 'Not Required' by the reconfiguration point identification mechanism. This is a positive result.
2. Required as Required. Reconfiguration proven to be required is identified as 'Required' by the reconfiguration point identification mechanism. This is a positive result.
3. Required as Not Required. Reconfiguration proven to be required is identified as 'Not Required' by the reconfiguration point identification mechanism. This is a negative result.
4. Unnecessary as Required. Reconfiguration proven to be unnecessary is identified as 'Required' by the reconfiguration point identification mechanism. This is a negative result, although not as detrimental as classification 3 in that it does not result in a discrepancy between the user's trail and their environment.

It was expected that the number of instances of classification #3 would be minimised at the higher  $\tau$  value, resulting in instances of classification #4 being relatively high by comparison. This is because the likelihood of reconfiguration being deemed necessary increases as  $\tau$  increases. At the lower  $\tau$  value it was expected that instances of classification #3 would increase and instances of classification #4 would decrease. Therefore, it was expected that the higher  $\tau$  value would result in the trail more accurately reflecting the contextual situation, but that achieving this accuracy would require sacrificing resources to unnecessary reconfigurations.

### 5.2.3.1 Day Planner Results

The results of processing 100 location change events in the day planner application are illustrated in Figure 5.9. With  $\tau = 0.95$ , 75 context events out of the 100 that are generated are handled correctly. Of the remaining 25 context events, 21 are handled by unnecessarily reconfiguring the trail and 4 events that should cause reconfiguration do not. This data is illustrated in the top half of Figure 5.9. With  $\tau = 0.85$ , the number of events handled correctly rises to 86. However, of the 14 context events that are mishandled, 3 cause unnecessary reconfiguration and 11 cause significant context events to be ignored. This data is illustrated in the bottom half of the figure. The difference between the results obtained using the two  $\tau$  values correlates with expectations for the experiment i.e., more context events resulted in classification #3 when  $\tau$  was at the lower value.

### 5.2.3.2 Music Festival Results

The results of the reconfiguration point identification experiment as conducted by generating 100 stage time context events in the music festival trail application are illustrated in Figure 5.10. With  $\tau = 0.95$  (illustrated in the top half of Figure 5.10), 83 context events are handled correctly. Of the remaining 17 events, 12 unnecessary reconfigurations are carried out and 5 necessary reconfigurations do not take place. When the  $\tau$  value is reduced to 0.85 (illustrated in the bottom half of the figure), the number of events identified correctly increases to 88. As expected, the remaining 12 events contain more cases in which reconfiguration should have taken place (7) than cases in which reconfiguration was unnecessary (5).

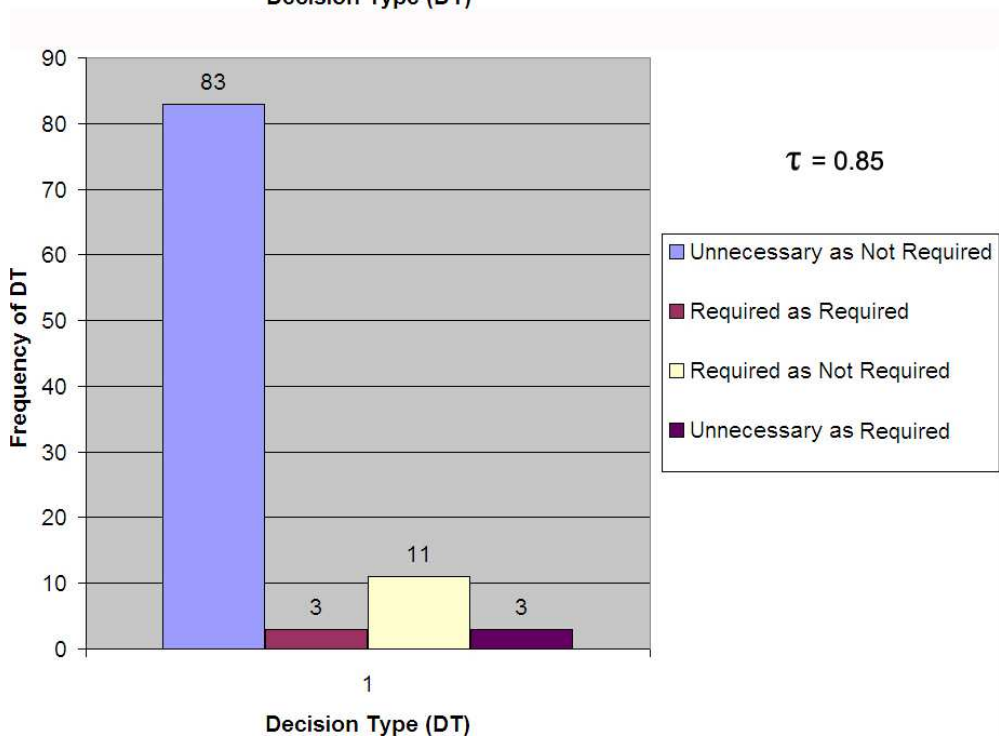
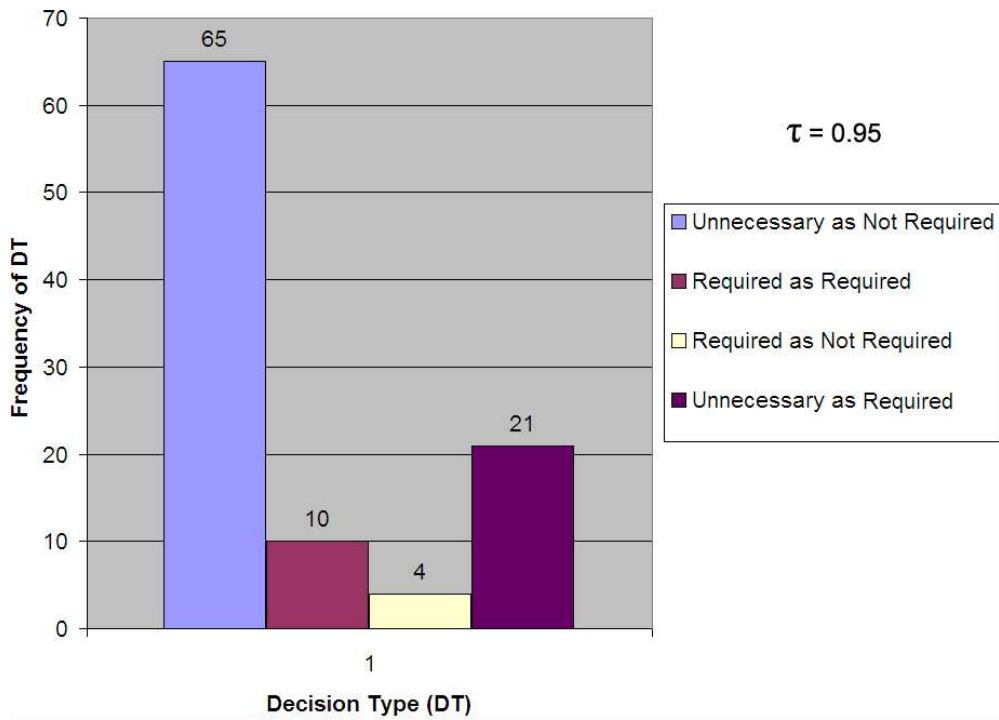


Figure 5.9: Day planner results with  $\tau = 0.95$  (top) and  $\tau = 0.85$  (bottom)



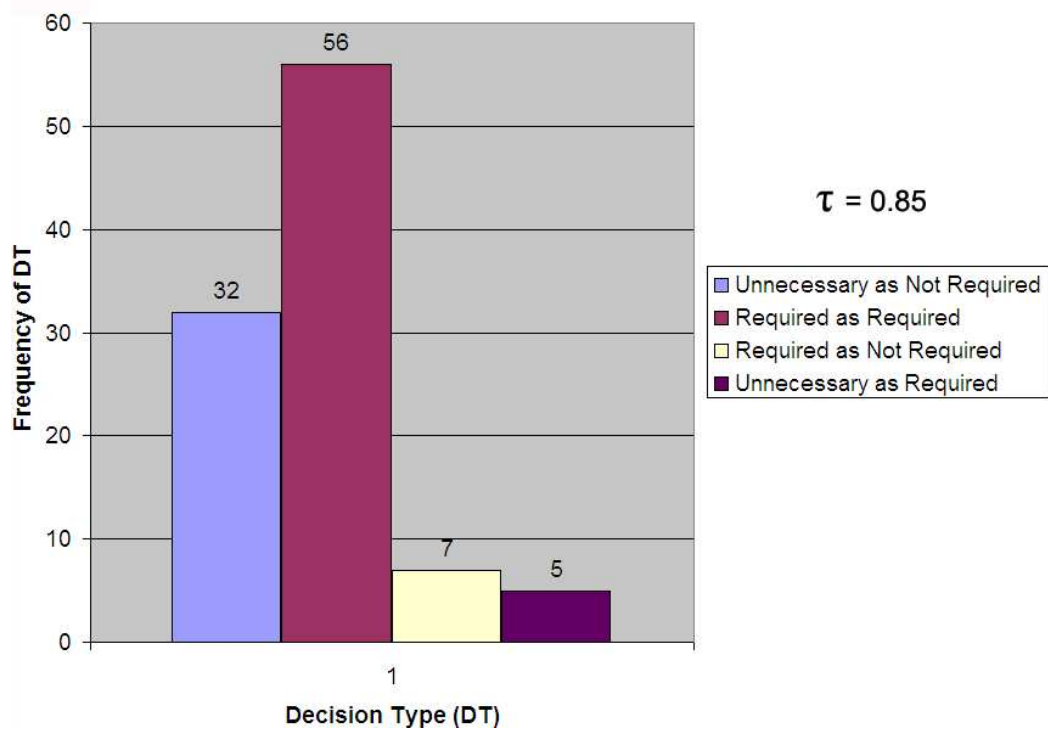
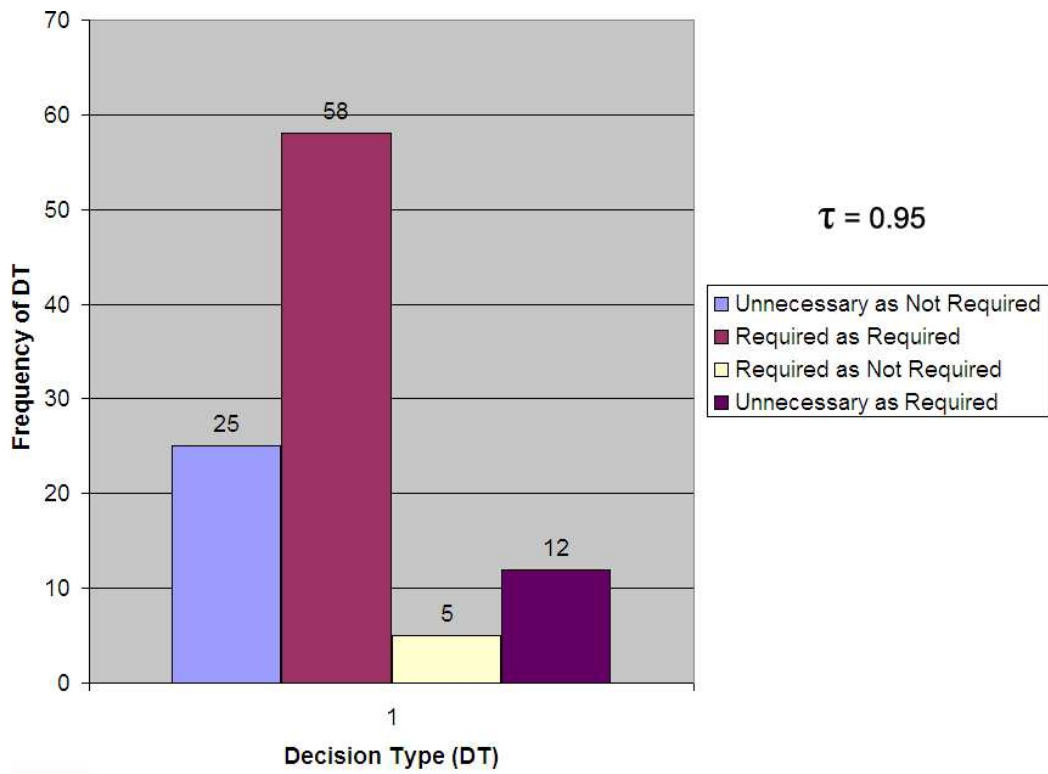


Figure 5.10: Music festival results with  $\tau = 0.95$  (top) and  $\tau = 0.85$  (bottom)

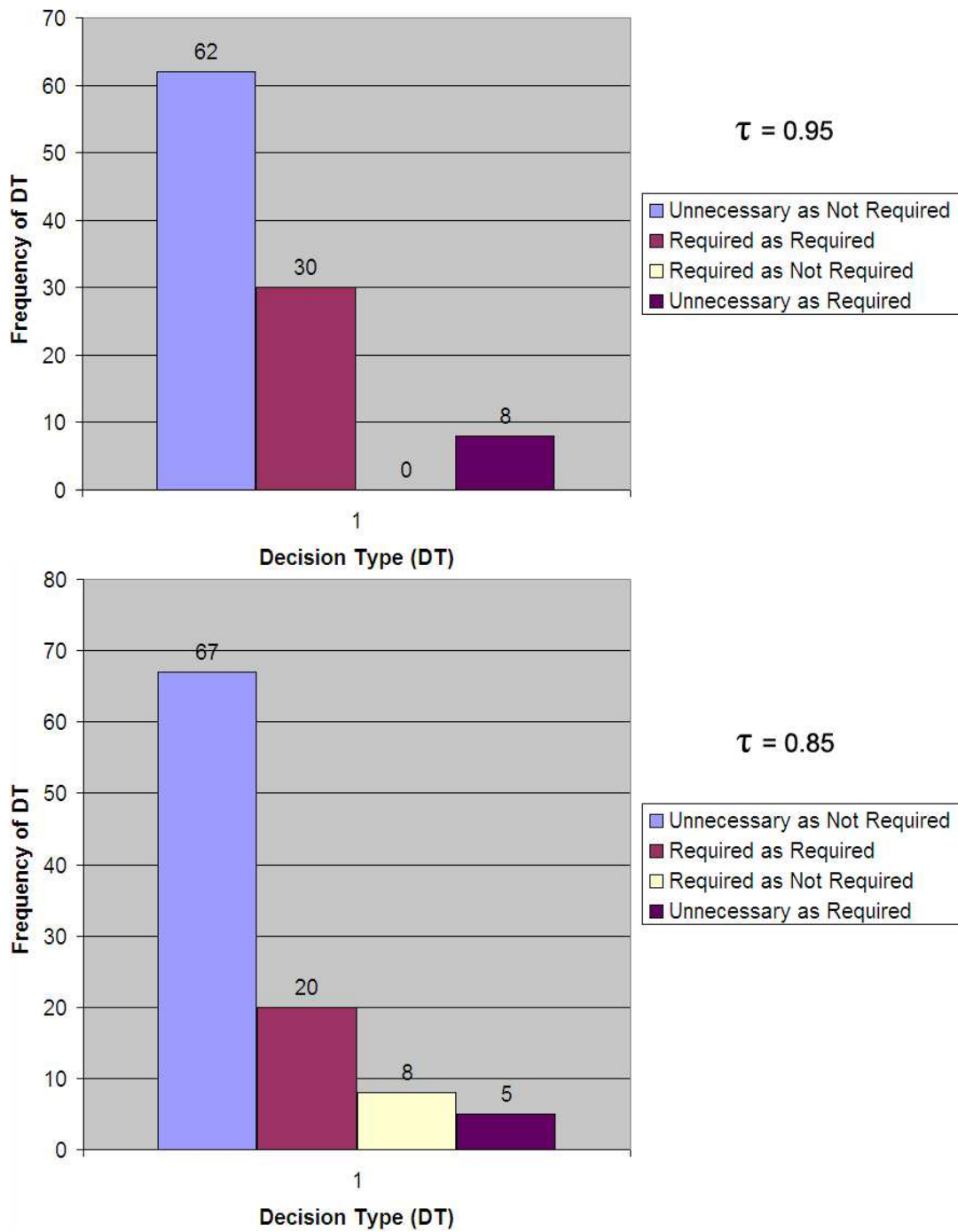


Figure 5.11: Theme park results with  $\tau = 0.95$  (top) and  $\tau = 0.85$  (bottom)

### 5.2.3.3 Theme Park Results

The experiment results for the theme park application dealing with 100 queuing time change events are illustrated in Figure 5.11. With  $\tau = 0.95$  (illustrated in the top half of Figure 5.11), 92 context events are correctly handled. Of the remaining 8 events, there were 8 unnecessary reconfigurations and 0 incorrectly classified necessary reconfigurations, meaning that the trail was never in a state inconsistent with the context. Moving  $\tau$  to 0.85 (illustrated in the bottom half of the figure) reduces the number of context events handled correctly to 87. The remaining 13 events are composed of 5 unnecessary reconfigurations and 8 required reconfigurations that were deemed to be not required. These results, along with those generated in the trials involving the day planner and music festival applications, correlate with the experiment expectations.

### 5.2.3.4 Analysis

Table 5.3 summarises the results of the reconfiguration point identification experiment.

	Unnecessary as Not Required	Required as Required	Required as Not Required	Unnecessary as Required	Positive Result	Negative Result
Day planner $\tau = 0.95$	65	10	4	21	75	25
Day planner $\tau = 0.85$	83	3	11	3	86	14
Music festival $\tau = 0.95$	25	58	5	12	83	17
Music festival $\tau = 0.85$	32	56	7	5	88	12
Theme park $\tau = 0.95$	62	30	0	8	92	8
Theme park $\tau = 0.85$	67	20	8	5	87	13

**Table 5.3:** Results of the reconfiguration point identification experiment

The results illustrate that, on average, the reconfiguration point identification technique handles 88.6% of context events correctly. Of the remaining context events, 4.3% result

	Reconfiguration Necessary	Classified Correctly	Reconfiguration Unnecessary	Classified Correctly
Day Planner	14	10	86	65
Music Festival	64	58	36	25
Theme Park	30	30	70	62
Total	108	98	192	152

**Table 5.4:** Further investigation of the trials with  $\tau = 0.95$

in unnecessary reconfigurations and 7.1% result in reconfiguration not being invoked when it should be. These figures are calculated by taking the ‘best’ result from each experiment i.e.,  $\tau = 0.85$  in the day planner, 0.85 in the music festival application and 0.95 for the theme park application. This assumes that ‘best’ means that as many context events as possible are handled correctly. However, if the trials that minimise the amount of miscategorised required reconfigurations (classification #3) are selected as the best results i.e.,  $\tau = 0.95$  in all applications, then 83.3% of the context events are handled correctly, 13.7% of the events result in unnecessary reconfigurations and 3% of the events that should cause reconfiguration do not. The reduction in miscategorisation of context events that necessitate reconfiguration relates primarily to the day planner application which has 4 when  $\tau = 0.95$  and 11 at 0.85.

Table 5.4 further illustrates the way context events were handled in the experiments that produced the best results (where the second definition of best is used). Of all the context events that necessitate reconfiguration (108 out of 300), the reconfiguration point identification mechanism correctly classified 98 of these events, or 90.7%. Of the remaining context events (192 out of 300) that do not require reconfiguration to occur, the mechanism (with  $\tau = 0.95$ ) correctly classifies 152 of these events, or 79.2%. This means the majority of the context events that are handled incorrectly result in computing

resources being needlessly consumed as opposed to the trail becoming out of sync with the world that it represents.

The analysis of the experiment results illustrates that the reconfiguration point identification in the application framework can categorise context events correctly in the majority of cases. Developers can customise the  $\tau$  value as appropriate on a per-application basis to achieve a suitable balance between unnecessary reconfiguration and miscategorisation of required context events. The importance of the conflicting goals of maintaining trail relevance and conserving system resources will dictate the  $\tau$  value used. As expected, a higher  $\tau$  value was shown to increase the amount of time that the trail accurately represents the user's context. Therefore, a higher  $\tau$  value is appropriate for use in trails applications where accuracy is critical e.g., in the healthcare application described in Section 5.1.4. However, a higher  $\tau$  value means there is a greater chance of incurring unnecessary reconfigurations. Lowering the  $\tau$  value has the effect of reducing unnecessary reconfiguration and increasing the number of required reconfigurations that are miscategorised. The use of a lower  $\tau$  value is appropriate in trails applications where constant accuracy is not paramount e.g., in a tourist guide application.

Finally, the average execution time of the reconfiguration point identification behaviour following the receipt of a context event is 777 milliseconds<sup>6</sup>. Therefore, it is preferable to incur this cost for each context event rather than risk a high number of unnecessary trail reconfigurations that are far more costly in terms of response time e.g., between 12-13 seconds for 6 activities using brute force or 8 activities using simulated annealing.

### 5.3 Trail Quality

The application framework provides a reusable and extensible way to produce context-aware trails for mobile users. While the case studies in Section 5.1 and lab experiments in Section 5.2 evaluate technical aspects, they do not provide any information regarding how humans perceive the decisions made by the framework. Consequently, the trail quality

---

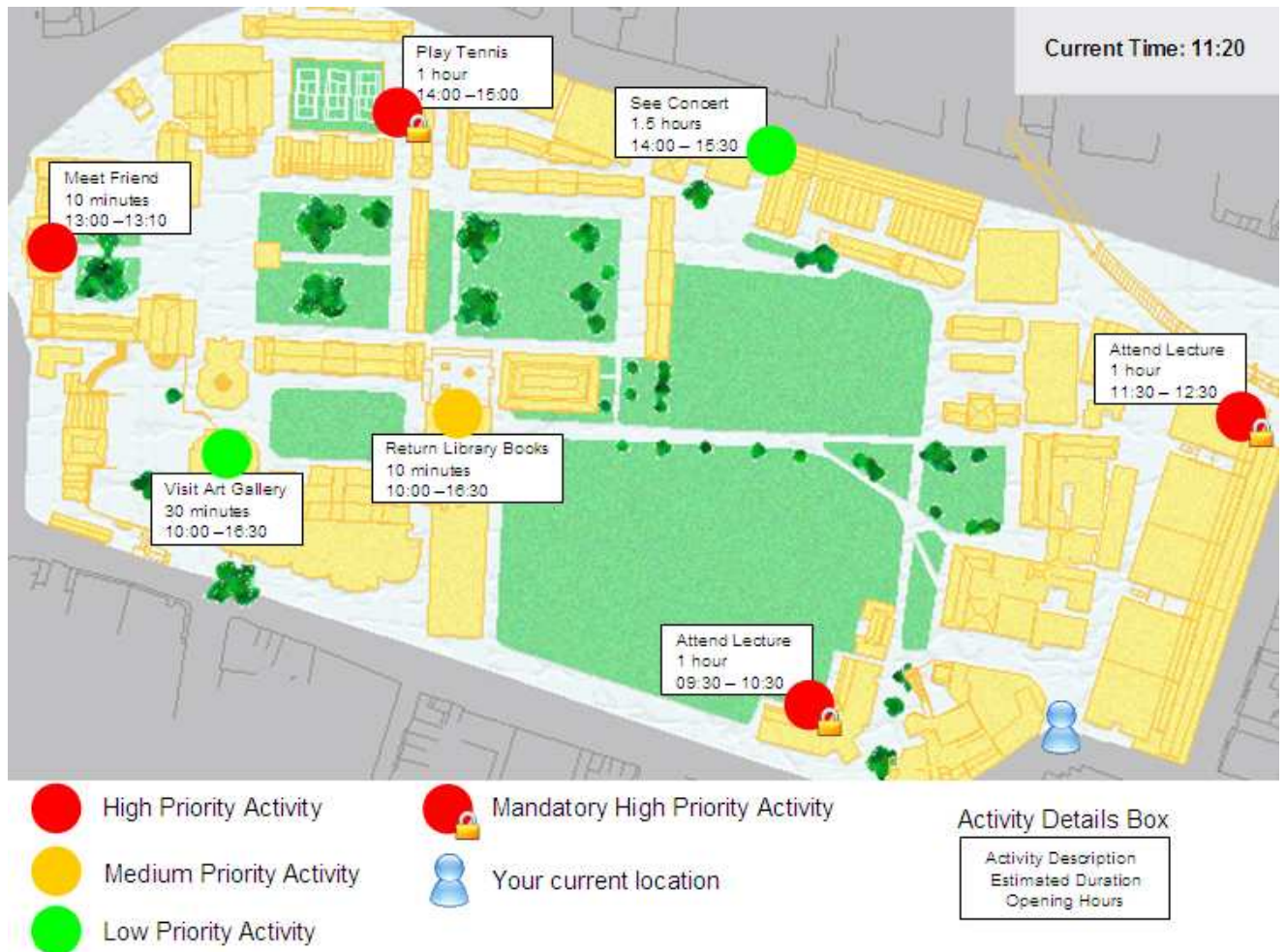
<sup>6</sup>This figure was calculated by averaging the time taken to execute the reconfiguration point identification behaviour in the day planner application (running on the iPAQ) 100 times with a subtrail of 8 activities.

experiment was conceived to evaluate this aspect. The objectives of the trail quality experiment were to determine human opinion on the quality of the trails generated, and to investigate if the application framework provides an advantage over manual trail ordering. Three hypotheses were tested:

1. If the trail generation mechanism in the application adequately models the key factors involved in making activity scheduling decisions in a given situation, then subjects, when in that same situation, will be satisfied with the trails produced by the application framework.
2. If subjects are presented with a trails solution to a context-based activity scheduling problem they will spend less time validating the solution (even when they have no reason to trust it) than they would spend devising their own solution to a similar problem.
3. If presented with a context-based activity scheduling problem to solve, subjects may not make the best use of their resources, primarily their time. Subjects, when shown the computationally generated trails solution to the problem, may be more satisfied with it than with their own solution.

Forty subjects participated in the experiment (29 male and 11 female). Ages ranged from 22 to 56, although the majority of subjects were aged between 24 and 32. Balancing gender and achieving a balance across several age categories were not considered as important as ensuring that all subjects were familiar with the geographic area in which the activity scheduling problems used in the experiment were based. For this reason, all subjects were selected based on their knowledge of the Trinity College campus. The activities in the scheduling problems were located on campus, and all subjects involved in the experiment had a good understanding of the layout of the campus, the majority being either Trinity students or employees. Subjects could therefore use their knowledge of the campus to make informed estimates about distances and related traversal times between activity locations. Additionally, subjects were familiar with the idea of having to carry out several activities at different locations around the campus.

Subjects were first asked to read an information sheet (presented in Appendix C.1) that explained the concept of a trail and described what the experiment would entail.



**Figure 5.12:** An activity scheduling problem given to trails experiment subjects

Next, a simple activity scheduling problem and solution validation problem (shown in Appendix C.2.2 and C.2.3) were worked through to familiarise subjects with the tasks involved in the experiment. Subjects were then asked to manually solve an activity scheduling problem involving seven activities and re-solve it following the introduction of new context<sup>7</sup>. Finally, subjects were also asked to validate that computer-generated solutions to similar problems were ‘reasonable’, where reasonable is intentionally subjective and measured on a Likert scale [75] (the questionnaire is included in Appendix C.3). The subjects were divided equally into two groups. Group 1 solved problems first and validated computer-generated solutions second. Group 2 did the tasks in reverse order. This facilitated the observation of potential learning effects. All tasks were timed. Figure 5.12 illustrates the first activity ordering problem that subjects in Group 1 were asked

<sup>7</sup>Appendix C.2 contains all of the activity scheduling problems and solutions used in the experiment.

to solve. Each activity has the following information associated with it: description, location, estimated duration, opening hours, priority, obligation and location. Subjects were required to identify impossible activities, resolve clashes and order the remaining activities (five activities remained for users to schedule after they had pruned the activity set).

### 5.3.1 Trail Quality Experiment Results

The trail quality experiment produced three types of result for each group of subjects:

1. Timing. These results relate to the amount of time subjects spent a) solving activity scheduling problems and b) validating computer-generated solutions to similar problems.
2. Solution validation. These results represent the extent to which subjects agreed that solutions to activity scheduling problems produced by the application framework are reasonable.
3. Solution quality. These results represent the quality of the activity scheduling solutions produced by subjects.

Table 5.5 contains the results of the timed element of the experiment. The median time taken to solve a problem in Group 1 was 108 seconds, with 58 seconds being the median time for validating a computer-generated solution. The results of the solution validation element of the experiment are contained in Table 5.6. When shown the computer-generated solution to a problem similar to that which they had solved, 95% of Group 1 totally agreed that the computer-generated trail was reasonable, 5% partially agreed. 100% agreed that the reconfiguration decision made by the computer was reasonable. The results of the experiment in relation to the quality of the solutions produced by subjects, and their view of how they compare with the computer-generated solutions to the same problems, are summarised in Table 5.7. 55% of subjects in Group 1 solved the trail generation problem in the same manner as the application framework. All subjects who solved the problem incorrectly agreed that the computer-generated solution was better than the one they had composed. 45% of subjects in Group 1 solved the reconfiguration problem



in the same manner as the application framework. All subjects who did not solve the re-configuration problem correctly indicated preference for the computer-generated solution over their solution.

	<b>Group 1</b>	<b>Group 2</b>
Solve problem (median time)	108 seconds	161 seconds
Validate solution (median time)	58 seconds	65 seconds

**Table 5.5:** Trail quality experiment timing results

The median time taken for solving a problem in Group 2 was 161 seconds, with 65 seconds being the median time for validating a solution. 95% of Group 2 totally agreed that the computer-generated trail for a similar problem was reasonable, 5% partially agreed. 85% of Group 2 totally agreed that the reconfiguration decision was reasonable, 15% partially agreed. 65% of subjects in Group 2 solved the trail generation problem in the same manner as the computer, while 45% percent of subjects composed a solution to the reconfiguration problem that was identical to that produced by the computer. As with Group 1, all subjects, when shown the computer-generated solution to both problems, indicated a preference for it over their own solution.

### 5.3.2 Analysis

In relation to the first hypothesis, which states that subjects will agree that trails produced by the application framework are reasonable if the framework adequately models how subjects make trail decisions, the results, illustrated in Table 5.6, show an average of 93.75% total agreement. This result validates hypothesis 1. The reasons for non-total agreement were collected via questionnaire. The main issue was the lack of leeway in the estimated activity durations used for trail generation. This lack of flexibility caused certain activities to be marked as impossible because the application framework calculated that undertaking them would involve overrunning the activity closing time. In cases where the overrun was only a few minutes, subjects felt that in reality they would still do the activity but not spend as long doing it. The concepts of estimated activity duration leeway and closing time leeway (discussed in Section 4.2.1) were added to the framework to address this.

	Totally Agree	Partially Agree	Neither	Partially Disagree	Totally Disagree
<b>Group 1</b>					
Solution reasonable?	95%	5%	0%	0%	0%
Reconfiguration solution reasonable?	100%	0%	0%	0%	0%
<b>Group 2</b>					
Solution reasonable?	95%	5%	0%	0%	0%
Reconfiguration solution reasonable?	85%	15%	0%	0%	0%

**Table 5.6:** Trail quality experiment solution validation results

It took subjects just over twice as long on average to manually solve an activity scheduling problem themselves than to validate a computer-generated solution to a similar problem. Subjects in Group 2 spent a significantly longer time solving problems than subjects in Group 1. It is thought that the increase in the median time taken by subjects in Group 2 to generate a solution is a result of being exposed to a solved activity scheduling problem before having to attempt to solve a problem themselves. As a result of having a greater understanding of how to solve activity scheduling problems, a higher percentage of subjects in Group 2 solved the activity scheduling problem in the same manner as the computer. The results of the timed aspect of the experiment, illustrated in Table 5.5, validate the second hypothesis, which states that users will spend less time validating computer-generated solutions than devising solutions to similar problems.

The results presented in Table 5.7 illustrate that, on average, 52.5% of subjects solved activity scheduling problems in the same manner as the application framework. 100% of the subjects that did not solve their activity scheduling problems in the same manner as the application framework agreed that the computer-generated solutions to the problems they attempted were better than those they had produced themselves. The reason subjects preferred the computer-generated solutions was because the computer-generated solutions made better use of the time available. This result validates hypothesis 3, which states that subjects may be more satisfied with the computer-generated solution to an activity scheduling problem than their own solution.

	Solved Correctly	Solved Incorrectly	Prefer Framework Solution
<b>Group 1</b>			
Solution generation problem	55%	45%	100%
Reconfiguration problem	45%	55%	100%
<b>Group 2</b>			
Solution generation problem	65%	35%	100%
Reconfiguration problem	45%	55%	100%

**Table 5.7:** Trail quality experiment solution quality results

In summary, the results of the trail experiment support the thesis that the trails generated by the application framework are reasonable, and are often superior to the solutions generated by humans. The results also indicate that presenting the user with a trail will significantly reduce the amount of time they spend scheduling activities, even in the case where they have no trust in the computer. It is expected that trust in the computer-generated trails will increase through positive experience, reducing the median time for solution validation and increasing time savings.

## 5.4 Chapter Summary

This chapter has described the evaluation of various aspects of the application framework. The case studies illustrate that the application framework fulfills its requirements in relation to supporting both reusability and extensibility, with the extent to which the framework can be extended depending on the developer’s knowledge of the framework.

The evaluation of the trail generation behaviour illustrates that the application framework is capable of considering just under 110 activities and scheduling between 5-7 of the most relevant activities within a reasonable response time on a resource-constrained mo-

bile device. The reconfiguration point identification mechanism was shown to correctly classify just over 83% of context events, with the majority of the incorrectly classified events not affecting the accuracy of the trail in relation to the user's contextual situation.

The trail quality study revealed that subjects typically spent almost 2 minutes solving an activity scheduling problem involving seven activities, and that 60% of subjects solved the problem correctly. The vast majority of subjects agreed that solutions to similar activity scheduling problems produced by the application framework were reasonable.

The following chapter summarises the most significant contributions of this thesis and its contributions to the state of the art. Related research issues that remain open for future work are also discussed.

# Chapter 6

## Conclusions and Future Work

The research presented in this thesis has investigated the development of an application framework for mobile, context-aware trails-based applications. More specifically, the research has focused on providing solutions to the challenges of trail generation and trail reconfiguration point identification that can be reused and extended by developers who wish to implement trails applications for deployment on mobile platforms. This chapter summarises the significant achievements of the work and its contributions to the state of the art, places them in a greater context, and outlines potential areas for future work relating to this thesis.

### 6.1 Achievements

The motivation for the work presented in this thesis arose from two observations on the state of the art research into context-aware activity scheduling for the mobile user. First, the approaches to trail generation and reconfiguration point identification used in existing applications are constrained by the number of activities they can consider and the number and types of contexts used to trigger reconfiguration respectively. Reconfiguration point identification techniques based on reasoning about specific context types e.g., location, are not scalable because they cannot consider different context types e.g., changes in activity availability, if/when they become available. Second, as a result of having research aims unrelated to providing generic support for mobile, context-aware activity scheduling, the research projects generally focus on the development of specific

applications as opposed to generic software to support the implementation of a range of similarly themed applications. While the existing application frameworks for mobile, context-aware application development discussed in Chapter 2 can be used to develop various types of applications e.g., mediascapes and museum guides, with partial activity scheduling support, none of them fully support trails application development.

To address these issues, this thesis presented an application framework composed of reusable and extensible trail generation and reconfiguration point identification behaviour. Chapter 3 describes how the behaviour in the application framework was designed to address the issues with the state of the art in relation to trail generation and reconfiguration point identification. An iterative, application-led design methodology was used to design generic, extensible approaches to trail generation and reconfiguration point identification. The trail generation mechanism uses context-based activity set pruning to reduce the number of activities considered during trail generation, and calculates a context-based relevance value for each activity. In cases where all activities cannot be scheduled due to response time requirements, the generated trail contains both scheduled and unscheduled activities, where the scheduled activities are those that are most relevant to the user based on the current context. This facilitates the implementation of applications that include a large number of activities. The reconfiguration point identification technique is based on the observation of differences between the set of activities scheduled on the current trail and the state of the activity set as a whole, ranked by relevance, following a context event. This generic approach facilitates the consideration of an extensible range of context types during reconfiguration point identification. The behaviour of the trail generation and reconfiguration point identification techniques can be customised by the developer or the user through the specification of preference values external to the application framework source code. The combination of the trail generation and reconfiguration point identification mechanisms proposed in this thesis has removed the restrictions associated with existing context-aware activity scheduling applications i.e., constrictive activity limits and context-specific reconfiguration point identification mechanisms, facilitating the development of trails applications that can consider a large number of activities and contexts during trail generation and reconfiguration point identification and can be deployed on mobile devices.

The implementation of the generic approaches to trail generation and reconfiguration point identification designed in Chapter 3 was described in Chapter 4, which illustrated how the application framework was implemented in a manner that facilitates both reuse of the base framework behaviour and the development of extensions to a number of areas of the framework. By implementing the trail generation and reconfiguration point behaviour in a reusable, extensible manner, the lack of generic support for mobile, context-aware activity scheduling application development in the state of the art research has been addressed. This supports developers in implementing trails applications without having to repeatedly address the common challenges associated with such applications.

The evaluation of the application framework was described in Chapter 5. The evaluation showed that the application framework is suitable for use as the basis to a range of mobile, context-aware applications that can generate and reconfigure trails in a manner that humans find reasonable. The evaluation also illustrated that the applications produced by the framework have the potential to save users a significant amount of time by relieving them of the burden of activity scheduling.

In summary, the research presented in this thesis has focused on investigating the provision of reusable and extensible techniques for trail generation and reconfiguration point identification to aid developers in implementing mobile, context-aware trails-based applications. The main contributions of this thesis can be summarised as follows:

- An overview of mobile, context-aware tourist guides, context-aware to-do lists and application frameworks for mobile, context-aware computing with respect to the provision of generic support for developing mobile, context-aware trails-based applications for deployment on mobile platforms.
- A user-preference driven approach to trail generation that uses context-based activity pruning and the notion of activity relevance to allow trails applications to consider a relatively large number of activities (between 100 and 110 when deployed on a HP iPAQ h6300 series PDA), with activities being selected for scheduling based on how relevant they are to the current contextual situation.
- A customisable approach to trail reconfiguration point identification that identifies significant context events that necessitate trail reconfiguration as they occur. The

evaluation of this technique illustrated that it correctly identifies just over 83% of context events, with the majority of the incorrectly classified events resulting in unnecessary reconfiguration and therefore not affecting the accuracy of the trail the user is following.

- An application framework that provides reusable, extensible implementations of the approaches to trail generation and reconfiguration point identification. The evaluation illustrated that the framework can be used in numerous ways, ranging from direct reuse to major extension. Each of the application development case studies discussed had a code reuse level of over 84%.
- A demonstration of the application framework's ability to serve as the basis to a range of mobile, context-aware trails-based applications that are capable of generating and reconfiguring trails in a reasonable manner. 93.75% of the subjects that took part in the trail quality experiment agreed that the trails generated by the application framework were reasonable.

## 6.2 Perspective

Mark Weiser, often referred to as the father of ubiquitous computing<sup>1</sup>, envisioned a world in which computing technology would weave itself into the fabric of everyday life until it became indistinguishable from it [131]. He proposed that computing technology would go through a similar process to that of writing, which he calls the first form of information technology. The constant presence of the written word does not require active attention from humans, but the information to be conveyed is ready for use at a glance. This is a result of people learning to read to a standard that allows them to cease to be aware that they are doing it. Weiser believed that the way to achieve a similar effect in relation to computing technology was to integrate computing technology into the world at large by making it available to people at all times. This would allow people to become more familiar with using computers of all types e.g., mobile devices and communal digital

---

<sup>1</sup>Ubiquitous computing integrates computation into the environment in the hope that this will enable people to interact with information-processing devices more naturally and casually than they currently do, and in ways that suit the context they find themselves in.



whiteboards, and to get on with achieving what they want to do.

Over the period in which the research described in this thesis has been conducted, Weiser's vision of the 'disappearing computer' has moved closer to becoming a reality. When the Hermes project began it was thought that trails applications would execute on PDAs, and therefore the relevance of the application framework discussed in this thesis was somewhat dependent on the widespread adoption of PDA-type mobile devices. It transpired that the general public did not adopt the PDA to the extent that the associated industry had hoped. However, while the PDA was failing to become ubiquitous, the mobile phone was becoming increasingly popular. The Irish mobile penetration rate<sup>2</sup> is currently 106% [42] and the average European penetration rate is over 100% [92]. These penetration rates illustrate that people are comfortable carrying and using mobile devices that, while perceived to be telephones, offer both telephonic and non-telephonic services such as text messaging and games. The rise in popularity of the mobile phone has resulted in mobile handsets becoming more and more technically sophisticated, to the point where they now have the capabilities of PDAs. For example, Apple Inc. have recently announced the iPhone [56], a powerful smart phone with a large graphical display, integrated wireless networking (including WiFi and Bluetooth) and support for Java applications and Google Maps<sup>3</sup>. Additionally, it is predicted that over the next two years, GPS will become a common feature in mobile phones, with eighty three million GPS-enabled handsets shipped in the last year alone [104]. The widespread adoption of mobile phones naturally provides an ideal environment for the development and deployment of context-aware applications.

Despite the progress towards Weiser's vision in terms of hardware, building context-aware applications for mobile users remains a significant undertaking. This is evidenced by the amount of research into framework support for such applications e.g., the work described in this thesis and the related work discussed in Section 2.3. A bird's eye view of the work described in this thesis is therefore that it has the potential to help software developers take advantage of the opportunities presented by the emergence of sophisticated mobile devices that provide wireless networking, positioning and mapping

---

<sup>2</sup>The mobile penetration rate is calculated based on the number of mobile subscribers and the population size. It must be noted that some subscribers may have more than one active SIM card.

<sup>3</sup><http://maps.google.com>

in a format that people have already adopted. It is expected that access to an application framework for developing trails applications will help developers to prototype, test and deploy trails applications in a reasonable amount of time, facilitating further exploration of the potential of context-aware activity scheduling for mobile users.

The widespread applicability of trail-based applications is evidenced by the existence of many commonly assumed business-related roles in which activity scheduling in a dynamic environment is an inherent requirement. Context-based activity scheduling is an aspect of the work conducted by individuals in workplaces such as hospitals (scheduling patient rounds and administrative tasks), warehouses (managing the order in which requests for items are fulfilled), hotels (managing the order in which rooms are serviced/cleaned) and prisons (managing the order in which inmates are monitored by guards). Those working in professions that involve greater mobility e.g., mobile salespeople and tradespeople (plumbers, electricians, office equipment technicians), on call care givers (doctors, veterinarians), taxi drivers and mobile delivery personnel (parcel/food/flower delivery couriers) also manage their working lives by using relevant context to schedule their pending and emergent activities. Away from the business world, context-based activity scheduling is used informally by many. At the simplest level, people use context to manage their day-to-day activities. Context-based activity scheduling is also used by people in more specific leisure-related situations such as when sightseeing, attending a music festival, playing treasure hunt-type games, visiting a theme park or going shopping at a particularly busy time e.g., Christmas time. The pervasiveness of the mobile phone and its recent technical advancement, combined with the trails-based application development support presented in this thesis, creates an environment in which computer support for context-based activity scheduling can be realised to support users in both business and leisure scenarios.

### **6.3 Future Work**

Throughout the process of designing, implementing and evaluating the application framework presented in this thesis, a number of issues worthy of further investigation were identified. This work relates to distributed trail generation, trail robustness and activity

dependencies and constraints.

### 6.3.1 Distributed Trail Generation

The application framework is designed to execute on mobile devices so that the trails service is not affected by wireless network disconnection. However, in situations where wireless network connectivity is available, it would be interesting to investigate the possibility of taking advantage of the increased processing power of remote servers. The trail generation algorithm is designed in a scalable manner, making it suitable for execution on a resource-rich platform without modification. This behaviour could be deployed as follows. The trail generation algorithm would reside on a remote server, with the subtrail size increased to take advantage of the server's processing power. The reconfiguration point identification mechanism would execute on the mobile device and trigger the upload of the activity set to the remote server when reconfiguration is deemed necessary. The remote server would generate a trail from the activities received and return it to the mobile device.

While the completion of this work would be relatively straightforward from a technical perspective, it raises the issue of data privacy. Privacy has long been noted as one of the major issues in mobile, context-aware computing [132, 111], and remains a popular research area e.g., [137, 21, 6, 108, 27, 102, 49]. As sending the activity set to a remote server involves sending details of what the user plans to do in the future, as well as other information about the user such as their current location, the privacy issue would need to be suitably addressed in order for distributed trail generation to be effective.

### 6.3.2 Trail Robustness

The robustness of a solution to any problem based on dynamic variables refers to how sensitive the solution is to minor environmental fluctuations. The concept of route robustness has been studied in mobile, ad hoc networking [126] and vehicle route generation [48, 62] in order to extend the lifetime of an individual route, therefore increasing its utility. Robustness is a desirable trail characteristic as it reduces the frequency of trail reconfiguration triggered by what users may perceive to be trivial context events. For

example, it is likely that a trail would not be considered robust if it reconfigured seconds after it had been generated because the user had not proceeded as expected. During the trail quality experiment it emerged that subjects in Group 2 were dissatisfied with the fragile nature of the computer generated solution they were shown (Section 5.3.2). This was addressed in the application framework with the addition of two activity attributes - estimated duration leeway and closing time leeway. This solution is satisfactory because application framework activities are subject to temporal constraints only. It would be interesting to further explore the issue of trail robustness and investigate a robustness mechanism that considers non-temporal as well as temporal activity constraints. An example of an activity with a non-temporal constraint is going to see a movie. The activity will eventually become impossible when the theatre reaches capacity attendance.

### 6.3.3 Activity Dependencies and Constraints

An activity constraint is a restriction set on the start and/or finish date of an activity. The activity model in the application framework uses activity opening and closing hours to model activity constraints. Therefore, by setting appropriate activity opening and closing times the framework can model common inflexible constraints such as ‘Must Start On’, ‘Must Finish On’, ‘Finish No Earlier Than’, ‘Finish No Later Than’, ‘Start No Earlier Than’ and ‘Start No Later Than’ [101]. However, the application framework does not support the specification of flexible activity constraints such as ‘As Soon As Possible’ and ‘As Late As Possible’.

A dependency between two activities exists when the start or end date of one activity is constrained by the start or end date of another activity. The activity model in the application framework does not explicitly cater for dependency relationships such as ‘Activity B can start *only* after Activity A has been completed’ and ‘Activity B *must* start directly after Activity A has been completed’. Dependency relations can be loosely modelled using activity priority, which can represent relative importance relationships between activities. Activity obligation (the designation of activities as either mandatory or optional) can also be used to loosely model activity dependency. However, neither of these approaches is guaranteed to enforce dependency relationships.

The activity model in the application framework could be extended to provide full support for activity constraints and dependencies. Such an extension would give developers the capability to define activity sets with more complex relationships between constituent activities. It would also provide more control over the trail generation process i.e., developers/users could specify partial trail orderings that would be maintained regardless of the context.

## 6.4 Chapter Summary

This chapter summarised both the motivations for and most significant achievements of the work presented in this thesis. In particular, it outlined how this work has contributed reusable and extensible approaches to trail generation and trail reconfiguration point identification that aid developers of mobile, context-aware trails-based applications. The chapter also placed the contributions of this thesis in a greater context and made suggestions for possible future work arising from the research undertaken in relation to this thesis.

# Appendix A

## User Study Results

### A.1 Oisín goes to Trinity - User Study Results

The results of the trail generation and reconfiguration questions from the ‘Oisín goes to Trinity’ user study questionnaire are listed below.

**Q1:** Did you notice the order of the activities change while you were using the application?

- Yes - 76%
- No - 5%
- Not Sure - 19%

**Q2:** Were you anticipating that the trail would be automatically reordered when it was?

- Yes, all of the time - 21%
- Yes, most of the time - 42%
- Some of the time - 16%
- No, not really - 11%
- No, not at all - 11%

**Q3:** Did you agree with the reordering decisions made by the application?

- Yes, all of the time - 55%
- Yes, most of the time - 28%
- Some of the time - 6%
- No, not really - 11%

**Q4:** The trail reordering decisions were better than those you could have made yourself?

- Totally agree - 28%
- Partially agree - 17%
- Neither agree or disagree - 44%
- Partially disagree - 0%
- Totally disagree - 11%

**Q5:** How did you feel about the amount of control the application had when reordering your trail?

- Far too much - 6%
- A bit too much - 11%
- OK - 60%
- A bit too little - 17%
- Far too little - 6%

# Appendix B

## Further Implementation Detail

### B.1 GPS Location Context

Figure B.1 illustrates the classes used to implement the GPS version of `LocationGenerator`. The `ContextGenerator`, `LocationGenerator` and `Subject` classes are the same as those discussed in Section 4.1. The remaining classes in Figure B.1 form part of the GIS world model component of the Hermes framework. These classes have been made available to the application framework so that it can be used in isolation from the Hermes framework if desired.

The `doLocationChange()` method is illustrated in Listing B.1. Line 2 creates a `Connection` object and lines 3-11 use this object to establish a `BufferedReader` object that is used to read GPS data from the mobile device's COM port. Line 18 sees the buffered reader instance (`gpsReader`) reading a line of GPS data. A check occurs to assess if the line contains latitude and longitude information, as GPS devices can deliver a range of information e.g., the current time and the direction the user is heading.

`GPSConversion` contains a `String` parsing method to evaluate lines of GPS data. If the current line contains the desired location data, the `convertToXY()` method of `GPSConversion` is used to convert the GPS coordinates to X, Y coordinates on the map-based interface used by the application.

---

Listing B.1: The `doLocationChange()` method in the `LocationGenerator` class

---

```
1 public void doLocationChange () {
```



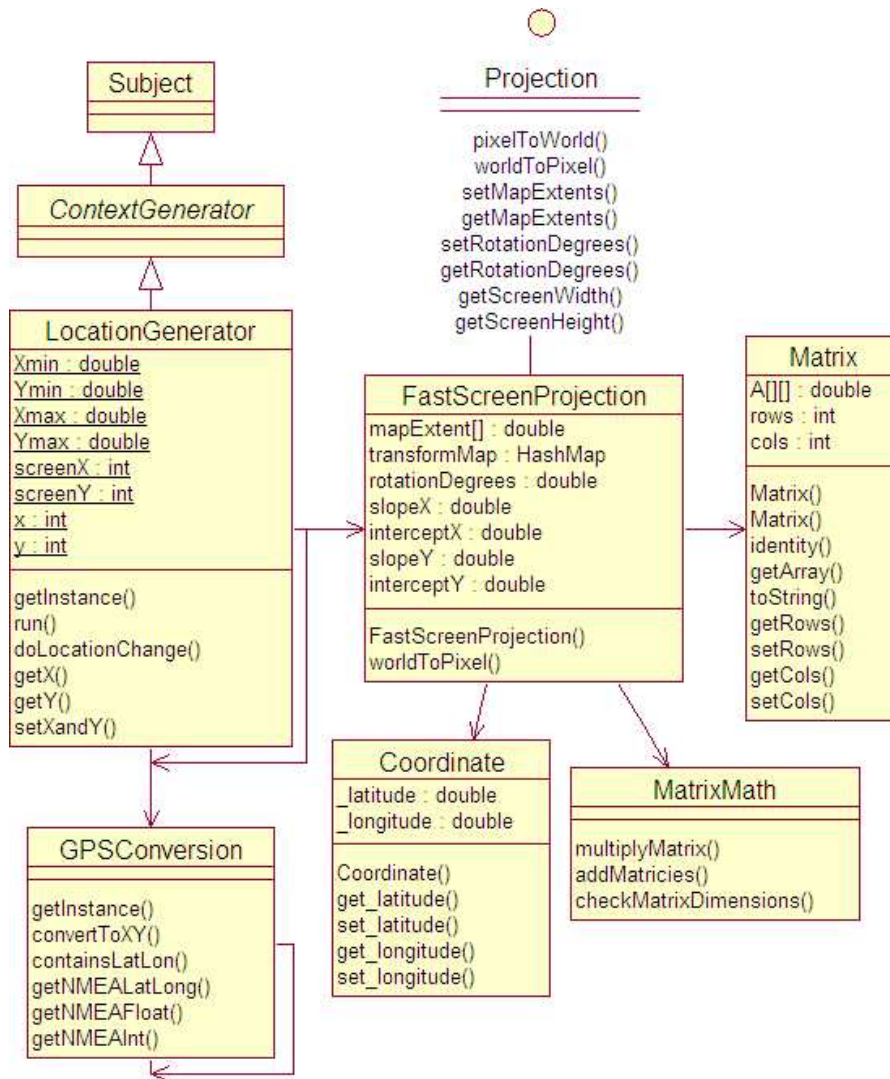


Figure B.1: GPS location context class diagram

```

2   Connection connection = new Connection();
3   try {
4       String[][] parameters = {"baudrate", (new Integer(9600)).
5           toString()});
6       connection.setParameters("1", parameters, 3, true);
7       InputStream inputStream = connection.openInputStream();
8       InputStreamReader inputStreamReader = new InputStreamReader
9           (inputStream);
10      BufferedReader gpsReader = new BufferedReader
11          (inputStreamReader);
12

```

```

13     String sentence = new String();
14     boolean noMoreGPSData = false;
15     while (!noMoreGPSData) {
16         if (!paused) {
17             try {
18                 sentence = gpsReader.readLine();
19                 if (sentence != null) {
20                     if (GPSConversion.getInstance().containsLatLon
21                         (sentence)){
22                         int [] xy = GPSConversion.getInstance().
23                             convertToXY(sentence, Xmin, Ymin, Xmax,
24                                 Ymax, screenX, screenY);
25                         if (x != xy[0] && y != xy[1]) {
26                             x = xy[0];
27                             y = xy[1];
28                             setXandY(x, y);
29                         }
30                     }
31                 }
32             } catch (IOException e) {...}
33         }
34     } catch (Exception e) {...}
35 }

```

---

The `convertToXY()` method of `GPSConversion` is illustrated in Listing B.2. The method takes the following arguments:

- **GPSSentence** - a GPS sentence containing latitude and longitude information.
- **Xmin** - the longitude reading at the real-world position represented by the top left-hand corner of the map being used.
- **Ymin** - the latitude reading at the real-world position represented by the top left-hand corner of the map being used.

- `Xmax` - the latitude reading at the real-world position represented by the bottom right-hand corner of the map being used.
- `Ymax` - the latitude reading at the real-world position represented by the bottom right-hand corner of the map being used.
- `screenX` - the width of the screen on the device the application is being deployed on.
- `screenY` - the height of the screen on the device the application is being deployed on.

Lines 4-7 generate two float variables from the GPS sentence using the `getNMEALatLong()` method from `GPSConversion`. This method extracts latitude and longitude information from a GPS sentence and converts it to decimal format. The `worldToPixel()` method from `FastScreenProjection` is then used (line 12) to convert the GPS data to X, Y coordinates. This method takes the screen dimensions and the map extents as parameters. The `worldToPixel()` method uses linear regression convert the GPS coordinates to X, Y screen coordinates. An `int` array of size 2 containing the X, Y values is returned on line 13.

Line 25 in Listing B.1 contains a check on the new X, Y values. If they are the same as the X, Y values already held by `LocationGenerator` then the new information is ignored. If either the X or Y value differs from the X or Y value already held then the `x` and `y` attributes in `LocationGenerator` are updated using the `setXandY()` method (line 28). This method invokes the `notifyObservers()` method that informs `ReconfigurationEngine` of the user's new real-world position.

---

Listing B.2: The `convertToXY()` method in the `GPSConversion` class

---

```

1 public int [] convertToXY(String GPSSentence, double Xmin, double
2   Ymin, double Xmax, double Ymax, int screenX, int screenY) {
3
4   double north = getNMEALatLong(GPSSentence.substring(7, 16),
5     GPSSentence.substring(18, 19));

```

```
6     double west = getNMEALatLong(GPSSentence.substring(19, 39),
7         GPSSentence.substring(32, 33));
8
9     Dimension d = new Dimension(X, Y);
10    double [] gps = new double [] { Xmin, Ymin, Xmax, Ymax };
11    FastScreenProjection fsp = new FastScreenProjection(d, gps);
12    int [] xy = fsp.worldToPixel(north, west);
13    return xy;
14 }
```

---

# Appendix C

## Trail Quality Experiment Materials

### C.1 Information Sheets

#### C.1.1 Group 1

Thank you for taking part in this experiment. It will take approximately 15 minutes. If you have any questions please feel free to ask.

A trail is an ordered collection of activities. Activities are ordered based on their various properties e.g., priority, opening hours, whether they are mandatory or optional, and their relationship to the person undertaking the activities e.g., proximity. A trail ordering aims to make the maximum number of activities possible while reducing the total distance the user must cover, and the time the user must spend, to complete the trail activities.

The experiment consists of two parts. In part one you will be presented with a map of the Trinity College Dublin campus annotated with a number of activities. You are required to:

1. *Identify the relevant activities based on their properties.* The relevant activities are those that are currently possible to complete, regardless of ordering. An activity is impossible if its opening hours have passed (or will pass before the activity can be completed) or it clashes with a more important activity.
2. *Order the activities.* The aim is to maximise the number of activities possible while

minimising the cost in terms of time and distance required.

You will then be shown another version of the map in which some activities have been completed. An unexpected event has occurred which may affect your ordering. You are required to:

- *Reorder the activities with respect to the unexpected event.*

In part two you will be shown a map of the Trinity College Dublin campus annotated with a computer-generated trail consisting of a number of activities (different to those in part one). You are required to:

- *Validate that the computer-generated trail is reasonable.* The impossible activities are identified and marked as impossible. Each possible activity is marked with a sequence number. Collectively these numbers indicate the order in which the activities will be completed. Estimated start and end times are shown, as well as paths between the activities.

You will then be shown another version of the map in which the some activities have been completed and an unexpected event has occurred. The trail has been reordered with respect to this event. You are required to:

- *Validate that the reordered trail is reasonable.*

You will be asked to fill out a short questionnaire following the completion of part two. All tasks will be timed.

### **C.1.2 Group 2**

Thank you for taking part in this experiment. It will take approximately 15 minutes. If you have any questions please feel free to ask.

A trail is an ordered collection of activities. Activities are ordered based on their various properties e.g., priority, opening hours, whether they are mandatory or optional, and their relationship to the person undertaking the activities e.g., proximity. A trail ordering aims to make the maximum number of activities possible while reducing the

total distance the user must cover, and the time the user must spend, to complete the trail activities.

The experiment consists of two parts. In part one you will be shown a map of the Trinity College Dublin campus annotated with a computer-generated trail consisting of a number of activities. You are required to:

- *Validate that the computer-generated trail is reasonable.* The impossible activities are identified and marked as impossible. Each possible activity is marked with a sequence number. Collectively these numbers indicate the order in which the activities will be completed. Estimated start and end times are shown, as well as paths between the activities.

You will then be shown another version of the map in which some activities have been completed. An unexpected event has occurred. The trail has been reordered with respect to this event. You are required to:

- *Validate that the reordered trail is reasonable.*

You will be asked to fill out a short questionnaire following the completion of part one.

In part two you will be presented with a map of the Trinity College Dublin campus annotated with just activities (different to those used in part one), no trail. You are required to:

1. *Identify the relevant activities based on their properties.* The relevant activities are those that are currently possible to complete, regardless of ordering. An activity is impossible if its opening hours have passed (or will pass before the activity can be completed) or it clashes with a more important activity.
2. *Order the activities.* The aim is to maximise the number of activities possible while minimising the cost in terms of time and distance required.

You will then be shown another version of the map in which some activities have been completed. An unexpected event has occurred which may affect your ordering. You are required to:

- *Reorder the activities with respect to the unexpected event.*

All tasks will be timed.

## C.2 Activity Scheduling Problems

### C.2.1 Activity Scheduling Problem Legend













-  *High Priority Activity* (can also contain a number indicating its position in a trail e.g.,  is done 3<sup>rd</sup>)
  -  *Medium Priority Activity* (can also contain a number indicating its position in a trail e.g.,  is done 1<sup>st</sup>)
  -  *Low Priority Activity* (can also contain a number indicating its position in a trail e.g.,  is done 4<sup>th</sup>)
  -  *Mandatory High Priority Activity* (must be done if possible, overrides priority)
  -  *Activity Not Possible* (low priority activity shown, though any activity can be impossible)
  -  *Activity Not Possible due to Clash* (low priority activity shown, any activity can be impossible due to a clash)
  -  *Activity Completed* (high priority activity shown, though any activity can be completed)
  -  *Your current location* (the location of this icon on the map represents your real-world position)
- Activity Details Box* (describes details of an activity – description, opening hours and estimated duration)
- |                      |
|----------------------|
| Activity Description |
| Estimated Duration   |
| Opening Hours        |
- Trails Activity Details Box* (describes details of a trail activity – adds estimated start and end time to the above info)
- |                          |
|--------------------------|
| Activity Description     |
| Estimated Duration       |
| Opening Hours            |
| Estimated Start/End Time |
- New Context Info* (Contains a message to describe the new information learned. Necessitates activity reordering)
- |   |
|---|
|  |
| Message   |
- The current time in each scenario is shown at the top right of the map

Figure C.1: Activity scheduling problem legend



## C.2.2 Example Problem

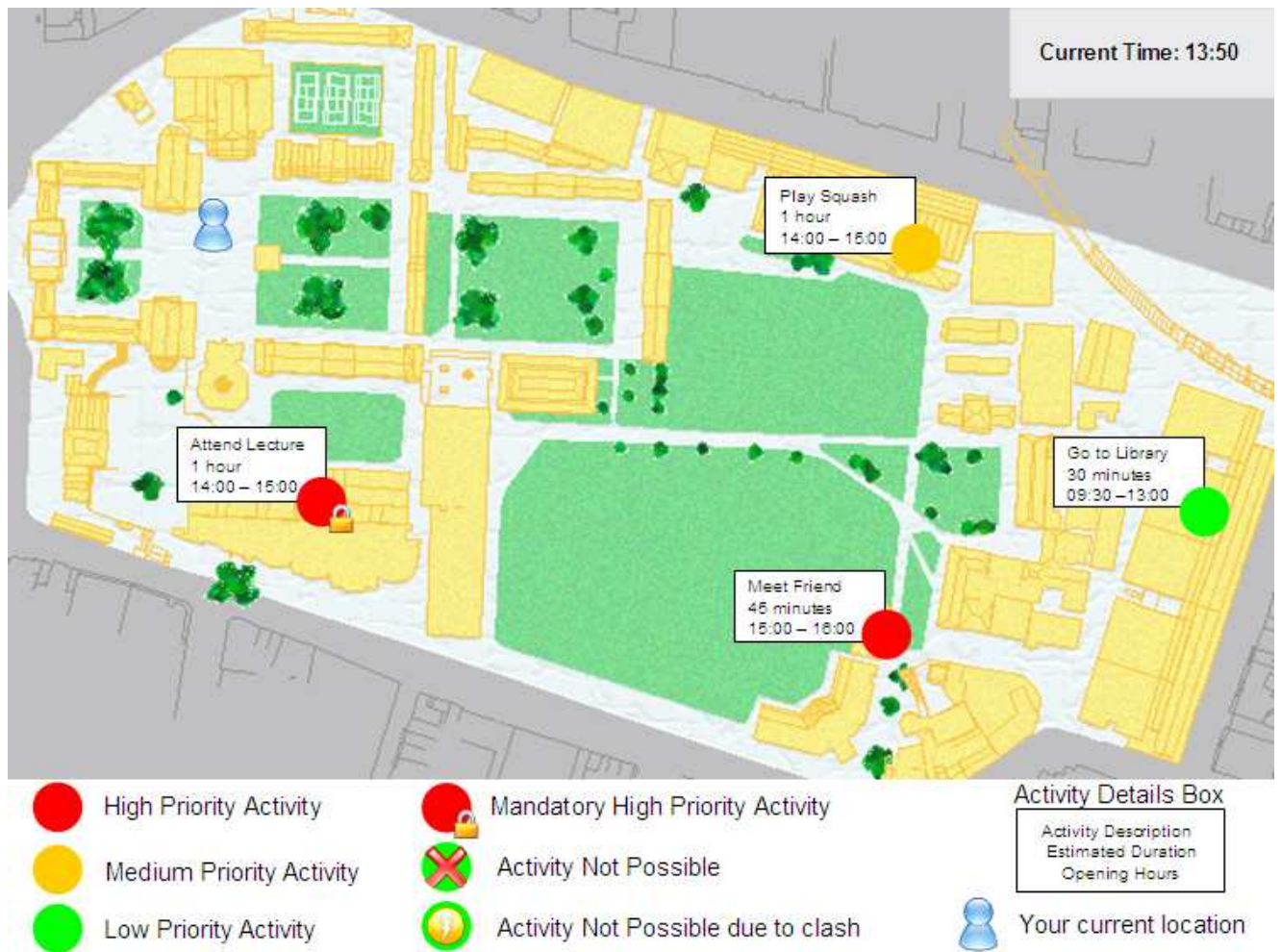


Figure C.2: Example activity scheduling problem

### C.2.3 Example Solution

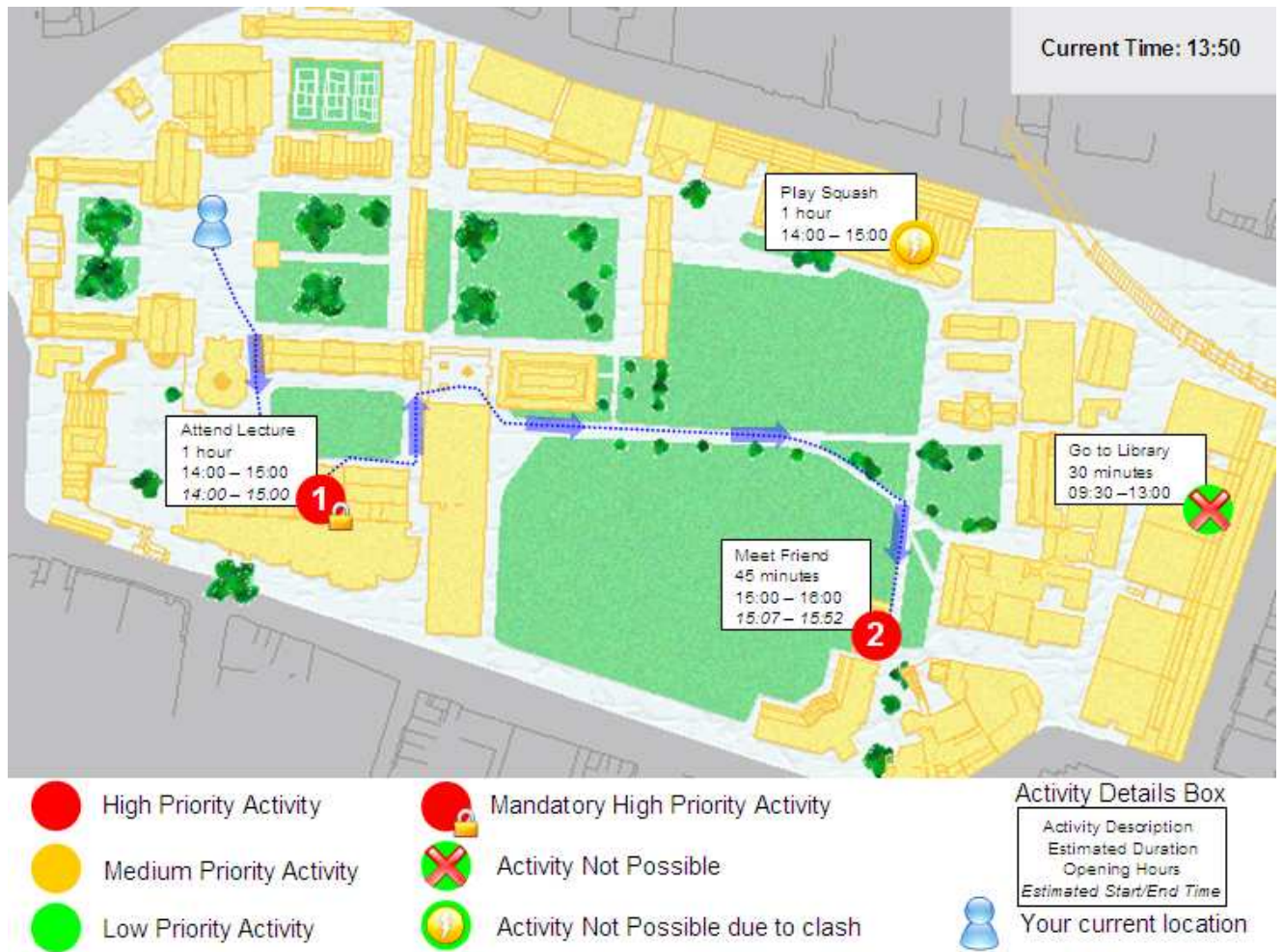


Figure C.3: Example activity scheduling problem solution

## C.2.4 Group 1: Problem 1

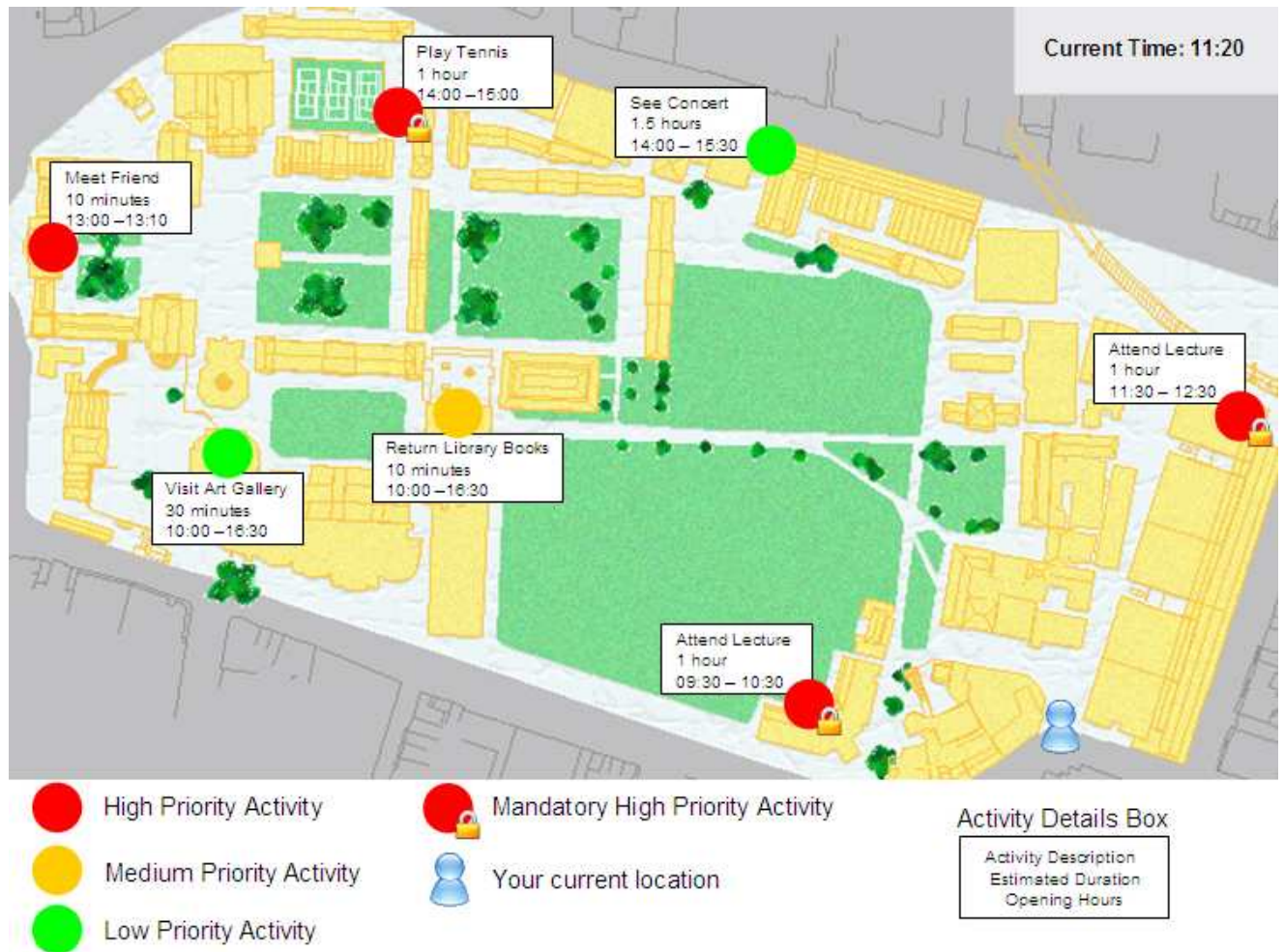


Figure C.4: Group 1: Activity scheduling problem 1

## C.2.5 Group 1: Problem 2

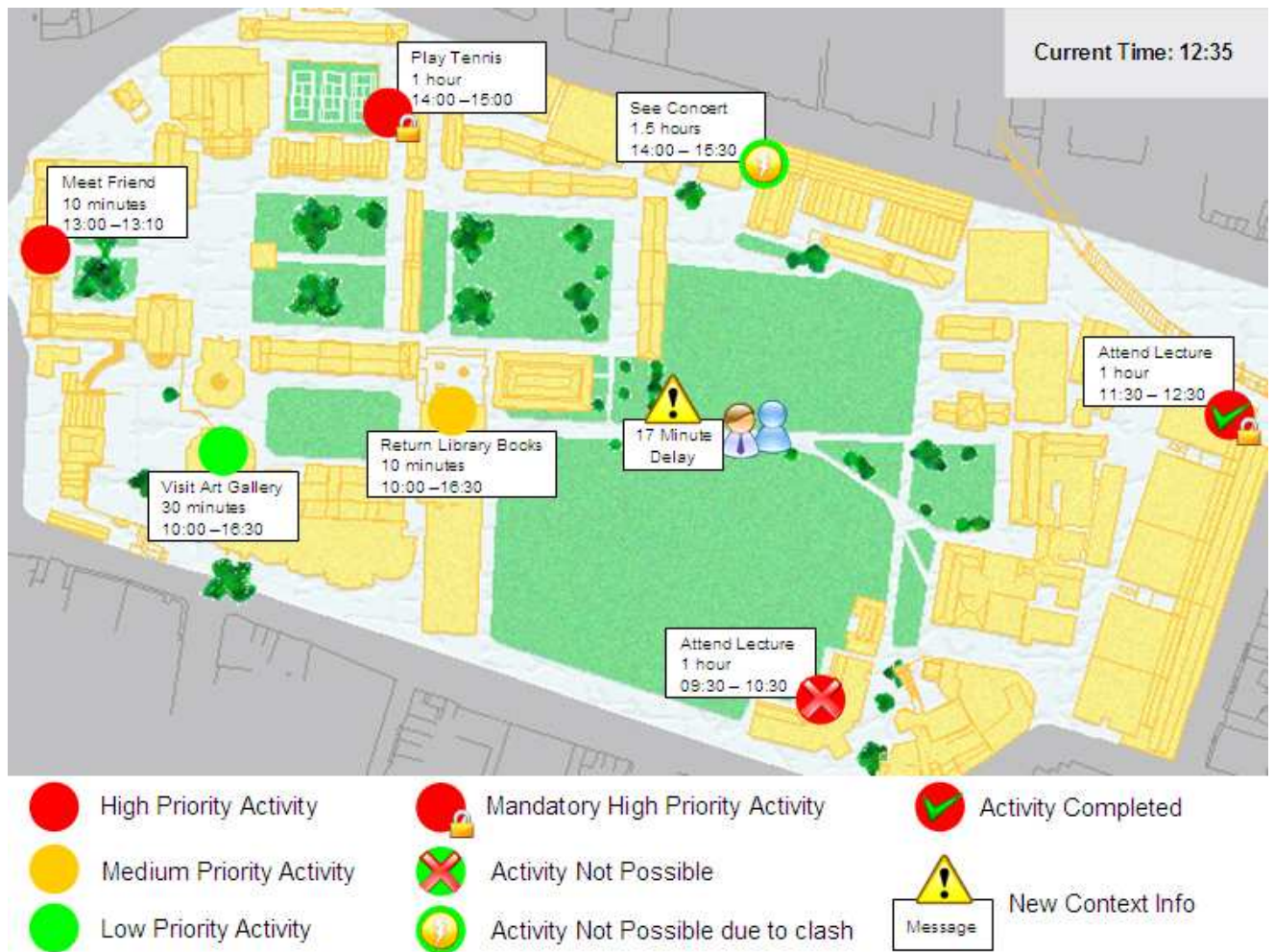


Figure C.5: Group 1: Activity scheduling problem 2

## C.2.6 Group 1: Solution 1

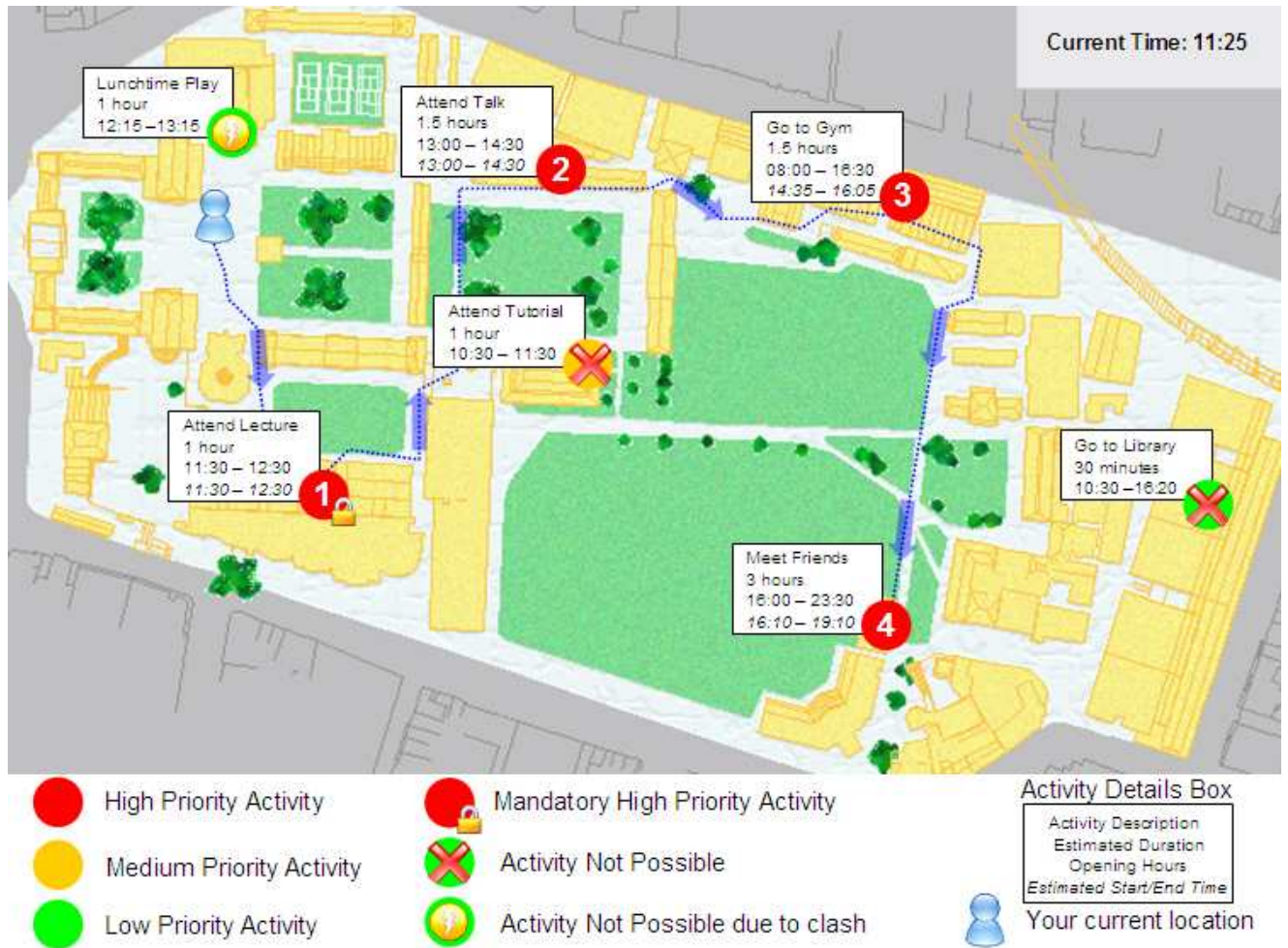


Figure C.6: Group 1: Activity scheduling problem solution 1

### C.2.7 Group 1: Solution 2

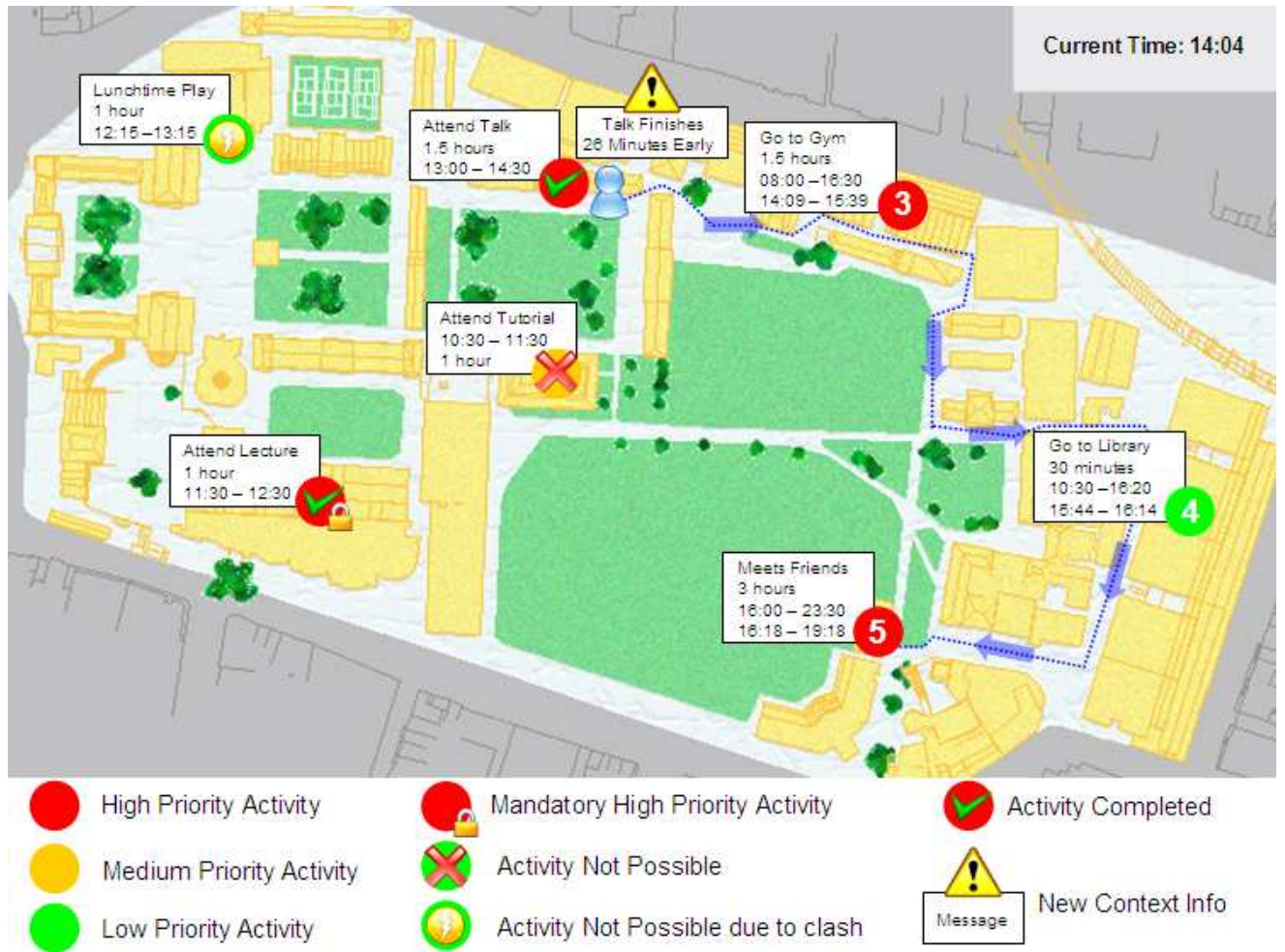


Figure C.7: Group 1: Activity scheduling problem solution 2

## C.2.8 Group 2: Solution 1

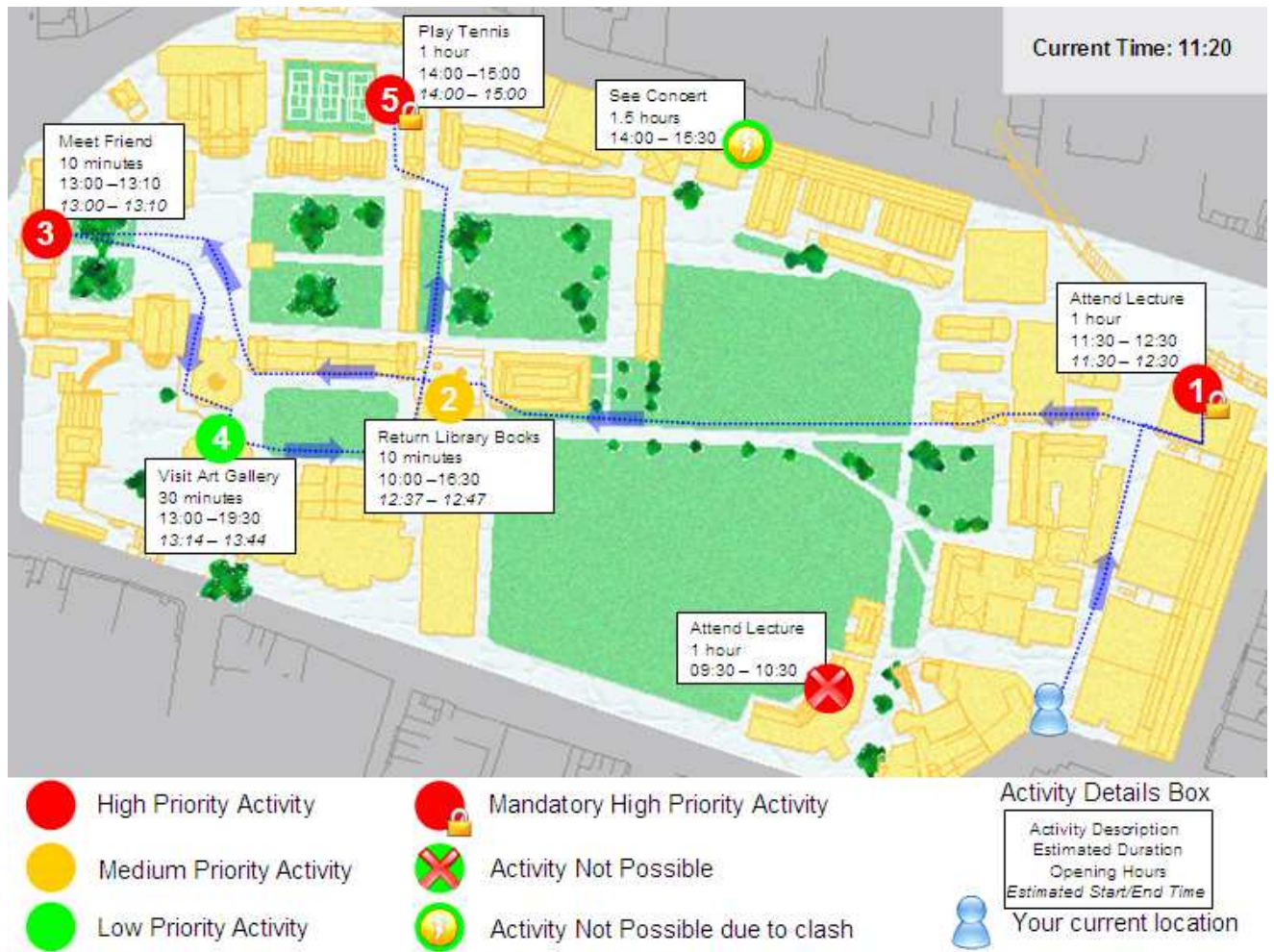


Figure C.8: Group 2: Activity scheduling problem solution 1

### C.2.9 Group 2: Solution 2

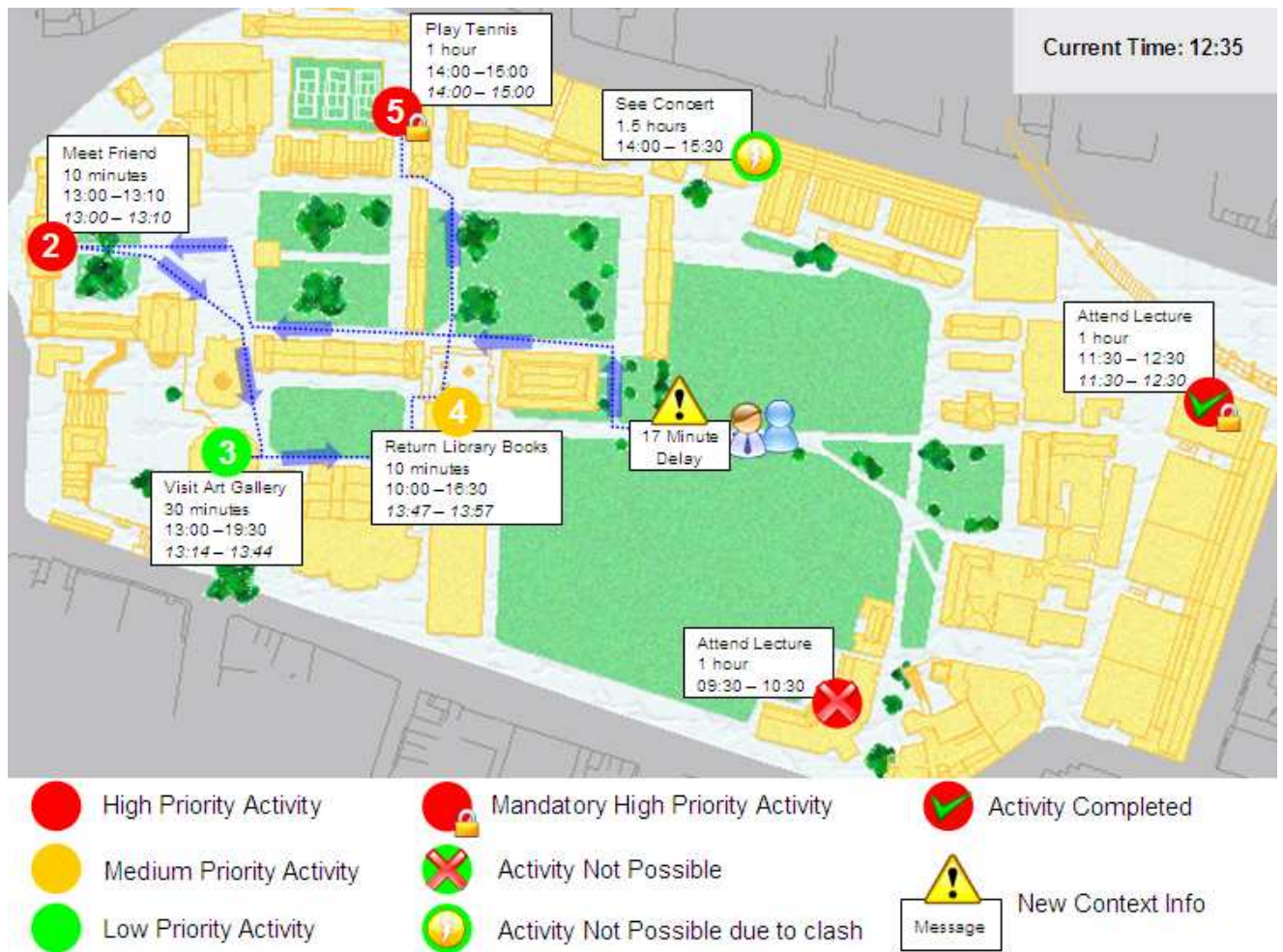


Figure C.9: Group 2: Activity scheduling problem problem 2



## C.2.10 Group 2: Problem 1

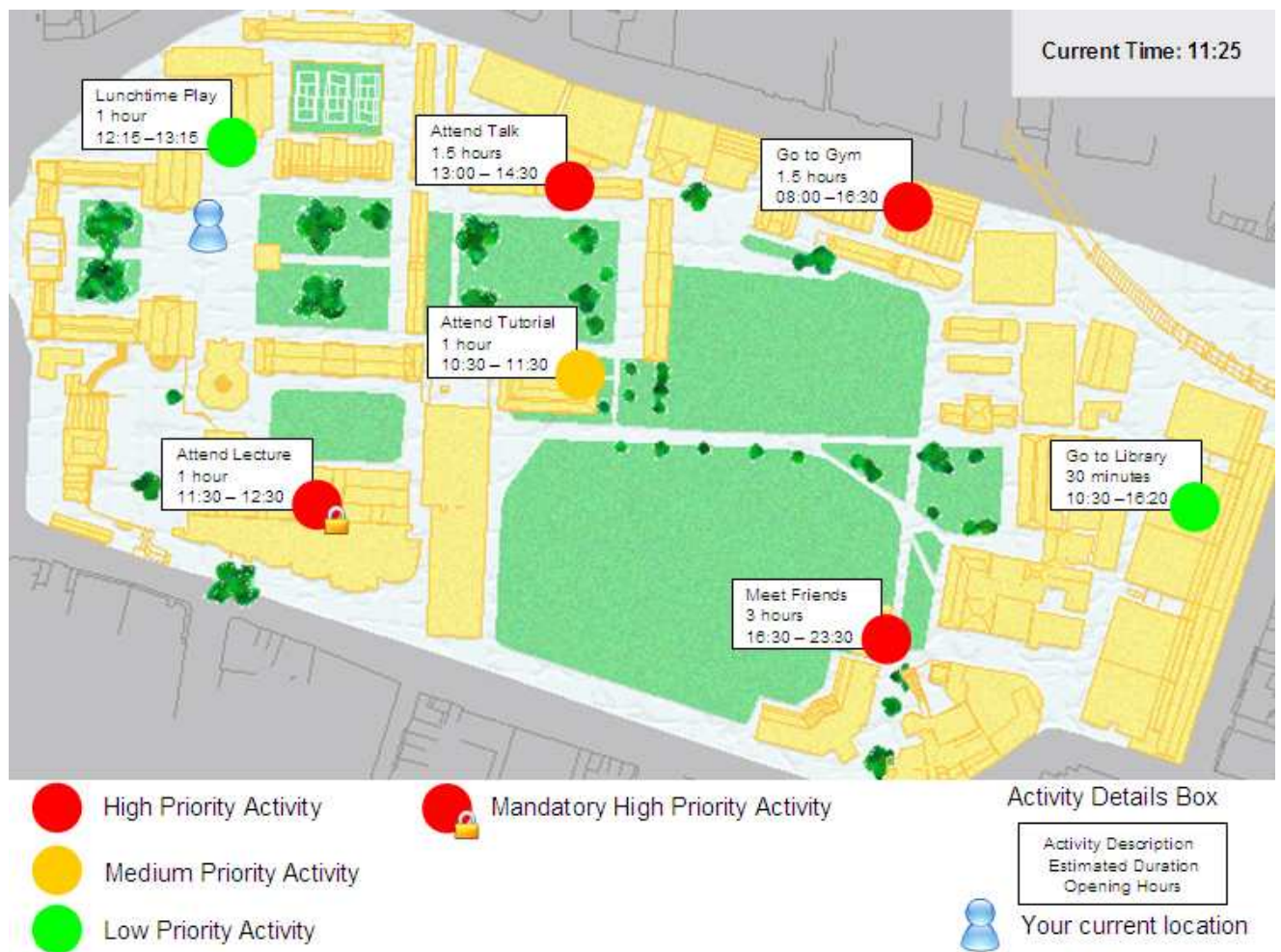


Figure C.10: Group 2: Activity scheduling problem 1

## C.2.11 Group 2: Problem 2

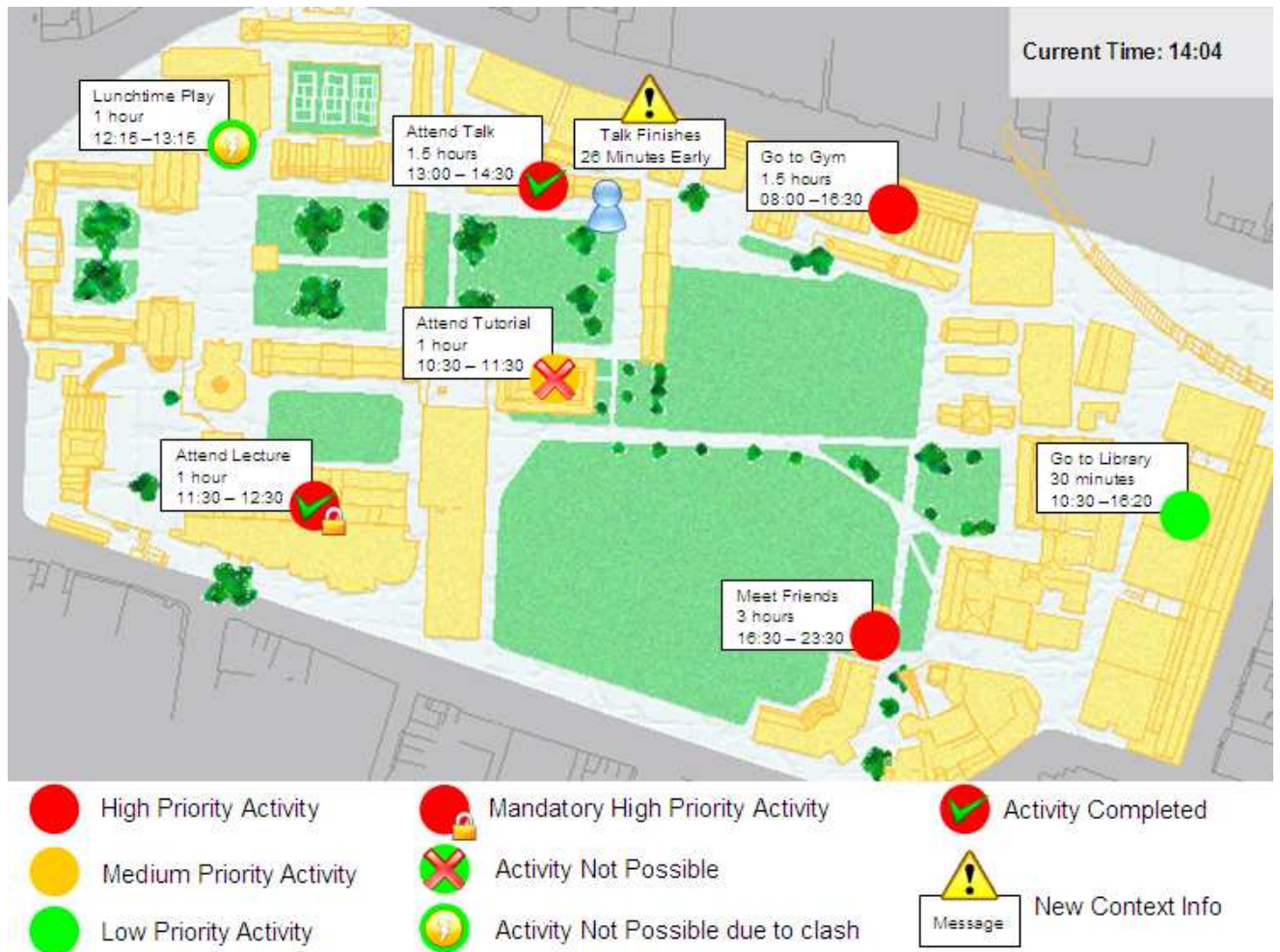


Figure C.11: Group 1: Activity scheduling problem 2

## C.3 Questionnaire

### Trails Decision Validation – please give your opinion on the statement below

S1: The first trail shown was reasonable.

- Totally agree
- Partially agree
- Neither agree or disagree
- Partially disagree
- Totally disagree

If you did not fully agree with the statement above please explain your reason(s) below.

---

---

---

---

---

### Reconfiguration Decision Validation – please give your opinion on the statement below

S2: The second, reordered trail was reasonable.

- Totally agree
- Partially agree
- Neither agree or disagree
- Partially disagree
- Totally disagree

If you did not fully agree with the statement above please explain your reason(s) below.

---

---

---

---

---

Figure C.12: The activity scheduling problem solution validation questionnaire

# Bibliography

- [1] 37signals. Ta-da Lists. Online; accessed 11-September-2006. <http://www.tadalist.com/>.
- [2] Robert P. Abelson and Ariel Levi. *The Handbook of Social Psychology, Vol. 1*, pages 231–309. Random House, 1985.
- [3] Gregory Abowd, Christopher Atkeson, Jason Hon, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3:421–433, 1997.
- [4] Anand Agarawala, Saul Greenberg, and Geoffrey Ho. The Context-Aware Pill Bottle and Medication Monitor. Technical Report 2004-752-17, Department of Computer Science, University of Calgary, 2004.
- [5] AKS-Labs. Outlook Task. Online; accessed 11-September-2006. <http://www.outlook-task.com/>.
- [6] Jalal Al-Muhtadi, Raquel Hill, Roy Campbell, and M. Dennis Mickunas. Context and Location-Aware Encryption for Pervasive Computing Environments. In *Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW '06)*, page 283, Washington, DC, USA, 2006. IEEE Computer Society.
- [7] Sahin Albayrak and Ralf Sessler. Serviceware Framework for Developing 3G Mobile Devices. In *Proceedings of the 16th International Symposium on Computer and Information Sciences (ISCIS 2001)*. Isik University Publications, 2001.

- [8] Hermann Anegg, Harald Kunczler, Elke Michlmayr, Günther Pospischil, and Martina Umlauf. LoL@: designing a location based UMTS application. *e&i - elektrotechnik & informationstechnik*, 119(2), February 2002.
- [9] Mathias Baldauf and Schahram Dustdar Florian Rosenberg. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2006.
- [10] Jakob E. Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proceedings of the 3rd International Conference on Pervasive Computing (PERVASIVE 2005)*. Springer-Verlag, 2005.
- [11] John Barton and Vikram Vijayaraghavan. UBIWISE: A Ubiquitous Wireless Infrastructure Simulation Environment. Technical Report HPL-2002-303, HP Labs, Palo Alto, CA, USA, 2002.
- [12] Francesco Bellotti, Riccardo Berta, Alessandro De Gloria, and Massimiliano Margarone. User Testing a Hypermedia Tour Guide. *IEEE Pervasive Computing*, 1(2): 33–41, 2002.
- [13] Arnold Bennett. *How to Live on 24 Hours a Day*. George H. Doran, New York, 1910.
- [14] Gregory Biegel and Vinny Cahill. A Framework for Developing Mobile, Context-aware Applications. In *Proceedings of the 2nd Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '04)*, page 361, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- [15] Soranna-Daniela Bolboaca and Lorentz Jantschi. Pearson versus Spearman, Kendall's Tau Correlation Analysis on Structure-Activity Relationships of Biologic Active Compounds. *Leonardo Journal of Sciences*, 9:179–200, 2006.
- [16] Craig Boutilier, Ronen Brafman, Christopher Geib, and David Poole. A Constraint-Based Approach to Preference Elicitation and Decision Making. In Jon Doyle and Richmond H. Thomason, editors, *Working Papers of the AAAI Spring Symposium*

- on Qualitative Preferences in Deliberation and Practical Reasoning*, pages 19–28, Menlo Park, California, 1997. American Association for Artificial Intelligence.
- [17] Niels Olof Bouvin, Bent G. Christensen, Kaj Grønbaek, and Frank Allan Hansen. HyCon: a framework for context-aware mobile hypermedia. *Hypermedia*, 9(1):59–88, 2003. ISSN 0955-8543.
- [18] Peter J. Brown. The Stick-E Document: A Framework for Creating Context-Aware Applications. In *Proceedings of the 6th International Conference on Electronic Documents, Document Manipulation, and Document Dissemination (EP '96)*, pages 259–272. John Wiley & Sons, 1996.
- [19] T. W. Butler. Computer response time and user performance. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '83)*, pages 58–62, New York, NY, USA, 1983. ACM Press.
- [20] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The Information Visualizer, an Information Workspace. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '91)*, pages 181–186, New York, NY, USA, 1991. ACM Press.
- [21] Roberto Speicys Cardoso and Valerie Issarny. Architecting Pervasive Computing Systems for Privacy: A Survey. In *The Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, 2007.
- [22] Guanling Chen, Ming Li, and David Kotz. Design and Implementation of a Large-Scale Context Fusion Network. In *Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous '04)*, volume 00, pages 246–255, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- [23] Peter Y. Chen and Paula M. Popovich. *Correlation: Parametric and Nonparametric Measures*. Sage Publications Inc, 2002.
- [24] 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 (MobiCom '00)*, pages 20–31, New York, NY, USA, 2000. ACM Press.
- [25] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Using and Determining Location in a Context-Sensitive Tour Guide. *IEEE Computer*, 34(8):35–41, August 2001.
- [26] Keith Cheverst, Nigel Davies, and Keith Mitchell. A Reflective Study of the GUIDE System. In Barbara Schmidt-Belz and Keith Cheverst, editors, *Proceedings of the 1st Workshop on Mobile Tourism Support*, pages 17–23, 2002.
- [27] John A. Clark, Richard F. Paige, Fiona A.C. Polack, and Phillip J. Brooke, editors. *Security in Pervasive Computing: Third International Conference*, 2006. Springer.
- [28] Siobhán Clarke and Cormac Driver. Context-Aware Trails. *IEEE Computer*, 37(8):97–99, August 2004.
- [29] Jon A. Cockle. Orchestrate. Online; accessed 11-September-2006. <http://www.orchestratehq.com/>.
- [30] Bristol Ferry Boat Company. Online; accessed 14-November-2006. <http://www.bristolferryboat.co.uk>.
- [31] Marshall J. Cook. *Time Management: Proven Techniques for Making the Most of Your Valuable Time*. Adams Media Corporation, 1998.
- [32] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, 1997.
- [33] Nigel Davies, Keith Cheverst, Keith Mitchell, and Adrian Friday. Caches in the Air: Disseminating Tourist Information in the GUIDE System. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*. IEEE Computer Society Press, 1999.
- [34] Dextronet. Daily To-Do List. Online; accessed 11-September-2006. <http://www.dextronet.com/daily-to-do-list.php>.

- [35] Anind K. Dey and Gregory D. Abowd. CybreMinder: A Context-Aware System for Supporting Reminders. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing (HUC '00)*, pages 172–186, London, UK, 2000. Springer-Verlag.
- [36] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. Technical Report TR2000-381, Georgia Institute of Technology, College of Computing, 1999.
- [37] Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. iCAP: Interactive Prototyping of Context-Aware Applications. In *Proceedings of the 4th International Conference on Pervasive Computing (PERVASIVE 2006)*. Springer, 2006.
- [38] Cormac Driver, Éamonn Linehan, Mike Spence, Shiu Lun Tsang, Laura Chan, and Siobhán Clarke. Facilitating Dynamic Schedules for Healthcare Professionals. In *Proceeding of 1st International Conference on Pervasive Computing Technologies for Healthcare*. IEEE, 2006.
- [39] Keith Edwards, Victoria Bellott, Anind K. Dey, and Mark Newman. Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Middleware. In Gilbert Cockton and Panu Korhonen, editors, *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '03)*. ACM Press, 2003.
- [40] Mohamed Fayed and Douglas C. Schmidt. Object-Oriented Application Frameworks. *Communications of the ACM: Special Issue on Object-Oriented Frameworks*, 40:32–38, 1997.
- [41] Bent Flyvbjerg. Five Misunderstandings About Case Study Research. *Qualitative Inquiry*, 12(2):219–245, 2006.
- [42] Commision for Communications Regulation (ComReg). Irish Communications Market: Key Data Report, December 2006. Online; accessed 24-January-2007. [http://www.comreg.ie/\\_fileupload/publications/ComReg0668.pdf](http://www.comreg.ie/_fileupload/publications/ComReg0668.pdf).
- [43] UMTS Forum. UMTS Forum website. Online; accessed 16-September-2006. <http://www.ums-forum.org/>.



- [44] Dennis F. Galletta, Raymond Henry, Scott McCoy, and Peter Polak. Web Site Delays: How Tolerant are Users? *Journal of the Association for Information Systems*, 5(1):1–28, January 2004.
- [45] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [46] David Griffiths. A Pragmatic Approach to Spearman’s Rank Correlation Coefficient. *Teaching Statistics*, 2:10–13, 1980.
- [47] Geoffrey Grosenbach. Rough Underbelly. Online; accessed 11-September-2006. <http://www.roughunderbelly.com/>.
- [48] Hisashi Handa, Lee Chapman, and Xin Yao. Robust Route Optimization for Gritting/Salting Trucks: A CERCIA Experience. *IEEE Computational Intelligence Magazine*, 1(1):6–9, 2006.
- [49] Urs Hengartner and Peter Steenkiste. Avoiding Privacy Violations Caused by Context-Sensitive Services. In *Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '06)*, pages 222–233, Washington, DC, USA, 2006. IEEE Computer Society.
- [50] Thomas Hill and Pawel Lewicki. *Statistics: Methods and Applications*. StarSoft, 2006.
- [51] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-Awareness on Mobile Devices - the Hydrogen Approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, page 292.1, Washington, DC, USA, 2003. IEEE Computer Society.
- [52] Jason I. Hong and James A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys '04)*, pages 177–189, New York, NY, USA, 2004. ACM Press.

- [53] John A. Hoxmeier and Chris DiCesare. System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications. In *Proceedings of Association of Information Systems Americas Conference (AMCIS 2000)*, 2000.
- [54] Markus C. Huebscher and Julie A. McCann. Simulation Model for Self-Adaptive Applications in Pervasive Computing. In *Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA '04)*, pages 694–698. IEEE Computer Society, 2004.
- [55] IBM. Lotus Notes Email. Online; accessed 11-September-2006. <http://www.ibm.com/software/lotus/>.
- [56] Apple Computer Inc. iPhone. Online; accessed 24-January-2007. <http://www.apple.com/iphone/>.
- [57] Novell Incorporated. Novell Evolution. Online; accessed 11-September-2006. <http://www.gnome.org/projects/evolution/>.
- [58] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. ACM Press Books, 1997.
- [59] Martin Jansson. Context Shadow: An Infrastructure for Context Aware Computing. In *Third workshop on Artificial Intelligence in Mobile Systems (AIMS)*, 2002.
- [60] Eric J. Johnson and John W. Payne. Effort and Accuracy in Choice. *Management Science*, 31:394–414, 1985.
- [61] Ralph E. Johnson. Components, Frameworks, Patterns. In *Proceedings of the 1997 Symposium on Software Reusability (SSR '97)*, pages 10–17. ACM Press, 1997.
- [62] Arun Jotshi and Rajan Batta. Finding Robust Paths for Routing Ambulances in a Dynamic Disaster Environment. In *Proceedings of the 13th Annual International Engineering Research Conference (IERC 2004)*, 2004.
- [63] A. Kamar. Mobile Tourist Guide (m-ToGuide). Deliverable 1.4, Project Final Report IST-2001-36004, Mobile Tourist Guide Consortium, 2003.

- [64] Maurice G. Kendall. *Rank Correlation Methods*. Hafner Publishing Company, New York, 1955.
- [65] Angela Kessell and Christopher Chan. Castaway: a context-aware task management system. In *Extended extended abstracts of the 2006 Conference on Human Factors in Computing Systems (CHI '06)*, pages 941–946, New York, NY, USA, 2006. ACM Press.
- [66] Emily Boyd and Omar Kilani. Remember The Milk. Online; accessed 11-September-2006. <http://www.rememberthemilk.com/>.
- [67] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. Case Studies for Method and Tool Evaluation. *IEEE Software*, 12(4):52–62, 1995. ISSN 0740-7459.
- [68] Panu Korpipää, Esko-Juhani Malm, Ilkka Salminen, Tapani Rantakokko, Vesa Kyllonen, and Ilkka Kansala. Context management for end user development of context-aware applications. In *Proceedings of the 6th International Conference on Mobile Data Management (MDM '05)*, pages 304–308, New York, NY, USA, 2005. ACM Press.
- [69] Ronny Kramer, Marko Modsching, Joerg Schulze, Marcel Hermkes, and Klaus ten Hagen. Context driven, adaptive tour computation and information presentation. In *1st International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP '05)*, 2005.
- [70] Ronny Kramer, Marko Modsching, and Klaus ten Hagen. Development and Evaluation of a Context-driven, Mobile Tourist Guide. *International Journal of Pervasive Computing and Communication*, 1(1), March 2005.
- [71] Tsvi Kuffik, Adriano Albertini, Paolo Busetta, Cesare Rocchi, Oliviero Stock, and Massimo Zancanaro. An Agent-Based Architecture for Museum Visitors' Guide Systems. In *Proceedings of the 13th International Conference on Information Technology and Travel and Tourism (ENTER 2006)*. The International Federation for IT and Travel & Tourism, January 2006.

- [72] Jussi Lahtinen, Petri Myllymaki, Tomi Silander, and Henry Tirri. Empirical Comparison of Stochastic Algorithms. In J. Alander, editor, *Proceedings of the 2nd Nordic Workshop on Genetic Algorithms and their Applications*, pages 45–59. Vaasa, 1996.
- [73] Brian M. Landry, Rahul Nair, Zach Pousman, and Manas Tungare. TaskMinder: A Context- and User-Aware To-do List Management System. Technical report, Georgia Institute of Technology, Gvu Center, 2003.
- [74] Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.
- [75] Rensis A. Likert. A Technique for the Measurement of Attitudes. *Archives of Psychology*, 21(140):5–54, 1932.
- [76] Wikipedia. Task list. Online; accessed 10-September-2006. [http://en.wikipedia.org/wiki/Task\\_list](http://en.wikipedia.org/wiki/Task_list).
- [77] Pamela J. Ludford, Dan Frankowski, Ken Reily, Kurt Wilms, and Loren Terveen. Because I carry my cell phone anyway: functional location-based reminder applications. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '06)*, pages 889–898, New York, NY, USA, 2006. ACM Press.
- [78] Mark M. Maccabee. Client/Server End-to-End Response Time: Real Life Experience. In *Proceeding of the 22nd International Computer Measurement Group Conference*. Computer Measurement Group, 1996.
- [79] Jason W. Mann and George D. Smith. *Modern Heuristic Search Methods*, chapter 14, pages 235–253. John Wiley & Sons, 1996.
- [80] Natalia Marmasse and Chris Schmandt. Location-Aware Information Delivery with ComMotion. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing (HUC '00)*, pages 157–171, London, UK, 2000. Springer-Verlag.

- [81] A. Maruyama, Naoki Shibata, Yoshihiro Murata, Keichi Yasumoto, and Minoru Ito. P-Tour: A Personal Navigation System for Tourism. In *Proceedings of the 11th World Congress on Intelligent Transport Systems*, volume 2, pages 18–21, 2004.
- [82] Tara Matthews, Hans-Werner Gellersen, Kristof Van Laerhoven, and Anind K. Dey. Augmenting Collections of Everyday Objects: A Case Study of Clothes Hangers as an Information Display. In *Proceedings of the 2n International Conference on Pervasive Computing (PERVASIVE 2004)*, 2004.
- [83] Jeffrey Mayer. *If You Haven't Got the Time to Do It Right, When Will You Find Time to Do It Over?* Simon and Schuster, New York, 1990.
- [84] Sun Microsystems. Java Micro Edition: Personal Basis Profile. Online; accessed 26-September-2006. <http://java.sun.com/products/personalprofile/>.
- [85] Robert B. Miller. Response Time in Man-Computer Conversational Transactions. In *Fall Joint Computer Conference 33 (part 1)*, pages 267–277. AFIPS Press, 1968.
- [86] Ricardo Morla and Nigel Davies. Evaluating a Location-Based Application: A Hybrid Test and Simulation Environment. *IEEE Pervasive Computing*, 3(3):48–56, 2004. ISSN 1536-1268.
- [87] Brad A. Myers. The importance of percent-done progress indicators for computer-human interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '85)*, pages 11–17, New York, NY, USA, 1985. ACM Press.
- [88] Fui Hoon Nah and Kihyun Kim. *Managing Web-enabled Technologies in Organizations: a global perspective*, chapter 7, pages 146–161. Idea Group Publishing, Hershey, PA, USA, 2000.
- [89] Jakob Nielsen. *Usability Engineering*. Academic Press Inc., 1993.
- [90] Jakob Nielsen. Response Times: The Three Important Limits. Online; accessed 02-October-2006. <http://www.useit.com/papers/responsetime.html>.
- [91] Gottfried E. Noether. *The Best of Teaching Statistics*, chapter 4, pages 41–43. The Teaching Statistics Trust, Nottingham, England, UK, 1986.

- [92] Wikipedia. List of mobile network operators of Europe. Online; accessed 25-January-2007. [http://en.wikipedia.org/wiki/List\\_of\\_mobile\\_network\\_operators\\_of\\_Europe](http://en.wikipedia.org/wiki/List_of_mobile_network_operators_of_Europe).
- [93] Eleanor O'Neill, Martin Klepal, David Lewis, Tony O'Donnell, Declan O'Sullivan, and Dirk Pesch. A Testbed for Evaluating Human Interaction with Ubiquitous Computing Environments. In *Proceedings of the 1st International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05)*, pages 60–69. IEEE Computer Society, 2005.
- [94] Microsoft Corporation. Microsoft Outlook. Online; accessed 11-September-2006. <http://www.microsoft.com/outlook/>.
- [95] Steven J. Parrillo and Fred G. Wenger. Time Management for the Academic Emergency Physician. *eMedicine Clinical Knowledge Database*, 2006. Online; accessed 11-September-2006. <http://www.emedicine.com/EMERG/topic673.htm>.
- [96] John W. Payne, James R. Bettman, and Eric J. Johnson. *The Adaptive Decision Maker*. Cambridge University Press, 1993.
- [97] PocketGear.com. Tree ToDo List. Online; accessed 11-September-2006. <http://www.pocketgear.com/>.
- [98] Stefan Poslad, Heimo Laamanen, Rainer Malaka, Achim Nick, Phil Buckle, and Alexander Zipf. CRUMPET: creation of user-friendly mobile services personalised for tourism. In *Proceedings of the 2nd International Conference on 3G Mobile Communication Technologies (3G 2001)*, pages 28–32, 2001.
- [99] Jeffrey S. Poulin. *Measuring Software Reuse: Principles, Practices, and Economic Models*. Addison-Wesley Professional, 1996.
- [100] Wolfgang Pree. Meta Patterns - A Means For Capturing the Essentials of Reusable Object-Oriented Design. In *Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP '94)*, pages 150–162, London, UK, 1994. Springer-Verlag.

- [101] Microsoft. Microsoft Office Project. Online; accessed 23-January-2007. <http://www.microsoft.com/project>.
- [102] Mika Raento and Antti Oulasvirta. Privacy Management for Social Awareness Applications. In *Proceedings of the workshop on Context Awareness for Proactive Systems (CAPS 2005)*, pages 105–114. Helsinki University Press, 2005.
- [103] Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications. *IEEE Pervasive Computing*, 04(2):51–59, 2005. ISSN 1536-1268.
- [104] IMS Research. GPS Positioned for Mass Cellular Uptake. IMS Research, October 2006.
- [105] Ben Clayton Richard Hull and Tom Melamed. Rapid Authoring of Mediascapes. In *UbiComp 2004: Ubiquitous Computing: 6th International Conference*, Lecture Notes in Computer Science, pages 125–142. Springer, 2004.
- [106] Don Roberts and Ralph Johnson. *ACM Pattern Languages of Program Design 3*, chapter Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks, pages 471–486. Addison-Wesley Longman Publishing Co., Inc., 1996.
- [107] Manuel Román, Christopher K. Hes, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, Oct–Dec 2002.
- [108] Peter Ruppel, Georg Treu, Axel Küpper, and Claudia Linnhoff-Popien. Anonymous User Tracking for Location-based Community Services. In *Proceedings of the 2nd International Workshop on Location and Context-Awareness (LoCa 2006)*, pages 116–133. Springer-Verlag, 2006.
- [109] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second international edition edition, 2003.
- [110] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *CHI '99: Proceedings of the*

- SIGCHI conference on Human factors in computing systems*, pages 434–441. ACM Press, 1999.
- [111] Mahadev Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.
- [112] Bill Schilit, Norman 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, US, 1994. IEEE Computer Society.
- [113] Barbara Schmidt-Belz, Heimo Laamanen, Stefan Posland, and Alexander Zipf. Location-based Mobile Tourist Services – First User Interaction. In *Proceedings of 10th International Conference on Information and Communication Technology in Tourism (ENTER 2003)*. Springer Computer Science, 2003.
- [114] Jean Scholtz and Sunny Consolvo. Toward a Framework for Evaluating Ubiquitous Computing Applications. *IEEE Pervasive Computing*, 3(2):82–88, 2004.
- [115] Wieland Schwinger, Christoph Grün, Birgit Pröll, Werner Retschitzegger, and Andrea Schauerhuber. Context-awareness in Mobile Tourism Guides - A Comprehensive Survey. Technical report, Vienna University of Technology, 2005.
- [116] Learning Commons Fast Facts Series. Making a Task List. The Learning Commons, University of Guelph, Canada, 2004. Online; accessed 11-September-2006. <http://www.learningcommons.uoguelph.ca/ByFormat/OnlineResources/OnlineFastfacts/OnlineLearningFastfacts/Fastfacts-MakingTaskList.html>.
- [117] Takayuki Shiraishi, Munenobu Nagata, Naoki Shibata, Yoshihiro Murata, Keiichi Yasumoto, and Minoru Ito. A Personal Navigation System with a Schedule Planning Facility Based on Multi-Objective Criteria. In *Second International Conference on Mobile Computing and Ubiquitous Networking*. Information Processing Society of Japan, 2005.
- [118] Daniel Siewiorek, Asim Smailagic, Junichi Furukawa, Andreas Krause, Neema Moraveji, Kathryn Reiger, Jeremy Shaffer, and Fei Lung Wong. SenSay: A Context-



- Aware Mobile Phone. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers (ISWC '03)*, page 248, Washington, DC, USA, 2003. IEEE Computer Society.
- [119] Pollen Software. Task-O-Matic. Online; accessed 11-September-2006. <http://www.pollensoftware.com/task-o-matic/index.html>.
- [120] Timothy Sohn, Kevin A. Li, Gunny Lee, Ian E. Smith, James Scott, and William G. Griswold. Place-Its: A Study of Location-Based Reminders on Mobile Phones. In *Proceedings of the 7th International Conference on Ubiquitous Computing (UbiComp 2005)*, Lecture Notes in Computer Science, pages 232–250. Springer, September 2005.
- [121] Charles Spearman. The Proof and Measurement of Association Between Two Things. *American Journal of Psychology*, 15:72–101, 1904.
- [122] Wikipedia. Kendall tau rank correlation coefficient. Online; accessed 20-October-2006. [http://en.wikipedia.org/wiki/Kendall\\_tau\\_rank\\_correlation\\_coefficient](http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient).
- [123] Klaus ten Hagen, Ronny Kramer, Marcel Hermkes, Björn Schumann, and Patrick Müller. Semantic Matching and Heuristic Search for a Dynamic Tour Guide. In *Proceedings of the 12th International Conference on Information and Communication Technology in Tourism (ENTER 2005)*. Springer-Verlag, 2005.
- [124] Klaus ten Hagen, Marko Modsching, and Ronnie Krammer. A Location Aware Mobile Tourist Guide Selecting and Interpreting Sights and Services by Context Matching. In *Proceeding of the 2nd International conference on Mobile and Ubiquitous Systems (MobiQuitous '05)*, pages 293–304. IEEE Computer Society, 2005.
- [125] Mozilla Corporation. Mozilla Thunderbird. Online; accessed 11-September-2006. <http://www.mozilla.com/thunderbird>.
- [126] Omesh Tickoo, Satish Raghunath, and Shivkumar Kalyanaraman. Route Fragility: A Novel Metric for Route Selection in Mobile Ad Hoc Networks. In *Proceedings of*

- the 11th IEEE International Conference on Networks (ICON 2003)*, pages 537–542. IEEE, 2003.
- [127] Edward P.K Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
- [128] Detlov von Winterfeld and Ward Edwards. *Decision Analysis and Behavioral Research*. Cambridge University Press, 1986.
- [129] Wallnote. Wallnote. Online; accessed 11-September-2006. <http://www.wallnote.com/>.
- [130] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The Active Badge System. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, 1992. ISSN 1046-8188.
- [131] Mark Weiser. The Computer for the 21st Century. *SIGMOBILE Mobile Computing and Communications Review*, 3(3):3–11, 1999. ISSN 1559-1662.
- [132] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, 1993. ISSN 0001-0782.
- [133] Morris Williams, Constance Fleuriot, John Reid, Richard Hull, Keri Facer, and Owain Jones. Mobile Bristol: A New Sense Of Place. In Peter Ljungstrand and Lars Erik Holmquist, editors, *Adjunct Proceedings of the 4th International Conference on Ubiquitous Computing (UBICOMP 2002)*, pages 27–28. Viktoria Institute, Goteborg, Sweden, September 2002.
- [134] Jens Wohltorf, Richard Cissée, Andreas Rieger, and Heiko Scheunemann. BerlinTainment - An Agent-Based Serviceware Framework for Context-Aware Services. In *Proceedings of the 1st International Symposium on Wireless Communication Systems (ISWCS 2004)*. IEEE, September 2004.
- [135] Jens Wohltorf, Richard Cissée, and Andreas Rieger. BerlinTainment: an agent-based context-aware entertainment planning system. *Communications Magazine*, 43(6):102–109, June 2005.

- [136] Jiyong Zhang and Pearl Pu. Survey of Solving Multi-Attribute Decision Problems. Technical Report IC/2004/54, Swiss Federal Institute of Technology, Lausanne, Switzerland, June 2004.
- [137] Feng Zhu, Matt W. Mutka, and Lionel M. Ni. The Master Key: A Private Authentication Approach for Pervasive Computing Environments. In *Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '06)*, pages 212–221, Washington, DC, USA, 2006. IEEE Computer Society.