An Application Framework for Mobile, Context-Aware Trails

Cormac Driver, Siobhan Clarke

Distributed Systems Group, School of Computer Science and Statistics, Trinity College Dublin, Ireland

Abstract

In this paper [1] we describe the design, implementation and evaluation of a software framework that supports 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 time management applications. Trails overcome limitations with traditional time management techniques based on static to-do lists by dynamically reordering activities based on emergent context.

Key words: trails, context-aware scheduling, pervasive computing, software frameworks

1 Introduction

Time management strategies for planning and scheduling activities increase the efficiency and effectiveness of either personal or corporate time use. Supporting techniques are commonly based on the use of prioritised to-do lists [15]. A prioritised to-do list is an ordered collection of activities that the list owner must complete, with tasks crossed off as they are completed. 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

Email address: `[firstname.lastname]@cs.tcd.ie` (Cormac Driver, Siobhan Clarke).

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 applications can discover and take advantage of contextual information (such as user location, time of day, nearby people/computing devices and user activity) [44]. Example applications of mobile, context-aware computing include location-aware telephone call forwarding [53], situation-aware self-managing mobile phones [46], context-aware medication monitors [2] and weather-aware clothes hanger-based information displays [37]. Mobile devices can acquire context information from on-board local sensors such as GPS devices, accelerometers and infrared sensors, and from remote sensors that transmit data about the environment in which they are embedded e.g., temperature and pulse rate data can be acquired from sensors implanted in the human body. Mobile, context-aware applications automatically adapt to discovered context by changing their behaviour as appropriate to better suit the user. This paradigm facilitates the automatic adaptation of a mobile user's schedule of activities 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 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.

The implementation of a mobile, context-aware activity scheduling application involves addressing two challenges common to this type of application. First, 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 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. Second, 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. 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 primarily interested in such as application deployment and user evaluation. In

this paper we present an application framework for the development of mobile, context-aware *trails*-based applications [14]. 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 provides structure and behaviour to support context-based activity schedule composition (a process know as *trail generation*), identification of whether or not schedule reordering is required following context change (*trail reconfiguration point identification*) and subsequent automatic schedule reordering as appropriate (*trail reconfiguration*). The framework supports the development of trails-based applications in a generic, extensible manner, enabling developers to both reuse common application components and extend the framework to support application-specific behaviour. Consequently, the development of a diverse range of trails-based applications is more accessible to software developers.

The remainder of this paper is organised as follows. Section 2 presents a short perspective on the trails concept and illustrates its applicability. Section 3 presents our application framework for mobile, context-aware activity scheduling. Section 4 describes the evaluation of the framework in terms of a) reusability and extensibility b) performance and c) human opinion on trail quality. Section 5 reviews related work and section 6 provides a summary.

2 A Perspective On Trails

The relevance of trails-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. The implementation of such an application, a trails-based day planner, is presented in Section 4.1.1 as part of our evaluation. 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. We discuss trails-based applications that support visitors to music festivals and theme parks as part of our evaluation in Section 4.1.2 and Section 4.1.3 respectively. In addition, much of the related work analysed in Section 5 is concerned with mobile, context-aware tourist guide applications.

The pervasiveness of mobile devices (particularly the mobile phone) and their recent technical advancement, combined with the trails-based application development support described in this paper, creates an environment in which computer support for context-based activity scheduling can be realised to support users in both business and leisure scenarios.

3 Application Framework

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 [42]. The term 'application framework' is used to describe a software framework that constitutes a generic application for a specific domain area [39]. In this section we introduce the Hermes project which is concerned with providing framework support for mobile, context-aware applications in general. We then discuss the challenges in mobile, context-aware trails management in particular and describe in detail how the application framework presented in this paper supports trail generation and trail reconfiguration point identification.

3.1 The Hermes Project

The Hermes project[2] at Trinity College Dublin is investigating extensible, generic components for mobile, context-aware applications. A screenshot from one such application, a trails-based mobile game called RiddleHunt developed using the framework, is shown in Figure 1(a). The screenshot illustrates the context-based positioning of a player on a map-based graphical user interface, the ordering of activities/riddles on the user's trail (which is determined by context such as game state and riddle type) and directions from the user's current position to the trail activities. The Hermes project is concerned with providing support 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

[2] `http://www.dsg.cs.tcd.ie/hermes`

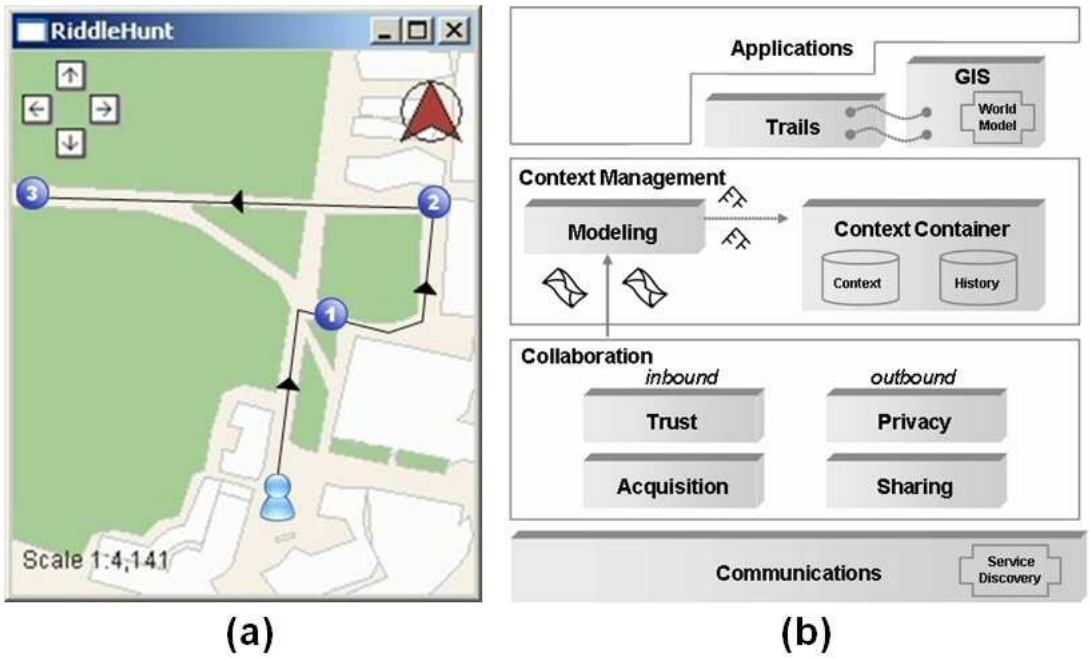

Figure 1. A screenshot from RiddleHunt and the Hermes architecture

[49]), context reasoning (deducing new and relevant context from various data sources), 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).

This paper is concerned specifically with the trails management component that enables Hermes to support the implementation of applications containing mobile, context-aware activity scheduling behaviour. Figure 1(b) illustrates the Hermes architecture. The Trails component resides beneath the Applications layer and assumes the availability of the context service below. The context service provides the Trails component with context information used for trail generation and reconfiguration point identification. This behaviour is briefly described below to illustrate how context can be acquired by trails-based applications (see [19] for full details).

3.1.1 Context Acquisition in Hermes

The Hermes framework contains reusable, extensible behaviour that supports applications in acquiring and modelling context from a range of sources e.g., remote mobile devices and local sensors. Peer-to-peer ad hoc service discovery is used to discover and communicate with remote devices that provide a context service. The Communication component illustrated in Figure 1(b) facilitates the discovery of remote devices and the transfer of context between devices. Proximate devices are discovered via an ad hoc service discovery protocol, with devices connecting directly to share context.

It is important to note that trails applications built using the application framework described in this paper are not restricted to using only the context acquisition support provided by Hermes. Developers can extend the classes responsible for context generation and acquisition by using the Hermes framework or they can use the context support provided by default in the application framework (discussed in Section 3.4).

3.2 Trail Generation

Trail generation is a combinatorial optimisation problem similar to the classic Travelling Salesman Problem (TSP) [32]. However, there are a number of differences between the TSP and the trail generation problem. First, the number of activities in a trail is variable (based on context) and therefore the solution space can dynamically increase and decrease in size. Second, in the general case it is not necessary to return to the starting point to complete a trail. Third, the TSP reasons about the distance between nodes whereas the generation of a trail involves reasoning about not only the distance between activities but also about the time it will take a user to traverse that distance, the number of activities possible if a specific activity sequence is chosen, the number of mandatory activities possible and the amount of the time the user will be idle while completing a given sequence of activities. 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 [13]. 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. 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

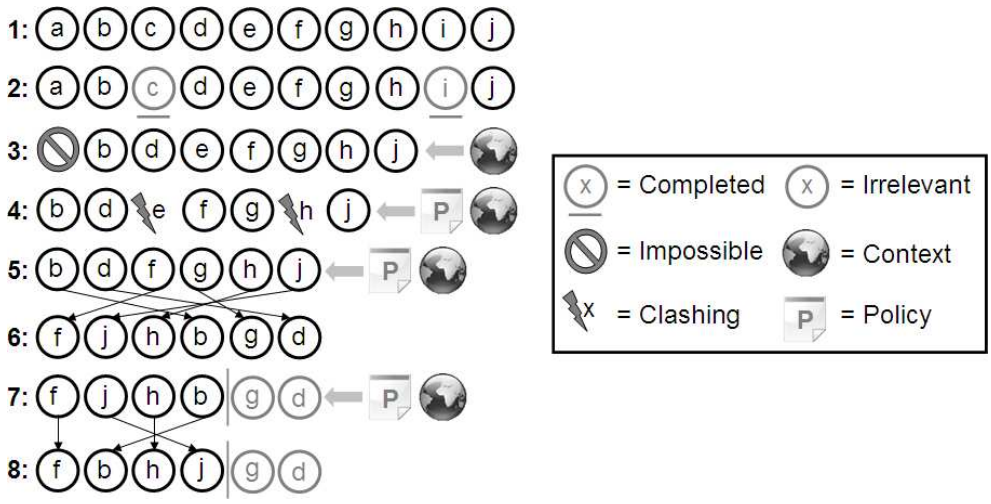

Figure 2. Context-based activity set reduction

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, 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, 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.

3.2.1 Trail Generation in the Application Framework

The trail generation mechanism in the application framework is based on context-based activity set reduction. 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 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 2. Line 1 in Figure 2 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.

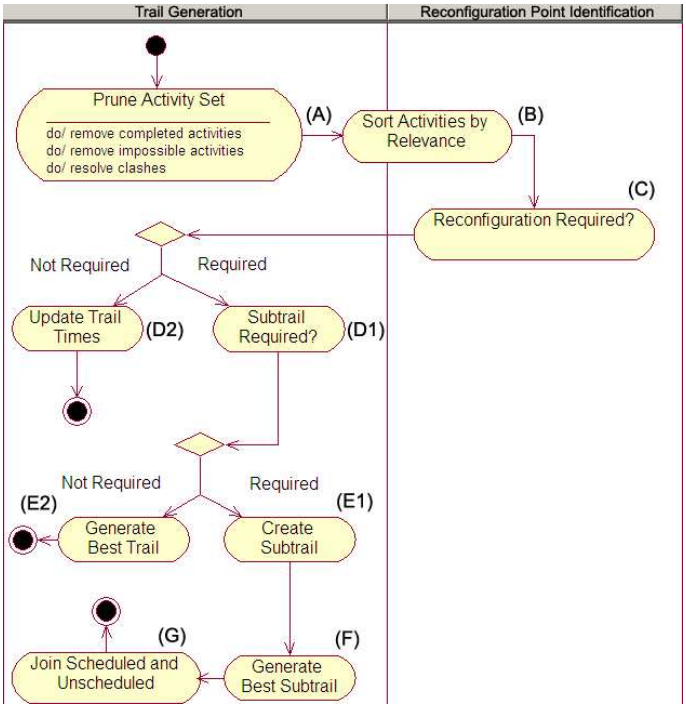

Figure 3. The trail generation process

3.2.1.1 Completed Activities An activity is completed when the activity status is manually changed to 'complete' via an application user interface. Line 2 in Figure 2 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, completed activities are removed in step (A).

3.2.1.2 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 or to retrieve an item from a warehouse if it becomes out of stock. Line 3 in Figure 2 illustrates the identification of an impossible activity, a, based on the current location context which is represented by the globe icon. a is then removed from X. Figure 3 illustrates that impossible activities are removed at step (A) following the removal of completed activities. Temporarily impossible activities i.e., impossible activities that can become possible based on significant context change, are monitored and reinstated to the activity set as appropriate.

3.2.1.3 Clashing Activities Two activities clash when, although independently possible, the completion of one activity renders the other impossible. 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 multi-attribute utility theory (MAUT) [52]. We also considered the use of conditional preference networks (CP-networks) [7] and constraint satisfaction problems [51] for multi-attribute decision making. CP-networks are suitable for use in instances where user preferences are incompletely specified and so were deemed unnecessary for use in trails applications (where preferences are known before trail generation). The use of constraint satisfactions problems was discounted as they do not map as naturally to the problem specification as MAUT [56]. The MAUT-based clash resolution technique considers user-specified weights when comparing the following activity properties: 1) proximity - the distance from the user's current location to the activity location 2) priority - the user-specified numerical priority level for the activity e.g., 5 for high, 1 for low and 3) obligation - whether the activity is mandatory or optional.

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. 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. Line 4 in Figure 2 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 shows that activities rejected as a result of a clash are removed following the removal of completed and impossible activities.

3.2.1.4 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 [23]). 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) and

a decision regarding whether or not reconfiguration is required (discussed in 3.3) is made in step (C). If reconfiguration is not required, the estimated 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. 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 activity proximity, priority, obligation and urgency (how urgently an activity must be addressed based on its opening hours). Lines 5 and 6 in Figure 2 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 $X_a$and the resultant formation of a subtrail.

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). 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.

3.2.1.5 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 evaluation function considers 1) the number of activities possible, 2) the number of mandatory activities possible 3) the total trail length/travel distance required 4) the trail duration and 5) the amount of idle time. The framework contains three approaches for generating candidate trails - brute force, simulated annealing and a genetic algorithm. The brute force approach represents the most straightforward method of attaining an exact solution, however it quickly becomes impractical as the number of activities under consideration

increases. The simulated annealing and genetic algorithms are approximation algorithms that produce good, but not necessarily optimal, solutions to combinatorial optimisation problems within reasonable time frames. These approaches, which illustrate how the NP-hard problem of trail generation can be addressed through the use of heuristics, are included in the application framework as an alternative to brute force trail generation, and provide developers with the facility to reason over larger sets of activities within a reasonable response time. The framework is designed to support extensions to the collection of trail ordering algorithms available to the developer/user, and therefore any exact or heuristic algorithm or can be plugged into the framework and assessed. The three approaches are evaluated in Section 4.2. Lines 7 and 8 in Figure 2 illustrate the trail ordering process, using current context and a user-weighted policy to order the relevant set to best serve the user.

3.3 Trail Reconfiguration Point Identification

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 [12]. 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 unexpectedly become impossible. 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.

3.3.1 Reconfiguration Point Identification in the Application Framework

The reconfiguration point identification technique in the application framework is based on the concept of activity relevance introduced by the trail generation mechanism. 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. 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.

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 1) activity migration between the relevant set and the irrelevant set 2) activity completion 3) an activity becoming impossible and 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.

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.

Kendall's rank correlation coefficient, known as Kendall's τ, calculates the correspondence between two rankings [27]. Kendall's τ was chosen over Spearman's ρ [48] and Pearson's product-moment correlation coefficient [11] due to

its more intuitive nature and ability to cope well with small sample sets respectively. A ranking of a set of n items may be written as a sequence, where each element of the sequence is a number between 1 and n, and no two elements are equal. This approach is used to calculate the degree of correspondence between the order of the activities after they are sorted by relevance and the order at the time they were used to generate the user's current trail. Kendall's τ 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 defined as follows. Suppose we have two rankings, (a_k) and (b_k). In the context of trail generation we can always assume that $(a_k) = (1, 2...n)$. Then:

$$P = \sum_{i=1}^{n} |B_i|$$

where $B_i = \{b_j \in (b_k) : j > i \text{ and } b_j > b_i\}$ and $1 \leq i \leq n$ and $1 \leq j \leq n$. For example, if we have the sequences $(a_k) = (1, 2, 3, 4, 5, 6)$ and $(b_k) = (3, 4, 2, 1, 5, 6)$, then $B_1 = (4, 5, 6)$ and $|B_1| = 3$. Similarly, $B_2 = (5, 6)$ and $|B_2| = 2$. Continuing this way we find that $P = 3 + 2 + 2 + 2 + 1 + 0 = 10$. Therefore, $\tau = \frac{20}{15} - 1 = 0.33$. 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.

A trail is reconfigured in the application framework if the τ value for correspondence between the new and previous relevant sets is below a threshold defined by the application developer or user. 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 4.2 during an analysis of the accuracy of the reconfiguration point identification technique.

The application framework, implemented in Java [3], is composed of reusable generic implementations of the trail generation and reconfiguration point identification techniques. The default implementations are developed in a manner that facilitates extension by developers who wish to specify application-specific trails behaviour. The application framework consists of 34 open source classes and 5 properties files that provide structure and behaviour for mobile, context-aware trails-based applications. This section describes the core classes and outlines their responsibilities (further implementation details can be found in [19]). The classes are logically grouped into six groups. The class groups that compose the application framework are as follows:

Group 1 - *Trails*. This collection of classes has four primary responsibilities. First, to provide a representation of the user's trail. Second, to provide behaviour to manipulate both the contents of the trail and the attributes of the activities. The `Activity` class contains attributes that describe a generic activity. Some attributes i.e., those that are known by the user/developer, are assigned values at application start-up, while dynamic attributes are assigned values calculated at runtime. The values assigned at application start-up are loaded from a persistent trail specification. These attributes include a textual activity description, activity location coordinates, activity opening and closing time, the amount of leeway in the closing time, the activity duration, the duration leeway, the activity priority and whether the activity is mandatory or optional. The dynamic activity attributes include whether the activity is scheduled or not, whether it is currently possible, the estimated start and end times, whether the activity has been completed and the revised activity duration. 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 a class called `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`. Third, 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`. Fourth, 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 3.2.1.5.

[3] Java 2 Micro Edition (J2ME) Personal Basis Profile [38]

Group 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.

Group 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 application framework uses a hierarchical, object-oriented context model. This model naturally lends itself to modelling real-world objects and their various relationships, as well as to extension. The Hermes framework (see Section 3.1), of which the application framework described in this paper is a component, provides object-to-XML mappings for context storage and exchange between mobile devices. 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 [20]. 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.

Group 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 [20]. 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`).

Group 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.

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 and activity specifications from disk and perform miscellaneous behaviour e.g., data format conversion and distance estimation, respectively.

The framework also contains properties files that store values used to customise application behaviour without requiring framework extension. The `trail.properties` file is used to specify properties such as the size of the subtrail to use, the trail generation strategy to employ and the τ value to use during reconfiguration point identification. The `userPreferences.properties` file stores user preferences for calculating activity relevance, resolving clashes and evaluating candidate trails. The `normalization.properties` file contains the upper limits for the trail and activity properties that are normalised during activity and trail comparison and evaluation.

4 Evaluation

The evaluation was conducted in order to meet three objectives. First, 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. Second, to quantify both the responsiveness of the trail generation implementation and the accuracy of the reconfiguration point identification implementation. Third, to evaluate human opinion on the quality of the trails generated by the application framework. Section 4.1 presents an overview of how the first evaluation objective was achieved through the examination three case study applications built using the application framework. Section 4.2 discusses how the second evaluation objective was achieved by means of lab experiments. Section 4.3 discusses the study conducted in

order to meet the third evaluation objective. In the interest of brevity we do not include full details of all case studies and experiments. Full details are available in [19].

4.1 Case Studies

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. We implemented three case studies that illustrate how the framework can be reused and extended to support the development of a range of mobile, context-aware trails-based applications. Each case study demonstrates different existing framework capabilities or possibilities for different kinds of framework extension.

4.1.1 Day Planner

The easiest way to use a framework is to use existing classes only i.e., without implementing any concrete subclasses [25]. The first case study explores how the application framework can be used to develop a day planner application that considers location context sensed from a GPS device, time data sensed from the host mobile device and activity properties and user preferences provided by the developer or user. Its key characteristic is that it represents the set of trails applications that can be built using the application framework without extension. 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.

In order to implement the day planner application using the application framework the developer is required to edit the `trail.properties` file to specify activity details and configure 1) the maximum number of activities that can be scheduled during trail generation i.e., the subtrail/relevant set size 2) the periodic reconfiguration time interval and 3) the τ 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. The developer can also edit user preferences for determining activity relevance, clash resolution and trail evaluation, although defaults can be used. The modification of these values in the `userPreferences.properties` file affects which activities are selected for scheduling and the ordering of the

user's trail. The properties in the `normalization.properties` file can also be set to specify trail and activity value ranges used for normalisation in the day planner application.

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.

4.1.2 Music Festival

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 [25]. The second case study explores how the application framework can be extended to cater for applications that require the use of a context type not provided by the application framework and the use of activities with an attribute not provided by the application framework.

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 goers 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.

The implementation of the music festival application involved extending the framework by adding a new context source, adding a new activity attribute and ensuring that the new behaviour is used by the rest of the framework.

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. `ReconfigurationEngine`, the class responsible for receiving context events and consequently invoking trail reconfiguration, is extended to take the new context source into account.

A new attribute, `genre`, is added to the default activity description so that musical performances can be better described. This necessitates the extension of the framework's default activity class. Consequently, the activity specification in the `trail.properties` file was extended accordingly. 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.

`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. This class is redefined so that the framework uses the extended versions of the classes that implement behaviour specific to the music festival application [4].

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%. Using the application framework in this manner is naturally more complicated than using it without extension. However, as the case study illustrates, 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.

4.1.3 Theme Park

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. 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.

[4] We are investigating the use of a plug-in architecture to remove the necessity to edit `TrailGeneration`, an existing framework class, in the case of framework extension.

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 1) the trail evaluation function, which is required to consider ride queuing time when scheduling the user's chosen activities 2) the clash resolution mechanism, which is required to consider the queuing time associated with each activity when assessing whether any of the user's chosen activities clash with each other and 3) the activity relevance measurement mechanism, which considers queuing time 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.

The addition of the new context source, queuing time, and the new activity attribute, are done in the same manner as the addition of the stage time context and genre attribute in the music festival application. The major difference between the two case studies is that the new type added in the theme park application necessitates the implementation of new ways in which to compare activities. 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`. 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 so that developers/users can specify how much of an impact activity queuing time should have on the respective behaviour's.

The trail evaluation function must also take the queuing time attribute into account when generating trail scores `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.

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%. 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 software framework, however it is also the most powerful [25]. 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.

4.1.4 Summary

The case studies presented in this section illustrate how the application framework can be reused either without modification to develop specific trails applications based on location and time 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 about 84% in all three case studies.

4.2 Trail Generation and Reconfiguration

This section presents the results of lab experiments conducted to assess 1) the number of activities that can be scheduled during trail generation while adhering to a response time of 12 seconds, 2) 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 as it was considered to be a reasonable proportion of the total response time (12 seconds) to dedicate to activity set pruning and activity relevance calculation) and 3) the accuracy of the trail reconfiguration point identification mechanism.

4.2.1 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. The

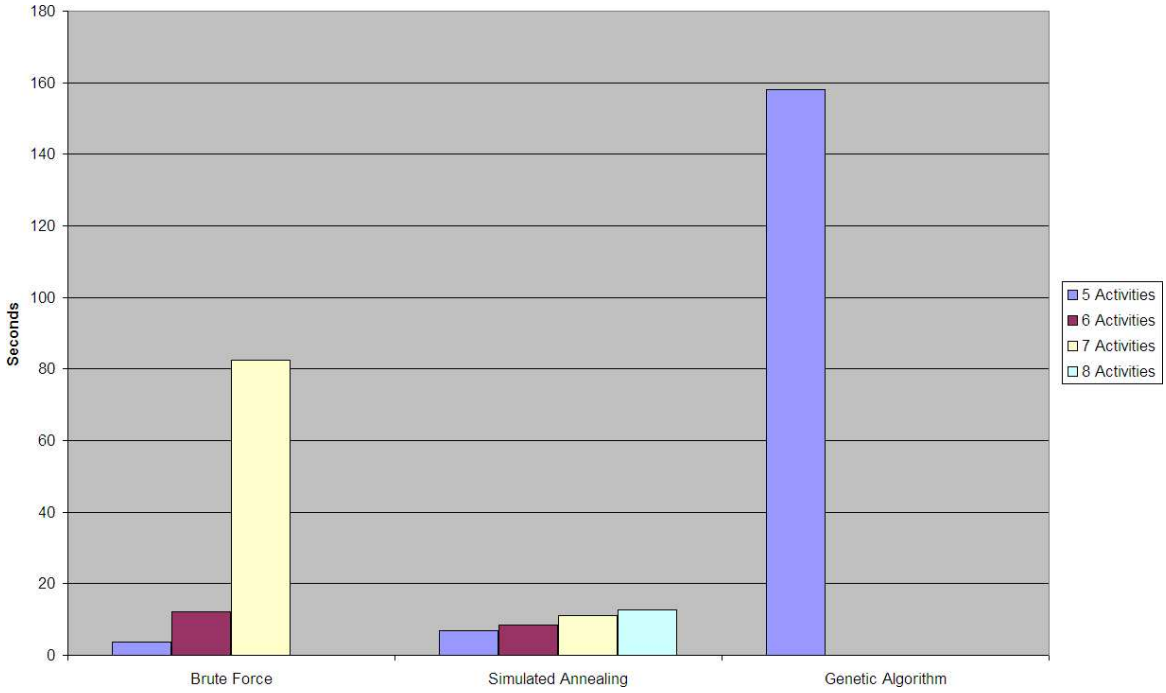

Figure 4. Trail generation response times

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.

Figure 4 illustrates the response times of the three trail generation algorithms for 5, 6, 7 and 8 activities over the course of 100 trail reconfigurations. 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 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. 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). The simulated annealing algorithm outperforms both the brute force algorithm and the genetic algorithm, with the results 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.

The results of this experiment indicate that simulated annealing is the best algorithm to select when developing an application as it can consider the most activities within a reasonable response time. However, using this algorithm requires an understanding of how the customisable parameters affect the behaviour of the algorithm in terms of execution time and solution qual-

ity. 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. The results also demonstrate that the application framework is capable of generating non-trivial trails within a reasonable response time. The trail quality experiment discussed in Section 4.3 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.

4.2.2 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 the activity scheduling experiment 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.

Figure 5 illustrates 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). The results show that between 100 and 110 activities can be considered during trail generation without spending more than 2 seconds on activity set preparation. 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 in which it is necessary to have a relatively large amount of activities.

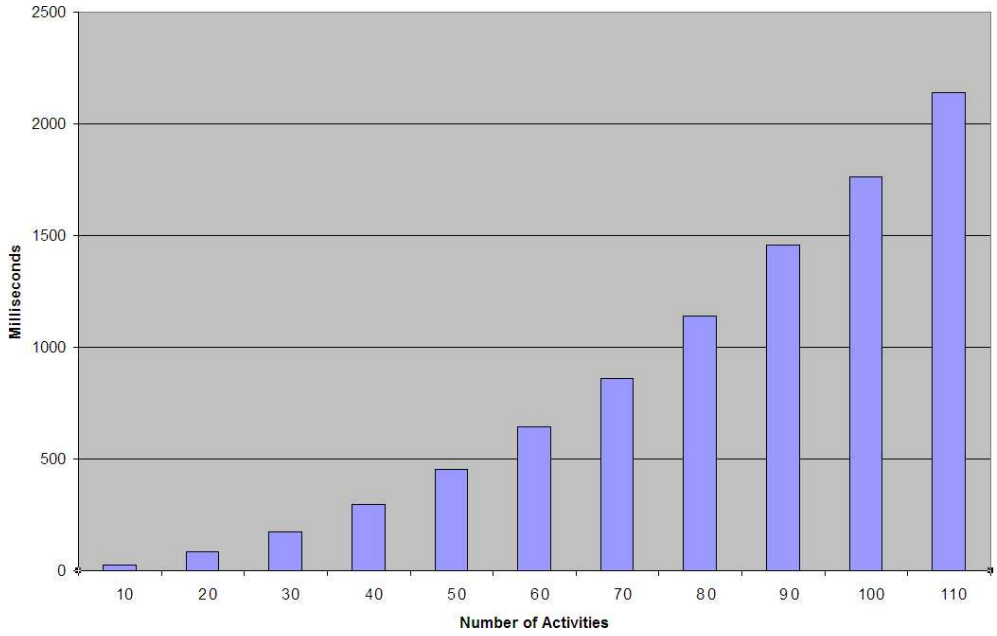

Figure 5. Results of the activity consideration experiment

4.2.3 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 and identifying if each context event was correctly handled by the reconfiguration point identification mechanism. Each application was executed twice, once with no reconfiguration 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 τ values - 0.95 and 0.85. These τ values were chosen because τ values in and around that region (0.8 - 0.95) produced 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 τ value, resulting in instances of classification #4 being relatively high by comparison. This is because the likelihood of reconfiguration being deemed necessary increases as τ increases. At the lower τ 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 τ value would result in the trail more accurately reflecting the contextual situation, but that achieving this accuracy would require sacrificing resources to unnecessary reconfigurations.

On average, the reconfiguration point identification technique was found to handle 88.6% of context events correctly. Of the remaining context events, 4.3% result in unnecessary reconfigurations and 7.1% result in reconfiguration not being invoked when it should be. These figures were 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.

Table 1 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

	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 1
Further investigation of the trials with = 0.95

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 experiment shows that the reconfiguration point identification in the application framework can categorise context events correctly in the majority of cases. Developers can customise the τ 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 τ value used.

4.3 Trail Quality Experiment

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. Forty subjects participated in the experiment (29 male and 11 female). Subjects were asked to manually solve an activity scheduling problem involving seven activities and re-solve it following the introduction of new context. 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 [33]. 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, facilitating the observation of poten-

tial learning effects. All tasks were timed. 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).

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 describe the quality of the activity scheduling solutions produced by subjects.

Table 2 contains the results of the timed element of the experiment. The results of the solution validation element of the experiment are contained in Table 3. Table 4 summarises the experiment results 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.

	Group 1	**Group 2**
Solve problem (median time)	108 seconds	161 seconds
Validate solution (median time)	58 seconds	65 seconds

Table 2
Trail quality experiment timing results

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 3, 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 were added to the framework to address this.

It took subjects just over twice as long on average to manually solve an ac-

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

Table 3
Trail quality experiment solution validation results

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

Table 4
Trail quality experiment solution quality results

tivity 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 2, 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 4 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.

In summary, the results of the trail experiment support our belief 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 Related Work

The intent of our work is to provide an application framework for development of mobile, context-aware trails-based applications, providing generic, extensible solutions to common challenges related to this application type. Therefore, we examine related work in the areas of mobile, context-aware tourist guides, context-aware to-do lists and application frameworks for mobile, context-aware application development.

5.1 Mobile, Context-Aware Tourist Guides

To date, most research into mobile, context-aware activity scheduling has focused on tourist guide applications [12,50,1,55,3,45,36,26,5]. However, many of

these applications do not support dynamic, context-based trail management. Some present the user with a static tour on a mobile device while others use context to generate a tour but offer no subsequent tour adaptation. In terms of trail generation and dynamic reconfiguration P-Tour [36] and the Dynamic Tourist Guide (DTG) [50] are the most sophisticated.

P-Tour is a personal navigation system that allows tourists to compose a multi-destination schedule, taking individual preferences and time restrictions into account. The end-user application resides on a mobile device and communicates via WiFi with a remote server. The server executes a genetic algorithm to compute tours to within 2% of optimal in 15.5 seconds (for a tour of 14 destinations). The optimal tour is transmitted to the tourist's mobile device. The system can recognise when the user has deviated from the recommended path by tracking their location periodically and either alerts the user or recomputes the schedule if the user has deviated significantly.

The Dynamic Tourist Guide is a mobile agent that selects tourist attractions from a predefined database (based on elicited user preferences) and plans a tour of these attractions. The end-user application is deployed on a mobile device and the tours are generated on a remote server. The DTG can compute a tour of 16 activities to within 5% of optimal in a response time of 5.5 seconds. The server-based agent uses a directed depth-first algorithm that incorporates a number of heuristics including using an average duration estimate for all attractions and trading off attraction relevance against the cost (in time) of travelling to and exploring the attraction. The DTG periodically checks if the user has deviated from their plan by more than a specified amount of time e.g., 30 minutes, and if they have then the tour is reconfigured through the addition or removal of activities as appropriate.

Both the P-Tour and DTG tour generation algorithms are server-based and assume an uninterrupted wireless network connection. Neither application provides support for disconnected operation in terms of tour management, something which we believe is necessary given the nature of unreliable nature of wireless networks in outdoor environments. It is unlikely this issue could be addressed by deploying the algorithms on the user's mobile device. The algorithms are designed to work on a powerful server platform and their performance would be significantly degraded by migrating to a resource-constrained mobile platform.

While the reconfiguration point identification mechanisms employed by P-Tour and the DTG are relatively sophisticated, they are not without limitations. The time-based periodic element of each raises the possibility that an important context event i.e., a location change, 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. It is unlikely that this would be a problem

in P-Tour or the DTG, where only location is the primary context 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 it is likely that inconsistencies between the real world and the computer representation would occur.

5.2 Context-Aware To-Do Lists

A number of context-aware to-do lists have been developed in recent years [31,16,35,34,47,28]. These applications typically allow users to specify contextual situations with which reminders are associated. The user is notified of a reminder when its associated contextual situation exists. 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, 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.

5.3 Application Frameworks for Mobile, Context-Aware Computing

Numerous application frameworks for mobile, context-aware computing have been developed [41,9,54,30,8,18]. Some of the frameworks, for example the Mobile Bristol toolkit [41] and the Context-aware Application Prototyper (iCAP) [18], facilitate the creation of context-aware applications through the use of a graphical user interface and extensible rule-base, 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. For example, all of the application frameworks can be used to create both static tour guide applications and context-aware to-do lists applications. 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 5.2, 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.

In addition to these application frameworks, a number of context-awareness frameworks have been proposed in recent years [4,6,10,21,24,22,29,40,43,17]. 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 management. The frameworks address common challenges such as acquisition, fusion and reasoning but due to their application-independent nature do not provide any support for context-aware activity scheduling.

6 Summary

In this paper we have presented an application framework for mobile, context-aware trails-based application development. The framework supports trail generation through context-based activity set reduction and trail reconfiguration point identification through identification of significant context events. Our evaluation illustrates how the application framework can be reused and extended to facilitate the implementation of a range of trails-based applications that can manage activities for people in a way they find useful. The evaluation demonstrates the flexibility and reusability of the framework, and the versatility of the trails concept as an underpinning for many different applications. The framework advances the state of the art by supporting scalable trail generation on mobile platforms and context-sensitive reconfiguration point identification. In addition, by supporting the development of trails-based applications in a generic, extensible manner, the framework aids the progression of the state of the art away from tourist-centric applications towards a more diverse range of trails-based applications.

7 Acknowledgements

We would like to acknowledge the support of Lero: The Irish Software Engineering Research Centre, funded by Science Foundation Ireland. Thanks also to Daire O'Broin and Éamonn Linehan for their comments on earlier drafts of this paper.

References

[1] 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.

[2] 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.

[3] Hermann Anegg, Harald Kunczier, Elke Michlmayr, Günther Pospischil, and Martina Umlauft. LoL@: designing a location based UMTS application. *e&i - elektrotechnik & informationstechnik*, 119(2), February 2002.

[4] 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.

[5] Francesco Bellotti, Riccardo Berta, Alessandro De Gloria, and Massimiliano Margarone. User Testing a Hypermedia Tour Guide. *IEEE Pervasive Computing*, 1(2):33–41, 2002.

[6] 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.

[7] 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.

[8] Niels Olof Bouvin, Bent G. Christensen, Kaj Grønbæk, and Frank Allan Hansen. HyCon: a framework for context-aware mobile hypermedia. *Hypermedia*, 9(1):59–88, 2003.

[9] 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.

[10] 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.

[11] Peter Y. Chen and Paula M. Popovich. *Correlation: Parametric and Nonparametric Measures.* Sage Publications Inc, 2002.

[12] 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.

[13] 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.

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

[15] Marshall J. Cook. *Time Management: Proven Techniques for Making the Most of Your Valueable Time.* Adams Media Corporation, 1998.

[16] 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.

[17] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16 (2-4):97–16, 2001.

[18] 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.

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

[20] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[21] 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.

[22] 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.

[23] 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.

[24] Martin Jansson. Context Shadow: An Infrastructure for Context Aware Computing. In *Third workshop on Artificial Intelligence in Mobile Systems (AIMS)*, 2002.

[25] Ralph E. Johnson. Components, Frameworks, Patterns. In *Proceedings of the 1997 Symposium on Software Reusability (SSR '97)*, pages 10–17. ACM Press, 1997.

[26] A. Kamar. Mobile Tourist Guide (m-ToGuide). Deliverable 1.4, Project Final Report IST-2001-36004, Mobile Tourist Guide Consortium, 2003.

[27] Maurice G. Kendall. *Rank Correlation Methods.* Hafner Publishing Company, New York, 1955.

[28] 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.

[29] Panu Korpipaa, 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.

[30] Tsvi Kuflik, 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.

[31] 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.

[32] 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.

[33] Rensis A. Likert. A Technique for the Measurement of Attitudes. *Archives of Psychology*, 21(140):5–54, 1932.

[34] 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.

[35] 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.

[36] A. Maruyama, Naoki Shibata, Yoshihiro Murata, Keiichi 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.

[37] 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.

[38] Sun Microsystems. Java Micro Edition: Personal Basis Profile. Online; accessed 26-September-2006. `http://java.sun.com/products/personalprofile/`.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] Charles Spearman. The Proof and Measurement of Association Between Two Things. *American Journal of Psychology*, 15:72–101, 1904.

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

[50] 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.

[51] Edward P.K Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[52] Detlov von Winterfeld and Ward Edwards. *Decision Analysis and Behavioral Research*. Cambridge University Press, 1986.

[53] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The Active Badge System. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, 1992.

[54] Jens Wohltorf, Richard Cissée, and Andreas Rieger. BerlinTainment: an agent-based context-aware entertainment planning system. *Communications Magazine*, 43(6):102–109, June 2005.

[55] 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.

[56] 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.