Supporting Personalised Recommendations in Context-aware Applications

Shiu Lun Tsang

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

March 2009

Declaration

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

Shiu Lun Tsang

Dated: March 30, 2009

Acknowledgements

I owe a very grateful thank you to many people who have helped me in countless ways during my doctorate experience.

I would like to thank my supervisor Siobhán Clarke for her guidance and assistance over the years. I very much appreciate her understanding, motivation, and patience. Thank you to the members of the Hermes project - Cormac Driver, Éamonn Linehan, and Mike Spence - for all the insightful discussions and expert opinions about my research. Thank you also to Anthony Harrington, Andronikos Nedos, Tim Walsh, and Raymond Cunningham for their review and valuable suggestions regarding the thesis.

Special thanks to Neil O'Connor, Andrew Jackson, Alan Gray, Ioanna Stamouli, and Jenny Munnelly for their constant support, encouragement, and friendship. You were always there when needed whether it's to discuss an idea, gain feedback about a problem, or simply just to listen.

To my other colleagues in the Distributed Systems Group, thank you for making it such a relaxed and enjoyable research environment to work in and it has been my pleasure to be a part of. I would also like to thank the Irish Research Council for Science, Engineering, and Technology who provided the funding for this research.

Finally, I would like to thank Laura and my family who have always been supportive of whatever I choose to do. I am forever grateful for their continuous love, understanding, and encouragement - thank you.

Shiu Lun Tsang
University of Dublin, Trinity College
March 2009

Abstract

Personalisation is the process of tailoring application behaviour to the requirements of its users. A primary concern of personalisation is providing users with behaviour recommendations that will aid them with their tasks. Making accurate recommendations benefits users by facilitating them with their activities, such as proposing actions preferred by the user, adapting resources to the user's situation, or informing users about news or events of interest. Accurate personalised recommendations also enable applications to make more effective use of user attention (e.g., by not interrupting users with irrelevant information) and facilitate greater user acceptance in proposed actions.

In general, context-aware systems use information about the surrounding environment, known as context, to make behaviour decisions that aid users with their activities. Personalisation in existing context-aware systems works by adapting to a description of user preferences, which is either explicitly defined by domain experts or users or implicitly learnt by observing user behaviour. Both approaches are suited to small applications with well-defined domains, but are less suited to personalising larger context-aware applications for two main reasons. Firstly, both approaches rely on human input to identify, relate, and prioritise different contexts and user preferences (usually as rules or cost/similarity functions) to ensure correct behaviour. However, context-aware systems are likely to support a large set of contexts, implying a large set of context relationships and user preferences. The human task of accurately defining this set of information is correspondingly time-consuming, complex, and error-prone. Secondly, existing approaches rely on the definition of algorithms that are inherently static. Specifically, relevant information and the relative importance (or utility) of information, from which the preferred actions of users is inferred, are defined at development time. However, user preferences are likely to evolve with new experiences, and the mobility of context-aware application users means they are likely to encounter events, which represent new relationships between existing context that may affect their preferences. Changing user preferences and unconsidered context relationships will render pre-defined information inaccurate, and the static nature of current techniques means that applications are unable to automatically adapt to the changes in information that are required. Explicit user feedback is a possible solution to these challenges. However, explicit feedback is usually unsuitable due to the large cognitive load associated with ranking recommendation criteria or alternative behaviours. In addition, user mobility means that providing input when necessary is not always possible. Given the limitations of existing techniques, a new approach to personalisation for context-aware applications is therefore necessary - one which does not rely on static, development-time definition of information or require domain experts or users to explicitly specify and maintain preferences over time. This motivates the research question addressed by this thesis, which is: what techniques/algorithms are necessary to support the dynamic and implicit determination of user preferences from user behaviour, including relevant information and correct information utility, to facilitate context-aware applications in making accurate personalised recommendations to users?

To address this question, this thesis describes an investigation into a novel approach to personalisation that supports context-aware applications in making recommendations without explicitly predefining the set of context relationships, user preferences and recommended behaviour. Instead, the solution dynamically relates this information at run-time. It addresses the limitations of existing work by providing software components that facilitate user autonomy, unconsidered context relationships, and changing user preferences. At the core of the approach is the identification and examination of associations between contexts and user choices. These associations, generated from past user interactions, represent patterns in behaviour from which the preferred actions of users, for a particular context, are inferred. The main contribution of this thesis is the provision of a multi-staged recommendation process consisting of a set of operations and algorithms that: automatically elicits user preferences from user behaviour; dynamically filters relevant information and adjusts the relative utility of information to reflect the current context and up-to-date preferences of the user; and dynamically generates and ranks competing recommendations. The process uses existing techniques and algorithms (employed in various other fields), which are adapted and combined in a novel manner to provide an integrated approach for supporting personalisation in the context-aware domain.

The effectiveness of the approach in providing accurate personalised recommendations is evaluated by comparing recommended behaviour against the preferred choice of users under different contexts. Two user studies and a set of computer simulated empirical experiments were conducted to assess accuracy. The results of two implemented recommendation strategies along with the accuracy measures for information filtering and utility adjustment algorithms are described. In addition, several of the most relevant recommendation approaches (rules, preference models, case-based reasoning, and neural networks) were implemented and evaluated, and the results of a study comparing the accuracy of these approaches against the solution described in this thesis are presented.

Publications Related to this Ph.D.

Shiu Lun Tsang and Siobhán Clarke. Mining User Models for Effective Adaptation of Context-aware Applications. In the *International Journal of Security and Its Applications* (IJSIA), Vol.2, No.1, January 2008. SERSC.

Éamonn Linehan, **Shiu Lun Tsang,** and Siobhán Clarke. Supporting Context-awareness: A Taxonomic Review. Technical Report, Department of Computer Science, Trinity College Dublin, TCD-CS-2008-37, `http://www.cs.tcd.ie/publications/tech-reports/reports.08/TCD-CS-2008-37.pdf.`

Shiu Lun Tsang and Siobhán Clarke. Mining User Models for Effective Adaptation of Context-aware Applications. In *Proceedings of the International Conference on Intelligent Pervasive Computing* (IPC), Jeju Island, South Korea, 2007. IEEE.

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

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

Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

Personalisation is the process of tailoring application behaviour to the requirements of its users. Personalisation benefits users by facilitating them with their work tasks, making more effective use of their attention, and increasing their satisfaction. Current approaches to personalisation are inadequate for context-aware applications for two main reasons. They rely on human input to identify and relate different contexts, user preferences, and appropriate behaviour - a task which is time-consuming, complex, and error-prone. They also rely on specifically defined algorithms that are inherently static and therefore unable to automatically adapt relevant decision making information to accommodate changing user preferences. This thesis examines the challenge of personalising context-aware applications and proposes an approach that supports the dynamic and implicit determination of user preferences, including relevant decision-making information and information utility (i.e., its relative importance), for a particular context. Behaviour recommendations that satisfy these preferences are subsequently generated, ranked and presented to the user. This introductory chapter provides the background and motivation to this work, introduces the proposed approach to personalisation of context-aware applications, presents the contributions of the thesis, and outlines the remainder of the document.

1.1 Background

There are many existing definitions of personalisation [181, 114, 95, 85, 31]. A central and common theme in all definitions is that personalisation is a process of adapting to the requirements of users. Personalisation is increasingly prevalent and necessary in today's applications due to the benefits it provides [95]. It accommodates differences between users and facilitates them by assisting with work goals and enabling better access to information [31]. A primary concern of personalisation is providing users with recommendations, tailored to their preferences, that will aid them with their tasks [86, 95, 31]. The more accurate personalised recommendations are, the more they will benefit users. Accurate recommendations will increase user motivation in carrying out their work tasks [144, 31], make better use of user attention [53], facilitate greater user acceptance in recommended actions [144, 53], and increase user satisfaction [114]. At the core of personalisation is a description of user preferences, usually referred to as a user model [100, 11]. User models contain information such as user preference criteria, the relative importance of criteria, and past user choices. Applications adapt and make recommendation decisions based on preference data contained in a user model. For example, the Travelling Salesman Problem (TSP) is a well known recommendation problem where applications evaluate and recommend travel routes based on a user's preference for the least costly [13].

Like personalisation, the aim of mobile, context-aware computing is to tailor application behaviour. Mobile, context-aware computing is a computing paradigm in which mobile devices have access to situational information (e.g., user location, time of day, nearby people, available resources etc.) [167, 201], defined initially by Dey as "context" [60]. The increased availability and access to context in a mobile setting facilitates applications in better adapting to its surrounding environment. In general, context-aware applications exploit context, to make behaviour decisions that will aid users with their activities. Many examples of context-aware applications exist including tourist guides [48], reminder services [61], and teaching/learning systems [76].

In existing context-aware applications and frameworks, personalisation is achieved by adapting to user preferences in the same manner as other types of context [65, 164, 107, 160]. This follows a *technology-centered* definition of context (classified by Dourish as a *representational* problem [63]), which is found in studies focused on context-aware frameworks, architectures, and components [60, 167]. A technology-centered definition defines context as a form of information that is delineable (i.e., what counts as relevant context can be defined in advance) and stable (i.e., the relevancy of context does not vary for different activities or events and can be made once and for all in advance) [63]. However, by supporting user preferences in the same manner as other forms of context, a technology-centered approach to personalisation ignores the dependencies that exist between user preferences and other forms of context i.e., the preferred behaviour of the user is predicated on their context. For example, users are more likely to accept work recommendations if it was a Monday than if it was a Sunday. The need to capture the effects of context on user preferences when building personalised systems is well supported [74, 22, 86, 121, 92]. Recent work by Palmisano and Gorgoglione in E-commerce, for example, concluded that context does matter when modelling the behaviour of people and that knowing the context in which a customer carries out purchases increases the ability to accurately predict the customers preferred behaviour [145, 74]. The context-dependent nature of user preferences is consistent with definitions of *user-centered* context [144, 22, 24, 63] (classified by Dourish as an *interactional* problem [63]) and supports the view of context as a set of conditions or factors that affect user preferences and subsequent behaviour. A user-centered definition of context is distinct from the widely adopted technology-centered approach, and defines context as a relational and dynamic property that may or may not be "contextually relevant" for a particular activity, the scope of which cannot be defined in advance, but should be dynamically determined [63].

Following from these definitions, this thesis adopts a user-centered definition of context. This work distinguishes between user preferences and other forms of *environmental* context (such as location, time of day, and weather). The thesis examines the context-dependent nature of user preferences (i.e., the effect of environmental context on user preferences) as well as the effect of preferences on the contextual relevance of environmental context in personalised recommendation decisions (i.e., the current user preference predicates the set of environmental context that is relevant for recommendation decisions). In particular, the thesis investigates possible approaches for implicitly capturing user preferences in context-aware applications, dealing with large sets of context and user preferences, identifying and supporting the effects of context on user preferences, adjusting the contextual relevance of environmental context to the current user preference, and tailoring context-aware application behaviour to users.

1.2 Motivation

Current mobile, context-aware applications, like previous personalised desktop applications, adapt to changing context and user preferences using either explicit or implicit techniques. Explicit techniques rely on users or domain experts to explicitly specify context information and the corresponding behaviour that is appropriate. With explicit techniques, experts are required to gather information about how applications should adapt. When adapting to user preferences for example, experts will require users to prioritise recommendation criteria or to rank alternative behaviours. This information is subsequently used to explicitly define the set of contexts (e.g., user preferences) supported by the application and the corresponding behaviour that should be recommended or executed in that context. Rule-bases are a common and widely adopted example of explicit techniques (e.g., if *Context A* then recommend *Behaviour 1*). Implicit techniques, in contrast, do not require users or domain experts to explicitly identify and specify how an application should behave under different context. Instead, applications are trained about how to adapt from past user behaviour or from training examples. However, experts are required to define a learning function (e.g., reward function or similarity function) that trains the application about how to respond to different user preferences or context. In case-based reasoning (CBR) for example, experts specify the information types and corresponding similarity measures that make up a similarity function (e.g., a travel application may consider origin, destination, price, and total time for comparing flights [55]).

Existing personalisation techniques provide a suitable approach for many current context-aware systems as they are generally confined to well defined domains, limited in the number of context types they support, and restricted to making recommendations of one particular type (e.g., a *route* recommendation in TSP). However, both explicit and implicit techniques used for specifying behaviour in current context-aware systems are limited in personalising larger context-aware applications that support context-dependent preferences. In particular, they are limited in supporting important personalisation tasks such as identifying and representing context-dependent user preferences (i.e., the effect of context on user preferences) and adjusting the contextual relevance and utility of environmental context in recommendation decisions to match the user's current context and up-to-date preferences. There are two main reasons for this:

- Current personalisation techniques rely on humans (domain experts or users) to identify, relate, and prioritise context and preferences to ensure correct behaviour. However, this task is complex, error-prone, and time-consuming particularly with large sets of information; and
- Current personalisation techniques rely on specifically defined algorithms or functions that are inherently static, with information relationships, relevant information, and the relative importance (or utility) of information used in making recommendations, defined at development time. The definition of these algorithms/functions is a non trivial task and they will result in inaccurate recommendations if relevant decision making information or utility changes, which occurs as a result of a user's preferences changing or when new relationships between existing context, preferences, or behaviour are required.

Many studies have shown that user preferences are affected by context and therefore need to be considered in order to maintain the utility of recommended behaviour [74, 22, 86, 136, 92]. In particular, the effect of context on user preferences and how preferences predicates the contextual relevance and utility of other environmental context are important factors that affect the accuracy of recommended behaviour (e.g., a user's location will be more relevant and important than the time when recommending nearby ATMs, but the opposite may be the case when recommending current TV programmes of interest to the user). To support such context-preference dependencies using existing techniques, experts are required to explicitly identify the set of possible user preferences and define the appropriate relationships between preferences, context and behaviour. However, this human task is difficult due to the dynamism of context-aware environments and the growing availability of sensor information. Context-aware applications operate in highly dynamic environments. The mobility and spontaneous interaction between different people and devices means that the current situation of the user is constantly changing [11, 147, 171, 23]. In addition, advances in sensor, mobile, and network devices are providing context-aware applications with greater availability and access to information. This dynamism and increased availability will see applications needing to support and respond to a larger number of more complex contexts, user preferences, and behaviours in order to adapt effectively. For example, an application that supports 10 independent contexts types, each with 5 different values, will need to consider 5^{10} (i.e., almost 10 million) possible different context combinations. To support context-dependent preferences with existing approaches would require humans (users or experts) to identify the set of context combinations that are *relevant*, the effect of each combination on user preference, and the corresponding appropriate behaviour. That is, a user preference will need to be correctly identified and related to each of the relevant 5^{10} possible context combinations and the appropriate behaviour recommendation will need to be determined and associated with each context and user preference pair. The large number of possible cases and the difficulty in accurately defining relationships between context, preferences, and behaviour makes the human task of correctly capturing this information extremely time-consuming, complex, and error-prone.

The task of accurately defining information relationships is further complicated because of changing user preferences. Various studies have shown that the needs of users are constantly changing [23, 11, 171, 147, 134]. Changes are either *abrupt* or *gradual*. An abrupt change is a change in preference as a result of events [113, 124]. Abrupt changes reflect the context-dependant nature of preferences and reaffirms the need to support changes to the preferred behaviour of a user as a result of a change in the user's context (e.g., a user's preferred mode of transport may change as a result of an abrupt change in weather). User preferences are also recognised to change gradually or slowly over time [26]. Gradual changes occur due to a person gaining new experiences or from growing older (e.g., a user's taste in music is likely to evolve over time). Abrupt changes make it difficult for developers to identify and define relationships between context and preferences (as a different preference may be required for each context combination) while gradual changes may result in defined information relationships, which were previously correct, becoming incorrect and therefore resulting in inaccurate recommendations. Reusing the previous example, the difficultly in identifying and representing information relationships and changing user preferences means that one of the 5^{10} possible contexts is likely to be missed or incorrectly defined. Inaccurate recommendations will therefore result until the relevant relationship definitions are corrected.

Recommendation algorithms that learn from user behaviour eliminate the limitations associated with the need to identify, relate, and pre-define all relevant relationships between context and user preferences. However, these algorithms are also pre-defined with details about information relationships, relevance, and utility that are to be considered in application decisions (e.g., a Bayesian network structure is statically defined to represent information relationships that are relevant in an application while a CBR similarity function pre-defines a set of relevant information used for measuring the similarity of cases). Changes to user preferences would therefore render pre-defined information as inaccurate and as these approaches rely on statically defined functions, they are unable to automatically adapt to necessary changes and require explicit human modification in order to be corrected (e.g., a defined CBR similarity function would need to be explicitly modified to accommodate any changes to the relevance or utility of information items used to compare case similarity).

The ability of a person to identify, relate, and define relevant context and preference information represents an example of the *Information Overload* problem [123, 77]. As available information increases, the human task of identifying, relating, and prioritising relevant context and user preferences to ensure correct application behaviour becomes correspondingly time-consuming, complex, and error-prone. Difficulties associated with this task such as knowledge discovery (e.g., determining what user preference and behaviour applies in a certain context), relating and prioritising information, and maintenance and efficiency are well recognised [149, 72]. The resultant effect is inefficient and ineffective decision-making [68, 187].

Explicit user input is a possible solution to these limitations and is adopted in the personalisation of traditional desktop applications to correct inaccurate recommendations and to maintain user preferences over time [157, 44, 174]. However, explicit user input is considered inappropriate as users are recognised as being poor at providing feedback about their preferences. There are a number of reasons for this. One reason is Zipf's *principle of least effort* [202], which states that people will adopt a course of action that involves the least average amount of work. This principle predicts that users will minimise the effort they expend on describing their preferences, even if it leads to inaccurate recommendations. This principle is supported by the results of various studies in linguistics [202] and information retrieval [161]. A second reason is the high cognitive load associated with explicit user input, particularly with tasks such as prioritising criteria or ranking alternative recommendations, resulting in users giving inaccurate feedback about their preferences [54]. A third reason is information overload. Users are presented with large amounts of preference-related data that they are unable to interpret, leading to an inaccurate description of their preferences. In addition, studies have shown that users do not have conscious access to their own preferences, and are therefore unable to give accurate feedback about them [159, 25, 170]. The mobility of context-aware users creates further difficulties for explicit user input. As users are mobile, providing input when necessary is not always possible and the primitive input capabilities and form factor of many existing mobile devices makes giving feedback difficult. In addition, a user's interaction with their device becomes a secondary task as they are engaged in other surrounding activities. Applications can no longer rely on prompt user input and any request for feedback may become frustrating and intrusive. In addition, a recent study by Barkhuus and Dey has shown that explicit user feedback is often inappropriate and that although users feel a loss of control, they prefer autonomy if accurate results are provided [21]. Other studies go a step further and indicate that the main reason for the success of personalised applications is because they do not require users to provide feedback or input [141, 116].

Given the limitations of existing personalisation techniques, a new approach to personalisation for context-aware applications is therefore necessary - one which does not rely on static, development-time definition of information or require domain experts or users to explicitly specify and maintain preferences over time. Existing context-aware systems and traditional personalised (recommender) systems have been analysed with a particular focus on investigating: how current applications acquire user preferences; how the effect of context on user preferences is identified and represented; how preferences affect the contextual relevance and utility of environmental context in recommendations; and how possible behaviour recommendations are generated and ranked. From this analysis, the following requirements for supporting the accurate personalisation of context-aware applications are derived:

1. Context-dependant user preferences - applications must support the implicit identification and representation of user preferences, the effect of context on these preferences, the effect of preferences on the contextual relevance of environmental context, and the mapping of context and preferences to appropriate behaviour. Personalised context-aware applications are required to support adaptation along two dimensions: user preferences and context. That is, either a preference change or a context change will influence an application's behaviour recommendation.

2. Changeable user preferences - personalisation in context-aware systems must implicitly accommodate changes to user preferences, which occur either abruptly in response to a change in context, or gradually over time as the user evolves with experience. User preferences encapsulate how users decide between alternative behaviours. An important element of a user's preference therefore, is the set of information and the relative importance of information users

consider when making decisions, sometimes referred to as a *multi-attribute utility estimation* problem [183]. In personalised context-aware applications, this set of information will need to be dynamically adjusted depending on the user's recommendation problem (e.g., determining a travel route between two locations) and context.

3. Complex context-aware environments - advances in sensor and mobile devices and the myriad of interactions between devices and people mean that accurately capturing all possible relationships between context, preferences, and behaviour is difficult. Applications must accommodate a large set of different context types, values, and combinations and also adapt to previously unconsidered or inaccurate context relationships while maintaining the utility of recommendations. The complexity of the environment also means that users will necessitate recommendations consisting of different output types (e.g., recommend both the preferred route *and* preferred mode of transport). Personalisation support in context-aware applications should therefore support the evaluation of multiple different output types, recommending those that are most appropriate for the current context and preference of the user.

4. Dynamic execution (to support developers) - the ability of a developer/domain expert to accurately pre-define the set of information relationships necessary for personalising context-aware applications is difficult due to the large number of possible contexts, the probability of discovering previously unsupported context relationships, and the diverse and changeable nature of user requirements. A dynamic approach eliminates the need for experts to specify relevant information relationships. Instead, relevant relationships between context, preferences, and behaviour are dynamically determined at run-time, when required (e.g., when a user's task changes) and tailored for the current context and user preference. Dynamic execution also enables applications to evolve to support changing user preferences or previously unconsidered context relationships, without explicit input, as relationships are generated from up-to-date information.

5. Autonomous execution (to support users) - explicit user input is used by personalised applications to start a recommendation process or to correct any inaccuracies in developer-defined preferences or recommendations. However, the problems associated with explicit user input necessitate applications to act with autonomy. Applications will have greater responsibility to automatically infer the desires of its users and act transparently on their behalf. Autonomous execution facilitates a move to achieving the aim of pervasive "invisible" computing [143].

These requirements motivate the research question addressed by this thesis, which is: what techniques and algorithms are necessary to support the dynamic and implicit determination of user pref-

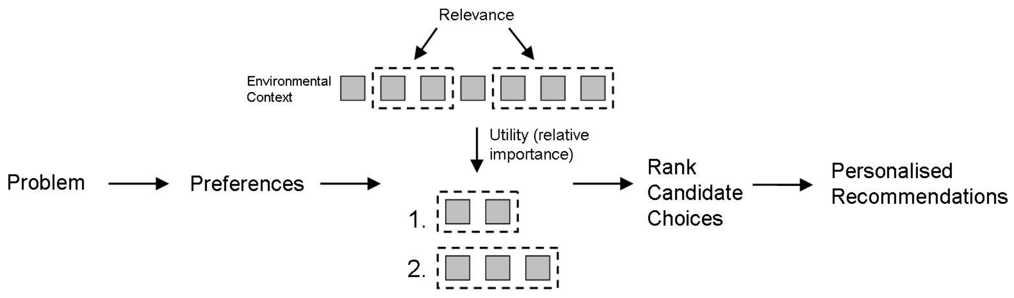


Figure 1.1: Context-aware Personalised Recommendation Process

erences from user behaviour, including relevant information and correct information utility, to facilitate context-aware applications in making accurate personalised recommendations to users?

1.3 Approach

To address the research question, this thesis describes an investigation into a novel approach to personalisation. This approach does not require information about relationships between context, user preferences, behaviour, or about the relevance and utility of information to be explicitly pre-defined. Instead, the approach dynamically determines context-dependant user preferences and relevant decision making information by implicitly identifying and reasoning appropriate information relationships at run time. It addresses the limitations of existing work by providing techniques and algorithms that facilitate user autonomy, unconsidered context relationships, and changing user requirements. Figure 1.1 illustrates the aim of the personalisation approach proposed in this thesis, which is to dynamically determine the preferences of a user for a particular problem and context. Information relevance and utility, tailored for the given problem and context, is subsequently inferred from the user's preferences and inferred knowledge is used to rank candidate recommendations. The run-time, implicit execution of these functions is designed to support applications in making recommendations that are specifically tailored to the problem, context, and preferences of users.

The approach uses existing techniques and algorithms (employed in other fields such as context-awareness, user modelling, information retrieval, and data mining) for tasks including the acquisition and management of context information, the filtering of large amounts of information, and the identification of patterns in context information and user behaviour. These techniques and algorithms are used, adapted, and combined in a novel manner, to provide an integrated approach for supporting personalisation in the context-aware domain. At the core of the approach is the identification and

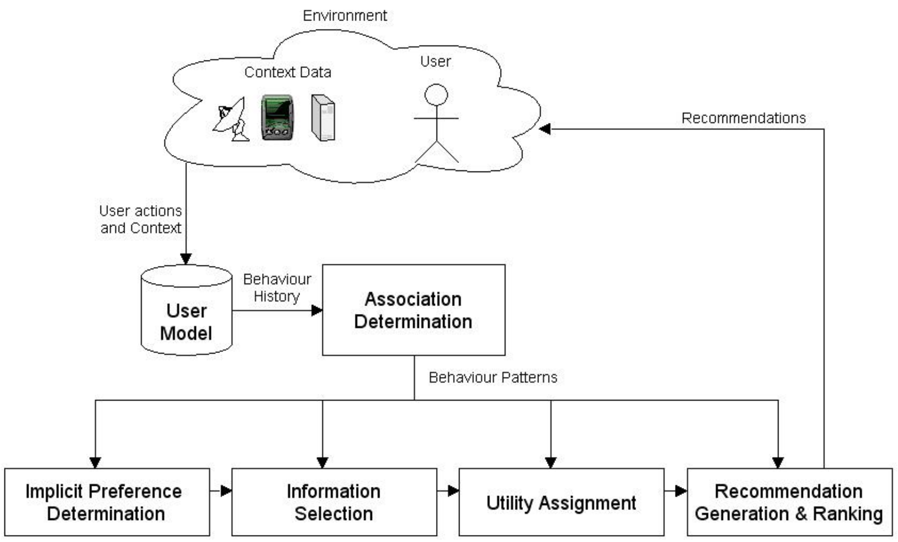


Figure 1.2: Personalisation of Context-aware Applications

examination of associations between contexts and user behaviour choices. These associations, generated from a user model of past user interactions, gathered from implicit user feedback, represent patterns in behaviour from which the preferred actions of users, for a particular context, are inferred. Details about relevant information, information utility, and criteria for ranking recommendations are also inferred from these patterns. The main contribution of this thesis is the provision of a multi-stage recommendation process consisting of a set of operations and algorithms that: automatically determines user preferences from past behaviour; dynamically filters relevant recommendation information and adjusts the relative utility of information to reflect the current context and up-to-date preferences of the user; and dynamically generates and ranks competing recommendations. Figure 1.2 illustrates this multi-stage process including the series of operations that support the personalisation of context-aware applications.

1.3.1 Implicit Preference Determination

At the core of personalisation is a description of user preferences. To ensure accurate tailored recommendations, applications must correctly determine the preferred behaviour of users for different contexts. Implicit preference determination is concerned with inferring the preferences of a user given their recommendation problem and context, without requiring explicit user input about preferences. Determined preferences should be up-to-date, capturing any abrupt or gradual changes to user preferences that occur with application use. Implicit preference determination is achieved by dynamically identifying behaviour patterns from past user behaviour stored in a user model (i.e., recurring associations between context and user choices). Several techniques adopted from other research fields are adapted and combined to complete the various tasks necessary for implicit preference determination. Methods from context-awareness are adopted for the acquisition, reasoning, and management of context [18, 132, 12, 42]. Functions from the Hermes framework [65] have been used for context-related tasks. User modelling techniques [100] and user profiles applied in content-based systems [6] were examined for representing user model data. User information such as past behaviour is gathered from implicit user feedback, therefore eliminating the need for users to explicitly specify details about their preferences. The user model is extended to also store the context at the time of each behaviour. User model data is represented in the form of cases in a similar manner as that used in case-based reasoning (CBR) [1]. Association data mining [8, 45] is used for mining user model data. Mining eliminates the need for explicit human input to identify and relate context and preferences. Instead, mining enables the autonomous and dynamic determination of association between context and user choices, which represent the context-dependent preferences of a user. These associations represent patterns in behaviour from which the preferred actions of users, for a particular context, are inferred. New user interactions and behaviour in addition to the context at the time of the behaviour are added to the user model with application use. As associations are generated at run-time using an up-to-date set of user interactions, the application can accommodate previously unconsidered context relationships or any changes to user preferences. Associations are also in a form that provides an intuitive and descriptive way of representing user behaviour and are compatible with explanation methods that enable users to query and scrutinise applications about their behaviour [27, 96]. Associations between different context types and values and user choices are generated, facilitating user preferences to be determined for multiple different combinations of context and multiple different types of recommendation problems. The CBR concept of using past similar cases to solve the current problem is adopted to determine user preferences from historic behaviour cases. The assumption is that behaviour patterns represent the preferred actions of users in a particular context [1, 53].

1.3.2 Information Selection

A user-centered definition of context defines context as a relational and dynamic property, which may or may not be contextually relevant depending on the particular activity, and should therefore be determined dynamically [63]. The growing availability of sensor information requires context-aware applications to support an increasing number of different context types and values. However, not all information available to and supported by an application is relevant for decision making [123, 4, 74].

Depending on their goal and their context, users will vary in the types of information they use for making decisions. An important task is therefore to dynamically and autonomously determine the subset of all context, preferences, and behaviour that are relevant for making recommendation decisions. Information selection (often referred to as feature selection [118]) is concerned with filtering information to determine the subset that is relevant to the current recommendation problem, context, and preference of the user.

Association mining techniques are used to discover patterns in user behaviour, which are iteratively compared to determine relevant information. Specifically, the user's action choice under various context settings is compared. From the analysis of similarities and differences that exist in patterns, relevant information that significantly influences use behaviour choices, and therefore should be considered in recommendation decisions, is inferred. Metrics from association mining, such as support and confidence [45] are used to filter generated association rules, ensuring only those that meet a particular accuracy and interestingness threshold are retained before comparisons. Techniques for textual matching and fuzzy comparisons used in information retrieval [17] and CBR [1] are facilitated for comparing information stored in behaviour patterns.

Determining the set of contextually relevant information for a particular context can also address the question of when to begin the recommendation process [65]. Starting the recommendation process at the incorrect time can lead to inaccurate recommendations, be intrusive to users, and result in sub-optimal performance (e.g., using a device's processing and battery power when unnecessary). By identifying the set of relevant information, information selection can be used to signal when a new recommendation should be evaluated and presented to the user. That is the preferred action of a user is likely to change when the value of a relevant context type changes (e.g., if a user's preferred transport option is based on the weather context, then a change in the value of the weather context (e.g., from dry to wet) will require making a new transport recommendation to the user).

1.3.3 Utility Assignment

As the set of information considered relevant for recommendation decisions varies depending on the user's preference and context, so also does the relative importance or weight information has in recommendation decisions. This is evident in existing recommender systems where users rank recommendation criteria differently in different context [114, 74]. While information selection is concerned with determining the set of environmental context that is relevant, utility assignment, in contrast, is focused on determining how contextually relevant each environmental context type is (i.e., the relative importance of each context type for the recommendation decision). The utility assignment stage therefore provides functions and algorithms for dynamically determining and assigning the correct weight (or utility) to information without the need for explicit user input, so that its relative importance reflects the preferences of a user in a particular context. The algorithm and heuristics used to dynamically determine the appropriate weight to assign to information for a particular context is constructed using information quality metrics from information retrieval [17] and data mining [45]. Precision, recall, confidence, support, and lift measures are combined and aggregated to create a customisable algorithm that determines the strength of user behaviour patterns, with information *stronger* rules given more weight in recommendation decisions. Multi-attribute utility theory [183] and decision theory [70] techniques that assign numeric weight values (or scores) to prioritise the importance of items are adopted to represent the utility of context information.

1.3.4 Recommendation Generation and Ranking

A primary goal of personalisation in context-aware applications is to provide users with behaviour recommendations that will aid them with their activities. The final stage of the recommendation process is concerned with generating and ranking candidate behaviours that best accommodate the current context and preference of the user. The implicit preference determination stage generated association rules representing the context-dependent preferences of users. These rules are then filtered, aggregated, and prioritised at the information selection and utility assignment stages to ensure that the most relevant and suitable rules that match the current user preferences and context are considered at this recommendation stage. Retained rules are used to rank any competing alternatives. Possible recommendations are generated from the current context from the domain (e.g., what actions are possible given the current context), determined using context-aware functionality [65], and a list of past actions chosen by the users, which are stored in the user model. The multi-attribute utility technique [183] is used to rank each of the possible behaviours using the outputs of information selection and utility assignment. The integration of this technique with the personalisation approach proposed in this thesis facilitates candidate behaviours to be scored using multiple dimensions with different levels of relative importance. The highest ranked behaviour (or a number of highest ranked behaviours) is then presented as recommendations to the user. Rules generated at the preference determination stage support the association of multiple different context and problem types, therefore facilitating the ranking and recommendation of multiple different recommendations (e.g., recommend both the preferred route *and* preferred mode of transport).

Presenting recommendations to users represents a *passive* approach, which enables user's to ultimately decide whether recommended behaviour should be executed or not [21]. It lies in between complete *personalization*, where users are in full control and specify how an application should behave, and an *active* approach, where applications act with full autonomy including making actual changes to application behaviour. Users have been shown to prefer a *passive* approach over *personalization,* while no significant preference was concluded between *passive* and *active* approaches [21].

1.4 Contributions

The personalisation approach described in this thesis contributes to the state of the art in the area of mobile, context-aware personalisation by addressing the following issues:

- Current approaches for determining context-dependant user preferences require users or domain experts to explicitly identify and represent *relationships* between context and preferences (typically as rules or learning functions) to ensure correct behaviour. However, the accurate definition of this information is difficult due to the large set of (complex) context and preferences relationships and the changeable nature of user preferences. This thesis describes a dynamic approach to determining context-dependant preferences, where preferences for a particular recommendation problem and context are automatically inferred from past user behaviour.

- Current approaches to personalisation require domain experts or users to statically specify the set of information to be considered *relevant* in recommendation decisions, typically at development time. However, when context, preference or behaviour definitions are incorrect, when previously unconsidered context relationships arise, or when user preferences change, a static approach is unable to automatically adapt and will therefore result in inaccurate recommendations. This thesis describes an approach that addresses this limitation by providing a technique that dynamically and autonomously determines, from past user behaviour, the set of relevant information, which is tailored for different recommendation problems, context, and user preferences.

- Current approaches to personalisation require domain experts or users to statically specify the relative *utility* of information considered in recommendation decisions. As with statically defining relevant decision-making information, a static approach to specifying utility means that applications are unable to automatically adapt when utilities, which represent user preferences, are incorrectly defined or when user preferences change, therefore resulting in inaccurate recommendations. This thesis describes an approach to utility assignment, where the relative weight attributed to information considered in recommendation decisions is dynamically and autonomously adjusted according to the problem, context, and preferences of the user.

- Current approaches for generating and ranking competing recommendations require explicit input from users or domain experts and are limited by the number of recommendations types they can evaluate. However, accurately determining and ranking behaviour is difficult due to the dependency between context and preferences as well as the changeable nature of user preferences. This thesis describes a technique which dynamically generates and ranks multiple recommendation types. The approach considers user preferences, context, and possible behaviours and prioritises recommendations by examining the strength of user behaviour patterns.

In summary, this thesis describes the development of a multi-stage recommendation process, which uses, adapts, and combines existing techniques and algorithms in a novel manner, for implicit preference determination, information selection, utility assignment, and recommendation generation and ranking to provide an integrated approach for supporting personalisation in context-aware applications.

The contributions of the thesis are evaluated by comparing behaviour recommendations against the preferred choice of users under different context. The ability of the approach to make recommendations which match actual user choices validates the contributions regarding the multi-stage recommendation process and the determination of context-dependant user preferences. The effectiveness of information selection, utility assignment, and recommendation ranking algorithms are also evaluated by comparing them with user choices. Two user studies and a set of computer simulated empirical experiments were conducted to assess the accuracy of two developed recommendation strategies. In addition, several of the most relevant recommendation approaches (rules, preference models, case-based reasoning, and neural networks) were implemented and evaluated, and the results of a study comparing the accuracy of these approaches against the algorithms described in this thesis are presented.

1.5 Thesis Outline

The remainder of this thesis is organised as follows. Chapter 2 presents an overview of the state of the art in personalisation of context-aware applications. Chapter 3 describes the design of the personalisation approach proposed in this thesis, including the techniques and algorithms designed for different stages of the approach. Chapter 4 describes the implementation of the approach. Chapter 5 presents its evaluation, including a description of user studies and simulated empirical experiments conducted to assess the accuracy of the approach in making personalised recommendations to users. Finally chapter 6 summarises the contributions of this thesis and discusses several directions for future work.

Chapter 2

State of the Art

This chapter examines state of the art projects that provide an approach to support personalisation in context-aware applications. Recommender systems, context-aware applications and frameworks, and projects that focus specifically on personalisation in context-aware systems are reviewed. A critical evaluation of how these projects support the requirements described in Section 1.2 is also presented. While a number of state of the art projects offer support for the development of personalised context-aware applications, this thesis concludes that no single approach offers support for the full set of requirements needed for effective personalisation in context-aware applications.

2.1 Recommender Systems

This section presents an overview of recommender information systems and discusses the approaches they provide to tailor behaviour recommendations to the preferences of users. An analysis of how these systems address the requirements outlined in Section 1.2 is also presented.

2.1.1 Overview

Recommender systems are designed to aid users by recommending items that match their preferences in situations where they have too many options to choose from [6, 134].

At the core of recommender systems is a description of user interests/preferences represented in a user model. Several techniques in existing recommender systems have been adopted for the representation of user models. The most commonly used representation approaches are history-based models, user-item ratings, and vector (or feature) space models: history-based models keep a list of a user's past interactions such as their purchasing history or navigation history to represent user interests; user ratings contain a record of user-item ratings that are explicitly inputted by users; and vector space models represent user preferences as a vector of features with an associated utility value (e.g., a song will have features such as artist and genre and an associated preference rating).

As user preferences and interests change over time [23, 11, 171, 147, 134], a user model should also change in order to maintain the utility of recommendations. Relevance feedback from the user is required for systems to monitor how user preferences change. Feedback is either explicit, which requires users to explicitly maintain user model information, or implicit, which automatically infers changing user's preferences by monitoring their behaviour.

There are three main methods for filtering and recommending candidate items so that only those that match the user's preferences are recommended. They are: content-based (or case-based), collaborative, and hybrid approaches [6, 134]. With content-based recommendations, users are recommended items similar to those they preferred in the past. A user profile is built from descriptions of items a user previously expressed interest in (e.g., by purchasing or rating a particular item). Recommendations are made by comparing features of candidate items with features contained in the user profile. Those that match or are sufficiently similar are recommended to the user. With collaborative recommendations, users are recommended items that *other* people with similar preferences have selected in the past. That is, the utility of items for a particular user is based on items previously rated by other similar users. A user profile is built from the behaviour of a group of similar users, which is then used to make recommendations for an individual user. A hybrid approach combines content-based and collaborative methods.

Many recommender systems from various domains are currently available. For example, the Personal TV system (PTV) [52] is a system that provides personalised TV listings based on the viewing preferences of users. User profile information is collected by asking users to explicitly provide feedback about TV programmes they like and dislike. A hybrid recommendation strategy using both collaborative and content-based methods is used when making programme recommendations to users. News4U [93] is a personal newspaper, which selects news items from a variety of sources based on the interests of its user. User models are built and maintained by explicit user feedback, which require users to express their topics of interest. News4U uses a content-based approach whereby topic keywords in the user model and news items are compared and when matched, are presented to the user. Letzia [115] is a system that recommends web-page hyperlinks by comparing them with pages previously visited by the user. The user's profile is constructed by extracting keywords from visited pages. These keywords are subsequently used to grade new pages to determine those to recommend to the user. Finally, Ama-

Name	Domain	Preference Representation	Recommendation Approach
Amazon [116]	E-Commerce	History	Hybrid
CASPER [33]	Recruitment	History	Content-based
CDNow [40]	E-Commerce	History, User-item rating	Hybrid
Entree [37]	Restaurant Recommender	Feature Vector	Content-based
GroupLens [106]	News	User-item rating	Collaborative
I-SPY system [19]	Web Search	History	Collaborative
Letzia [115]	Web Recommender	History	Content-based
LIBRA [30]	Book Recommender	User-item rating	Hybrid
MovieLens [73]	Entertainment	Feature Vector	Hybrid
Musicmatch jukebox [191]	Entertainment	Feature Vector	Content-based
News4U [93]	News	Feature Vector	Content-based
NewT [123]	News	Feature Vector	Content-based
PResTo [97]	Web Search	History	Content-based
PTV [52]	Entertainment	User-item rating	Hybrid
SmartRadio [82]	Entertainment	User-item rating	Content-based
SMMART [112]	E-Commerce	Feature Vector	Content-based
TURAS [127]	Travel Route Recommender	History	Content-based
WebSail [46]	Web Search	Feature Vector	Content-based

Table 2.1: Recommender Systems Overview

zon [116] supports personalisation by maintaining a user's set of purchased products and other similar products purchased by similar users are recommended. Table 2.1 illustrates the domain, preference representation method, and recommendation approach of a number of state of the art recommender systems.

2.1.2 Analysis

The set of surveyed systems are examples of recommender systems that adapt to user model information such as user interests and user preferences. However, user preferences only give a partial expression of a user's needs. Studies such as those by Gorgoglione [74] and Barnard [22] show that other types of (environmental) context such as location and time (in addition to user preferences) are also important as they impact on user preferences and affect the accuracy of recommendations. Consequently, the main limitation of these existing recommender systems is the limited support for environmental context in their design. For example, in the PTV application the fact that people watch TV with other people is not taken into account and shopping systems such as Amazon and CDNow do not consider that users may be shopping for other people who have different interests. Several appli-

Name	Preference Feedback	Supported Context
Amazon [116]	Explicit, Implicit (purchase history)	purchased items
CASPER [33]	Implicit (job application history)	search terms
CDNow [40]	Explicit, Implicit (purchase history)	purchased items
Entree [37]	Explicit	food preferences
GroupLens [106]	Explicit, Implicit (time spent)	movie features
I-SPY system [19]	Implicit (search history)	search terms
Letzia [115]	Implicit (links, time spent)	web page content
LIBRA [30]	Explicit	search terms
MovieLens [73]	Explicit	music features
Music Match jukebox [191]	Explicit, Implicit (playlist history)	music features
News4U [93]	Explicit	news topics
NewT [123]	Explicit	news topics
PResTo [97]	Implicit (search terms, links)	search terms
PTV [52]	Explicit	programme ratings
SmartRadio [82]	Explicit, Implicit (playlist history)	music ratings, current playlist
SMMART [112]	Explicit, Implicit (purchase history)	item ratings, purchased items
TURAS [127]	Implicit (travel history)	start, end location, travel history
WebSail [46]	Explicit	search terms

Table 2.2: Recommender Systems Supported Feedback and Context

cations such as CASPER and Entree provide limited support for context by enabling user to explicitly specify their current context through search terms. Other applications such as Letzia and SmartRadio provide support for implicitly inferring user context by recording the user's current behaviour. However, these applications are limited by the number of context types they support. Existing approaches for representing user models such as user history, user-item ratings, and vector spaces also are limited in their support for context in their representation. As an extension to the limited support for context, methods that will aid users and developers to identify and represent dependencies between user preferences and context are also unsupported by current recommender systems, therefore restricting the ability of recommender applications to adapt user preferences to different context.

Content-based and collaborative approaches used in recommender systems are also inherently static and rely on users or developers to define relevant information and information utility. In state of the art projects, relevant user model features (content-based) and similarity measures (collaborative) are explicitly defined by developers. Preferences information such as the utility of features and item ratings typically require explicit user feedback or input. The static nature of content-based and collaborative approaches also means that human input is required to accommodate any changes to user preferences. That is, users are required to re-rank items or developers are required to re-define relevant information and information utility to ensure that the system evolves with user preference changes. Some systems (e.g., Letzia and SmartRadio) do provide implicit techniques for acquiring and evolving user preferences, but these are restricted to very specific application types such as history of purchases or navigation history, and more importantly are limited by the number of context types they consider when determining preferences. While human input is acceptable for many traditional desktop based personalised systems, the problems outlined in Section 1.2 show that autonomous techniques for identifying and maintaining context-dependent user preferences are required. This view is supported by Keenoy and Levene [97]. As part of their survey of web search personalisation techniques, they outlined a set of requirements for an "*ideal*" personalised search system. Central to these requirements are the need for the implicit collection of user preference information and the automatic adaptation to current up-to-date user preferences. Table 2.2 summarises the preference feedback approach and context types supported by the surveyed state of the art personalised recommender systems.

Other limitations with content-based and collaborative approaches also exist. A content-based approach is limited by its tendency to over specialize to particular items described in the user's profile and the need to represent features as textual descriptions. Collaborative approaches are limited by their lack of scalability in terms of efficiency and the sparse nature of data. They are also restricted by privacy concerns associated with sharing data and the difficulty in identifying similar users.

The limited support for context and dependencies between context and user preferences, and the reliance on explicit human input limits the support for personalisation techniques used in existing recommender systems to provide accurate personalised recommendations in context-aware applications. However, a survey of these personalised systems, along with the techniques and approaches they utilise, provide a useful insight into how preferences can be identified, captured, and represented in context-aware applications.

2.2 Context-aware Applications and Frameworks

The aim of context-aware systems, like personalised recommender systems, is to tailor application behaviour. However, context-aware applications generally consider more types of situational information (e.g., location, time, weather) when making behaviour recommendations. This section presents a description of the basic functionality of several context-aware applications and frameworks along with an analysis of how they support the requirements outlined in Section 1.2.

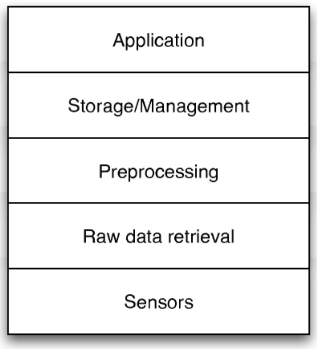


Figure 2.1: Layered Conceptual Structure for Context-aware Systems [18]

2.2.1 Overview

A large body of research in the context-awareness domain has focused on creating architectures, infrastructures, middlewares and frameworks that provide functionality to support developers in building context-aware applications. The primary goal of these frameworks is to make context information available to application developers so that they do not need to concern themselves with the complexities of context-related tasks. These frameworks provide "uniform abstractions of common functionality and reliable services for common operations" [88]. Core context-related functionality provided by these frameworks includes operations for the collection, management, and use of context information, which serve as the building blocks for developing context-aware applications.

The functionality provided by existing context-aware frameworks and applications have been reviewed and categorised in many recent studies [18, 132, 12, 42, 2, 117]. The conceptual structure defined by Baldauf et al. (shown in Figure 2.1) illustrates the set context-aware functionality common to existing context-aware applications and frameworks. The structure consists of the following layers (from bottom to top):

- Sensors - this layer consists of a collection of data sources that may provide context information. Sensors are classified as physical, virtual, or logical. *Physical* sensors are hardware devices such as colour sensors, microphones, and GPS systems, which provide light, audio, and location context respectively; *virtual* sensors are software applications or services such as electronic calendars, emails, and keyboard input; and *logical* sensors combine physical and logical sensors e.g., determining user location by analysing logins at desktop PCs and subsequently querying a mapping database about the location of devices.
- Raw data retrieval - this layer encapsulates functionality associated with the retrieval of raw

context data. It uses appropriate drivers and APIs to provide reusable methods that hide the low level details of sensor access, which supports the exchange of underlying sensors (e.g., a RFID system can be replaced by a GPS system) without affecting current and upper layers.

- Preprocessing - this layer is responsible for reasoning and interpreting context information and for transforming context into a high-level representation that is internally understood by the application.
- Storage/Management - this layer organises context data acquired from lower levels and provides clients (i.e., applications) with synchronous or asynchronous access to context.
- Application - this layer defines the actual reaction to different context events such as the suitable behaviour to recommend to application users.

Personalisation is not a primary concern of existing context-aware applications and frameworks - context-related functionality such as acquiring, reasoning, and managing context information is. However, personalisation *is* supported by existing context-aware applications and frameworks and this is achieved by supporting user preferences in the same manner as other types of context. User preferences, like other context types, are acquired through the sensor and raw data retrieval layers (e.g., through user keyboard input), are interpreted and modelled at the preprocessing and storage/management layers, and used by applications to recommend appropriate personalised behaviour.

Several existing context-aware *applications* explicitly mention user preferences in their design. For example, the GUIDE system [48] is a mobile context-aware tour guide that provides city visitors with information about tourist attractions based on their location and profile information. User profile information is captured by asking users about their preference for *architecture*, *maritime*, or *history* and about their preferred *reading language*. Environmental information such as the *time of day* and *opening/closing time* of attractions are also considered when personalising information to present to users. Similarly, P-Tour [125] is a personal navigation system for tourism, which computes a personalised travel schedule for tourists that maximises the number of destinations they can visit, considering both user preferences and other environmental context. User preferences are acquired by requesting users to assign rankings to candidate *destinations* and their preferred *staying time* for each destination. Environmental context such as the user's *current location, the location of each destination, and the destination time restrictions* (i.e., opening and closing times) are also considered when scheduling destinations. Schedules are automatically modified based on context changes, which are detected by monitoring the time and current user location or by users modifying their preferences (e.g., a user changing their staying duration at a destination). Other context-aware tourist guide applications and smart home applications which adapt their behaviour to both user preferences and context in a similar manner to GUIDE and P-Tour include Dynamic Tourist Guide [109], the museum guide by Zancanaro et al. [196], CRUMPET [148], Personal Home Server [139], Smart Home [122], Aware Home [179], and Microsoft Easy Living [129].

Several context-aware *frameworks* also explicitly mention user preferences in their design. User preferences in these frameworks are also supported in the same manner as other types of context information. For example, the Stick-e Document [35] and Stick-e Note frameworks [146] enable items (documents or notes) to be attached to particular context, so that when a user enters such a context, these items will be invoked. Users express their preferences by indicating the *types* of documents or notes they wish to receive. Only document and note types specified as part of these preferences are triggered and presented to the user when detected by the system. In the Gaia project [160], the *user centrism* property requires applications to provides mechanisms that allow users to configure applications to their personal preferences and ensures that applications move with the users and adapt according to changes in the available resources. User preferences are mounted onto the *Context File System* and are accessed by applications when necessary. Rule-based and machine learning approaches (Bayesian Networks, Neural Networks, and Reinforcement Learning) are supported for adapting application behaviour to context such as user preferences. Personalisation in MyCampus [164] is supported by providing users with access to a *personal environment* from which they control access to their personal preferences and contextual attributes such as calendar information and current user location. Users can personalise the operation of their personal environments on their PDAs by installing *context-aware agents* (in a similar way to mobile phone users downloading new ring tones) to perform specific tasks while taking into account relevant user preferences and context. For example a restaurant concierge agent would provide suggestions about where to have lunch depending on the user's food preferences, time available before the next class, and the weather. Ontologies are used to describe context data and user preferences. A *STRIPS*-like (Stanford Research Institute Problem Solver) [69] formalism is used to define behaviour. The Hermes framework [65] supports user preferences using *multi-attribute utility estimation* techniques [183], enabling users to specify preferences regarding activity scheduling in context-aware trails-based applications. Users express their preferences by adjusting the relevance and utility of context considered in recommendation decisions. For example, users can express their preference for activities that are close to their current location by increasing the utility of the *proximity* variable. Finally, Merino [111] provides a layered architecture for context-awareness. User models are stored in a distributed context database using the *Personis* user modelling framework [96]. User model data such as user location or preferences, termed *evidence*, are acquired from sensors or user input. *Resolvers* enable the user to control the classes of evidence made available while the *view* mechanism enable subsets of the user model to be queried. A critical element of the Personis framework is to provide interfaces, enabling users to query any part of the user model and make changes where necessary. Other context-aware projects, like Merino and Personis, have also focused on providing appropriate interfaces that support users in specifying preferences such as those by Weis [185], Riekki [156], and Korpipaa [107].

Other context-aware frameworks such as the Context Toolkit [62], the Java Context-Awareness Framework (JCAF) [20], the Sentient Object Model (SOM) [29], and the Technology Enabling Awareness (TEA) layered architecture by Chen [41] do not explicitly mention user preferences in their design. However, personalisation can be achieved by these frameworks in a similar manner to the previously described applications and frameworks (i.e., by acquiring and adapting to preferences in the same manner as other context types). For example, in the Context Toolkit, widgets and interpreters can be applied to the acquisition and interpretation of user preferences as they would other context such as user location. Similarly JCAF provides Java-based operations, SOM provides context modelling and rule-based inference functionality, and TEA provides a set of commands organised in a layered fashion that can be applied to support the gathering and management of, and adaptation to, user preferences. Other examples of frameworks which do not explicitly include user preferences in their design but support preferences as they do other context types include ContextFabric [89], Context Shadow [94], Hydrogen [87], HyCon [32], SOCAM [78], and CoBrA [43].

2.2.2 Analysis

A review of existing context-aware applications and frameworks has shown that personalisation is provided by supporting user preferences in the same manner as other types of context. This enables operations designed for context acquisition, reasoning, and adaptation to be reused and applied to the management of user preferences.

However, by supporting preferences in the same manner as other types of context, existing frameworks are limited in their support for expressing the dependencies that exist between user preferences and other *environmental* context. Specifically, users or developers are responsible for identifying and representing a user's behaviour preference under different context settings. This requires identifying the set of relevant context relationships and subsequently associating these relationships with user preferences and appropriate behaviour. Identifying and defining the utility of different context (and context relationships) for recommendation decisions is also required. No methods or techniques are currently provided by existing context-aware applications and frameworks to aid users or developers in dynamically identifying and representing the effect of different and changing context on the preferred actions of users. For example, with rule-based adaptation in SOM, developers are responsible for specifying and prioritising a set of context-action rules to ensure correct personalised behaviour. Similarly, with Bayesian Network adaptation in Gaia, expert knowledge and input is required to define the structure of the network, including modelling the relevant set of context relationships and calculating the associated set of probability distributions. The difficulty of this human task of identifying and specifying context-dependent preferences highlights the need for techniques and algorithms that will automatically identify and represent the preferred actions of a user in different contexts.

In addition, any changes to user preferences are not autonomously supported by existing context-aware applications and frameworks. To accommodate changing user preferences and to maintain the utility of recommendations over time, existing approaches currently require explicit human (developer or user) knowledge and input. Updates to defined context, preference, or behaviour information as well as to the relevance and utility of information in recommendation decisions will be necessary. State of the art frameworks require users or developers to manually modify preference details such as rule definitions or model structures to cater for preference changes. For example, the Hermes framework enables users to modify their preferences by adjusting the relevance and utility of context in recommendation decisions. In SOM, rules will need to be re-defined or re-ordered. In Gaia, the Bayesian Network structure will require re-modelling or the probability distribution tables will require updating. Online learning facilitates applications to automatically adapt to changing user preferences. However, online machine learning also requires extensive expert knowledge and current context-aware applications and frameworks do not explicitly support online learning in their designs. Explicit human input provides a possible approach to maintaining user preferences over time (and is the approach adopted by existing context-aware applications and frameworks). However, this thesis contends that explicit user feedback is not always accurate or possible (as described in Section 1.2), and dynamic methods that autonomously react and adapt to changing context and user preferences at the appropriate time are required for effective personalisation in context-aware applications.

Logically, the autonomous determination of context-dependent user preferences would occur at the preprocessing stage of the conceptual structure of context-aware functionality; the storage/management layer would provide methods for representing and accessing these preference; and the application layer would support functionality that interprets the preferences of a user for the current context, including the determination of context relevance and utility, for making personalised behaviour recommendations to users.

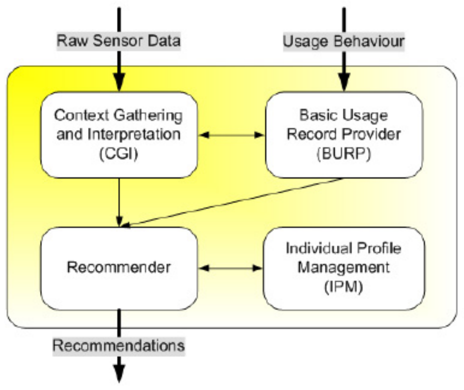

Raw Sensor Data
Usage Behaviour
Context Gathering and Interpretation (CGI)
Basic Usage Record Provider (BURP)
Recommender
Individual Profile Management (IPM)
Recommendations

Figure 2.2: MobiLife Contextual Personalisation Architecture [176]

2.3 Approaches to Context-dependant Personalisation

While current context-aware applications support adaptation to context and user preferences, they are not specifically focused on supporting dependencies between user preferences and other types of context. This section reviews the most related state of the art projects in context-awareness that focus directly on addressing the challenge of supporting personalisation (and in particular identifying, representing, and adapting to context-dependent user preferences) in context-aware applications. These projects are evaluated against the requirements described in Section 1.2.

2.3.1 MobiLife

The EU-IST MobiLife project [176, 165] conducts research aimed at developing an architecture to support the contextual and personalised adaptation of multimodal mobile applications for users. The architecture of MobiLife is based on a generic context management framework, which provides approaches for managing, reasoning about, and exchanging context data.

A primary element of MobiLife is the *Contextual Personalization Architecture*, developed as part of the overall MobiLife architecture, which provides personalisation support for services and applications. The central aim of the architecture is to provide support for developers so they do not have to repeatedly tackle the challenges associated with context processing and personalisation. The architecture provides components to support tasks such as acquiring and interpreting context, inferring appropriate behaviour, and storing and learning user preferences. At present, several components of the architecture have been developed and are illustrated in Figure 2.2: *Context Gathering and Interpretation* (CGI), *Basic Usage Record Provider* (BURP), *Individual Profile Management* (IPM), and *Recommender*. The CGI component is responsible for collecting sensor data and determining the user's current context from this data; the BURP stores historical user behaviour; the IPM manages user preferences and is responsible for linking user preferences with a particular context; and the Recommender uses data from BURP and IPM to model user behaviour patterns for a particular context (i.e., context-dependent user preferences) and makes recommendations accordingly. User data is described in user models, which contain *domain* submodels and *conditional* submodels. Domain submodels describe user preferences for a particular domain or function (e.g., a user's preferred newspaper content) and conditional submodels describe user information for a particular context (e.g., user preference for newspaper content at a particular time of day). User data is represented using RDF(S) and OWL representation languages. Two approaches are currently supported by the Recommender - rule bases and Bayesian Networks - to provide personalised service adaptation to users.

2.3.1.1 Analysis

The main aim and contribution of the MobiLife project is the provision of a context management architecture that supports the collection, management and adaptation of context and preference information. However, while MobiLife provides a high level architecture for context-awareness and personalisation support, implementation is ongoing and precise descriptions about supported functionality are currently unavailable. MobiLife does not currently support the implicit elicitation of user preferences. The use of RDF(S) and OWL indicates that developers are required to explicitly identify and statically specify the set of context relationships and preferences that are relevant and supported by the application. Rule-based and Bayesian Network adaptation approaches currently supported also require explicit expert input, either to construct rules or structure network models. Context-dependent preferences are supported by developers defining all possible context and corresponding user preferences, therefore limiting the number of context and output (behaviour) types an application can accurately consider. Techniques such as association mining could be applied to the architecture to implicitly link user actions and context to determine context-dependent user preferences. However, this is not supported and existing functionality would need to be extended in order to incorporate techniques for implicit preference elicitation with RDF(S) and OWL representation languages used by the architecture to model context. Learning from user feedback (i.e., online learning) is also not supported by the MobiLife architecture and the static nature of supported adaptation approaches therefore means that applications are unable to adapt if pre-defined context, preference, and behaviour information is incorrect or if a user's preferences change over time. In addition, the contextual relevance and utility of information is also defined at development time, and no techniques are provided to support changes to the contextual relevance of information in recommendation decisions that occur when a user's context or preferences changes.

2.3.2 Daidalos

Daidalos is an ongoing European research project that provides a platform architecture consisting of features for context-awareness and personalisation for telecommunication systems and services [186]. The architecture consists of six main software components. *Context Management, Pervasive Service Management, Event Management, Rule Management, Security and Privacy Management,* and *Personalisation Module.* The first five components are responsible for managing context information, discovering and composing services depending on the current context, firing events to signal relevant context changes, maintaining the set of rules that control the systems, and tracking user identities to ensure security and privacy protection respectively. In terms of personalisation support, the Personalisation Module component provides functionality for handling personalisation tasks such as learning user preferences, monitoring user preferences, and selecting and composing services. A number of subcomponents are responsible for these tasks: *Personalized Selection, Adaptation of Composition Process, Parameter Configuration, Preference Manager, User Monitor,* and *Inference Engine* [192]. The current implementation of Daidalos requires user preferences to be manually specified by users through the device interface and preferences are stored in a rule-based format. Future implementations plan to support other means of establishing user preferences such as the use of stereotypes and the learning of preferences from user behaviour [126]. Personalisation in Daidalos is adopted for a number of functions including personalised call redirection, personalised service selection, and personalised network or device selection.

2.3.2.1 Analysis

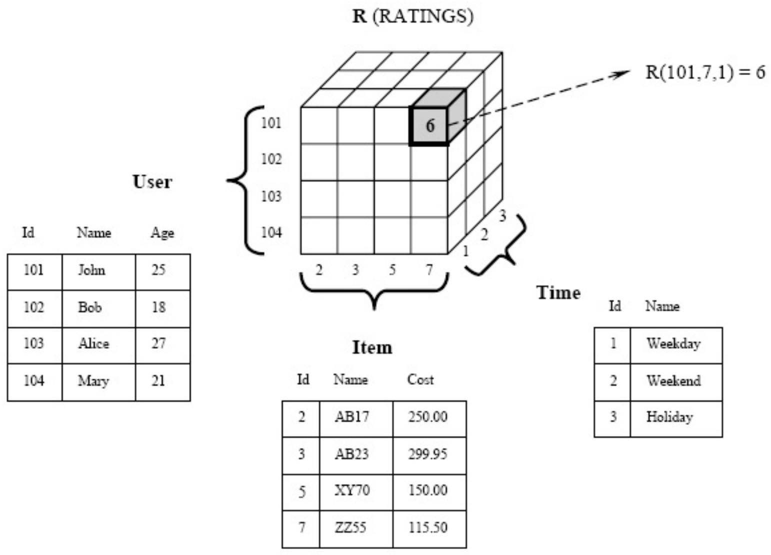
The platform architecture developed by the Daidalos project provides functionality to manage context information and to personalise telecommunication services such as service selection and call redirection. Like MobiLife, research work in Daidalos is ongoing and precise implementation details are currently unavailable. While Daidalos provides personalisation support in context-aware applications, the developer's current view is that *"the process of managing user preferences was regarded as less important than that of using them, and hence the simplest approach was adopted, namely manual entry of preferences"* [126]. The main limitation of personalisation in the current Daidalos implementation is the reliance on system users to provide details about their preferences. No techniques are currently provided for automatically eliciting user preferences (although this is planned for future implementations, the details of which are yet to be published) and context-dependent preferences are currently acquired from users who explicitly rank candidate services in different context. There is also no support for dynamically adjusting the relevance and utility of information in recommendation decisions for different context. Users are again required to explicitly specify this information. Preferences are represented as rules and the manual modification of these rules is required to accommodate changing user preferences. A rule-based approach to store preferences also suffers from maintenance and scalability problems [149].

2.3.3 Multidimensional Recommendation Model

Adomavicius et al. [4] present a *Multidimensional Recommendation Model* (MD model) that provides personalised recommendations while incorporating context information. The MD model extends the traditional two dimensional (2D) *Users x Items* paradigm to make recommendations based on multiple dimensions. Systems using the 2D paradigm support only two types of entities, users and items. The MD model differs as it is extended to support multiple other entities such as context information in the recommendations process. The model is based on an adjusted version of the multidimensional *Online Analytical Process* (OLAP) model. User preferences are expressed by providing ratings for each dimension in the MD model. For example, a three dimensional recommendation space User x Item x Time with a rating R(101, 7, 1) = 6 represents a rating of 6 for a given User 101 to Item 7 for weekday (represented by the value 1). Figure 2.3 illustrates the specification of this user preference in the MD model [4]. Dimension hierarchies (e.g., the Time dimension may be broken down into minutes, hours, days, months) and the aggregation of multiple dimensions (through aggregation functions such as summation) are also supported by the MD model. A rating estimation method based on the *reduction-based* approach is provided to estimate unknown ratings in a multidimensional recommendation space. Reduction enables ratings to be filtered based on a particular dimension (e.g., filtering movie recommendations by time) or expanded to include some superset of the context (e.g., predicting movie ratings for a Monday could use ratings given for weekday). A *Recommendation Query Language* (RQL) is supported as part of the MD model approach and enables users to create complex recommendations queries that consider multiple dimensions of context and user preference information.

2.3.3.1 Analysis

The MD model supports the representation of context-dependent preferences by representing context as dimensions in a multidimensional space, and preferences as item (behaviour) ratings. However,



User

Id	Name	Age
101	John	25
102	Bob	18
103	Alice	27
104	Mary	21

Item

Id	Name	Cost
2	AB17	250.00
3	AB23	299.95
5	XY70	150.00
7	ZZ55	115.50

Time

Id	Name
1	Weekday
2	Weekend
3	Holiday

Figure 2.3: Multidimensional Recommendation Model [4]

explicit user input is currently required to acquire user ratings for items under different dimensions, which is a complex and difficult task, particularly when there are a large number of different dimensions. Similarly, users are required to explicitly modify ratings in the appropriate dimension space to accommodate any changes in their preferences. Automatic preference elicitation techniques are not provided by the MD model, but could (according to our analysis) be integrated to determine user ratings. There are no provisions for supporting differences in dimension utility (i.e., in a particular multi-dimensional space, all the context types that make up that dimension have the same utility), and the utility of different dimensions are not adaptable to different context or user preferences (i.e., the utility of context remains the same regardless of the value of other context types). RQL enables context-dependent user preferences to be retrieved by supporting the construction of complex recommendation queries consisting of different relevant dimensions. However, the set of context to include in RQL queries requires explicit human input as there are no provisions for automatically determining or adjusting the set of dimensions that are relevant for recommendation decisions for a particular context.

2.3.4 The Personal Digital Secretary

The Personal Digital Secretary (PDS) is a context-aware personal assistant application designed by Byun and Cheverst at Lancaster University [39, 38]. The central aim of this work is to investigate the integration of user modelling, machine learning, and context-awareness as an approach for providing proactive personalised behaviour recommendations that will aid users with their daily activities. In particular, their approach is based on utilising context history together with user modelling and machines learning to infer patterns from user behaviour, which are then used to dynamically adapt to changes in the user's situation or their lifestyle. An important element of their work is the provision of explicit and understandable explanation enabling users to query the application about recommended behaviour.

The conceptual structure of the PDS is made up of five main modules and is illustrated in Figure 2.4: *context management agent, user modelling agent, recommender agent,* and *remembrance controller.*

- The Context Management Agent is responsible for the acquisition and merging of context gathered from sensors, the representation of context using XML technologies, and the storage of context in a persistent database.
- The Scheduler controller requests explicit user input regarding their future events (e.g., a users lecture timetable).
- The User Modelling Agent is responsible for predicting a user's future behaviour, using machine learning algorithms, from combined information about the user's past events, their explicit schedules, their current context, and explicit user feedback.
- The Reminder/Recommender Agent informs users of events based on their schedule and current context. It can also proactively recommend behaviour to users based on information learnt from the user model (e.g., if a user had left to play tennis at 5pm everyday for a week, the application can notify the user of this behaviour the following week).
- The Remembrance Controller stores a user's current context and enables the user to retrieve past events based on similar current context.

Three types of inferences designs are supported for predicting user behaviour and the corresponding behaviour recommendation: s*ingle learning algorithm, single learning algorithm with pre-defined rules,* and *multi-learning algorithms.* The single learning algorithm design uses a single machine learning algorithm to infer the correct behaviour for the current context from a user's context history. The Näive Bayes Classifier machine learning algorithm has been selected in their implementation. A single learning algorithm with pre-defined rules uses rules, pre-defined by the developer, to specify appropriate adaptation behaviour with machine learning used only to determine behaviour in more specific

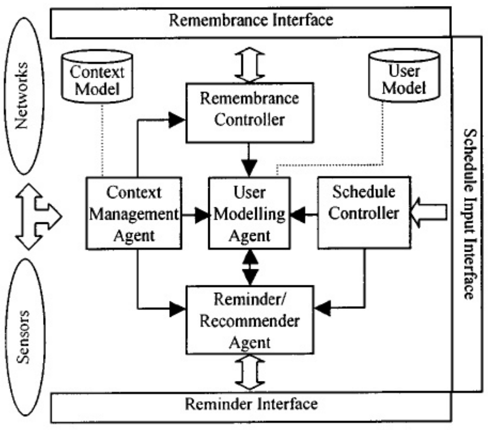


Figure 2.4: Conceptual Structure of the PDS [39]

exceptional cases. A multi-learning algorithm design supports the composition of several learning algorithms depending on the characteristics of the situation to be learnt. Example scenarios that have been implemented include reminder notices informing users of regular meetings and determining when a user will return to their office given the state of their office door and temperature of their coffee.

2.3.4.1 Analysis

The Personal Digital Secretary application and its design provides a valuable contribution by demonstrating the effectiveness of integrating techniques from user modelling, machine learning and context-awareness to support proactive, personalised behaviour. The inference of context-dependent user preferences is achieved by learning from past user actions. However, explicit expert knowledge and input is required to pre-define rules and design learning functions used to infer user behaviour. For Näive Bayes Classifiers, the probabilities for each possible context are calculated by experts in order to train the application. As the result, the current implementation is limited to supporting a small number of context types and values. The system does not support learning from user feedback (i.e., online learning), and therefore will result in inaccurate behaviour when rules and machine learning definitions are incorrect, or if a user's preferences change. Support for changing user preferences could be possible through addition of new user actions to the user model. However, domain expert input would be required to re-calculate probabilities of a particular context occurring to include new user actions. In addition, the definition of rules and machine learning algorithms by experts is at development time, whereby experts are required to explicitly determine the set of relevant attributes (context), the set of hypotheses (context and appropriate behaviour), and the explicit probabilities of each hypothesis.

The contextual relevance and utility of different environmental context types are therefore static, and do not dynamically adjust to different context or user preferences. Adjusting the contextual relevance to the current situation and user preference will enable applications to extract more precise patterns in user behaviour and consequently provide more accurate recommendations. For example, using the scenario for determining when a user will return to their office presented in this work [39], the *TempCup* context should be considered more contextually relevant than other environmental context (such as user location) in the current context as history indicates that the user usually returns to the office when the value of the *TempCup* context changes.

2.3.5 Pervasive Autonomic Context-aware Environments

As part of the Pervasive Autonomic Context-aware Environments (PACE) project, Henricksen and Indulska at the University of Queensland, developed a software engineering framework for context-aware pervasive computing [84, 86]. Their framework contains a number of layers that provide components to support the acquisition, management, storage and dissemination of context information:

- The Context Gathering layer acquires context from sensors, decouples sensors from processing components, and minimises problems associated with sensor or network disconnection.
- The Context Reception layer translates inputs from sensors into a *fact-based* representation.
- The Context Management layer maintains a set of context models, integrating information from heterogeneous context sources.
- The Query layer enables applications to query for context information from the Context Management layer, using context modelling abstractions supported by the framework.
- The Preference Management layer stores user preferences and evaluates preferences on behalf of applications.
- The Application layer provides toolkit support that enables applications to query the framework for context as well as providing methods for preference-based adaptation, known as *branching*.

A novel contribution of this work is the provision of a preference model that enables users to specify context-dependent preferences. Specifically, the model enables users to link preferences to context and to combine preferences into composite preferences and preference sets. Users express their preferences by assigning a score to possible (candidate) behaviour choices for a particular context. A central requirement of the model is usability, enabling users to easily link preferences to context and to

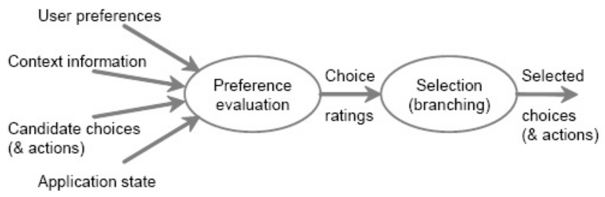


Figure 2.5: Context- and Preference-based Decision Process in PACE [85]

p1 = *when Urgent (priority)* $\wedge$ *SynchronousModel(channel)*
rate ♮
p2 = *when Urgent (priority)* $\wedge$ $\neg$*SynchronousModel(channel)*
rate 0.5

Figure 2.6: Example Communication Preferences in PACE [86]

combine preferences into sets or composite preferences. The model considers user preferences, context information, candidate choices and application state when making behaviour decisions. Figure 2.5 illustrates this decision process. Preferences are specified as context and score pairs, where a score is either a numeric value or a special score supported by the model (e.g., the ♮ symbol represents veto and the $\perp$ symbol represents indifference), and represents the score assigned to a candidate choice. Preferences are either *simple*, which express atomic user preferences, or *composite*, where different preferences are combined into a single score. Preferences can also be grouped into preference sets to reflect certain choice problems or the application domain. Figure 2.6 illustrates two example preferences for a context-aware communication tool [86]. A *branching* model examines the scores assigned to different candidate choices for a particular context and recommends the highest scoring behaviour to the user. The approach has been applied for specifying preferences in various application types such as a context-aware email client and a context-aware communication tool (which assists users in selecting an appropriate communication channel, such as telephone, email, text. etc. to interact with others).

2.3.5.1 Analysis

The PACE project provides an effective model for representing context-dependent preferences. However, the main limitation of this work is that explicit human input is required to specify, prioritise and combine user preferences. Manual modification of preferences by human input is also used as a means to accommodate any changing user preferences. Their evaluation results show that inaccurate recommendations do occur and are mainly due to the difficulties involved with users writing preference sets, especially when the set of context or preferences is large or complex. The adjustment of context relevance and utility for different contexts and preferences is also reliant on explicit user input and is explicitly expressed by users through ranking scores to various possible behaviours. Currently, no techniques are supported for the automatic preference elicitation or the dynamic adjustment of information relevance and utility, although the authors have highlighted these as possible avenues for future work [86].

2.3.6 SenSay

SenSay is a context-aware mobile phone developed by Krause, Smailagic, and Siewiorek at Carnegie Mellon University [110]. SenSay adapts its behaviour to the user's context and preferences using wearable sensors. This research focuses on how machine learning techniques can enable applications to identify the current user state (context) and learn context-dependent user preferences to automatically modify behaviour from experience. A two stage approach is described. At the first stage, *Context Identification*, the current user context is identified using Kohonen Self-Organising Maps. The second stage is concerned with personalisation and the learning of context dependent user preferences from context variables (which describe evidence about a user's context) and system variables (which describe evidence about a user's preferences). Bayesian networks are used at this second stage. Specifically, the *K2 algorithm*, a fast algorithm for structure learning, is adopted to construct their Bayesian network model. Three studies were performed to evaluate the approach. The first study illustrates that a *näive threshold-based* approach (which requires experts to explicitly define context and corresponding behaviour) does not generalise to support larger sets of context and users. The second study confirms that their approach to context identification is as successful with clustering results as manual annotation is. The final study concludes that their approach can reliably determine contexts and learn user preferences.

2.3.6.1 Analysis

The SenSay application contributes to existing work by investigating the use of wearable sensors to identify a user's context and context-dependent preferences. Their work also highlights the difficulties associated with generalising a *threshold-based* approach to adaptation, where context-aware behaviour is pre-defined. The proposed approach supports the dynamic and autonomous determination of context-dependant user preferences. However, a large amount of domain expert/developer knowledge is still required to explicitly construct the Bayesian network model necessary for learning a user's preferences. Difficulties associated with Bayesian network construction include identifying the correct dependencies between variables and estimating probabilities and joint distribution values of relevant context variables. The large numbers of context variables supported by SenSay means that typical techniques used to describe Bayesian networks, such as acyclic graphs, are unsuitable. Consequently, the K2 structure learning algorithm was adopted for learning the appropriate structure of the network. However, a strong limitation of this approach is that the ordering of variables needs to be known, which in reality is often not possible. Hence, assumptions regarding the ordering of variables are assumed by the authors. Learning user preferences is supported, but this learning requires the explicitly definition of a Bayesian network structure, which is inherently static. Consequently, changes to user preferences are not automatically captured and may lead to inaccurate behaviour. Adapting to changing user preferences could be possible but would require explicit developer modification of the Bayesian network structure and/or probability distributions. A better understanding of relations between context and preferences, such as the adjustment of context relevancy and utility for recommendations under different context, is also currently unsupported, but highlighted as an important avenue of future work [110].

2.3.7 A Context-aware Preference Database System

At the University of Ioannina, Stefanidis et al. [172] investigate the integration of context into database management systems. A central aim of their work is to provide a logical model for the representation of context-dependent user preferences. Users explicitly express their *basic* preferences, using a rating score between 0 and 1, based on single context parameters (such as location or weather). Basic preferences can be combined to compute *aggregate* preferences. For example, a user may give a restaurant a higher score when it is raining than when it is sunny. An independent score would also be given to the same restaurant by the user depending on their current location. The basic preferences for each are then combined to provide an aggregate score that depends on more than one context parameter. Data cubes are used to store basic user preferences (i.e., an association between a single context parameter and user preference) and *Online Analytical Process* (OLAP) techniques are used to process context-aware database queries. Each cube contains a dimension for the user, a dimension for an output (e.g., restaurant), and a dimension for a context parameter (as shown in Figure 2.7). Each cell of the cube contains a rating, representing the user's preference for that output and context parameter value. Hierarchies of preferences are supported by using cubes, enabling different levels of abstraction of context. A context tree structure, where the path in the context tree corresponds to an assignment of scores to context parameters, is used for evaluating aggregate preferences. To retrieve context-dependent preference information, users are required to explicitly construct queries using either SQL or OLAP operators. A similar approach is supported by the MD Model, however

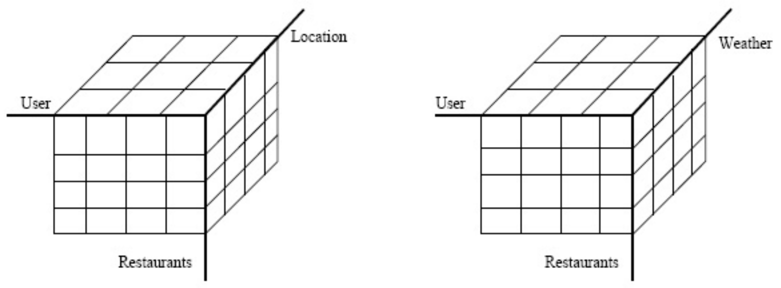


Figure 2.7: Context Parameter Data Cubes [172]

the MD Model supports multiple dimensions space for any number of context types, while the OLAP approach here requires a separate cube for each context type.

2.3.7.1 Analysis

The Context-aware Preference Database System provides a novel means to represent context-dependent preferences using data cubes. The main focus of their work is on efficiently combining context, preference and database operations. The approach is therefore limited in supporting personalisation tasks such as implicitly determining user preferences or adjusting the relative importance of information to reflect the preference of a user in different context. The main limitation is the need for users to explicitly specify their preferences by rating possible outputs (e.g., rating restaurants). Adjusting the relative importance of context parameters also requires explicit user input whereby users specify weights for individual context parameters when computing aggregate preferences. While not supported or discussed, techniques for automatically eliciting user preferences could be integrated with this work, although extensions would be required to integrate preference elicitation techniques with the data cube representation of context and preferences. Specifying context-dependant preferences as cubes also poses a scalability problem as a separate cube is required for each context parameter. It becomes more difficult for users to provide preference details as the number of context parameters (and therefore number of cubes) increases. There are also no provisions for automatically adapting the relevance and utility of context parameters. Users are currently required to explicitly construct queries to retrieve context-dependent preferences and the approach does not support the use of preferences as part of a recommendation process.

2.3.8 Context-aware Content Filtering and Presentation

Xu et al. [190] investigates the adaptation of content information based on context-dependent preferences. The central aim of this work is to select the right type of *content* and presentation *format* based on the user preferences for the current context. Similar to the approach developed by Stefanidis et al. for their Context-aware Preference Database, the approach by Xu et al. for discovering context-dependent preferences is based on Online Analytical Process (OLAP) techniques. The history of a user's application usage is stored as data (OLAP) cubes, which model different context parameters and the content type and format desired by the user for that corresponding set of context parameters. Co-occurrence analysis is used to compare the current context with context parameters stored in the cubes to determine the appropriate content and format for presentation. Content and format are ranked by counting the number of entries in the data cubes that match the current context.

2.3.8.1 Analysis

The main contribution of the work by Xu is the use of OLAP techniques to determine contextual preferences from usage logs. Their approach is also designed to support multiple arbitrary information types, which alleviates the need for developers to define a fixed set of context parameters and models. Changing user preferences are supported by executing OLAP queries on up-to-date usage data. While OLAP techniques eliminate the need for developers to pre-define the relationships between context parameters and user preferences, representing context-dependent preferences as data cubes presents a scalability problem. As separate cubes are required for each context parameter, the storage and maintenance of these cubes becomes correspondingly difficult as the number of context parameters increase. The ranking of alternative behaviours (in this case content and format) by counting the occurrences of a specific behaviour for a particular context does not accommodate differences in the relative importance of different context parameters. Methods for weighting according to how recent a user action is have been suggested, but selecting and weighting parameters according to different context and user preferences is also necessary.

2.3.9 Amigo

The EU Amigo project presents work on a user modelling service that enables personalised services to be provided in smart home environments [150, 182]. Amigo home is designed to recognise users, their preferences, their location and other context when executing services in the home such as automatically recording movies or supporting communication between home inhabitants. A primary element of the Amigo project is the *User Modelling and Profiling Service* (UMPS) architecture, which has been developed to support two main functions for capturing and representing context-dependent user preferences. Users can control the system via a GUI (static modeller component), allowing them to explicitly edit personal details and to specify and maintain context-dependent preferences, which are modelled in a tree-based ontology representation. Alternatively, a dynamic approach to user modelling, which eliminates the need for explicit user input is also supported. Dynamic user modelling stores interaction history as context-action tuples and context-dependent preferences are learnt (dynamic modeller component) using Case-based Reasoning techniques (CBR) and Support Vector Machines (SVM).

2.3.9.1 Analysis

The UMPS architecture of the Amigo project provides a set of services that aid developers to personalise context-aware systems for smart home environments. Both static and dynamic techniques are provided to support the acquiring and modelling of context-dependent user preferences. Both CBR and SVM provide a viable approach to automatically learning context-dependent user preferences in a smart home environment, but both have limitations when applied to large context-aware systems or when user preferences are likely to change. The accuracy of CBR is reliant on an accurately predefined similarity function, the construction of which is a non-trivial process, particularly when a large number of context types and values are supported. CBR also requires context relevance and utility to be statically defined as part of a similarity function. Consequently, the relevance and utility of information is not dynamically adjusted when determining recommendations in order to reflect the different context or changing user preference. Similarly, SVM (which is a method that classifies items between positive or negative classes) does not support the adjustment of relevance and utility to different context and user preferences. Large amounts of training data, memory and processing power are also required to train SVMs to a suitable level of accuracy [197]. SVMs will also require offline retraining by experts to accommodate changing user preferences.

2.3.10 Passepartout

Passepartout is a personalised digital TV guide developed by Berkovsky et al. that recommends programs to users that match their context-dependent preferences [28]. Context-dependent preferences in Passepartout are termed *experience*, which is described as user feedback (explicit or implicit) as a result of consuming a particular item in a particular context. Feedback is in the form of a rating. The work is focused on providing semantically-enriched descriptions of experiences for the construction

context.motivation=work
context.time=afternoon
item.meal.price=moderate
rating=0.8

Figure 2.8: An Example Experience in Passepartout [28]

of user models and a means to reason about user models for providing personalised behaviour. An extension of the GUMO user modelling ontology (RDF/OWL format) [83] is provided to facilitate various dimensions of context. Ratings represent a user's desire for a particular item in a specified context. Figure 2.8 provides an example of an experience *E*, representing a situation, an item, and a rating, where the user gives a rating of 0.8 to represent their preference for a moderately priced meal in the afternoon or a workday [28]. Rules and CBR are supported for reasoning and making recommendations.

2.3.10.1 Analysis

Passepartout highlights the need to include context in the personalisation process and their concept of an *experience* provides a viable means for describing context-dependant user preferences. However, the automatic determination of user experiences is not supported and no provisions are provided for automatically determining or prioritising context features most relevant for recommendations in a particular context. The use of RDF/OWL suggests that experiences, including the relevance and utility of context features, are required to be explicitly identified and defined by users or experts. The approach is flexible and different reasoning mechanisms can be integrated, however current reasoning approaches based on rules and CBR are limited for personalising context-aware applications due to their reliance on explicit human input.

2.3.11 Context-aware Case-based Reasoning Applications

This section evaluates projects that extend the case-based reasoning (CBR) concept for making personalised recommendations to users in context-aware applications. These projects are evaluated and grouped together as they support the implicit determination of context-dependent user preferences by determining the preferred actions of users using past cases that have a similar context to the current recommendation problem.

2.3.11.1 AmbieSense

AmbieSense is a project conducted as part of the Information Society Technologies (IST) Programme, the goal of which is to develop a set of tools to facilitate context-aware computing [101, 102]. AmbieSense makes use of context information provided by its system architecture and CBR for making behaviour recommendations to users. The AmbieSense system architecture is divided into three major parts: *the Context Middleware, the multi-agent systems,* and *reasoning.* The Context Middleware provides a context management infrastructure, which collects and maintains context. Context is modelled as a semantic network based on five main aspects: *personal context, task context, social context, spatio temporal context,* and *environmental context.* The multi-agent system, based on the Jade platform, maintains a set of autonomous application agents capable of executing actions associated with the current situation (e.g., a context agent receives new current context notifications from the Context Middleware and sends them to the relevant receivers). An extended version of the Creek CBR system, is used at the reasoning stage. The CBR cycle begins when new context arrives from the Context Middleware. The system classifies the current situation and the appropriate sequence of actions from past cases that match the current context, are executed. Cases are stored as a triplet consisting of the context describing the situation, the problem (goal) associated with the situation, and the solution employed by the system's application agents. Applications of this approach have been demonstrated as part of a tourist guide and a healthcare patient diagnosis and treatment scenario.

2.3.11.2 LISTEN

The LISTEN project conducted at the Fraunhofer Institute provides a system that personalises audio pieces to the time, position and head orientation of its users [200]. A LISTEN prototype has been made available to visitors of the Kunstmuseum in Bonn, enabling visitors to experience personalised audio information about exhibits through their headphones as they move through the museum. The design and implementation of LISTEN is based on general layered framework for context-aware systems (sensor layer, semantic layer, control layer, and actuator layer) which provides functionality for the acquisition and management of context and for personalising behaviour. CBR is used to personalise adaptation behaviour (i.e., which audio clip to play) by comparing the current context with past similar cases. Context contains four dimensions: identity, location, time, and environment/activity; and integrated with CBR cases. The similarity assessment used places a high weight on the location, the time, and the interests of the user. Cases are represented by mapping problem-solution pairs of a CBR case description to context-recommendation pairs.

2.3.11.3 Personal Travel Assistant

The Personal Travel Assistant (PTA) is a recommender system that takes requests from a user and recommends a set of flights that match the user's request. In PTA, user preferences are implicitly inferred from their past behaviour (i.e., previously purchased flights). The context-dependent nature of preferences is supported by considering features of a flight request (e.g., is a request for a long haul or a short haul flight). Two recommendation strategies based on CBR are provided: *offer-based* and *session-based*. Offer-based recommendations are made on the basis that a user prefers similar offers to those they previously purchased (i.e., the context of the request is not considered). Session-based recommendations, in contrast, are made on the basis that a user prefers similar offers to those they selected in *similar* previous interactions (i.e., past purchases that are similar in context). A technique to adjust the relative utility of features in recommendation decisions is also provided. Feature utilities are adjusted according to a user's flight purchases such that selected cases are ordered highest. This technique adjusts feature weights according to individual users, but there is currently no support for adjusting weights for different context. Evaluation results show that an offer-based strategy averages 61% accuracy and the session-based recommendation with feature weight learning averages 67% accuracy.

2.3.11.4 Ticketyboo

The Ticketyboo system is a case-based music concert recommender system. When making recommendations, Ticketyboo takes into account the user's music preferences and other context such as concert dates and user's location context. The system acquires its context information from the Construct context-aware infrastructure [173]. Ticketyboo operates in two stages; the first stage selects information about upcoming concerts (artist, location, date, time, ticket price), while the second stage filters the number of concerts by the user's music preferences and their location at the time of the concert. Music preferences are automatically inferred from the user's media play while location information is acquired from the user's calendar. Concerts remaining after this filtering stage are recommended to the user.

2.3.11.5 Analysis

In each of the projects described in this section, the CBR method is used for making personalised behaviour recommendations to its users. Context-awareness is supported by extending the traditional CBR case to include context information (such as location, time, and weather). Similarity function definitions incorporate relevant context types, enabling applications to automatically infer the preferred behaviour of users in past similar context (i.e., context-dependent user preferences). CBR therefore enables the autonomous determination of context-dependent preferences (from historical) behaviour. However, the effectiveness of CBR to accurately personalise behaviour recommendations requires extensive domain knowledge and relies on experts to define an accurate similarity function. Experts are required to identify and represent the set of information that is relevant, its relative importance (i.e., utility), the relationships between different information types, and the threshold values that measure similarity as part of the construction of a similarity function. However, this is a non-trivial task that is time-consuming and error-prone, particularly when large amounts of context and values are to be supported. The need to define a CBR function at development time also means that the approach is inherently static with context relevance and utility statically defined as part of a similarity function. While CBR can adapt to different context by selecting contextually similar past cases, the specification of what context is relevant and how relevant it is, is static (i.e., the similarity function remains the same regardless of the current context). Consequently, the relevance and utility of information is not dynamically adjusted in recommendation decisions in order to reflect different context or changing user preference. However, in reality, the contextual relevance and utility of information varies depending on the context, meaning that in different situations, a user's definition of what more similar is likely to change (i.e., similarity functions should be context-dependent). For example, when recommending flights, context types such as flight duration, departure time, destination, price, and carrier are considered, but depending on the context values, certain context types may be considered more relevant than others by the user. A user, for example, may consider the departure time of the flight as irrelevant when the flight duration is over 8 hours, but for short haul flights the user historically prefers to travel during the morning. Therefore, to ensure accurate recommendations, a means to adjust the relevancy and utility (attribute weights) in similarity functions depending on the context are also necessary. That is, CBR can automatically accommodate gradual changes in user preferences through the addition of new cases to the case base. However, more significant preferences changes, which affect what users consider similar context, would require experts to explicitly modify the similarity function in order to maintain the utility of recommendations. Zimmermann refers to this as *"realizing different views on cases"* [199].

2.3.12 The 1:1 Pro System

The 1:1 Pro System builds personal profiles based on a customer's previous transactions in order to make personalised recommendations about future purchases [5]. Data mining rule discovery algorithms such as Apriori for association rules and CART (Classification and Regression Trees) for classification

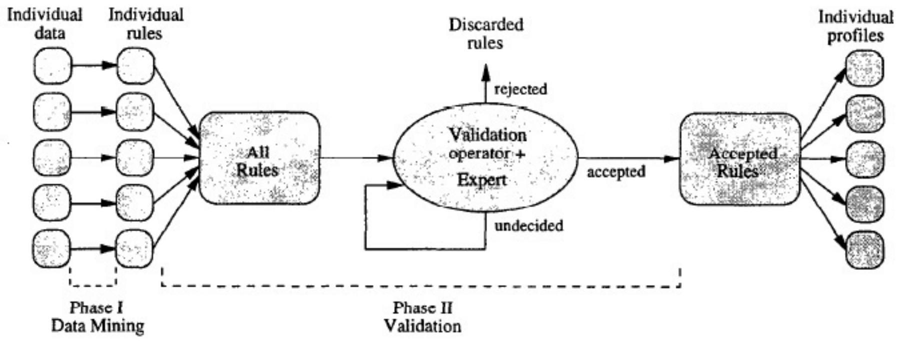


Figure 2.9: Rule Discovery Profile Building in 1:1 Pro System [5]

rules are used to discover and model user behaviour. An example of customer behaviour as a rule is *"when purchasing cereal, John usually buys milk"*. Experts post analyse rules to determine those that are trivial, conflicting, or irrelevant. Several validation operations are supported to aid human experts to validate large numbers of rules. A similarity based rule grouping operator puts similar rules into groups based on expert-defined similarity criteria, a template-based rule filtering operator filters rules that match expert specified rule templates, and a redundant rule elimination operator eliminates rules that can be defined from other rules. Figure 2.9 illustrates the process of building user profiles from rule discovery in the 1:1 Pro System.

2.3.12.1 Analysis

This work demonstrates that mining is a viable approach for discovering and modelling user behaviour from historical data. The rule based format for modelling behaviour is also an intuitive and descriptive way to represent behaviour. Support for various environmental context types (e.g., location, weather) is not explicitly described in this work. However, context could be supported by adding context values to a user's transactional history. The main limitation of this approach for context-awareness is the need for human experts to validate rules at development time. Techniques have been developed to aid human experts in selecting relevant rules, however support for autonomous methods to validate and prioritise rules (to represent the utility/preference of user behaviour in a particular context) and to select those rules most appropriate for recommendation is not provided. The validation of rules at development time also limits the ability of applications to evolve as they are unable to autonomously correct invalid rules or adapt to changing user preferences. The application of mined rules to a recommendation process is also not supported.

2.3.13 Summary and Analysis

A survey and evaluation of the state of the art projects shows that there are a range of approaches that aid developers in personalising context-aware applications. Table 2.3 summarises the extent to which these approaches support the requirements described in Section 1.2. While each of these projects provide some level support for the outlined requirements, the techniques they provide are limited and no approach fully supports all requirements.

	Context-dependent User Preferences	*- Adjusting Context Relevance and Utility*	Complex Context-aware Environments	Dynamic Execution	Changing User Preferences	Autonomous Execution
Recommender Systems	○	-	-	-	○	-
Context-aware applications and frameworks	○	○	○	-	○	-
MobiLife	○	○	○	-	○	-
Daidalos	○	○	○	-	○	-
The MD Model	$\surd$	-	○	-	○	-
The Personal Digital Secretary	$\surd$	○	○	-	○	-
PACE	$\surd$	○	○	-	○	-
SenSay	$\surd$	○	○	○	○	-
Context-aware Preference Database	$\surd$	$\surd$	○	-	○	-
Context-aware Content Filtering and Presentation	$\surd$	○	○	-	○	-
Amigo	$\surd$	○	○	○	○	○
Passepartout	$\surd$	○	○	-	○	-
Context-aware Case-based Reasoning Applications	$\surd$	○	○	○	○	○
The 1:1Pro System	○	○	○	○	○	-

$\surd$ = support ○ = limited support − = no support

Table 2.3: Summary of Context-aware Personalisation Support

Recommender systems provide methods for adapting to user preferences. Progress has been made in recommender system in recent years to incorporate user and item profiles into recommendations. However, preference data used in these systems are quite basic (based on features such as keywords) and in particular, these systems are limited by the types of context they consider when making behaviour recommendations. In many cases such as Amazon, PTV, and LIBRA no context information other than user preferences are considered. Many state of the art recommender systems are also limited by the need for explicit user input when acquiring preference data. More advanced, implicit feedback techniques have been used but are mainly restricted to the context of web usage analysis i.e. to discover patterns in a user's page viewing sequence and use this information to make better web page recommendations.

State of the art context-aware applications and frameworks support personalisation by adapting to user preferences in the same manner as other context types. Relevant context relationships and the definitions regarding utility of information that influence recommendation decisions are statically specified by experts at development time. No techniques are provided to aid experts with identifying or representing context and preference relationships and utility. Accommodating changing user preferences also requires explicit expert or user input. For example (as described in Section 2.2), Hermes enables users to modify their preferences by adjusting the relevance and utility of context in recommendation decisions. Other frameworks require developers to manually modify rule definitions or the structure of behaviour models (e.g., Bayesian Networks) in order to facilitate changing user preferences.

Other projects have specifically focused on the challenge of personalising context-aware applications. Architectures such as MobiLife and Daidalos provide components that facilitate user preferences as part of an overall context-aware architecture. Other projects such as PACE, and the OLAP based techniques used by the MD Model, by Stefanidis, and by Xu aid users and developers by providing novel representation models that ease the task of specifying dependencies between context and user preferences. However, these projects remain limited by their need for users to input preference details and support for techniques that enable the automatic inference of user preferences at run time are not provided. Methods for dynamically adjusting the relative relevance and utility of information in recommendation decisions depending on the current context are also unsupported. Approaches for implicitly determining user preferences at run time from past behaviour such as CBR are available and go some way to supporting personalisation and context-dependent user preferences. However, these techniques are limited by their static developer defined similarity functions, the definition of which does not automatically adapt to different context or changing user preferences. The rule discovery approach proposed in the 1:1 Pro System supports the automatic identification of context-dependent user preferences without the need for experts to explicitly identify context, preference, and behaviour relationships. However, explicit expert input is required to validate and identify those preferences that are relevant and interesting. The lack of automated techniques for identifying and specifying context and preference relationships (which currently require human input) also limits the amount of context and recommendation types existing state of the art projects support. Rule-based, model-based, and CBR techniques all require significant expert knowledge and input to construct, which becomes corresponding time-consuming, complex and error-prone with the number of context and preference types/values to be supported. OLAP based cubes also suffer from storage and processing problems.

2.4 Chapter Summary

This chapter has presented a number of state of the art approaches to support the personalisation of context-aware applications. In particular, the chapter has examined state of the art projects in the areas of recommender systems, context-aware applications and frameworks, and projects that focus specifically on personalisation in context-aware systems. Each of the surveyed projects have been evaluated against the requirements outlined in Section 1.2. While a number of state of the art projects exists that offer support for the development of personalised context-aware applications, this thesis concludes that no single approach offers full support for the complete set of requirements. These projects therefore cannot, without significant extension, be used to support accurate and effective personalisation in context-aware applications. Specifically, support for personalisation functionality such as techniques for identifying context-dependent user preferences, adjusting relevance and utility of information for recommendation decisions, and adapting to changing user preferences, which will aid developers in building personalised context-aware applications, remains limited. The next chapter describes the design of a novel approach to personalisation for context-aware applications. It discusses techniques and algorithms that provide implicit determination of context-dependent preferences, support the automatic identification of relevant context information and the adjustment of information utility for the current user preference and context, and facilitate the generation and ranking of candidate behaviours that are personalised to the current context-dependent preferences of users.

Chapter 3

Design

The analysis of the state of the art approaches to personalisation in context-aware applications in the previous chapter shows that they are limited in their support for the requirements outlined in Section 1.2. An approach to context-aware personalisation is required to support the determination of user preferences for different context settings, including the determination of relevant information and information utility for different recommendation problems. State of the art approaches are limited by the need for explicit human input (from users or developers) to identify, prioritise, and maintain relevant relationships between context and user preferences to ensure correct recommendation behaviour - a human task which is complex, time-consuming, and error-prone, especially when applications are to support a large number of context types or values. An approach to context-aware personalisation should also support applications to autonomously adapt to any changing user preferences or unconsidered relationships between context and preferred user behaviour, which is necessary in order to maintain the accuracy of recommendation decisions over time. The state of the art approaches use algorithms or functions that are inherently static and do not automatically adapt to newly acquired relationship information. Users or developers are required to manually modify context and behaviour relationship definitions as well as definitions of information relevance and utility in order to accommodate changing user preferences or previously unconsidered information relationships.

This chapter describes the design of a novel approach to support personalisation in context-aware applications. This approach provides techniques and algorithms that address the requirements outlined in Section 1.2. At the core of the approach is a set of techniques and algorithms that: implicitly determines the preferred actions of users for different context; identifies the set of relevant context information for different recommendation problems; adjusts the utility of information for different problems and context; and generates a set of candidate behaviours and ranks them according to the inferred context-dependent preferences of the user. These techniques combine to form a multi-stage recommendation process that addresses the limitations of existing personalisation approaches in context-awareness by eliminating the need for experts to explicitly define relationships between context and preferred user behaviour and details about information relevance and utility. The need for users or developers to explicitly modify defined information to cater for new relationships between context and user preferences is also removed.

The remainder of the chapter provides a detailed description of the personalisation approach proposed in this thesis. A description of the *Hermes* project, the umbrella project under which the work presented in this thesis was conducted, is also presented. The majority of the chapter is dedicated to describing the personalisation approach and the individual techniques and algorithms that have been designed for specific personalisation functionality. The chapter concludes with a summary of how the approach is designed to address the requirements outlined in Section 1.2.

3.1 Overview

This section describes the design methodology used for the development of the personalisation approach proposed in this thesis. It also provides an overview of the *Hermes* project including a description of the set of applications and the application framework developed under Hermes.

3.1.1 Design Methodology

The personalisation approach described in this thesis has been developed as part of the *Hermes* project with the aim of providing generic support for user preferences for mobile, context-aware applications. Like other research work conducted as part of Hermes, the proposed personalisation approach is developed using the *Three Examples* methodology proposed by Roberts and Johnson [158]. Three Examples is built on the rationale that people develop abstractions about functionality by generalising from concrete examples. The general rule proposed by this approach is to develop at least *three* prototype applications whereby each subsequent application is extended and builds on the lessons learnt from the previous one. Although specifically focused on the development of application frameworks, the Three Examples methodology provides an effective and iterative approach that enables the identification and design of techniques, algorithms, and models required for the effective development of particular processes and application types. Consequently, Three Examples provides an effective means for developing novel personalisation algorithms and techniques by supporting the investigation of a variety of personalisation methods and their applicability to context-aware applications. It also supports the assessment of a collection of personalisation theories and concepts, a comparative analysis of different personalisation approaches, and an evaluation of individual personalisation methods.

3.1.2 Hermes

The Hermes project investigates models, techniques, and algorithms for supporting the development of mobile, context-aware applications[1]. As part of the project an application framework has been developed, which provides developer support for common context-aware functionality, organised in a layered structure. The Hermes architecture distinguishes itself from other context-management frameworks through its support for *trails-based* applications [50]. A trail is a set of activities or tasks, together with associated information (e.g., spatial and temporal information or user preferences) and a dynamically reconfigurable recommended visiting order. Activities in a trail are contextually scheduled and adapt to changes in personal and environmental context [65]. An example is an application to optimally schedule ward rounds for physicians [66].

3.1.2.1 Applications

The initial requirement for personalised recommendations in context-aware applications was identified from the experience of investigating and designing several context-aware applications. From the development of these applications, user preferences and the effect of context on these preferences was investigated and a suitable approach for personalising context-aware applications was developed. The following personalised mobile, context-aware applications have been investigated in the Hermes project:

- A Campus-based Student Support System - this application supports students in completing a set of compulsory and optional activities on their first day on a new college campus. The application makes use of environmental context and user preferences to order activities and generate a suitable route between activities. Environmental context such as the user's location, the location of activities, activity obligation (i.e., activities are either compulsory or optional), and activity opening hours are considered. A student's preferences regarding activity obligation and routes types (e.g., shortest, most scenic) are also considered.
- A Riddle Hunt Game - this application supports users with navigating between different riddle locations. Activity order and routes between riddles are generated based on a player's preferences

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

for factors such as riddle type and difficulty level.

- A Scheduler Application for Healthcare Professionals - this application facilitates healthcare professionals with task scheduling such as ordering the set of tasks that require completion on a doctor's ward round. Tasks are ordered according to a stated policy such as the most urgent or shortest path between patients. A doctor's preferences such as being alerted when behind schedule, preference for visiting new patients first, and minimal walking are also considered.

- A Restaurant Recommender System - this application evaluates and presents restaurant recommendations to users using information about their dining preferences and their current context. The application supports dining preferences by making recommendations about cuisine type, food quality and service rating, while context types that are supported are the user's location, the type of meal (i.e., breakfast, lunch or dinner), and the type of day it is (i.e., weekday or weekend).

- A Travel Recommender System - this application facilitates users with travelling between two locations by recommending a suitable travel route and transport option. Unlike, the Travelling Salesman Problem [13], which evaluates routes using a single static cost metric, this application evaluates the users preferred travel route and transport using their past travel habits under different context settings. Different travel route optimisations are supported: distance, time, number of turns, and number of junctions; while transports options for travelling by bus, by car, or by walking are supported. Context types supported include, start and end locations, the type of day, time of day, and weather. This application also supports the evaluation of multiple recommendation types (i.e., provides *both* route and transport recommendations).

The design and analysis of each of these applications has shown the presence of dependency relationships between context and user preferences and that different users consider different sets of information with varying degrees of relative importance when making behaviour decisions[2] [118]. Following the Three Examples methodology, the development of each application is built on the findings of the previous ones and provides improved support for personalisation behaviour. To this end, the final two applications provide the most sophisticated level of personalisation support including the set of algorithms and techniques developed for automatically determining user preferences and ranking candidate behaviours. The final application also supports functionality for dynamically determining information relevance and information utility. These two applications were used as part of the user

[2]These findings are validated by user studies conducted as part of the evaluation of this personalisation approach described in Chapter 5.

study evaluations conducted to assess the accuracy of the personalisation approach described in this thesis. These results are outlined in the Evaluation chapter (Chapter 5).

3.1.2.2 Application Framework

As part of the Hermes project, a layered application framework, which provides developer support for common context-aware functionality, has been developed. Functionality provided by this framework is utilised by techniques and algorithms for personalisation described in this thesis. Specifically, integration with Hermes makes available a set of services for: ad hoc discovery and exchange of information with mobile devices and fixed infrastructures; filtering and transforming acquired information from heterogeneous information sources; and on-demand access to context information. Figure 3.1 illustrates the Hermes application framework.

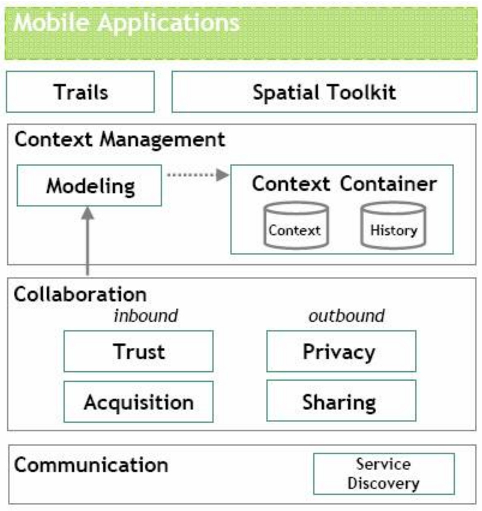


Figure 3.1: The Hermes Application Framework

Communication, Collaboration and Context Management layer functionality provides the set of services responsible for gathering, modelling, and providing access to context information. At the lowest level of the architecture is the *Communications* and *Service Discovery* layer, which is responsible for discovering and managing communication with sensors and fixed infrastructures over a variety of networks. Above this layer is the *Collaboration* layer. *Collaboration* divides its functionality into inbound (*Acquisition* and *Trust*) and outbound (*Sharing* and *Privacy*) operations. Acquisition and Sharing functionality act as a gateway for the transfer of information and are responsible for controlling the exchange of context information with other devices. Specifically, Acquisition is responsible for proactively acquiring specific types of context on behalf of the application while Sharing is responsible for responding to incoming requests for context. Trust and Privacy control the set of information that is acquired or shared. Trust interacts with Acquisition and augments incoming context with trust-related meta-data. Similarly, Privacy works with Sharing and protects personal data by controlling which types and values of context are exchanged with which peers. The *Context Management* layer is responsible for converting, reasoning, modelling, and storing context information. Context Management contains *Modelling* and *Context Container*. Modelling is responsible for fusing different types of context and its conversion into the framework's internal context representation while Context Container stores current context and persists information as history when new context arrives. The Context Container also provides functions that enable applications to retrieve information using either push or pull techniques [49]. Unique to the Hermes framework are layers for Trails and Spatial Toolkit functionality. Trails functionality is responsible for the generation and management of trails and Spatial Toolkit provides functionality for the representation and interpretation of spatial (geographical) properties of the user's environment. Finally, applications operate above this layer and use functionality provided by the lower layers to adapt or make recommendations about behaviour most appropriate for the current context and user preference.

The personalisation approach described in this thesis is designed to integrate with the Hermes framework. Specifically, personalisation techniques directly collaborate with Acquisition in order to specify the context types that are to be acquired. It also interacts with the Context Management layer, which provides personalised applications with access to both current and historic context information.

The initial version of the Hermes framework supports user preferences as part of Trails functionality. However, this initial version requires users to explicitly specify preference information including relevant decision making information and their associated utilities. The personalisation approach proposed in this thesis is designed to extend personalisation functionality in Hermes and provide more generic support for user preferences. In particular, it provides functionality that enables user preferences, including information relevance and utility, to be dynamically determined from past user behaviour for different recommendation problems and context settings.

3.2 Multi-Stage Recommendation Process

This section describes the personalisation approach proposed in this thesis, including a description of the set of techniques and algorithms provided to support context-aware applications in making accurate personalised recommendations to users.

3.2.1 Overview

The research question addressed by this thesis is: what techniques and algorithms are necessary to support the dynamic and implicit determination of user preferences from user behaviour, including relevant information and correct information utility, to facilitate context-aware applications in making accurate personalised recommendations to users? To address the research question, this thesis describes an investigation into a novel approach to personalisation. Following a user-centered definition of context discussed in Section 1.1, a user, depending on their preferences, is likely to consider different types and combinations of context information with different levels of relative importance when deciding on their preferred behaviour. This approach does not require information about relationships between context, user preferences, behaviour, or information relevance and utility to be explicitly pre-defined and maintained. Instead, the approach determines and ranks candidate behaviour choices by dynamically inferring knowledge about user preferences, information relevance, and information utility for a given problem and context. Relevant relationships between context and user behaviour are determined at run-time when required e.g., when a user's activity changes. Relevant decision making information and utility are also determined when the application is executed using knowledge inferred from user behaviour patterns and implicit user feedback. The approach is designed to support applications to dynamically adapt to the varying effect of context on the preferred behaviour of users. Limitations of existing work are addressed by providing techniques and algorithms that facilitate user autonomy, unconsidered context relationships, dynamic determination of information relevance and utility, and changing user preferences.

Figure 3.2 illustrates the design of the personalisation approach described in this thesis[3]. The approach is organised as a set of processes that execute in sequence to support personalised recommendations in context-aware applications. Existing techniques from various other fields, along with newly developed techniques and algorithms, are used, adapted and combined in a novel manner to provide a set of techniques and algorithms that: automatically determines user preferences from past behaviour; dynamically filters relevant recommendation information; adjusts the relative utility of information to reflect the current context and up-to-date preferences of the user; and dynamically generates and ranks competing recommendations. Context information and information about recommended behaviour selected by the user is acquired from sensors and from user-device interactions using Hermes framework functionality. User recommendation selections and the context at the time of these actions are stored in a user model along with other past user interactions. At the core of

[3] * indicates functionality that was added and # indicates functionality that was improved for the extended strategy.

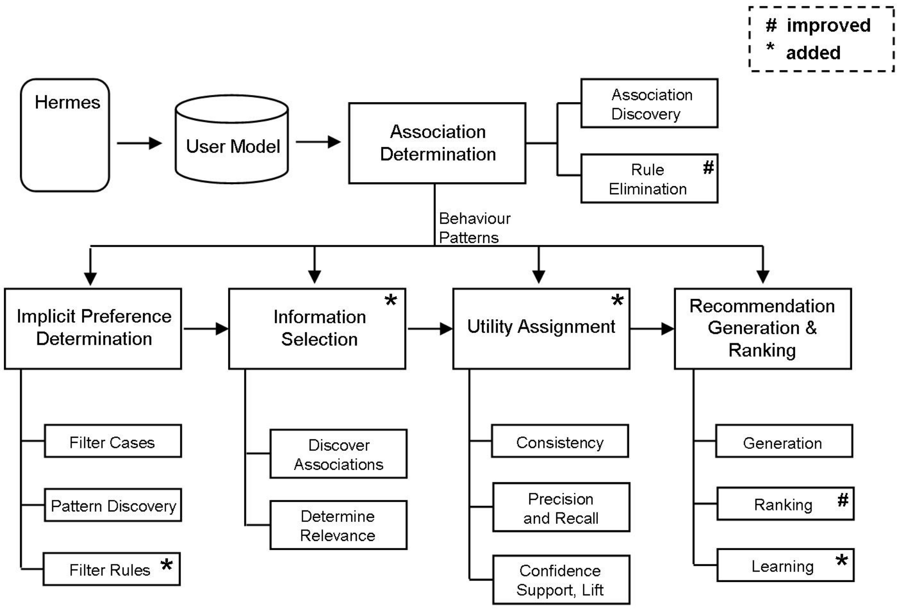


Figure 3.2: Personalisation Support for Context-aware Applications

the approach is the identification and examination of relevant relationships between contexts and user behaviour choices. Relevant relationships discovered on filtered user model cases represent the preferred behaviour of a user for a particular context. Relevant information, information utility, and the ranking of candidate choices, are determined at subsequent stages using techniques and algorithms that utilise knowledge inferred from discovered user behaviour patterns. The set of techniques and algorithms designed for each stage of the recommendation process will be discussed in greater detail in later sections of this chapter.

While relationships between context and user choices and relevant decision making information and utility are dynamically determined, the approach does require the development time specification of the types of context and recommendation information supported by the application. Specifically, the following set of information is to be defined:

- the set of context types and their possible values supported by the application e.g., *day* context has values for *weekdays* and *weekend*; *weather* context has values such as *rain*, *sun*, *cloudy.*
- the set of recommendation problems/output types, features of each output type, and their set of possible values supported by the application e.g., a *restaurant* recommendation output type has features such as *cuisine*, *food quality*, *service/atmosphere rating*, and *price* while a *route*

recommendation type has features such as *distance*, *travel time*, and *number of junctions*.

Both context and output information are stored as *type-value tuples* (e.g., day=Monday, cuisine= Italian). A boolean approach is currently used to compare information types but more fuzzy based comparisons such as similarity measures used in case-based reasoning can also be applied. Context types with sequential values are supported by aggregating a series of values into a higher level value e.g., time between 12.00pm and 5.00pm is grouped together in an *afternoon* value.

3.2.1.1 Recommendation Strategies

Following the three examples design methodology (Section 3.1.1), the personalisation approach described in this thesis was developed in an interative manner with improvements made to techniques and algorithms with each iteration. As part of this process, two main recommendation strategies were constructed. The first (i.e., *original*) strategy provides personalised recommendations using context-dependent user preferences inferred at the Implicit Preference Determination stage (Section 3.2.5). The determination and ranking of candidate behaviour choices to recommend is executed at the Recommendation Generating and Ranking stage (Section 3.2.8). With this strategy, all context supported by an application is considered relevant and of equal utility in recommendations decisions.

While this first strategy produces accurate personalised recommendations (illustrated in Section 5.2 in Chapter 5), its limitation is its assumption that users use the same set of context types and place the same relative importance to different context types when making behaviour decisions. Consequently, Information Selection (Section 3.2.6) and Utility Assignment (Section 3.2.7) techniques and algorithms were developed to provide functionality that adjusts the relevance and utility of information for different recommendation problems. The second (i.e., *extended*) recommendation strategy therefore builds on the first by providing techniques to dynamically determine information relevance (Information Selection) and utility (Utility Assignment). Figure 3.2 illustrates the set of techniques and algorithms that are supported in each recommendation strategy.

3.2.2 Context Management

The personalisation approach proposed in this thesis assumes the availability of context data as input for personalisation tasks. Functionality that supports the timely access to context information is provided by the Hermes framework. The framework supports the acquisition of context from sensors and fixed infrastructures and also processes and transforms context data from its raw form into an internal representation that can be queried by personalised applications. These tasks, which make context available to applications, are the responsibility of the Hermes Communications, Collaboration, and Context Management layers. Although not a contribution of this thesis, work on context acquisition and modelling provided by the Hermes framework has been conducted in conjunction with work on personalisation functionality. As a note, the proposed personalisation approach is not limited to the acquisition and modelling functionality supported by the Hermes framework. Approaches provided by other frameworks and architectures such as the Context Toolkit [62] or the ACoMS context management system [91] can be adopted as an approach to obtaining context data. The following sections briefly describe the context acquisition and modelling functionality provided by the Hermes framework and how they provide context information to personalisation functions.

3.2.2.1 Context Acquisition

Hermes provides services that enable applications to discover and communicate with remote devices in a generic manner by hiding the complexities associated with interacting with heterogeneous devices with different communication protocols and interfaces. Hermes provides functionality for acquiring both *local* and *remote* context. Local context is collected from sensors that are directly connected to the user's personal mobile device. For example, the user's location is acquired from a connected or integrated GPS device and the current time is acquired from the system clock on the mobile device. Local context can also be inputted by the user directly. For example, the user can input details about their current task or select a given behaviour recommended to them. In contrast, remote context is gathered from sensors or infrastructures not directly connected with the user's device, but require collaboration with remote context sources. For example, the location of a friend is acquired by communicating with that person's mobile device. Other information services may also act as remote sensors, such as a weather service, which would provide information about the current (and future) weather context. Peer-to-peer ad-hoc service discovery provided by the Hermes *Service Discovery* and *Communication* function are used to locate and communicate with remote context sources. Remote devices within proximity are discovered via an ad hoc service discovery protocol. Devices are subsequently directly connected to share context.

The Service Discovery and Communication functions support the receiving of advertisement and context information messages and the sending of context request messages. An application is aware of the context it requires in order to complete its operating tasks, which is usually a subset of the context types supported by the application, and when necessary, it explores the environment for context advertisements to determine the context that is available. Devices advertise context by sending a service description message containing the context types the application is to share to remote devices that are in proximity. Once a device discovers context it wishes to receive, a context type request message is sent to the remote device advertising that context type. The remote device handles this request message by responding with the requested context value. Figure 3.3 illustrates the service discovery and communication of context between devices as supported by the Hermes application framework.

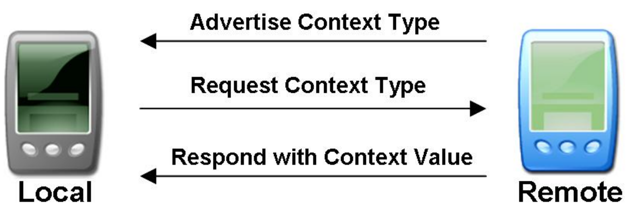


Figure 3.3: Context Acquisition in Hermes

The following incoming and outgoing messages are supported by the Hermes framework: context data messages from different context sources; context request messages from remote devices; service description (i.e., context advertisement) messages to remote devices, application messages; broadcast messages advertising IP address and port information used to communicate with the user's device.

Both *pull* and *push* communication approaches [49, 154] are supported by the Hermes framework. A pull approach enables remote devices to request for information on demand, while a push approach enables devices to register with remote devices for context updates. Each time relevant context types are updated locally, remote devices which have registered an interest in these context types will be sent the updated context value (including the current time stamp). This therefore prevents devices from having to repeatedly request certain context types that are of interest. A queue based mechanism enables applications to send and receive multiple messages at once without hindering an application's processing response.

3.2.2.2 Context Modelling

The Context Management layer of Hermes provides functionality for modelling, storing, and accessing context information through its Modelling and Context Container functionality. Modelling functions reason and combine different context types and converts context into an internal representation. Context acquired from both local and remote sources are transformed into a single representation structure. Hermes adopts a hierarchical, object-oriented (OO) model to represent context information. An OO approach lends itself to modelling real-world objects and their relationships and provides a high level of formality through the use of well-defined interfaces [175]. New types of context can also be easily added while limiting the impact on other existing context types. The Context Container is responsible for maintaining the context model, such as updating the values of current context and persisting context to history when new context arrives. Like acquisition from sensors, current context and context history can be accessed through the Context Container using either pull or push techniques. The context model, stored as part of the Context Management layer, is hierarchically structured with the following four context types at the root of the hierarchy as illustrated in Figure 3.4.

- Activity - information about real world tasks that users complete e.g., travelling to a certain location. Behaviour recommendations evaluated by applications are designed to aid users with completing these tasks.
- People - information about people e.g., user preferences.
- Artefact - information about real-world objects or services e.g., a particular restaurant or travel Route.
- Feature - information about the environment e.g., weather or latitude-longitude coordinate.

Application specific context types are represented as subclasses of one of these context types. Context information retrieved from the context model as part of this personalisation approach are in type-value tuple form.

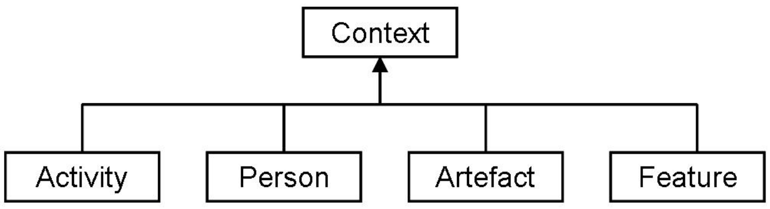


Figure 3.4: Hermes Context Model

3.2.3 User Modelling

A user model is defined as *"collections of information and assumptions about individual users (as well as user groups) which are needed in the adaptation process [...] if a system is supposed to automatically adapt to the requirements of the current user"* [99]. The construction of an accurate user model is therefore a critical element of personalised systems, the success of which is largely dependent on the ability to accurately represent a user's preferences [134].

Existing personalised recommender systems have adopted two main types of user models: feature vector models and history-based models[4] [134]. Common to both of these model types is a collection of features or items. A feature vector model contains a set of features where each feature has an associated rating. For example, a restaurant recommender system (e.g., Entree) would contain features such as cuisine type, cost, service/atmosphere, and food quality. Each instance of a feature value (e.g., Italian food or low cost) would have a corresponding rating to represent a user's preference for that feature. In contrast, history-based models contain information about a user's past experiences, which are represented as a set of cases. Cases are typically described as problem-solution pairs - the problem represents a description of the previous situation and the solution represents the recommendation selected by the user for that situation. Re-using the restaurant recommender example, a sample case in such a system would contain a restaurant name (e.g., Roma) in the solution portion of the case and properties associated with that restaurant are stored in the problem portion of the case (e.g., Italian food or low cost). In future, when the user queries for a low cost Italian restaurant, Roma will be recommended.

The user model constructed for the personalisation process proposed in this thesis borrows properties from both feature vector and history-based user models. This user model, like feature vector models contains a set of features and associates ratings with each feature. In this model, ratings represent the relative importance (or utility) of features in past decisions. Specifically, this user model is concerned with attaching utility values to environmental context information to represent how much weight different context types have in deciding the appropriate behaviour to recommend. The process of determining the utility value to associate with different context is described in Section 3.2.7.

Like history-based models, this user model is designed to store a set of cases that capture the past behaviour of a user, gathered from implicit feedback. However, where history-based cases are mainly structured as problem-solutions pairs, the structure of this user model case is extended in a number of directions. Firstly, the user model cases record information about environmental context that existed at the time of each behaviour (e.g., user location, weather). Secondly, each case is extended to store information about features of a particular solution (e.g., cuisine type, cost for a restaurant solution). Finally, cases are extended to store information about multiple different recommendation problem types and correspondingly multiple different solution features. For example a restaurant recommendation problem has solution features such as cuisine, food quality, and cost, while a route

[4]Due to their similarity, feature vector models and user item rating models as outlined in Section 2.1 are grouped together here.

recommendation problem will have solution features such as distance and travel time.

To support the extended set of information, the user model used for this personalisation approach consists of five main parts[5]: a problem type, a solution, a set of solution features, a set of environmental context information, and a set of weights that represent the utility of context information that was considered for that case. The problem type part of the case records the type of recommendation problem that case refers to (e.g., a restaurant recommendation problem), the solution records the recommendation that was selected by the user (e.g., Roma), and the solution features represent the characteristics of the selected solution (i.e., the characteristics of Roma restaurant, e.g., Italian cuisine, low cost). Different problem types will have different solutions and different sets of solution features that correspond to the problem type. For example, cases associated with restaurant recommendations will have one set of (restaurant related) features, while cases about travel routes will have a different (route-related) set of features. The set of context feature values records the values of environmental context at the time of the problem (e.g., user at work, dry weather) and the set of weights represent the utility assigned to relevant context features for that recommendation problem.

The structure of this user model is more formally defined as:

$$U = \{Cs_1,\ Cs_2,\ Cs_3 ... Cs_n\} \tag{3.1}$$

where Cs is a user model case. Each Cs is defined as:

$$Cs = \{P_t,\ S,\ (F_1,\ F_2,\ F_3 ... F_x),\ (C_1,\ C_2,\ C_3 ... C_y),\ (W_1,\ W_2,\ W_3 ... W_y)\} \tag{3.2}$$

where P_t is the problem type, S is the solution selected by the user, $\{F_1,\ F_2,\ F_3 ... F_x\}$ are the set of features associated with a given solution, $\{C_1,\ C_2,\ C_3 ... C_y\}$ are the set of environmental context, and $\{W_1,\ W_2,\ W_3 ... W_y\}$ are the set of weights representing the relative importance of context features.

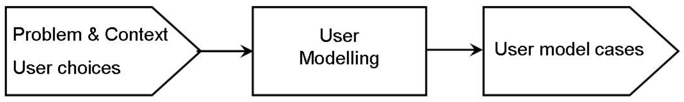


Figure 3.5: User Modelling Inputs and Outputs

Figure 3.5 illustrates the set of inputs required for populating user model cases, which consist of user recommendation selections and environmental context information. This set of information

[5]Note that, in addition, a time stamp value associated with each case is also stored.

is acquired from users and from context sources using the Hermes framework and is accessed by applications through Context Management layer functionality. The output of User Modelling is a set of user model cases, which are created for each recommendation problem and persisted when a user selects a particular recommendation solution. For example, when requesting a restaurant recommendation, the application will make a recommendation using the user's context and their preferences (inferred from their past restaurant selections using techniques and algorithms provided by the multi-stage process described in subsequent sections of this chapter). When the user selects one of the recommended restaurants, their selection, including its feature values, the environmental context of the recommendation problem, and the weights assigned to different context features as part of the recommendation decisions are stored as a new case in the user model. This is viewed as recording a snapshot of the environment and the associated behaviour selected by the user [199].

The current design assumes that users explicitly select their preferred solution when presented with a set of different recommendations. However, it is important to note that users are not required to input any preference related information such as providing ratings for candidate solutions. The solution selected by the user is the only form of feedback the application receives from the user. Approaches for monitoring user actions to determine their selected behaviour are currently not provided but have been investigated. Existing approaches to monitoring user behaviour such as those in the area of activity recognition e.g., the CIGAR framework by Hu et al. [90] and the RFID-based approach by Liu et al. [119] can be supported and integrated with the approach to automatically determine the recommendation selected by the user.

To support applications in addressing the requirements outlined in Section 1.2, this user model extends the set of information stored in previous models by recording information about solution features, environmental context, and context utility. Firstly (Requirement 1, Section 1.2), support is provided to applications to tailor decisions to different context values and relationships because information about the environmental context at the time of each behaviour is recorded as part of user model cases. The combination of problem types and environmental context enables applications to focus on a specific set of relevant cases. Storing environmental context information along with solution information also provides a set of information from which dependencies between context and user behaviour selections can be identified. This supports applications to infer knowledge about the effect of different context on the preferred behaviour of users and applications can therefore provide more accurate recommendations. Secondly (R2), new user behaviour selections along with the context at the time of the behaviour, which are automatically acquired, are added to the user model with application use. Storing cases as the application is used ensures that the user model remains up-to-date without necessitating explicit user input and enables the application to evolve to support previously unconsidered context relationships and any changes to user preferences (as a series of cases will capture how a user's habits change over time). Thirdly (R3), storing cases for different problem types supports applications in making recommendations about different types of problems (e.g., a restaurant recommendation differs from a route recommendation). The design also facilitates applications to make recommendations about multiple different types of problems using the same process (e.g., present a restaurant recommendation *and* a route recommendation). Also, by storing features of a solution, and not only a solution identifier, the user model accommodates changing properties associated with a particular solution. Therefore, if certain properties a restaurant were to change (e.g., from low cost to high cost), or if the restaurant no longer exists, the application would still be able to make useful recommendations (i.e., by recommending a restaurant with similar characteristics). This also supports applications in making *diverse* recommendations (i.e., by recommending a restaurant with similar preferred features that was *not* previously recommended) minimising the problem of *over-specialisation*[6] [6, 134]. The application of this user model with techniques and algorithms described in later sections supports the requirements for dynamic (R4) and autonomous (R5) execution.

3.2.4 Association Determination

Once a set of user model cases is available, they can be analysed to determine patterns in user behaviour that will aid with personalised recommendation decision making. The personalisation approach proposed in this thesis adopts data mining [80, 188, 79], and in particular the association discovery technique [8] for identifying user behaviour patterns.

Data mining, also referred to as knowledge discovery, is the process of *"nontrivial extraction of implicit, previously unknown and potentially useful information"* [45]. Mining enables interesting knowledge, high level information, and relevant patterns to be extracted from large sets of information and provides a valuable and reliable source for knowledge generation and verification [105, 45]. One of the main techniques of data mining is association rule discovery. Association rule discovery is a process that discovers associations or correlations among sets of items in a data structure [7, 8]. The discovery of association rules was first introduced for market basket analysis where a large collection of basket items is mined with the aim of increasing sales by knowing what items people bought together [7]. For example, an association rule $bread \Rightarrow milk$ indicates that when a user purchases bread, they also purchase milk.

[6] *over-specialisation* results when applications are restricted to only providing recommendations that have been presented to the user in the past [134].

While association rule discovery is effective at discovering associations between items in a collection [8], its main limitation is that generated associations may be irrelevant or redundant [15, 195]. Therefore, an important challenge is to analyse and filter discovered rules to ensure that only those that are useful are retained.

The personalisation approach proposed in this thesis includes an Association Determination process as part of its multi-stage recommendation process. It consists of two sub-processes: *association discovery* and *rule elimination*. Association discovery adopts the association rule discovery technique as a method for identifying patterns in user behaviour. Specifically, this technique is applied to discover associations between context and user behaviour choices from relevant user model cases, which provides the knowledge from which the preferred behaviour of a user in a particular context is inferred. Rule elimination provides techniques responsible for removing uninteresting and redundant rules from the discovered rules set. A newly developed technique for removing rules that do not represent relationships between context and user actions is combined with existing techniques designed to remove uninteresting and redundant rules. As input, Association Determination requires a set of user model cases and outputs a set of non redundant association rules representing dependencies between context and user actions that are present in user model cases (Figure 3.6).

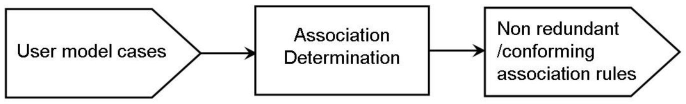

User model cases
Association Determination
Non redundant /conforming association rules

Figure 3.6: Association Determination Inputs and Outputs

3.2.4.1 Association Discovery

Associations between information items are discovered by determining the frequency of items occurring together (known as itemsets) in a data collection. More formally, the process consists of the following: $I = \{i_1, i_2, i_3 ... i_n\}$ is the set of distinct attributes, called *items*; T is the set of transactions (i.e., user model cases), where each transaction t contains a set of items such that $T \subseteq I$; an association rule is an implication of the form $X \Rightarrow Y$ (i.e., X implies Y), where X, $Y \subset I$ called itemsets in I and $X \cap Y = \phi$ [7]. The left hand side of the rule X is known as the *antecedent* or *premise* and the right hand side of the rule Y is known as the *consequent*. This process of discovering association rules generally consist of the following steps [45]:

- discover the set of large itemsets i.e., the sets of itemsets that have a support value above a specified minimum s. Itemsets that do not have the minimum support are discarded

- use the large itemsets to generate association rules.

The *Apriori* algorithm [8] is adopted as the association rule discovery algorithm used in the personalisation approach[7]. Apriori discovers associations by iteratively constructing candidate sets of large itemsets, counting the number of occurrences of each candidate itemset, and determining large itemsets based on a pre-determined minimum support. Figure 3.7, drawn from information in [80, 188], illustrates the Apriori algorithm.

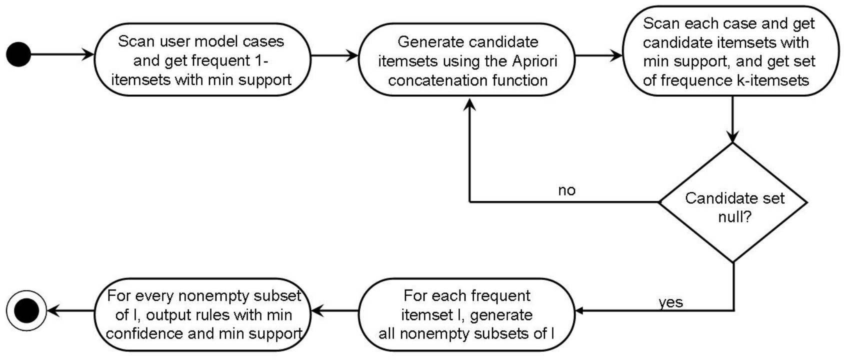

Scan user model cases and get frequent 1-itemsets with min support
Generate candidate itemsets using the Apriori concatenation function
Scan each case and get candidate itemsets with min support, and get set of frequence k-itemsets
Candidate set null?
no
yes
For each frequent itemset l, generate all nonempty subsets of l
For every nonempty subset of l, output rules with min confidence and min support

Figure 3.7: Apriori Algorithm

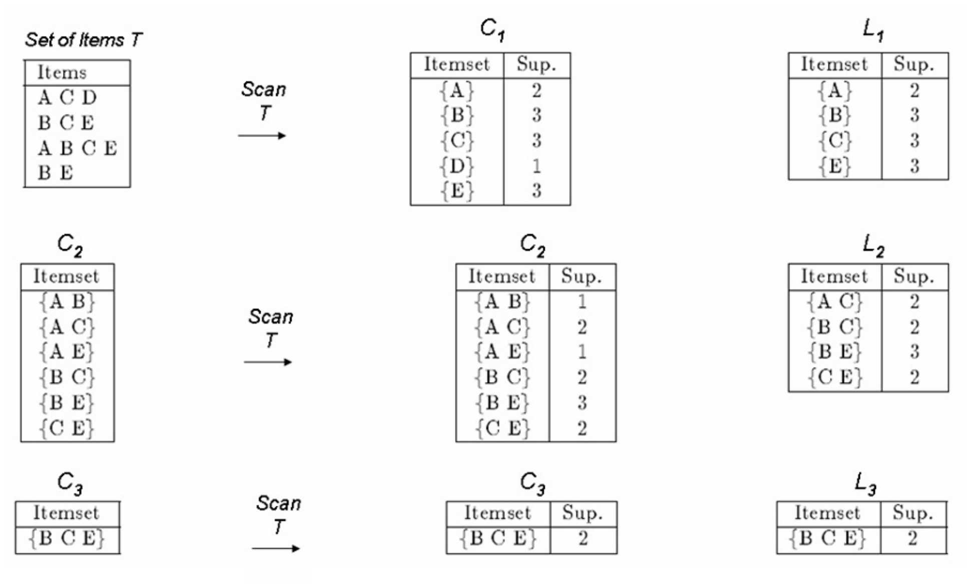
The first pass of the algorithm counts the number of occurrences for each item (i.e., 1-itemsets/ itemsets with 1 item). Those that do not meet the specified minimum support are discarded. Each subsequent pass k consists of two phases. First, the large itemsets L_{k-1} are used to generate the candidate itemsets C_k. Apriori uses $L_k * L_k$ to generate candidate sets of itemsets where $*$ is a concatenation operation of the form:

$$L_k * L_k = \{X \cup Y \mid X, Y \in L_k, \mid X \cap Y \mid = k-1\} \tag{3.3}$$

The second phase scans each transaction and the support of each candidate itemset in C_k is counted. Those that do no meet the minimum support are again discarded. This process continues with increments to k until no more large itemsets can be generated. Once complete, non empty itemsets are scanned and association rules are outputted.

The example taken from [45] shown in Figure 3.8 illustrates the generation of itemsets for the Apriori rule discovery process with sample values. The first pass generates the set of candidate *1-itemset* C_1. Assuming a minimum support of *2*, those itemsets that do not meet this support value

[7]Apriori is the original association algorithm and provides the basis for other association mining algorithms that have been developed [108]


Scan T
Scan T
Scan T

Set of Items T

Items
A C D
B C E
A B C E
B E

C_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A B}
{A C}
{A E}
{B C}
{B E}
{C E}

C_2

Itemset	Sup.
{A B}	1
{A C}	2
{A E}	1
{B C}	2
{B E}	3
{C E}	2

L_2

Itemset	Sup.
{A C}	2
{B C}	2
{B E}	3
{C E}	2

C_3

Itemset
{B C E}

C_3

Itemset	Sup.
{B C E}	2

L_3

Itemset	Sup.
{B C E}	2

Figure 3.8: Apriori Candidate and Large Itemsets Generation [45]

are discarded. The large itemsets L_1 are then used to discover the set of *2-itemsets* and candidate itemsets C_2 using the * operation. From C_2 the set of large 2-itemsets L_2 are generated. This process is repeated for the generation of candidate itemsets C_3 and large itemset L_3. As there are no 4-itemsets to be discovered from L_3, the Apriori process of discovering large itemsets ends.

Figure 3.9 illustrates a sample set of association rules with real values generated for a restaurant recommendation problem. Rules generated show associations between context and past user choices for solutions and individual solution features (i.e., the solution and solution features portion of a user model case).

1. time=afternoon weather=rain → solution=Roma
2. time=evening company=friends → cuisine=irish price=expensive
3. day=weekend company=alone → cuisine=italian service=three
4. time=evening weather=rain → solution=Gallaghers
5. day=weekday company=alone → solution=Roma

Figure 3.9: Sample Association Rules

3.2.4.2 Rule Elimination

A limitation of association discovery is that it outputs a large number of associations, many of which are uninteresting and/or redundant [15]. Rule elimination is the second part of Association Determination, and provides functionality to remove uninteresting and redundant rules. The personalisation approach proposed in this thesis uses three different methods to remove unnecessary association rules.

Firstly, measures of *interestingness* are used to restrict the number of association rules. In association mining, there are two main metrics for measuring the *interestingness* of rules: *confidence* and *support* [45]. Generally speaking, confidence measures the significance of a rule as the number of times a rule is correct and support measures how often a rule occurs in a data set . In mathematical terms, confidence is the percentage of transactions that contain X *and* Y to the total number of transactions that contain X. Support is the percentage of transactions that contain X *and* Y together to the total number of transactions in the *database*.

$$Confidence\,(X \Rightarrow Y) = \frac{number\,transaction\,containing\,X\ \cap\ Y}{total\,number\,of\,transactions\,containing\,X} \tag{3.4}$$

$$Support\,(X \Rightarrow Y) = \frac{number\,transaction\,containing\,X\ \cap\ Y}{total\,number\,of\,transactions} \tag{3.5}$$

More formally, the rule $X \Rightarrow Y$ has a confidence c if $c\%$ of transactions in T that contain X also contain Y. The rule $X \Rightarrow Y$ has support s if $s\%$ of transactions in T contain $X \cup Y$. Confidence indicates the strength of the implication, so an association rule $X \Rightarrow Y$ with 80% confidence means that 80% of transactions that contain X also contain Y, while support denotes the frequencies of recurring patterns in a rule. *Strong* rules have high confidence and high support and the higher the support and the confidence the more accurate and reoccurring the rule [45]. Values of confidence near value 1 are expected for important association rules. As the number of transactions and items may be large, support and confidence threshold values are useful for constraining rules so that only those that are interesting or useful (i.e., strong rules) are selected for decision making.

Secondly, this personalisation approach provides a technique that filters rules by restricting retained rules to a certain form. The execution of association rule discovery is designed to discover relationships between any items in an itemset. Consequently, rules of the form $context \Rightarrow context$ or $solution/solution\,feature \Rightarrow solution/solution\,feature$ may be generated. However, context-aware personalisation is interested in rules that represent patterns in user behaviour. In particular, personalisation is interested in association between context and user behaviour choices (i.e.,

$context \Rightarrow solution/solution\,feature$). Rules in the rule set that do not conform to the form $context \Rightarrow solution/solution\,feature$ provide no useful knowledge and can therefore be discarded. Remaining rules will be of the appropriate form where the antecedent part of the rule consists of context types and the consequent part of the rules consists of solutions and solution features. Reducing the number of rules to be considered in this way enables applications to focus on those that provide relevant user preference knowledge.

The third approach to rule elimination, that was added as part of the extended strategy (shown in Figure 3.2), is designed to remove redundant rules from the rule set. While selecting rules using their support and confidence measures and restricting rules to the $context \Rightarrow solution/solution\,feature$ form will reduce the number of rules that are retained, redundant rules (i.e., rules that have the same meaning) may still exist in the rule set [47, 14]. For example, consider three rules $X \Rightarrow Y$, $X \Rightarrow Z$, and $X \Rightarrow YZ$. As the third rule is a simple combination of the first two rules, either the first two rules *or* the third rule can be marked as redundant as the two sets of rules convey the same information. The design was extended to remove redundant rules to ensure that recommendation decisions are not biased as a result of considering more than one rule with the same meaning. An approach proposed by Ashrafi et al. [15] is adopted to remove redundant rules from the rule set. Ashrafi provides two algorithms for removing redundant rules - one algorithm is designed for rules with multiple antecedents and the other for rules with multiple consequences - which eliminate redundant rules without losing any higher confidence rules that cannot be substituted by other rules. Figure 3.10 illustrates the algorithm for removing redundant rules with multiple consequence items but have the same antecedent items, drawn using information in [14]. The algorithm begins by selecting a rule from the rule set that has multiple consequence items, referred to as the *proper rule* (e.g., $X \Rightarrow YZ$). Once it finds such a rule, a check is carried out to determine if there are other rules in the rule set that have the same antecedent, referred to as *sub-consequence* rules (e.g., $X \Rightarrow Y$, $X \Rightarrow Z$). If so, the consequence items of these rules are examined to determine if their combination equals the consequence of the proper rule. If this applies, either the proper rule or the set of sub-consequence rules is considered redundant. Ashrafi et al. describe several analogies to aid with determining the rule(s) (either proper or sub rules) that should be considered redundant using confidence and support measures: total dominance, partial dominance, and indifference dominance [15]. For rules with multiple consequences: total dominance is evident when either the proper rule or sub-consequence rules have higher support and confidence values; partial dominance is evident when several sub-consequence rules, but not all, have a higher confidence value than its corresponding proper rule; and indifference dominance occurs when a proper rule and its sub-consequence rules have the same support and confidence. Given the structure of rules with multiple consequences (i.e., single antecedent and multiple consequences), total and partial dominance will result in the proper rule being considered redundant; and indifference dominance will result in the set of sub-consequence rules being redundant [15]. The removal of redundant rules with multiple antecedents but same consequences follows a similar sequence of steps, except it first finds rules with multiple items in the antecedent, looks for other rules that have the same consequence (*sub-antecedent* rules), and determines if the combination of antecedents of these sub-antecedent rules equals the antecedent of the proper rule. Ashrafi et al. describe similar analogies for rules with multiple antecedents as outlined previously for rules with multiple consequences. For rules with multiple antecedents: total dominance is evident when either the proper rule or sub-antecedent rules have higher confidence; partial dominance is evident when several sub-antecedent rules, but not all, have a higher confidence value than its corresponding proper rule; and indifference dominance occurs when a proper rule and its sub-antecedent rules have equal confidence. Given the different structure of rules with multiple antecedents (i.e., multiple antecedents and single consequence), total dominance considers either the proper rule or sub-antecedent rule, whichever has a lower confidence, as redundant; partial dominance will result in sub-antecedent rules being redundant; and indifference dominance will result in the proper rule being redundant [15].

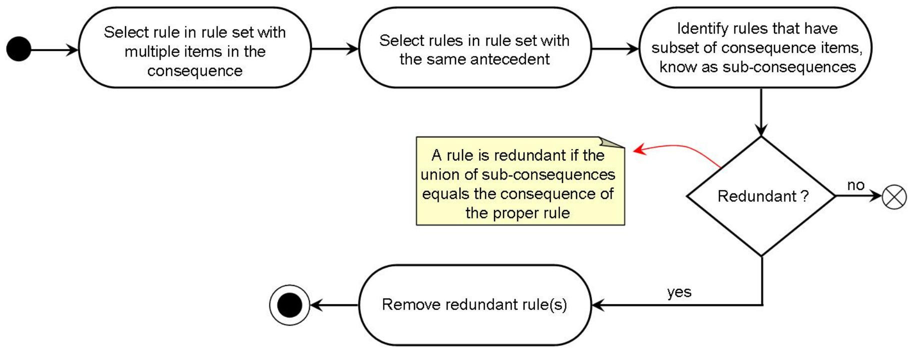

Select rule in rule set with multiple items in the consequence
Select rules in rule set with the same antecedent
Identify rules that have subset of consequence items, know as sub-consequences
A rule is redundant if the union of sub-consequences equals the consequence of the proper rule
Redundant ?
no
yes
Remove redundant rule(s)

Figure 3.10: Redundant Rule Elimination

The combination of techniques and algorithms adopted at this Association Determination stage is designed to facilitate context-aware applications with dynamically and implicitly determining interesting relationships between context and user behaviour choices. Data mining, and in particular association discovery, was selected because it is not a specific algorithm that is explicitly defined for a particular application or domain. Instead it provides a general purpose approach that determines key relationships in the data by following a data mining cycle (data collection and pre-processing, pattern discovery, and recommendation) [131]. Association discovery can also concurrently look at numerous multidimensional information relationships, highlighting those that are exceptional or interesting [137] and is more scalable than other approaches due to the focus of data mining techniques on efficient pattern discovery [131, 137].

The integration of the association discovery technique into the personalisation approach described in this thesis was taken to ensure greater flexibility (than specifically pre-defined recommendation algorithms) for supporting different types of applications, recommendation problems, preferences, and context - therefore facilitating personalisation tasks to be more easily and effectively integrated into existing applications. Its application, along with techniques and algorithms incorporated for rule elimination, to the previously described user model is intended to facilitate applications to address the requirements outlined in Section 1.2. Firstly, as association discovery is not restricted to discovering associations between specific information types. Therefore its application can facilitate the identification of relationships between context and user actions, for multiple different context types/values and multiple different action types/values, supporting applications with determining user preferences for different combinations of context and different types of recommendation problems using the same process. Techniques and algorithms for filtering and prioritising discovered associations are intended to ensure that applications focus only on rules that are most accurate (or strongest) and relevant for a given problem and context (R1 and R3, Section 1.2). Secondly, the approach is designed so that associations between context and user choices are explicitly identified. This enables inferred knowledge to be used at later stages of the decision making process to infer and tailor information relevance and utility for different recommendation problems and context (Section 3.2.6 and Section 3.2.7) (R1 and R2). Also, by explicitly identifying associations in rule form, explanation methods can be easily integrated, enabling users to query and scrutinise applications about their recommended behaviour, and enabling applications to provide an intuitive and description response to user queries. Thirdly, the decision to discover associations between context and user actions at run-time from up-to-date user model data was taken to ensure that context-aware applications have the ability to evolve to support previously unconsidered context relationships and changing user preferences (R2). Run time execution, when required, using association discovery and information about past user interaction also limits the need for developer and user input (R4 and R5). Finally, the application of association discovery provides an approach to discovering interesting information relationships that is easily configurable and can be customised in various ways according to the requirements of an application. For example the number of rules that are discovered can be specified, as can values for minimum support used during the candidate generation process. Section 4.3 in the following chapter expands on the configuration details related to association discovery.

However, it is important to note that association rule discovery is not without its limitations. The main limitation, particularly with the Apriori algorithm, is its discovery of large numbers of often redundant and uninteresting rules and its bottlenecks in terms of its processing and memory requirements. The association determination in the personalisation approach proposed in this thesis has been designed to address the first problem by including techniques and algorithms that remove and filter discovered rules, ensuring that only the most accurate and relevant are retained (Section 3.2.4.2). No techniques have been integrated to specifically address the efficiency problem of Apriori, but this approach is designed in a pluggable manner and is not restricted to using the Apriori algorithm. Other, more efficient, association discovery algorithms can be applied such as those by Wang and Tjortjis [184], Wu et al. [189] and Liusheng et al [120]. These algorithms will discover association rules in a similar manner to Apriori, but are designed to discover rules with greater efficiency [108].

Apart from association discovery, *clustering* provides a possible data mining alternative for solving recommendation problems [71]. The aim of clustering is to divide a data set into groups or clusters so that the data in each subset shares some similarity, defined by some *distance* measure[8]. In a personalised context-aware application, clustering techniques could be adopted to classify problems based on the type of recommendation to be made. However, the main limitation of clustering is the need to accurately define the concept of distance. In general, relevant domain expert knowledge is required to define an optimum concept of distance. When applied to personalisation and context-awareness, this provides a non-trivial challenge due to the nature of available data such as interactions between different context types and user preferences and the difficultly in specifying many context types in numeric form. Clustering algorithms are also ineffective when applied to problems with multi-dimensional data as there are not enough items to populate the vector space [71]. In addition, clustering does not support the presence of variances in information relevance and utility that exist as a result of differences in context.

3.2.5 Implicit Preference Determination

The primary goal of Implicit Preference Determination is to automatically infer user preferences for a given problem and context by identifying patterns from past user behaviour stored in the user model. Implicit Preference Determination determines user preferences from past user behaviour, without the need for users to specify their preference details explicitly. Instead, preferences are

[8]distance measures the degree of similarity between data.

automatically inferred from past user behaviour and implicit user feedback. Current applications of implicit preference techniques have been restricted to web search and information retrieval (document) systems, which are limited in the number of environmental context types they consider when executing (Section 2.1.2). The Implicit Preference Determination technique described in this thesis differs as it supports different types of environmental context and automatically discovers relevant relationships between them. More importantly for personalisation, the approach also automatically determines how different context and combinations of context affect the preferred actions of users. This knowledge enables applications to identify the effect of different context values on user preferences and therefore supports applications in providing personalised recommendations to users for different user problems and context settings.

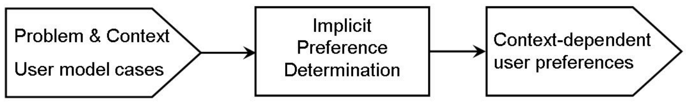

Problem & Context
User model cases
Implicit Preference Determination
Context-dependent user preferences

Figure 3.11: Implicit Preference Determination Inputs and Outputs

The Implicit Preference Determination process begins when a user queries the application for a recommendation about a particular problem (e.g., a user requests a restaurant recommendation from the application). As illustrated in Figure 3.11, this process takes a set of relevant user model cases as input and outputs the preferred actions of the user for the given problem and context (i.e., context-dependent preferences). The Association Determination process is used as the technique for identifying patterns in user behaviour (Section 3.2.4). As previously described, this process does not require explicit human knowledge or input for identifying relationships between context, user preferences, and appropriate behaviour. Instead, associations between context and user choices, from which recommendation decisions are made, are autonomously and dynamically inferred at run time.

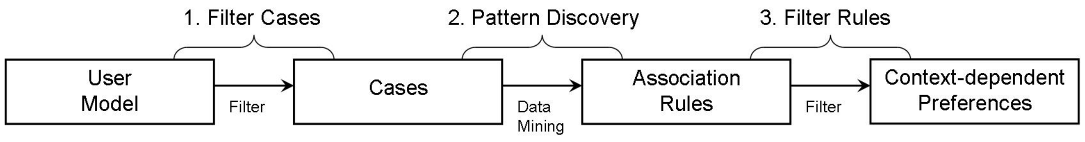


Figure 3.12: Implicit Preference Determination Stages

The sequence of steps design for Implicit Preference Determination is illustrated in Figure 3.12. Association rules are discovered using the previously described Association Determination technique (Section 3.2.4), while the personalisation approach proposed in this thesis adds techniques that filter both user model cases and discovered associations. These steps are designed to ensure that the most accurate and relevant set of user preferences for a user problem and context are determined and applied to ranking candidate solutions. The *Filter Cases* stage acts as the pre-processing stage for association discovery and provides the functionality that filters the set of user model cases so that only those cases relevant for the given recommendation problem and context are considered. The *Pattern Discovery* stage provides the data mining functionality for discovering user behaviour patterns from selected user model cases using the association determination process described previously (in Section 3.2.4). Finally, the *Filter Rules* stage, added as part of the extended recommendation strategy, filters discovered association rules to ensure only those relevant to the user problem and context are retained.

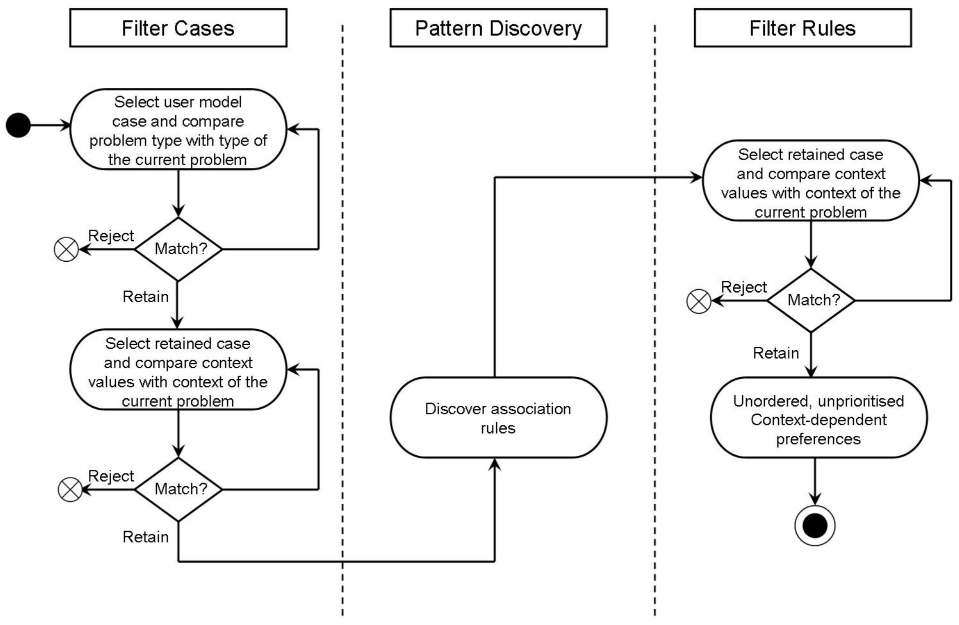

Filter Cases
Pattern Discovery
Filter Rules
Select user model case and compare problem type with type of the current problem
Reject
Match?
Retain
Select retained case and compare context values with context of the current problem
Reject
Match?
Retain
Discover association rules
Select retained case and compare context values with context of the current problem
Reject
Match?
Retain
Unordered, unprioritised Context-dependent preferences

Figure 3.13: Implicit Preference Determination

Figure 3.13 provides a more detailed illustration of Implicit Preference Determination, including the sequence of functions that execute. The first stage, Filter Cases, provides functionality for determining the set of cases that should be used in pattern discovery. The aim is to ensure that only the relevant set of user model cases is considered for recommendations decisions. Two approaches are supported for filtering cases: *semantic* and *context -based* filtering. Semantic-based filtering filters user model cases based on the type of recommendation problem being processed. For example, a user model may have cases pertaining to restaurant recommendations, travel route recommendations, and many others. The aim of semantic filtering is to select the correct set of cases for the current user problem (i.e., recommendation decisions regarding restaurant should be made on user model restaurant cases). To achieve this, the *type* of the current user problem is compared with the *problem type* value of user model cases. Those that match are retained. Semantic filtering can also be adjusted to collect cases that are semantically similar to the current problem type (for example if there are a limited number of cases associated with a particular problem type). Semantic similarity is defined by an application's domain model hierarchy structure. For example, take a dining object with restaurant and cafe children objects. When determining restaurants to recommend, all cases associated with restaurants are retrieved, but cases related to cafes can also be selected i.e., users who like comfortable restaurants will prefer comfortable cafes. The approach is flexible and can be adjusted to traverse any number of levels or particular branches of the hierarchy. This flexibility also enables applications to easily adjust to different types of user problems.

Context-based filtering, in contrast, filters user model cases so that only those that are relevant to the context of the current user problem are retained. This process compares the values of *context* in each case with the context values of the current problem. For example, if the problem context is *afternoon time* on a *weekend*, then only user model cases that contain context features whose values match these are selected. Cases associated with *evening time* and *weekdays* will not be retained for the remainder of the decision process. The approach is designed to retain cases that match at least one context type, therefore ensuring that any case that may provide knowledge about a user's preferences is not discarded. While this is the approach adopted by the current design, it is adaptable so that only cases that match a certain specified number of context values are retained (Section 4.3). Figure 3.14 illustrates both the semantic and context-based filtering process[9]. In the illustrated example, Case 1 and Case 3 will be retained as they semantically match the current problem type, as underlined. They will also both be retained as they have context values that match the problem context, also underlined.

The second stage of Implicit Preference Determination, Pattern Discovery, provides functionality that identifies patterns in user behaviour using cases retained after the Filter Cases stage. Association Determination (described in Section 3.2.4) provides the technique used for discovering associations between context and user behaviour choices. These associations are in rule form and represent the general context-dependent preferences of a user for the current problem.

The Pattern Discovery stage will generate a large set of rules of the form *context* $\Rightarrow$ *solution/solution feature*. However, because cases retained at the Filter Cases stage may include context values that do not match that of the current problem, a certain number of discovered rules will not be relevant.

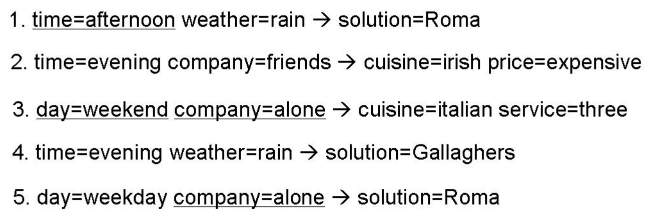
[9]weight values stored as part of a user model case are omitted from this illustration

```
Problem   {
and       (ptype=restaurant),
context   (time=afternoon, day=weekend, weather=sun, company=alone)
          }

Case 1    {
          (ptype=restaurant),
          (solution=Roma),
          (cuisine=italian, price=cheap, food-quality=three, service=three)
          (time=afternoon, day=weekday, weather=rain, company=alone)
          ...
          }

Case 2    {
          (ptype=route),
          (solution=grafton-westmoreland-o'connell-parnell),
          (route-type=shortest, transport=walk, toll=false)
          (time=night, day=weekend, weather=sun, company=alone)
          ...
          }

Case 3    {
          (ptype=restaurant),
          (solution=Gallaghers),
          (cuisine=irish, price=expensive, food-quality=four service=three)
          (time=evening, day=weekend, weather=rain, company=friends)
          ...
          }
```

Figure 3.14: Filtering User Model Cases

Filter Rules was added as part of the extended strategy to further ensure that only rules relevant for the current recommendation problem and context are considered. Therefore, Filter Rules provides functionality to filter irrelevant associations that are generated during pattern discovery. Filtering is performed in a similar manner to context-based case filtering. That is, if a rule has any context value that matches the context of the current problem, then it is retained. This, like case filtering, is also adaptable so that only rules that match a certain specified number of context values are retained (Section 4.3). Figure 3.15 illustrates a set of sample association rules. Given the same context types and values illustrated in Figure 3.14, Rules 1, 3, and 5 will be retained as part of their antecedent values match the context of the current problem as underlined. The final set of retained rules represents the unordered and unprioritised context-dependent preferences of a user. Inferred from patterns in user behaviour, they represent the preferred behaviour of a user that are specifically customised to the current problem and problem context.

The sequence of steps of Implicit Preference Determination is designed to support the automatic identification of preferred user behaviour for different problems and context. Association determina-



1. time=afternoon weather=rain → solution=Roma
2. time=evening company=friends → cuisine=irish price=expensive
3. day=weekend company=alone → cuisine=italian service=three
4. time=evening weather=rain → solution=Gallaghers
5. day=weekday company=alone → solution=Roma

Figure 3.15: Filtering Association Rules

tion is incorporated as part of this technique to support the automatic determination of user preferences, eliminating the need for expert knowledge and input for identifying and specifying relationships between context, user preferences, and recommendation behaviour. The use of association mining is also intended to facilitate the identification of associations between multiple different data types, therefore supporting the inference of user preferences for multiple different context combinations and problem types. The design and incorporation of filtering techniques before and after this association discovery process is intended to ensure that applications focus only on cases and discovered rules that are relevant for the current user problem and context. Also, the implicit preference determination technique is designed to discover relationships between context and user actions at run-time using an up-to-date set of user model cases, facilitating the determination of the most up-to-date user preferences for the current problem and context, and accommodating any changes to preferences that may have occurred over time.

3.2.6 Information Selection

The availability of information from different sources provides context-aware applications with a myriad of different context types and values. However, not all context types may be relevant at all times and different subsets of information will be relevant to decision making under different context situations. In addition, different context relationships (or interacting features [118]) may exhibit varying information relevance when coupled together, which are not evident in individual context types alone. One example is the XOR problem [118] in which the effect of considering information types separately differs from the effect of grouping different types together. This problem is referred to as information overload or the curse of dimensionality [81].

Information Selection is the technique designed to address this problem by providing functionality to determine the set of information (i.e., context types) that are necessary to consider when making recommendation decisions. The ability to select the subset of context types that are relevant from the original set of supported context types will lead to more effective performance [118, 178, 67].

To understand the issues of information relevance, consider the following movie recommendation example. Assume that a single context type *Location* has one of two values *Home* and *Cinema*. Also assume that the ratings given to movies watched at home (*Location* = *Home*) have the same distribution as movies watched at the cinema (*Location* = *Cinema*). This indicates that the location in which movies are watched does not affect the user's preferences regarding movies, and therefore the *Location* context can be considered irrelevant when making movie recommendations to the user.

Kohavi and John [104] provide three categories of information relevance: *strong relevance*, *weak relevance*, and *irrelevance*, which are important when determining what information types to conserve and what types to eliminate. The strongly relevant types should always be considered by any information selection process and necessary for an optimal information subset. These types cannot be removed without inversely affecting the effectiveness of recommendations. Weakly relevant types could be important and may be necessary for an optimal subset at certain conditions - depending on other types and their values already selected. Irrelevant types are not necessary and should not be considered. An optimal subset should include all strongly relevant information types, none of the irrelevant types, and a subset of weakly relevant types. The Information Selection technique described here enables developers to, if necessary, define strongly relevant types. Weakly relevant types (and irrelevant) types for the current user problem and context are dynamically determined at run-time.

Existing literature broadly classifies information selection (also commonly referred to as feature selection) into two categories: a *wrapper* model approach uses the predictive accuracy of a predetermined learning algorithm to determine the relevance of a selected subset of information types; while a *filter* model approach utilises various measures of generated characteristics of data to determine relevance [118]. However, both approaches are typically developed as a pre-processing operation and therefore do not adapt the relevance of information to different context or user problems. In existing context-aware applications and frameworks, a human expert or developer is typically responsible for defining the set of strongly and weakly relevant context types for recommendation decisions. For example, in the Sentient Object Model [29], relevant information types are defined as part of behaviour rules, while in Gaia [151] these types are defined as part of Bayesian network structures. The explicit specification of information is highly reliant on human knowledge, which is difficult especially when there are a large number of context types supported. Such an approach also does not adapt to changing information relevance as a result of changing user preferences.

The goal of Information Selection is to address these limitations by eliminating the need for explicit developer knowledge and input for specifying information relevance and also by adapting the set of

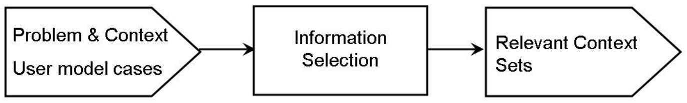


Figure 3.16: Information Selection Inputs and Outputs

information used in recommendation decisions to a given user problem and context. To achieve this goal, a filter model approach to feature selection is used as part of the personalisation approach proposed in this thesis. Specifically, the approach uses association rules discovered from user model data to determine information relevance and executes at run-time for each recommendation problem and problem context, meaning that the relevance of information is dynamically tailored. That is, the relevance of information is determined for a given recommendation problem and context, and is re-determined each time this information changes. Unlike the wrapper model approach to information selection, there is no requirement for a developer to pre-define any specific learning algorithm. A filter model approach is also more efficient than a wrapper model approach, which is computationally expensive for data with a large number of items [118].

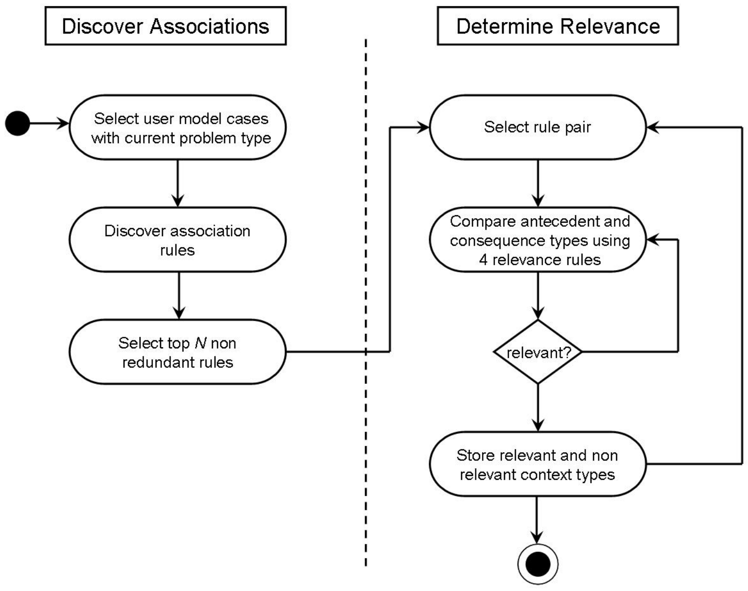

Discover Associations
Determine Relevance
Select user model cases with current problem type
Discover association rules
Select top N non redundant rules
Select rule pair
Compare antecedent and consequence types using 4 relevance rules
relevant?
Store relevant and non relevant context types

Figure 3.17: Information Selection

As illustrated in Figure 3.16, the approach taken to Information Selection in the personalisation approach described in this thesis requires data about the user's recommendation problem and a set of relevant user model cases. It uses this information to determine and output the set of context types that are relevant and to be considered for ranking candidate choices for the recommendation problem. Information Selection is divided into two stages, *Discover Associations* and *Determine Relevance*. Figure 3.17 illustrates the set of sub-tasks performed at each stage. The first stage, Discovers Associations, is designed, like Implicit Preference Determination (Section 3.2.5), to discover associations between context and user actions. User model cases are first filtered so that only those cases that match the current problem type are retained. Cases, as part of Discover Association, are not filtered by context, as the aim is to determine the set of context that in general are relevant for recommendation decisions of a given problem type. Association rules are discovered using the previously described Association Determination technique (Section 3.2.4) and the top N rules of the form $context \Rightarrow solution$ are retained. N defines the number of rules to be compared during Information Selection and can be configured according to the requirements of the application (Section 4.3). The second stage, Determine Relevance, is responsible for determining relevant context types or sets of types using the set of retained association rules. The design supports the storing of sets of types to facilitate the relevance of certain types given the presence of other types. At the first stage, a pair of rules in the rule set are selected and iteratively compared in a similar manner to the movie example outlined previously. That is, the solution of rules are compared and context types present in both rules are also compared. The aim of comparing context and solutions is to determine the set of context types that are relevant for making recommendations about a particular type of problem. Four information selection *relevance rules* have been designed as part of the personalisation approach described in this thesis to determine if a context type or set of context types are relevant.

1. When comparing two rules, if the values of a context type/set of context types in each rule are *equal* and the value of the solution are *equal*, then that context type/set of context types is *relevant.*

2. When comparing two rules, if the values of a particular context type/set of context types in each rule are *equal* and the value of the solution are *not equal*, then that context type/set of context types is *not relevant.*

3. When comparing two rules, if the values of a particular context type/set of context types in each rule are *not equal* and the value of the solution are *equal*, then that context type/set of context types is *not relevant.*

4. When comparing two rules, if the values of a particular context type/set of context types in each rule are *not equal* and the value of the solution are *not equal*, then that context type/set

of context types is *relevant.*

Figure 3.18 illustrate the Information Selection technique with three example association rules. Each pair of rules is compared in turn. For each comparison, the solution and the set of corresponding context types are compared. For example, when comparing association Rule 1 and Rule 2, the values for context type's *time* and *weather* are compared (as these types exist in both rules) and the values of the solutions in the two rules are compared. The *context set* (i.e., time *and* weather) is inferred as relevant using *relevance rule four* (i.e., corresponding context and solution values in each rule are not equal). The second comparison, comparing Rule 1 and Rule 3, found no corresponding context types (i.e., day and time) to be relevant using *relevance rule three.* Similarly, when comparing Rule 2 and Rule 3, a context set consisting of the *company* context type is evaluated as being relevant using *relevance rule four.* The number of times a context type is evaluated as being relevant and irrelevant is also recorded and this knowledge is used in Utility Assignment (Section 3.2.7). In this example the context set containing *company* and the context set containing both *time* and *weather* are considered relevant once and considered irrelevant twice.

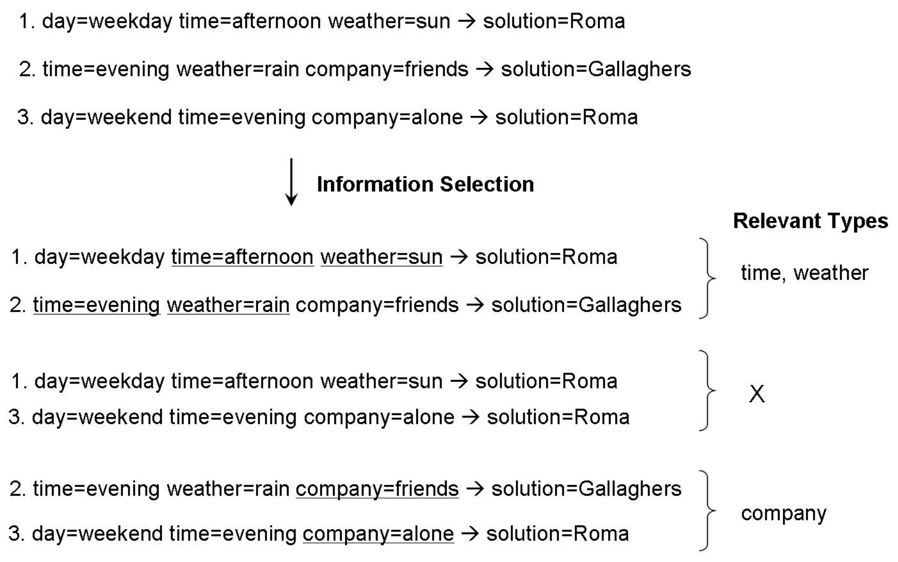

1. day=weekday time=afternoon weather=sun → solution=Roma
2. time=evening weather=rain company=friends → solution=Gallaghers
3. day=weekend time=evening company=alone → solution=Roma
Information Selection
Relevant Types
1. day=weekday time=afternoon weather=sun → solution=Roma
2. time=evening weather=rain company=friends → solution=Gallaghers
time, weather
1. day=weekday time=afternoon weather=sun → solution=Roma
3. day=weekend time=evening company=alone → solution=Roma
X
2. time=evening weather=rain company=friends → solution=Gallaghers
3. day=weekend time=evening company=alone → solution=Roma
company

Figure 3.18: Information Selection Example

While some approaches support relevance by placing weights on information, Information Selection is a boolean process. That is, a context set is either relevant or not relevant for the given problem. The subsequent Utility Assignment technique is responsible for assigning weights to context that are inferred as relevant. Distinguishing between these two methods eliminates the need for applications to evaluate appropriate information utility for the entire set of support context types. This design is taken to ensure that complexities associated with determining the relative importance between different items will be reduced given that less items are to be considered.

Determining the set of relevant types for a particular user problem and context can also be used to address the challenge of when to begin the recommendation process. While new user tasks will require a new recommendation process to be started so also may changes to values of relevant context types [65]. By identifying the set of relevant context, information selection serves as a technique for identifying when a new recommendation process should begin (and consequently a new case should be created and added to the user model). That is, the preferred action of a user is likely to change when the value of a relevant context type changes. This facilitates applications to avoid starting the recommendation process at the incorrect time, which can lead to inaccurate recommendations, be intrusive to the user, and result in sub-optimal performance.

The decision to incorporate association determination within Information Selection was taken for similar reasons as outlined for Implicit Preference Determination, namely the need for developers to explicitly specify relationships between context and user choices and the set of relevant context types that should be considered for different recommendation problems is eliminated. The decision to compare rules is taken to utilise knowledge discovered by the association determination process about patterns in user behaviour. Storing sets of data types is supported to facilitate dependencies between context, whereby certain context types are relevant depending on the type and value of other context. Information Selection is designed to execute at run time so that information relevance can be tailored to specific user problems and context. Information Selection rules are also designed to ensure that relevance for different types of problems and individual solution features can be determined. Also, as associations used during Information Selection are generated from up-to-date user model information, changes to how users decide which behaviour recommendation to select are likely to be captured.

3.2.7 Utility Assignment

Utility Assignment is the task of determining *how* relevant context types are relative to other types in recommendation decisions. While Information Selection produces a boolean output where a particular context type is either relevant or not relevant, Utility Assignment determines and assigns the appropriate weight to each relevant context type so that its relative importance reflects the preferences of a user for a given user problem and context. This process occurs after Information Selection so that the number of context types to be evaluated is minimised.

Current state of the art classifies the process of determining appropriate weights for different information items into *individual evaluation* and *subset evaluation* [194]. The personalisation approach in this thesis adopts an individual evaluation approach to assigning weights to context. Individual evaluation, also known as feature weighting/ranking, assesses individual features (i.e., context types in this thesis), and assigns weights according to their degrees of relevance. The main motivation for adopting such an approach is its efficiency with high-dimensional data, which is evident in a context-aware environment where there are a large number of relationships between different context types and user behaviour. The main limitation with individual evaluation is its inability to remove redundant context types. The design of Utility Assignment is taken to minimise this limitation by using Information Selection, which provides functionality for removing redundant association rules and reducing the number of context types to be evaluated. The alternative subset evaluation, in contrast, generates feature subsets based on a search strategy such as forward or backward search. Each candidate subset, along with associated weights, is evaluated by a certain evaluation measure and compared with the previous best subset, which it replaces if better. This process is repeated until a defined stop criterion is satisfied. Its main advantage is that it takes into account the existence and effect of redundant features, which is no longer necessary in this personalisation approach, as such a task is performed at an earlier stage. The main limitation of subset evaluation for context-aware applications is that it suffers from problems associated with searching through feature subsets and multidimensional data, and does not scale well to large sets of data [193]. Subset evaluation also fails to consider variance in the relevance of features in the selected subset.

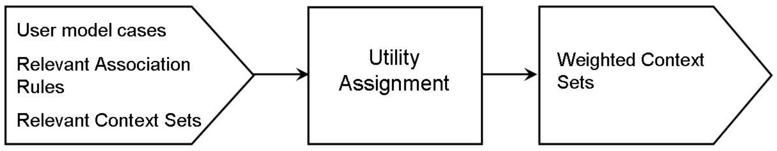


Figure 3.19: Utility Assignment Inputs and Outputs

Figure 3.19 illustrates the inputs and outputs of Utility Assignment. The association rules and relevant context sets evaluated at Information Selection are required as input. A set of weighted context sets for the current recommendation problem are then outputted. The larger the weight, the more influence a context type has on the recommendation decision.

The appropriate weight to assign to relevant context sets is determined using different properties of association rules. In general, context in stronger, more accurate, and more consistent rules will be given higher weights as they provide a more accurate indicator as to the preferred behaviour of users. This approach uses several measures of accuracy and consistency of association rules, which are combined to compute the utility of each relevant context set for a given recommendation problem. These measures are: consistency, precision and recall, and confidence and support and lift. These measures are selected as they represent the accuracy, consistency, and relevance of discovered association rules and the information they contain. They also represent the set of quality measure that can be calculated using the discovered rules. Measures of *coverage* and *leverage* related to association mining are not considered due to their similarity with confidence and support measures. A consistency value, calculated as part of Information Selection, represents how the occurrence of certain context types as indicators of past user actions. Precision and recall [168, 17] are two widely known and adopted metrics for measuring the relevance of information, while confidence and support values, as described in Section 3.2.4.2, provide measures that indicate the accuracy, occurrence, and *interestingness* of generated association rules. Lift is an additional metric at this stage, which measure the strength of a given association. Lift is the ratio of confidence to expected confidence (shown in formula 3.6). It will have a value of 1 if the antecedent and consequence parts of a rule are independent. The higher this value, the more likely that the existence of antecedent and consequence together in a transaction is not just a random occurrence, but because of some relationship between them. Therefore, lift provides an important and relevant measure related to the preference of a user action in a particular context.

$$Lift\,(X \Rightarrow Y) = \frac{Confidence\,(X \Rightarrow Y)}{Expected\,Confidence\,(X \Rightarrow Y)} \tag{3.6}$$

where

$$Expected\,Confidence\,(X \Rightarrow Y) = \frac{number\,transaction\,containing\,Y}{total\,number\,of\,transactions} \tag{3.7}$$

The combination of these values (i.e., consistency, precision, recall, confidence, support, and lift) provide an indicator as to the overall *quality* of association rules, and as higher quality rules provide a more accurate and relevant indicator as to the preferred behaviour of users, context types in these rules will be given more weight. Figure 3.20 illustrates the sequence of functions of Utility Assignment.

Relevance knowledge inferred during the previous Information Selection stage is used to determine the consistency of relevant context sets. During Information Selection, generated rules are compared and relevance is determined based on matches (and mismatches) between context and solution types. As part of this process, a count of the number of times a particular context set is evaluated as being relevant and not relevant is also recorded. By subtracting the latter value from the former, a consistency value for each context set is determined. This value represents how often or consistent a

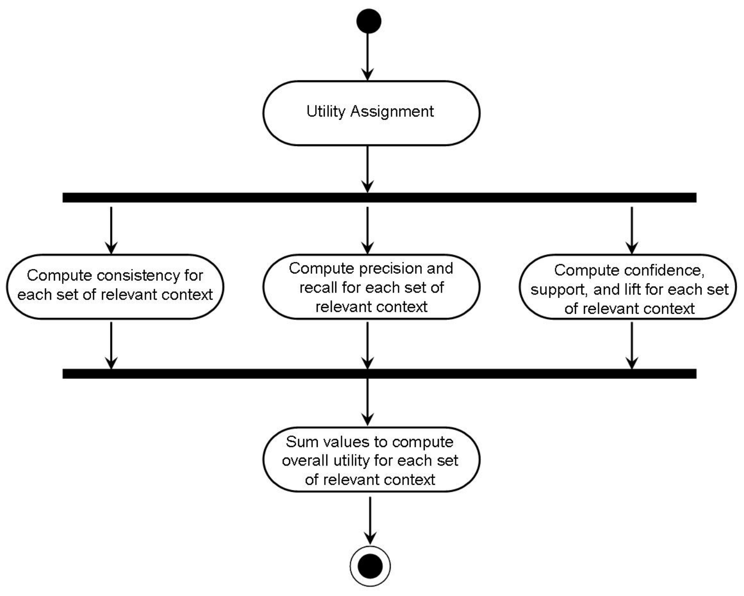

Utility Assignment
Compute consistency for each set of relevant context
Compute precision and recall for each set of relevant context
Compute confidence, support, and lift for each set of relevant context
Sum values to compute overall utility for each set of relevant context

Figure 3.20: Utility Assignment

given set of context types results in an accurate behaviour recommendation. Context sets that are more consistent, will therefore provide more accurate knowledge regarding preferred user actions, and consequently will be weighted more highly than less consistent ones.

The second operation incorporates *precision* and *recall* measures into context utility. *Precision* and *recall* are two common and widely used measures for evaluating the relevance of results in Information Retrieval and statistical classification [36]. Precision is a measure of the proportion of items retrieved that are relevant, while recall is the proportion of available relevant items that have been retrieved. These measures assume a ground truth notion of relevancy: every item is known to be either relevant or non-relevant to a particular recommendation decision for a given user tasks and context. Precision and recall provide useful measures of association rules as they highlight rules that contain a large number of relevant decision making information. Rules with high precision and high recall are therefore more likely to result in accurate recommendations and consequently context sets and the types they include that are contained in these rules are weighted higher based on the knowledge that they are included in rules that lead to accurate recommendations.

The same association rule set used during Information Selection is used during Utility Assignment to determine precision and recall values for relevant context sets. Precision and recall are calculated for each rule using the formulas 3.8 and 3.9 outlined. Figure 3.21 illustrates an example of how precision and recall is calculated for each rule. Rule 1 contains *two* context types that are relevant as

Relevant Types: time, weather, company

1. day=weekday time=afternoon weather=sun → solution=Roma } precision = 2/3, recall = 2/3
2. time=evening weather=rain company=friends → solution=Gallaghers } precision = 1, recall = 1
3. day=weekend time=evening company=alone → solution=Roma } precision = 2/3, recall = 2/3

Figure 3.21: Determining Precision and Recall for Association Rules and Context Types

evaluated during Information Selection and therefore has a *relevant items* $\cap$ *items retrieved* value of 2 (i.e., Rule 1 contains *time* and *weather* context types that are relevant types). As three relevant types are retrieved (i.e., time, weather, and context), the *relevant items* value is 3. Therefore the precision value for Rule 1 is $2/3 = 0.67$ (rounded). The recall value for Rule 1 evaluates as being the same as its precision value (i.e., 0.67) as the *items retrieved* value and the *relevant items* value is the same (i.e., *items retrieved* is 3 as Rule 1 contains a total of three context types). Precision and recall values for context types are then assigned these evaluated values. Each rule, and subsequently each context type in a rule are selected in turn and the precision and recall values of the rule is assigned to each context set in that rule. Re-using the example illustrated in Figure 3.21, each context set evaluated as relevant in Information Selection will be assigned the precision and recall value of the rule they are contained in. Consequently, in rule 1, relevant context type's time and weather will be assigned a precision value of 0.66 and a same recall value.

$$Precision = \frac{\mid relevant\,items \;\cap\; items\,retrieved \mid}{\mid relevant\,items \mid} \tag{3.8}$$

$$Recall = \frac{\mid relevant\,items \;\cap\; items\,retrieved \mid}{\mid items\,retrieved \mid} \tag{3.9}$$

The next stage of Utility Assignment incorporates confidence, support, and lift measures into context utility. Similar to precision and recall, rules with high confidence, high support, or high lift are more accurate and recurring. Consequently, utility assignment will favour and assign more weight to context contained in rules with higher confidence and support as there are more likely to result in accurate recommendations that match the user's preferences.

The technique for assigning confidence, support, and lift values to relevant context sets is similar to that for assigning precision and recall values. That is, the set of relevant context sets in each rule are assigned a confidence, support, and lift value of the rule they are contained in[10]. For example, as illustrated in Figure 3.22, relevant context time and weather in Rule 1 will be assigned a confidence value of 0.8, a support value of 0.5, and a lift value of 0.6.

Relevant Types: time, weather, company

1. day=weekday time=afternoon weather=sun → solution=Roma	} confidence 0.8 support 0.5 lift 0.6
2. time=evening weather=rain company=friends → solution=Gallaghers	} confidence 0.4 support 0.5 lift 0.2
3. day=weekend time=evening company=alone → solution=Roma	} confidence 0.8 support 0.2 lift 0.6

Figure 3.22: Determining Confidence and Support for Context Types

Figure 3.23 summarises the procedure for assigning precision, recall, confidence, support, and lift values to relevant context sets - each rule in the rule set is selected in turn, the set of relevant context types are selected and assigned a precision, recall, confidence, support, and lift value corresponding to the rule value, and the overall value is calculated by averaging the total value for the rule set.

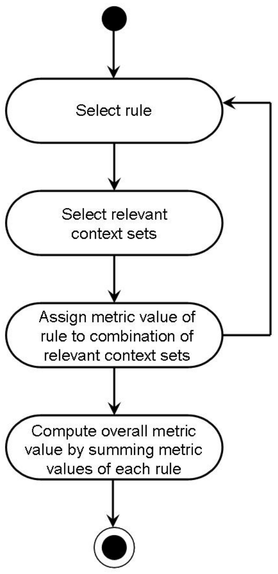

Select rule
Select relevant context sets
Assign metric value of rule to combination of relevant context sets
Compute overall metric value by summing metric values of each rule

Figure 3.23: Determining Precision, Recall, Confidence, Support, and Lift

The set of indicators for association rule accuracy, consistency, and relevance (i.e., consistency,

[10]Confidence, support, and lift values for association rules are calculated during the association discovery process described in Section 3.2.4 using formulas 3.4, 3.5, and 3.6 respectively.

precision, recall, confidence, support, and lift) are subsequently combined to determine a utility value to assign to each relevant context sets. The following algorithm is used to determine the appropriate utility for a context set for a particular association rule:

$$utility\,(context\,set/rule) = consistency \times (precision \times recall) \times (confidence \times support \times lift) \quad (3.10)$$

The overall utility for a context set is:

$$utility\,(context\,set) = \sum_{n=1}^{n} utility\,/\,n \quad (3.11)$$

where n is the number of association rules in the rule set that that context set is contained in. The Utility Assignment algorithm is designed to combine accuracy, consistency, and relevance measures in a manner such that more weight is assigned to context that are more likely to result to recommendations that match the user's preferences.

Each of the metrics used in the utility algorithm measures a quality property (i.e., accuracy, consistency, or relevance) of a particular context-action association rule as a proportion of all actions taken by a user in the past. The values are combined into a single measure for each relevant context set in the rule to represent how significant a context set is as an indicator of the preferred actions of a user in a particular context. As each measure represents an independent proportion of the overall utility, values are combined to give an overall measure using the conditional probability $\cap$ (i.e., *and*) property[11].

In addition to the described utility assignment algorithm, Utility Assignment also looks to incorporate utility knowledge learnt and stored in past cases. At present, utility values associated with corresponding context sets in user model cases of the same problem type are acquired and averaged. These utility values are compared with the evaluated utility of relevant context sets for the current problem. If the difference between these utility values is above a pre-defined threshold value, then the inferred utility is adjusted to this threshold value to closer match the utility value of past user problems of the same type. Future work will investigate techniques that make greater use of utility values stored in the user model. In particular, a process for mining utility values associated with different context types and sets and incorporating knowledge inferred from this into the computation of context utility is planned.

[11]i.e., P(A∩B) = P(A).P(B) [138].

3.2.8 Recommendation Generation and Ranking

The final stage of the recommendation process is concerned with generating and ranking candidate behaviours that best accommodate the preferences of the user and the context of the user problem. Outputs from the previous implicit preference determination, information selection, and utility assignment stages are used as inputs to generate and rank the set of candidate behaviours (Figure 3.24).

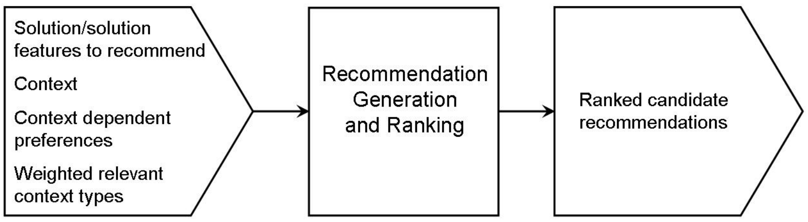

Solution/solution features to recommend
Context
Context dependent preferences
Weighted relevant context types
Recommendation Generation and Ranking
Ranked candidate recommendations

Figure 3.24: Recommendation Generation and Ranking Inputs and Outputs

Figure 3.25 illustrates the process of generating and ranking candidate behaviour recommendations. The first step determines the set of possible candidate behaviour from context-dependent preferences discovered from Implicit Preference Determination.

The design supports any number of rules for solution or solution features to be ranked depending on the requirements of the application (e.g., *three* rules, each containing competing recommendation solutions will be ranked if an application is designed to present *three* possible recommendations to its users). The configuration of this value is described in Section 4.3.

As outlined in Section 3.2.1.1, two recommendations strategies are included in the personalisation approach proposed in this thesis. As the *original* strategy does not include Information Selection or Utility Assignment functionality, the ranking of candidate behaviours differs in each strategy. In the original strategy, each context type is considered to be equally important. That is, all context types are considered relevant and have the same utility value. In this strategy, candidate behaviours are ranked according to their *applicability* and *strength.* Firstly, applicability considers how relevant a given user preference rule is for the current context. Applicability is measured by counting the number of context values in a preference rule that matches the context of the current problem. Therefore, a rule that matches three context values is more applicable than a rule that matches two and the behaviour association and the most applicable rule is recommended to the user. In addition, context values in a rule that do not match the current context are considered to negatively impact on the applicability of a rule. For example a rule containing three context types, two of which match the

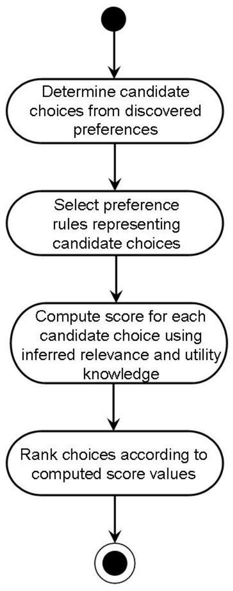
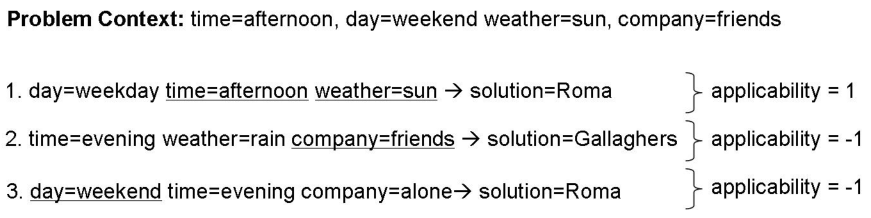

Determine candidate choices from discovered preferences
Select preference rules representing candidate choices
Compute score for each candidate choice using inferred relevance and utility knowledge
Rank choices according to computed score values

Figure 3.25: Recommendation Generation and Ranking

current context, is less applicable than a rule which contains two context types both of which match the current context. Figure 3.26 provides an example of how rule applicability is measured. As rule one matches two of the problem context values, but mismatches one, its applicability is 1. If rules are evaluated to be equally applicable, the confidence and support measures are used to further rank rules, with those with higher confidence and support values favoured. If rules are still equally matched then the time stamp of the case from which the rule is inferred is used and rules associated with more recent cases will be ordered higher. The view is that a rule is more likely to be accurate given it was preferred by the user more recently.



Problem Context: time=afternoon, day=weekend weather=sun, company=friends

1. day=weekday time=afternoon weather=sun → solution=Roma } applicability = 1
2. time=evening weather=rain company=friends → solution=Gallaghers } applicability = -1
3. day=weekend time=evening company=alone→ solution=Roma } applicability = -1

Figure 3.26: Rule Applicability Example

The second (i.e., *extended*) recommendation strategy extends on this first strategy to support the variance of context relevance and utility and adapts their values according to the user problem and context. The *multi-attribute utility theory* (MAUT) technique [183] is adopted to score candidate behaviour choices using inferred knowledge about context relevance and utility. MAUT is the technique currently used in Hermes to support multi-attribute decision making [65]. However, unlike the initial design which requires explicit user input, the application of MAUT here is adapted so that users are not required to explicitly specify relevant decision making dimensions and utility. Instead, this information and its utility are dynamically determined at Information Selection and Utility Assignment and used to rank candidate behaviour choices. Similar work has been conducted by Zhang and Pu [198] and later by Reilly et al. [155] whereby weights associated with decision making dimensions or attributes are dynamically updated. However, their approach relies on an interactive process between the user and the application and does not dynamically modify the set of decision making dimensions or attributes for different problems or context.

Briefly, MAUT is a utility theory decision making technique for evaluating candidate choices using information about user interest in various dimensions of each candidate choice. It provides a technique that enables the worth of a set of candidate choices to be evaluated using the values of properties or dimensions associated with each choice. The set of relevant dimensions and weights can also be dynamically adjusted for different problems and context without the need to change the underlying approach. With MAUT, the overall evaluation $v(x)$ of a particular behaviour choice x is defined as the weighted addition of its evaluation with respect to its relevant value dimensions. For example, a restaurant can be evaluated on dimensions such as cuisine, cost, service, food quality and any other environmental context values. The ranking technique for the personalisation approach proposed is designed to rank candidate choices along context value dimensions, which supports the evaluation of the preferred choice of a user in a given context. The choice evaluation is defined by the overall value function taken from [183]:

$$v(x) = \sum_{i=1}^{n} w_i v_i(x) \tag{3.12}$$

where $v_i(x)$ is the evaluation of the choice on the i-th value dimension d_i, and w_i is the weight (or relative importance of a dimension) determining the influence of the i-th value dimension on the overall utility value, n is the number of different value dimensions and $\sum_{i=1}^{n} w_i = 1$, meaning that the sum of the weights for the different value dimensions equals 1. As boolean matching is used as part of this personalisation approach, the value of an dimensions is typically 1 or 0. However, variants of values are also possible when relevant context values contain more than one type. For example, if a relevant context combination contained two types and a preference rule contained one of those values, then an attribute value of 0.5 is assigned.

MAUT also supports the separation of each value dimension d_i into a set of attributes composing the dimension. For example, the service dimension of restaurant can be calculated by considering attributes such as waiting time, speed of waiting staff, politeness of waiting staff, and restaurant atmosphere. The following formula is used to evaluate attributes of a dimension [183]:

$$v_i(x) = \sum_{a \in A_i} w_{ai} v_{ai}(l(a)) \tag{3.13}$$

where A_i is the set of all attributes relevant for d_i and $v_{ai}(l(a))$ is the evaluation of the actual level $l(a)$ of attribute a on d_i. w_{ai} is the weight determining the influence of the attribute a on value dimension d_i (i.e., the relative importance of an attribute for a given dimension). As with evaluating dimensions, the sum of the weights of attributes also sum to 1. This use of MAUT to evaluate attributes is not necessary for ranking candidate choices as the design does not currently support the separation of context types into individual attributes.

Figure 3.27 illustrates the algorithm designed for ranking candidate behaviour choices. The relevant context sets and their utility as inferred from Information Selection and Utility Assignment are acquired. The set of utility values are then normalised so their summed value equals 1 as required by MAUT. Each preference rule is then evaluated in turn to determine the context sets they contain and they are scored accordingly using the normalised weight values using a MAUT approach. This process is repeated for the set of preference rules and when complete these rules are ranked in score order. The recommendation represented by the highest ranked (or set of highest ranked) rules are presented to the user. This algorithm provides a novel application of MAUT as the set of candidate choices to be ranked, relevant decision making dimensions, and weight values associated with each dimensions are dynamically determined.

Figure 3.28 illustrates an example of how candidate preference rules are ranked. Relevant context sets - time and weather, and company - and their utility values are acquired. Preference rules are subsequently evaluated and scored according to the relevant context sets they contain. Given that the calculated normalised utility values for time and weather, and company are 0.66 and 0.34 respectively, preference rules are scored accordingly. Rule 1 and Rule 2 both contain context sets that are applicable (i.e., match the context of the current problem) and are scored accordingly. Note that if Rule 1 had

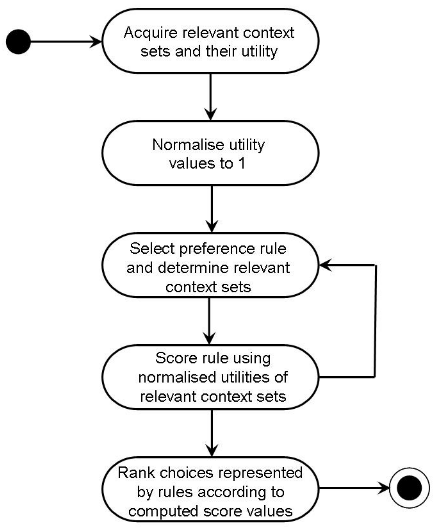
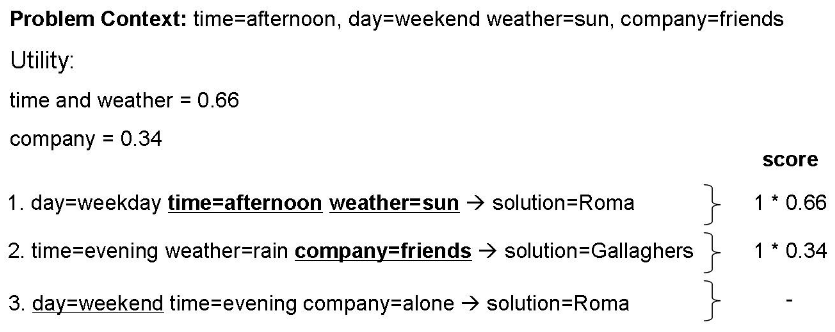

Acquire relevant context sets and their utility
Normalise utility values to 1
Select preference rule and determine relevant context sets
Score rule using normalised utilities of relevant context sets
Rank choices represented by rules according to computed score values

Figure 3.27: Recommendation Ranking Algorithm

not contained either an applicable time or weather value, then the attribute value assigned would be 0.5 instead of 1 (as it matches half of the relevant context set). Rule 3 does not match any of the problem context values and therefore has a default score of 0. The higher the score, the higher ranked a recommendation.



Problem Context: time=afternoon, day=weekend weather=sun, company=friends

Utility:

time and weather = 0.66

company = 0.34

	score
1. day=weekday **time=afternoon** **weather=sun** → solution=Roma	1 * 0.66
2. time=evening weather=rain **company=friends** → solution=Gallaghers	1 * 0.34
3. day=weekend time=evening company=alone → solution=Roma	-

Figure 3.28: Recommendation Ranking Example

3.2.8.1 Learning from Implicit User Feedback

Learning from implicit user feedback was added as part of the *extended* recommendation strategy (Section 3.2.1.1) to facilitate the personalisation approach to learn when incorrect recommendations are made (i.e., when the user chooses a recommendation that is not the most highly ranked) and therefore improve future recommendations. Learning is achieved by further adjusting the utility of context types so that the recommendation selected by the user is ranked highest. It is important to note that this form of application learning is only executed if the user does not select the behaviour recommended by the application and learning is a retroactive process (often referred to as lazy methods in machine learning [9, 58]). That is, the learning is too late to the applied to the current problem, but is recorded with the aim of aiding in future recommendation decisions.

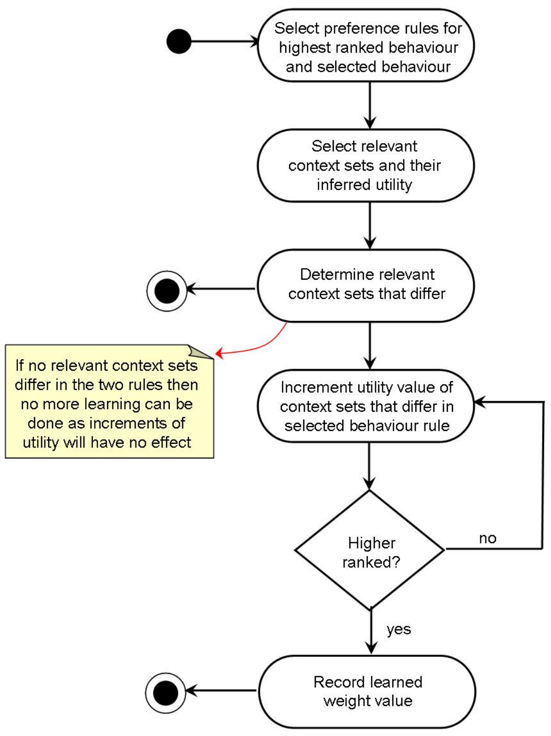

Select preference rules for highest ranked behaviour and selected behaviour
Select relevant context sets and their inferred utility
Determine relevant context sets that differ
If no relevant context sets differ in the two rules then no more learning can be done as increments of utility will have no effect
Increment utility value of context sets that differ in selected behaviour rule
Higher ranked?
no
yes
Record learned weight value

Figure 3.29: Implicit User Feedback Learning Algorithm

Figure 3.29 illustrates the algorithm for learning from user feedback. The aim of the algorithm is to adjust the weights of relevant context sets so that the behaviour selected by the user is ranked highest. The preference rule representing the highest ranked behaviour recommendation that was presented to the user (*recommended rule*) and the preference rule representing the behaviour selected by the user are retrieved (*selected rule*) are selected. The relevant context sets used for the recommendation are also retrieved. Differences in the relevant context sets in each rule are identified. If there is no difference, no learning can be done as adjusting the weights of context sets contained in both rules will have no effect. If different context sets are identified, then the utility of context sets contained in the selected rule and not the recommended rule are iteratively incremented by a given value until the selected rule is ranked higher than the recommended rule. If any other preference rules are ranked higher than the selected rule as a result of incrementing context set utility, then this process is repeated until the selected rule is ranked the highest or no more learning is possible. If no learning is required, then the weights determined at Utility Assignment are stored in the *weights* section of a new user model case. However, if learning was necessary, the set of adjusted weight values are persisted. This knowledge is subsequently incorporated into inferred context utility in for future recommendations as described in Section 3.2.7.

3.3 Chapter Summary

This chapter described the design of the personalisation approach proposed in this thesis. The approach is organised as a multi-stage recommendation process, which provides a set of techniques and algorithms designed to support personalised recommendations in context-aware recommendations. The chapter presented: the design methodology used for the development of this process; an overview of the Hermes project, the umbrella project under which the personalisation approach proposed in this thesis was developed; and a description of the multi-stage recommendation process including a detailed description of the algorithms and techniques designed and used at each stage.

Design decisions during the development of the approach were made to ensure that the combination of techniques and algorithms address the requirements outlined in Section 1.2. Although no single technique or algorithm at each stage fully addresses any single requirement, the combination of all techniques and algorithms are designed to address the full set of requirements. Implicit Preference Determination is designed to provide techniques that support the automatic determination of preferred user actions for a particular problem and context and to adapt preferences according to different context combinations. Similarly, Information Selection and Utility Assignment are designed to facilitate the tailoring of context relevance and relative importance of environmental context for different user problems and context. The combination of these techniques and algorithms therefore support applications in determining user preferences and adapting the relevance and utility of decision making information for different problems and context (R1). The user model structure was designed to store information about recommendation problems, context, and user choices. Such a design facilitates the identification of patterns about the preferred action of users for different problems as well as information about the context in which these patterns apply. The decision to incorporate association discovery was taken to facilitate the implicit discovery of relationships between context and user choices, eliminating the need for relevant information relationships to be explicitly defined by developers (R4). As association discovery is not a specific algorithm, relevant relationships between multiple different context types and user actions can be discovered, supporting applications to identify context-dependent preferences for different combinations of context and different user problems. Also, the focus on data mining techniques on efficient pattern discovery [131, 137] ensures that the personalisation approach and its use of association discovery is scalable and can be applied identify patterns in large sets of multi-dimensional data (R3). Techniques for filtering user cases and generated context-action associations were integrated to ensure that the most relevant knowledge about user preferences is discovered. The decision to execute association discovery at run time, when required, ensures that decision making information is inferred based on up-to-date user interaction and context relationship information stored in a user model and is tailored for specific recommendation problems. This coupled with the addition of new cases to the user model as the application is used and the learning from implicit user feedback about recommendation selections facilitates applications to accommodate changes in user preferences and new context relationships without the need for explicit user input (R2 and R5).

The next chapter describes the implementation detail of the personalisation approach, including the set of techniques and algorithms used at each stage of the multi-stage recommendation process, described in this chapter.

Chapter 4

Implementation

The previous chapter described the design of a multi-stage personalisation approach and discussed how its techniques and algorithms provide personalisation support to context-aware applications. Specifically, the multi-stage recommendation process was described in detail along with a description of how techniques and algorithms designed for each stage combine to address the requirements and research question outlined in Section 1.2.

This chapter describes in detail the straightforward Java implementation of the techniques and algorithms. The chapter begins with a high-level overview of the set of classes that compose the personalisation approach. This is followed by a description of the relevant attributes, functions, behaviour, and interactions for each stage of the personalisation process - user modelling, associating discovery, implicit preference determination, information selection, utility assignment, and recommendation generation and ranking. The set of classes involved and the sequence of operations that are executed at each stage are presented. Configuration details and implementation code for significant operations are also illustrated and discussed.

4.1 Architecture Overview

The implementation of personalisation functionality described in this thesis is divided into several high-level functional groups, each with a *Manager* class responsible for coordinating behaviour associated with specific personalisation functionality. These classes and their interactions provide the structure and behaviour that support the various personalisation functions developed to address the requirements outlined in Section 1.2.

This section describes the implementation of personalisation functionality for context-aware applications at a high-level, outlining the core functional groups and their responsibilities. Figure 4.1 illustrates the architecture of the personalisation approach and the main groups of functionality at a high-level. `Personalisation` provides the entry point for applications into personalisation functionality and acts as a mediator between different personalisation functions designed for the multi-stage recommendation process. At the same level as `Personalisation` is `Context Management`, which functions as a coordinating interface that facilitates the request and acquisition of context information from the *Hermes* application framework. Below this level is the set of functionality that represent the techniques and algorithms designed for the specific personalisation functions that make up the personalisation approach described in this thesis. This level also includes the user model which stores the set of past user actions. Updates to the user model are made through `Personalisation` on completion of a recommendation problem, while other functions can access the user model to retrieve data used in specific personalisation tasks. The lowest level contains the set of functionality responsible for the execution and management of specific operations that are used by higher level groups. This consists of functions related to association discovery, user model cases, and association rules.

Each core functional group contains a *Manager* class, which acts as a facade, mediates with other *Manager* classes, and supports the exchange of information necessary for specific personalisation functionality[1] e.g., the `MiningManager` is responsible for data mining functionality including coordinating classes that provide association discovery functionality. The set of *Manager* classes and their responsibilities are:

- `ContextManager` - provides and coordinates operations associated with the management of context information. This class, and the set of classes it coordinates, are implemented as part of the Hermes application framework and provide other components and classes with access to current and historical context information.

- `PersonalisationManager` - provides the entry point for personalisation functionality and also coordinates operations between different stages of the multi-stage recommendation process. Specifically, this class mediates and coordinates interactions between other personalisation related *Manager* classes. It also interacts with the `ContextManager` to acquire environmental context information and is responsible for communication with applications (e.g., providing recommendations to applications).

- `UserModelManager` - manages user model functionality such as operations for accessing and updating the user model and persisting user model cases to disk.

[1]Each functional group also contains a `Utility` class responsible for the implementation of low level utility operations.

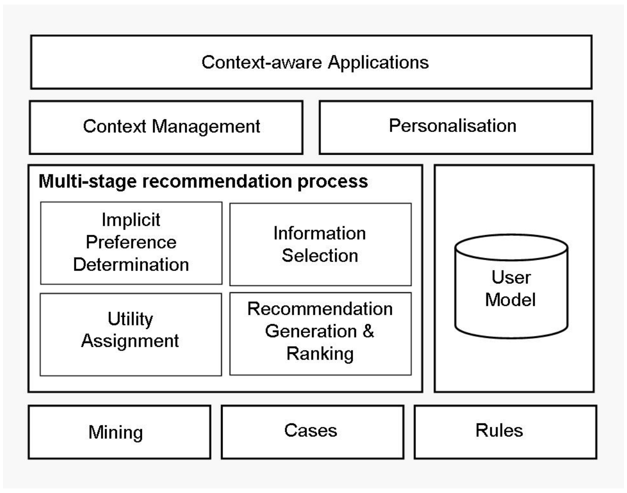

Context-aware Applications
Context Management
Personalisation
Multi-stage recommendation process
Implicit Preference Determination
Information Selection
Utility Assignment
Recommendation Generation & Ranking
User Model
Mining
Cases
Rules

Figure 4.1: High-level Architecture of Personalisation Functionality

- `CaseManager` - provides and coordinates operations associated with individual user model cases.
- `MiningManager` - coordinates all data mining operations such as the execution of association discovery and the management of discovered association rules.
- `RuleManager` - provides and coordinates operations associated with individual association rules.
- `PreferenceManager` - provides and coordinates operations associated with Implicit Preference Determination.
- `RelevanceManager` - provides and coordinates operations associated with Information Selection.
- `UtilityManager` - provides and coordinates operations associated with Utility Assignment.
- `RankingManager` - provides and coordinates operations associated with Recommendation Generation and Ranking.

Figure 4.2 illustrates the sequence of interactions at a high level that execute between the *Manager* classes when generating and evaluating context-aware recommendations[2]. As illustrated in this figure, `PersonalisationManager` is responsible for mediating and coordinating personalisation related functionality. Figure 4.3 illustrates this `PersonalisationManager` object. The multi-stage process begins when `PersonalisationManager` receives either a recommendation request from the user via a concrete instance of `Application` (e.g., restaurant recommender application) or a context event signalling the need for a new recommendation. `PersonalisationManager` interacts with other *Manage*r classes and

[2]As this figure shows the sequence of operations at a high level, some classes have been omitted from the illustration.

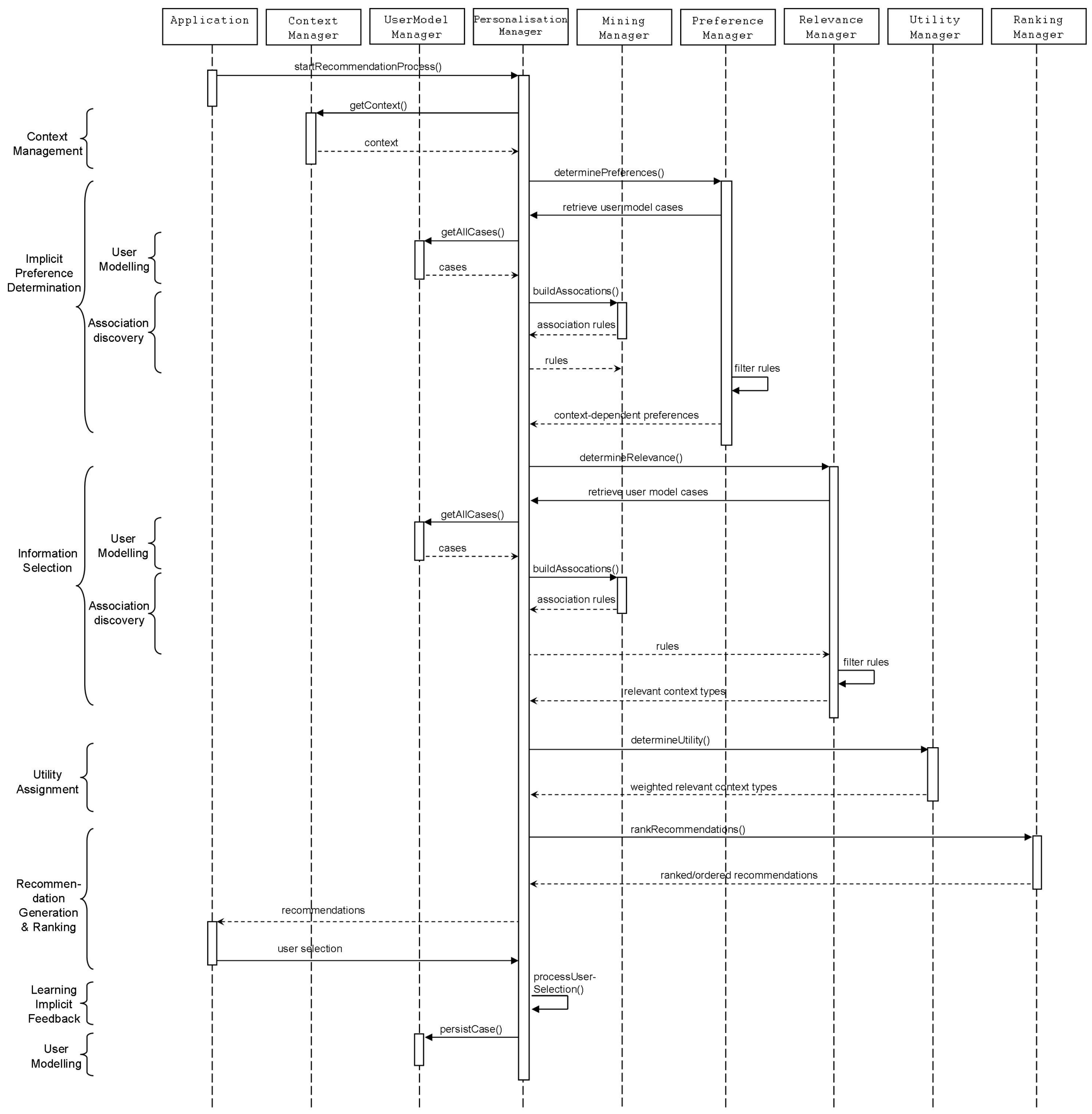


Figure 4.2: High-level Sequence of Operations

executes the sequence of operations required for providing a personalised, context-aware recommendation. Listing 4.1 illustrates a code excerpt from the `startRecommendationProcess()` method in `PersonalisationManager` showing the set of operations that execute for the various stages of the personalisation approach described in this thesis. The invocation of this method signals the start of the recommendation evaluation process and includes the execution of functions for Implicit Preference Determination, Information Selection, Utility Assignment, and Recommendation Generation and Ranking. Each function is implemented corresponding to the design described in the previous chapter.

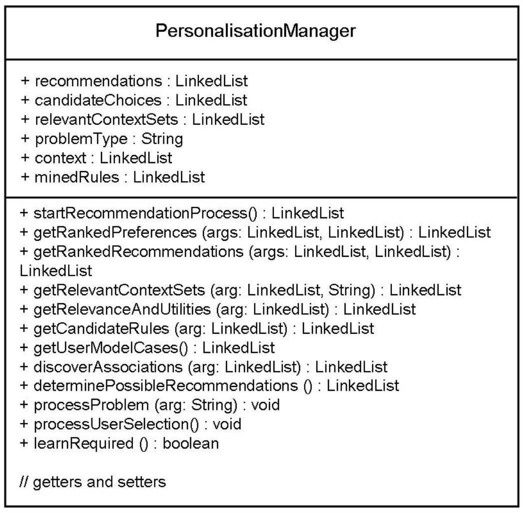
Listing 4.1: The `startRecommendationProcess()` method

```
// Implicit Preference Determination
LinkedList candidateRules = PreferenceManager.determinePreferences();
// Information Selection
LinkedList contextSets = RelevanceManager.determineRelevantContext();
// Utility Assignment
LinkedList weightedSets = UtilityManager.determineUtility();
// Recommendation Generation and Ranking
LinkedList rankedRecommendations = RankingManager.rankRecommendations(
    candidateRules, weightedSets);

// present recommendations to user
```

Later sections of this chapter will expand on these operations including a description of the set of classes and sequence of operations associated with each call. Specifically, the set of *Manager* classes, other classes that are implemented, their interactions, and the implementation details of their operations and algorithms are described.

4.2 Context Management and Hermes Integration

As described in the previous chapter, the Hermes framework encapsulates functionality for the acquisition of context information from sensors and the modelling of context in an internally understandable representation. The personalisation approach described in this thesis acquires context information from the Hermes *Context Management* layer via a `ContextManager` object. `ContextManager` coordinates the transfer of context information, which is either pulled using explicit context request calls



```
PersonalisationManager

+ recommendations : LinkedList
+ candidateChoices : LinkedList
+ relevantContextSets : LinkedList
+ problemType : String
+ context : LinkedList
+ minedRules : LinkedList

+ startRecommendationProcess() : LinkedList
+ getRankedPreferences (args: LinkedList, LinkedList) : LinkedList
+ getRankedRecommendations (args: LinkedList, LinkedList) :
LinkedList
+ getRelevantContextSets (arg: LinkedList, String) : LinkedList
+ getRelevanceAndUtilities (arg: LinkedList) : LinkedList
+ getCandidateRules (arg: LinkedList) : LinkedList
+ getUserModelCases() : LinkedList
+ discoverAssociations (arg: LinkedList) : LinkedList
+ determinePossibleRecommendations () : LinkedList
+ processProblem (arg: String) : void
+ processUserSelection() : void
+ learnRequired () : boolean

// getters and setters
```

Figure 4.3: `PersonalisationManager` Class

or pushed by registering interest in certain context types and notified by observers. `ContextManager` also maintains a definition of the set of context types and their possible values that are supported by the application. The process of how Hermes transparently acquires context from sensor devices of behalf on applications and provides applications with access to acquired context information is briefly described here.

As outlined in Section 3.2, Hermes contains *Service Discovery* and *Communication* components that provide applications with operations for discovering and communicating with both remote and local devices to support the exchange of context information. Listing 4.2 shows an example of a basic Hermes communications configuration, which controls the service discovery and communication components of the Hermes framework.

Listing 4.2: Hermes Basic Communications Configuration

```
# COMMUNICATION CONFIGURATION PROPERTIES
hermes.comms.thread.incoming=2
hermes.comms.heartbeattimeout=30000
hermes.comms.broadcast.port=446
hermes.comms.broadcast.tick=10000
hermes.comms.listening.timeout=10000
hermes.comms.listening.tick=5000
hermes.comms.receive.timeout=5000
hermes.comms.receive.tick=5000
```

Communication with devices is supported by listening and exchanging information on a specified well known port. Context exchange with remote devices requires the broadcast and direct exchange of a series of messages - containing information about device addresses, communication ports, and context request and response information - before context is successfully transferred from a remote source to the local device. A Connection between two devices is maintained until no messages are exchanged between devices for a specified period of time. An absence of information exchange means that devices are no longer in communication range or communication is complete, and resources used for communication can be reclaimed. Communication messages exchanged between devices are represented in XML. Hermes supports a first-in first-out message-handling thread pool which enables multiple messages to be processed at once.

The Hermes framework also needs to be informed of any local sources it should connect to. Listing 4.3 shows the addition of a location (GPS) source to the Hermes framework, which informs Hermes about the context source, the type of context information it provides, and how to communicate with it.

Listing 4.3: Adding a Local Context Source in Hermes

```
SerialAddress serialPort = new SerialAddress(commPort, baudRath);
// Class modelling sensor device
GPSReceiver gps = new GPSReceiver(serialPort);
// Get description of type of context provided by source
ContextServiceDescription[] sd = (ContextServiceDescription)gps.
    getContextServiceDescriptions();
// Discover the new local context source
Hermes.getCommunication.getServiceDiscovery().updateContextSourceList(gps,
    sd);
```

Once informed about context sources it should interact with, the Hermes framework can begin to acquire context information from these sources. `ContextManager` informs Hermes of the context information it is interested in using the `Hermes.setInterestInContextTypes()` command, which takes three parameters - a reference to an object that is listening for events notifying it of a change in a particular context value, the type of context to acquire, and the maximum frequency of context change notifications. Acquired context information is managed at the *Context Management* layer of

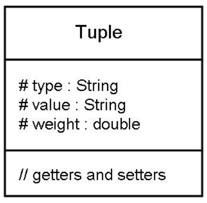


Figure 4.4: `Tuple` Class

Hermes, which provides the set of functionality for maintaining an accurate model of environmental context information. As described in the previous chapter, Hermes represents context as a hierarchical object-oriented (OO) model and raw context information received from the *Collaboration* layer as XML information is parsed and transformed into objects, which are stored as part of the OO context model. An object to XML mapping supports the transformation of XML message to objects and from objects to XML. `ContextManager` provides the set of operations that enable `PersonalisationManager` to transparently acquire and receive up to date context information stored in the Hermes *Context Container* model. Listing 4.4 illustrates an example of how context information is retrieved from the Hermes framework. `ContextManager` gets a handle on the system service that supplies location context. Location context information is acquired with the `getLocation()` method call. The implementation of this service for supplying specific context types is influenced by The Open Handset Alliance Android 1.0 application framework for mobile devices [51].

Listing 4.4: Retrieving Location Context from Hermes

```
// Ask Hermes for its location manager service (part of the context model)
LocationManager locManager = (LocationManager) Hermes.getSystemService(
    Context.LOCATION);
// Get the current location
Location loc = null;
try{
        loc = locManager.getLocation();
} catch (LocationException le) {
```

Context information retrieved from Hermes is stored in tuple form using the `Tuple` object as illustrated in Figure 4.4.

4.3 Personalisation Configuration

Various techniques and algorithms designed as part of the personalisation process may be configured before execution. Listing 4.5 illustrates a sample personalisation configuration file. Configuration values for association determination (Lines 2-5), filtering user model cases and association rules (Line 6-7) information selection (Line 8), utility assignment (Line 9), learning from implicit user feedback (Line 10), and recommendation generation and ranking (Line 11) are specified. The configuration file, specified using `java.util.properties`, allows static parameters to be persisted and loaded at run time, enabling the behaviour of personalisation operations to be customised without requiring source code modification.

Listing 4.5: Personalisation Configuration File

```
1  # PERSONALISATION CONFIGURATION PROPERTIES
2  association.rulesToGenerate=500
3  association.minSupport=1
4  association.confidenceThreshold=1
5  association.supportThreshold=1
6  filtering.caseContextMatches=1
7  filtering.ruleContextMatches=1
8  relevance.rulesToCompare=50
9  utility.weightDifferenceThreshold=0.8
10 learning.learningRate=0.025
11 ranking.rulesToRecommend=3
```

The determination of values for *association, filtering,* and *relevance* properties represents a trade off between generating too much information that makes the retrieval of relevant knowledge difficult and not discovering enough information meaning that relevant and useful information is missed. For example, the *association.rulesToGenerate* property specifies the number of rules that should be discovered as part of Association Determination (Section 3.2.4). The discovery of a large number of rules makes identifying those rules that are relevant difficult. The computation time for processing data also increases in relation to the number of rules to be discovered. In contrast, generating fewer rules will decrease computation time but may mean that important associations between context and user actions are not discovered at all and therefore are not available for recommendation decisions. A similar trade off applies for the number of rules that are compared during Information Selection to determine relevance (i.e., *relevance.rulesToCompare* property). The result of having too many comparisons is additional computation overhead and the generation of too many context sets, which makes identifying the most accurate and relevant set of information difficult. In contrast, having fewer comparisons reduces computation effort but can result in relevant context being missed. A similar trade off applies to other properties related to *association* and *filtering*. It is important to note that the design of the personalisation approach to incorporate techniques for filtering user model cases and discovered rules is taken to ensure that a suitable number of relevant associations can be discovered, minimising the loss of relevant decision making knowledge and increasing the ease of identifying relevant knowledge.

As knowledge about information used in the personalisation approach is based on information about past user interactions, the decision to assign a particular value to a property is influenced by the properties of user model data being searched. For example, a larger user model is likely to have more associations between its data than one that contains less cases (provided user model cases are not repeated). Larger user models will therefore require the generation of a larger number of rules in order for the full set of possible associations between data to be discovered. Similarly, a user model is likely to have more associations than a model that supports a lower number of data types (e.g., context types). Therefore, a user model that supports a higher number of context types will require a higher number of associations to be generated to facilitate the discovery of all relationships between different context and user actions. The personalisation approach proposed in this thesis is therefore designed to support the configuration of properties (without source code modification) related to association and relevance depending on the requirements and properties of data in different applications and domains.

The utility of context inferred by Utility Assignment is designed so that it incorporates utility knowledge from past user decisions stored in the user model (Section 3.2.7). The *utility.weightDifferenceThreshold* specifies the point at which the inferred utility value should be adjusted (i.e., utility is adjusted when its inferred value is abnormally different from utilities used by the user in the past). A high threshold value therefore means that utility values are only adjusted in instances where inferred utilities are largely different from historic values. The risk of specifying a high threshold value is that inaccurate recommendations may result as utility values which should have been adjusted are not. In contrast, a low threshold value will result in utility values being altered even when unnecessary.

The *learning.learnRate* property specifies how much context utility values should be incremented by when learning from implicit feedback in user selections (i.e., when an incorrect recommendation is made). This property represents a learning rate value. Higher values lead to faster learning but the accuracy of learning suffers as utility values may be adjusted in excess of what is necessary.

Finally, the *ranking.rulesToRecommend* specifies the number of candidate recommendations to be presented to the user, and is specified depending on the requirements of the application. For example some applications may present the user with three possible recommendations, while others may only recommend one. This value therefore configures the execution of the approach to only rank a specified number of association rules to reflect the number of candidate recommendations to be presented to the user.

While the set of configuration information is currently configured by developers or users, future works will investigate techniques that will support the dynamic and autonomous adaptation of these values so that they are customised to different user problems, contexts, and preferences (Section 6.2.2).

4.4 User Modelling

The implementation of User Modelling provides the set of operations that manage (i.e., add, remove, access, and persist) the user model and the cases it contains. Figure 4.5 illustrates the main set of classes involved in User Modelling. `UserModelManager` contains the set of functionality for creating, storing, and accessing cases from a persistent user model structure stored on disk. The user model is persisted and accessed as a *comma separated value* (csv) file using `java.io`. Listing 4.6 illustrates a sample user model file with the three sample cases introduced in Figure 3.14. Each case contains values following the case structure described in Section 3.2.3.

Listing 4.6: A sample User Model csv file

```
restaurant ,afternoon ,weekday ,rain ,alone ,italian ,cheap ,two ,three ,Roma
    ,0 ,0.54 ,0.46 ,0
route ,night ,weekend ,sun ,alone ,shortest ,walk ,false ,grafton-westmoreland-o'
    connell-parnell ,0.4 ,0.2 ,0.4 ,0
restaurant ,evening ,weekend ,rain ,friends ,irish ,expensive ,two ,three ,Gallaghers
    ,0 ,0.54 ,0.46 ,0
```

`UserModelCase` objects represent user model cases in application code (Figure 4.5). Subclass instances of `UserModelCase` (e.g., `RestaurantCase` and `TravelCase`) are implemented for different problem types. The *problem type* and *solution* items contained in a user model case are stored as attributes in the `UserModelCase` object. The set of *environmental context* items in a case, which represents the set of context supported by the application, are also stored in `UserModelCase`. *Solution feature* items, which are associated with specific problem types, are stored in subclass instances of `UserModelCase`. *Weights* that represent the utility of context in recommendation decisions are stored as part of each context items `Tuple` object.

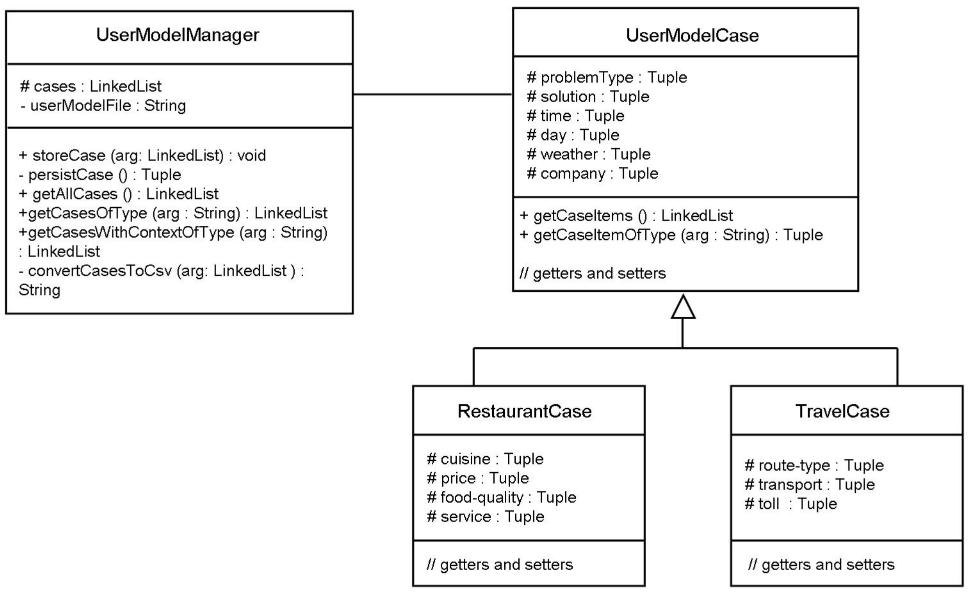


Figure 4.5: User Modelling Classes

Figure 4.6 illustrates the sequence of operations executed for persisting a user model case to memory. The operation is initiated when a user selects a given recommendation. The problem type, solution, and solution features associated with the selected recommendation are acquired along with the environmental context and context utilities of the problem. If necessary (i.e., if the user selects a recommendation that is not ranked highest) learning from relevance feedback is also executed. Values for the set of items stored in a user model case (i.e., problem type, solution id, environmental context values, and context weights) are transferred to `UserModelManager` as part of the `storeCase()` command. A new case is created (`createCase()`) and persisted to the user model csv file (`persistCase()`). Listing 4.7 shows the set of operations executed for the `persistCase()` method. The `convertCasesToCsv()` command converts the values of case attributes into the comma separated value representation of the user model file.

Listing 4.7: Persisting User Model Cases to Memory

```
try {
        BufferedWriter out = new BufferedWriter(new FileWriter(fileName,
            true));
        for (int i = 0; i < cases.size(); i++) {
                UserModelCase aCase = (UserModelCase) cases.get(i);
                String caseInCsv = convertStringListToCsv(aCase.getValues())
                    ;
                out.write("\n");
                out.write(caseInCsv);
        }
}
```

User model cases are retrieved from persistent memory using one of three methods in `UserModelManager`: `getAllCases()` returns all user model cases, `getCasesOfType()` returns cases with a specific problem type, and `getCasesWithContextOfType()` returns cases containing a particular context type. When retrieving user model cases from persistent memory, each case is individually read from memory using a `CSVReader` object and `readNext()` command. `getAllCases()` returns the entire set of user model cases, while `getCasesOfType()` and `getCasesWithContextOfType()` filter cases according to problem type. Listing 4.8 illustrates the filtering of cases by problem type by comparing the problem type of each case with the type requested by the application. Filtering cases by context is performed in a similar manner by comparing the context types stored in each case with what is required.

Listing 4.8: Filtering User Model Cases

```
String problemTypeWanted = "travel";
while ((nextLine = csvReader.readNext()) != null) {
        String problemType = nextLine[0];
        if(problemTypeWanted.equals(problemType)
                //retain case
        }
}
```

4.4.1 Extensibility

The personalisation approach can be extended to support new types of context and new types of recommendation problems by extending the `UserModelCase` object and implementing new subclasses. Specifically, new types of context can be supported by adding new attributes, representing new context types, to `UserModelCase`. New types of recommendation problems and their associated features can

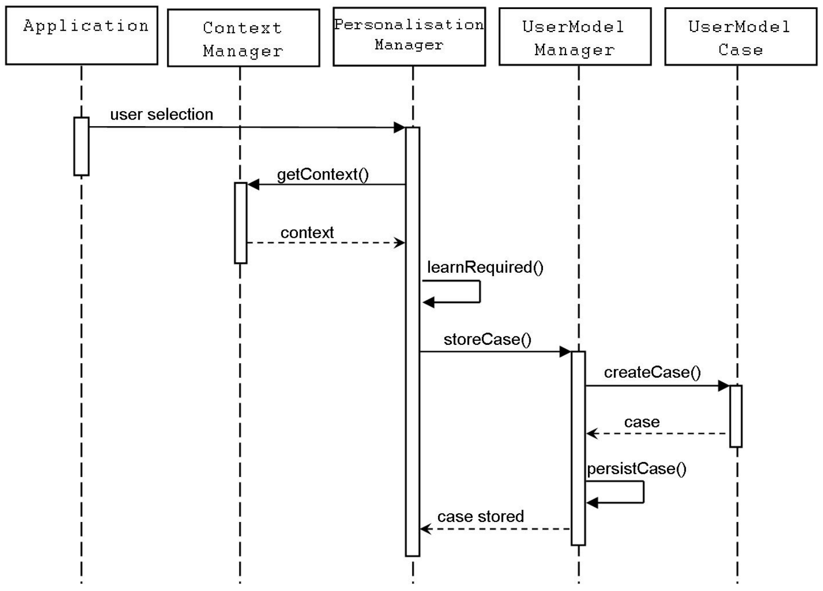


Figure 4.6: Storing a New User Model Case - Sequence of Operations

be supported by creating new subclass instances of `UserModelCase`.

4.5 Association Determination

Association Determination (Section 3.2.4) discovers associations between context types and solutions/ solution features that exist in user model cases. Figure 4.7 illustrates the set of classes involved with association determination. `MiningManager` is responsible for coordinating data mining behaviour and initiates association discovery. `Miner` provides the functionality for discovering association rules using `Apriori` and discovered rules are maintained by the application as a collection of `Rule` objects. The `Rule` object is designed to represent discovered associations between context and user actions. `Rule` contains attributes for premises, consequences and various metrics for evaluating the quality of the rule. Rule `premise` are designed to stored the set of context type-value tuples contained in the rule (e.g., time=afternoon), while `consequence` store tuples for user actions (e.g., solution=Roma). `RuleManager` provides functionality for managing `Rule` objects including operations for creating new rules and removing uninteresting and redundant rules.

Figure 4.8 illustrates the sequence of operations executed for Association Determination. The process begins when the `startAssociationDiscovery()` method is invoked on `MiningManager`, which receives a set of user model cases as input. When `Miner` is invoked, various properties of association dis-

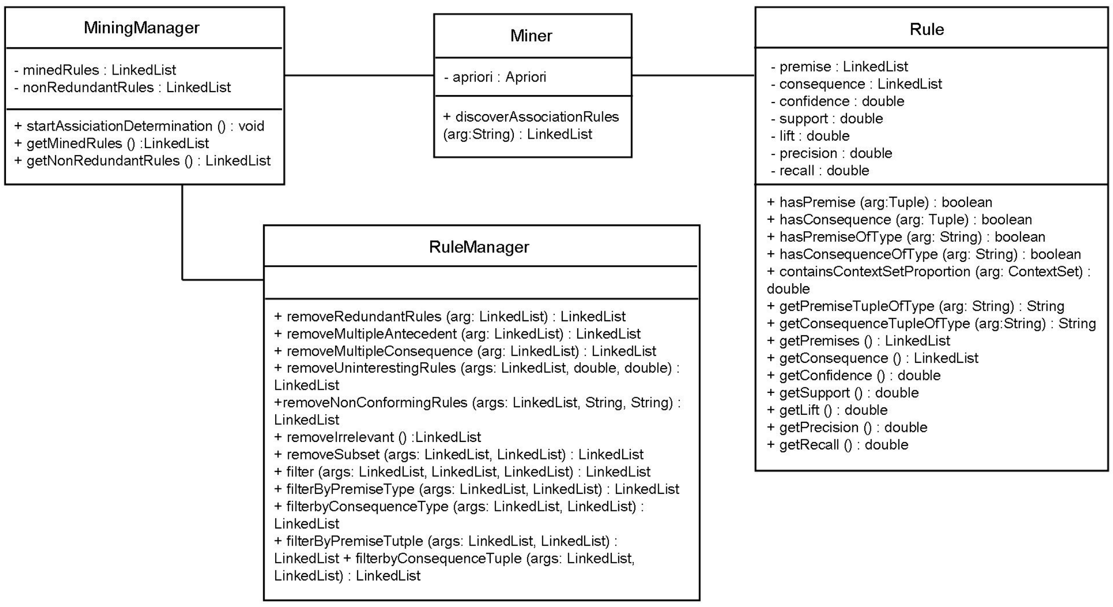

MiningManager
- minedRules : LinkedList
- nonRedundantRules : LinkedList
+ startAssiciationDetermination () : void
+ getMinedRules () :LinkedList
+ getNonRedundantRules () : LinkedList
Miner
- apriori : Apriori
+ discoverAssociationRules (arg:String) : LinkedList
Rule
- premise : LinkedList
- consequence : LinkedList
- confidence : double
- support : double
- lift : double
- precision : double
- recall : double
+ hasPremise (arg:Tuple) : boolean
+ hasConsequence (arg: Tuple) : boolean
+ hasPremiseOfType (arg: String) : boolean
+ hasConsequenceOfType (arg: String) : boolean
+ containsContextSetProportion (arg: ContextSet) : double
+ getPremiseTupleOfType (arg: String) : String
+ getConsequenceTupleOfType (arg:String) : String
+ getPremises () : LinkedList
+ getConsequence () : LinkedList
+ getConfidence () : double
+ getSupport () : double
+ getLift () : double
+ getPrecision () : double
+ getRecall () : double
RuleManager
+ removeRedundantRules (arg: LinkedList) : LinkedList
+ removeMultipleAntecedent (arg: LinkedList) : LinkedList
+ removeMultipleConsequence (arg: LinkedList) : LinkedList
+ removeUninterestingRules (args: LinkedList, double, double) : LinkedList
+removeNonConformingRules (args: LinkedList, String, String) : LinkedList
+ removeIrrelevant () :LinkedList
+ removeSubset (args: LinkedList, LinkedList) : LinkedList
+ filter (args: LinkedList, LinkedList, LinkedList) : LinkedList
+ filterByPremiseType (args: LinkedList, LinkedList) : LinkedList
+ filterbyConsequenceType (args: LinkedList, LinkedList) : LinkedList
+ filterByPremiseTutple (args: LinkedList, LinkedList) : LinkedList + filterbyConsequenceTuple (args: LinkedList, LinkedList) : LinkedList

Figure 4.7: Association Determination Classes

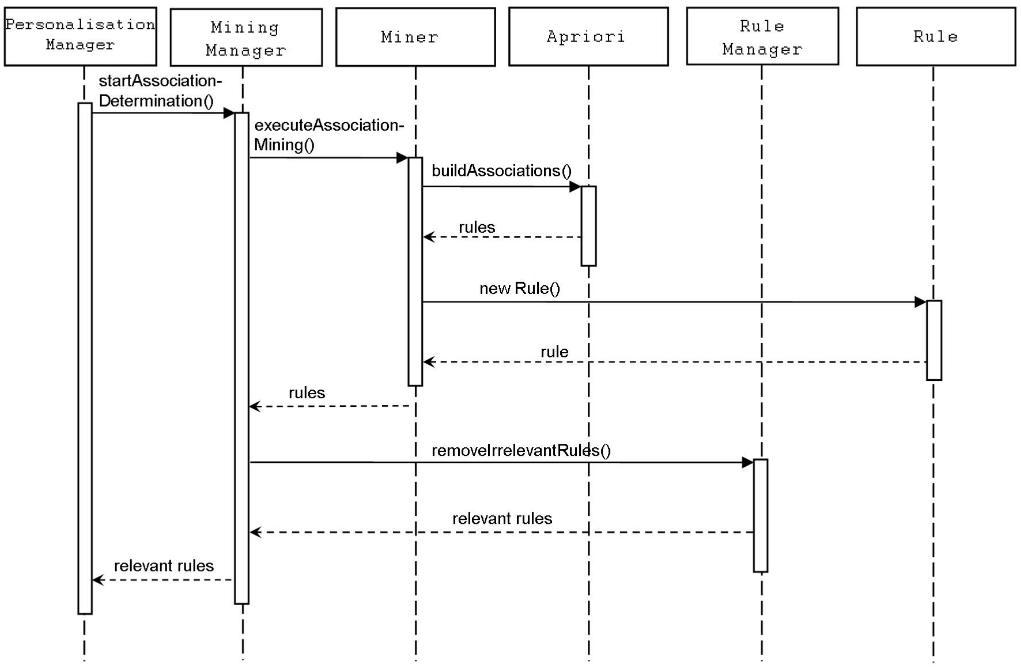


Figure 4.8: Association Determination - Sequence of Operations

covery such as number of rules to discover and minimum support of rules are read from the personalisation configuration file. Association rules are then discovered using the `Apriori.buildAssociations()` method taken from the Weka[3] open source implementation of the Apriori algorithm [188]. The *instances* parameter represents the set of user model cases to be mined. For each discovered rule a new `Rule` object is created and the set of discovered rules are maintained in and accessed from `MiningManager`. Listing 4.9 summarises the set of operations invoked for discovering association rules and creating `Rule` objects.

Listing 4.9: Discovering and Creating Association Rules

```
Apriori apriori = new Apriori();
apriori.setNumRules(numberOfRules);
apriori.buildAssociations(instances);
aprioriRules = apriori.getAllTheRules();
// for each rule, extract premise, consequence, confidence, support, lift
// e.g., FastVector premises = aprioriRules[0];
rule = new Rule(premise, consequence, confidence, support, lift);
rules.add(rule);
```

The `removeIrrelevantRules()` command shown in Figure 4.8 consists of three operations designed to remove uninteresting and redundant rules as described in Section 3.2.4.2. Firstly, `removeUninterestingRules()` removes rules by evaluating each rule's confidence and support measures. Only rules that have a confidence and support value higher than a specified threshold are kept. Threshold values are defined in the personalisation configuration properties file. Secondly, the `removeNonConformingRules()` command removes rules that do not conform to the *context* $\Rightarrow$ *solution/solution features* form. Finally, the `removeRedundantRules()` method iterates the remaining rule set to remove rules that are redundant using the algorithm described in Section 3.2.4.2. Firstly, redundant rules with multiple *antecedents* are removed. From the remaining rule set, redundant rules with multiple *consequences* are removed. Listing 4.10 shows the implementation of the algorithm for removing redundant rules with multiple consequences. The algorithm begins by finding a rule in the rule set with multiple consequences (i.e., consequence rule). n is the number of consequences contained in the consequence rule (Line 1). For all other rules in the rule set, it is determined: if that rule has the same antecedent or premise (Line 4-7); if that rule has a *n-1* consequence length (Line 9); and if that rule has a consequence that is a subset of the consequence rule's consequence (Line 11). If n number

[3]http://www.cs.waikato.ac.nz/ml/weka/

of rules that satisfy these conditions are found (i.e., the set of sub-consequence rules) (Line 18), then the consequence rule or the set of sub consequence rules (whichever has a lower strength measure) is deleted from the rule set (Line 20).

Listing 4.10: Removing Redundant Rules

```
1  // find rule with multiple consequences (i.e., consequenceRule)
2  consequenceLength = consequenceRule.consequenceRule.getConsequences().size()
       ;
3  // loop through other rules in the rule set
4  for (int i = 0; i < rules.size(); i++) {
5          rule = (Rule) rules.get(i);
6          // if rules have the same premise
7          if (hasSamePremise(consequenceRule, rule)) {
8                  // if so see if it have consequence of n-1
9                  if (rule.getConsequences().size() == (consequenceLength - 1)
                       ) {
10                         // if so see if its consequence is contained in
                               consequence rule
11                         if (consequenceRuleConsequences.containsAll(rule.
                               getConsequences())) {
12                                 subconsequenceNumber++;
13                                 subConsequenceRules.add(rule);
14                         }
15                 }
16         }
17 }
18 if(subconsequenceNumber == consequenceLength) {
19         // remove redundant rule(s) with lower confidence/support from rule
               set
20         RuleManager.removeSubset(consequenceRule, subConsequenceRules);
21 }
```

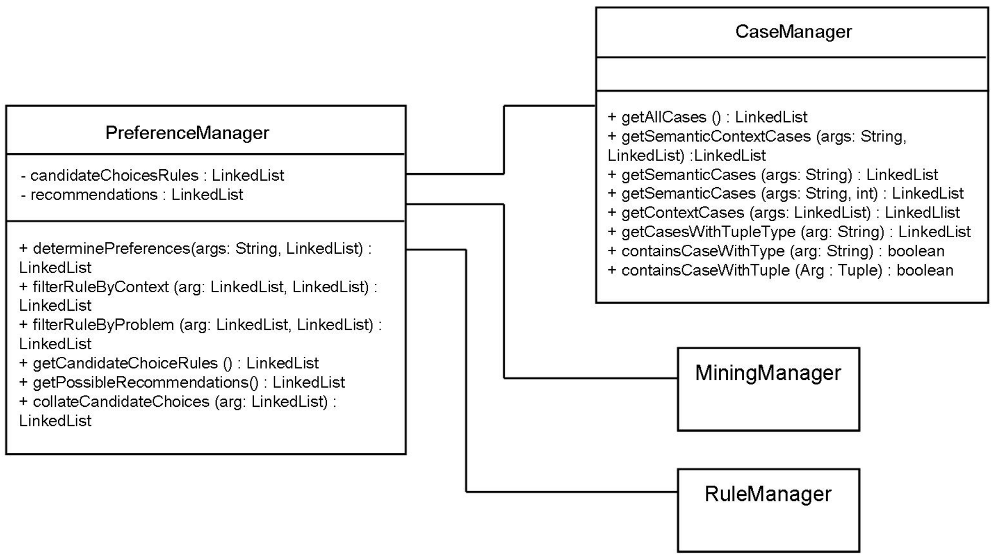

CaseManager
+ getAllCases () : LinkedList
+ getSemanticContextCases (args: String, LinkedList) :LinkedList
+ getSemanticCases (args: String) : LinkedList
+ getSemanticCases (args: String, int) : LinkedList
+ getContextCases (args: LinkedList) : LinkedLlist
+ getCasesWithTupleType (arg: String) : LinkedList
+ containsCaseWithType (arg: String) : boolean
+ containsCaseWithTuple (Arg : Tuple) : boolean
PreferenceManager
- candidateChoicesRules : LinkedList
- recommendations : LinkedList
+ determinePreferences(args: String, LinkedList) : LinkedList
+ filterRuleByContext (arg: LinkedList, LinkedList) : LinkedList
+ filterRuleByProblem (arg: LinkedList, LinkedList) : LinkedList
+ getCandidateChoiceRules () : LinkedList
+ getPossibleRecommendations() : LinkedList
+ collateCandidateChoices (arg: LinkedList) : LinkedList
MiningManager
RuleManager

Figure 4.9: Implicit Preference Determination Classes

4.6 Implicit Preference Determination

Implicit Preference Determination (Section 3.2.5) is designed to determine the preferred actions for users for a given problem and context without the need for explicit input about context and preference relationships. Figure 4.9 illustrates the main set of classes involved in Implicit Preference Determination. `PreferenceManager` is the main class responsible for Implicit Preference Determination behaviour. When determining preferences for the current problem and context, `PreferenceManager` interacts with `CaseManager` to retrieve a filtered set of cases from the user model, and with `Mining-Manager` to discover associations from these cases. It also collaborates with `RuleManager` to filter user model cases and discovered association rules.

Implicit Preference Determination begins when `determinePreferences()` in `PreferenceManager` is invoked. The recommendation problem type, and the set of solution features to be recommended are its input parameters. Listing 4.11 illustrates the `determinePreferences()` method.

Listing 4.11: Implicit Preference Determination Operations

```
// get filtered user model cases
LinkedList userModelCases = CaseManager.getSemanticCases(problemType);
// discover association rules from filtered cases
```

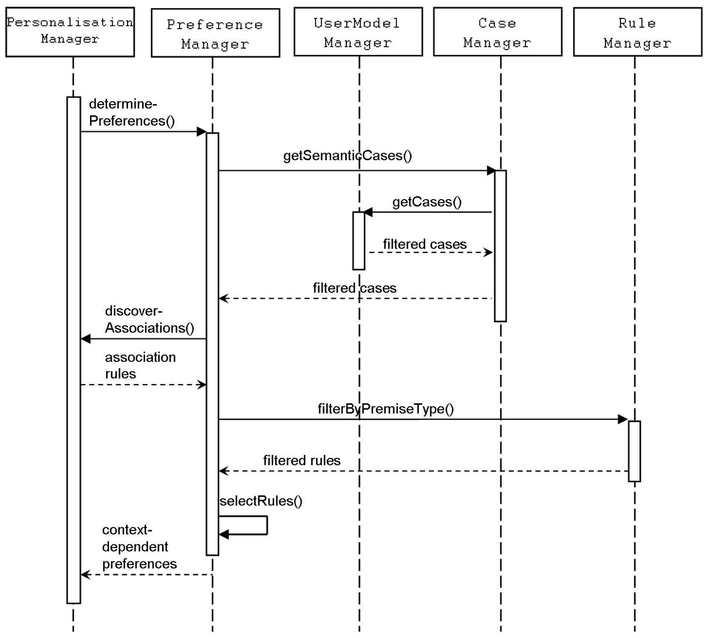


Figure 4.10: Implicit Preference Determination - Sequence of Operations

```
4 LinkedList associationRules = PersonalisationManager.discoverAssociations(
      userModelCases);
5 // filter discovered rules by context
6 LinkedList filteredRules = RuleManager.filterByPremiseType (associationRules
      , context)
7 // filter rules by number to be presented as specified in config file
8 LinkedList preferences = getPreferencesToRank(filteredRules);
9 return preferenes;
```

`PreferenceManager` collaborates with `CaseManager` to acquire a filtered set of user model cases from `UserModelManager`. The decision to access user model cases using `CaseManager` ensures that all functionality related to filtering and manipulating user model cases is encapsulated at a single point. Several methods are provided by `CaseManager` to retrieve filtered user model cases (Figure 4.9). For example, `the getSemanticCases()` method enables the retrieval of cases from the user model that are of a particular problem type (Line 2). This method invokes the `getCasesOfType()` method in `UserModelManager` described in Section 4.4 and Listing 4.8. Filtering cases by context is performed in a similar manner. Filtered cases are passed to `MiningManager` via the `PersonalisationManager` and mining is executed to determine association rules using the set of filtered cases (Line 4). Rules that are returned are further filtered to ensure that the most relevant rules for the problem and context are identified. `RuleManager` provides the functionality for filtering rules (Figure 4.7). For example, `filterByPremiseType()` filters a set of rules so that only rules that contain a specified context type are retained (Line 6). Listing 4.12 illustrates the filtering of association rules using the `hasPremiseOfType()` command in the `Rule` object. Remaining rules represent the preferred actions of a user for the given user problem and context and are returned to `PersonalisationManager`. These rules, are further filtered so that only a certain N number of rules for each solution or solution feature are retained and later ranked (Line 8 of Listing 4.11). The N value is specified as part of the personalisation configuration file (Section 4.3).

Listing 4.12: Filtering Association Rule by Context

```
for (int i = 0; i < rules.size(); i++) {
        Rule rule = (Rule)rules.get(i);
        if(rule.hasPremiseOfType(contextType)){
                // retain rule
        }
}
```

4.7 Information Selection

Information Selection (Section 3.2.6) determines the set of context types that are relevant for a given recommendation problem. Figure 4.11 illustrates the classes involved in Information Selection. `RelevanceManager` provides and coordinates Information Selection functionality, which begins in a similar set of processes as executed for Implicit Preference Determination. `RelevanceManager` interacts with `CaseManager` to retrieve and filter user model cases for the current recommendation problem. Retained cases are subsequently passed to and mined by `MiningManager`, which returns a set of discovered association rules. Rules are filtered by `RuleManager` and returned rules are iteratively compared in `RelevanceManager` to determine the set of context types that are relevant for the current recommendation problem using the algorithm described in Section 3.2.6. The number of rules to be compared is specified as part of the personalisation configuration file (Section 4.3).

Figure 4.12 illustrates the sequence of operations and associated classes that are executed during Information Selection[4]. Information selection is triggered following the execution of Implicit Preference

[4]For clarity, the Association Determination operation provided by `MiningManager` is omitted.

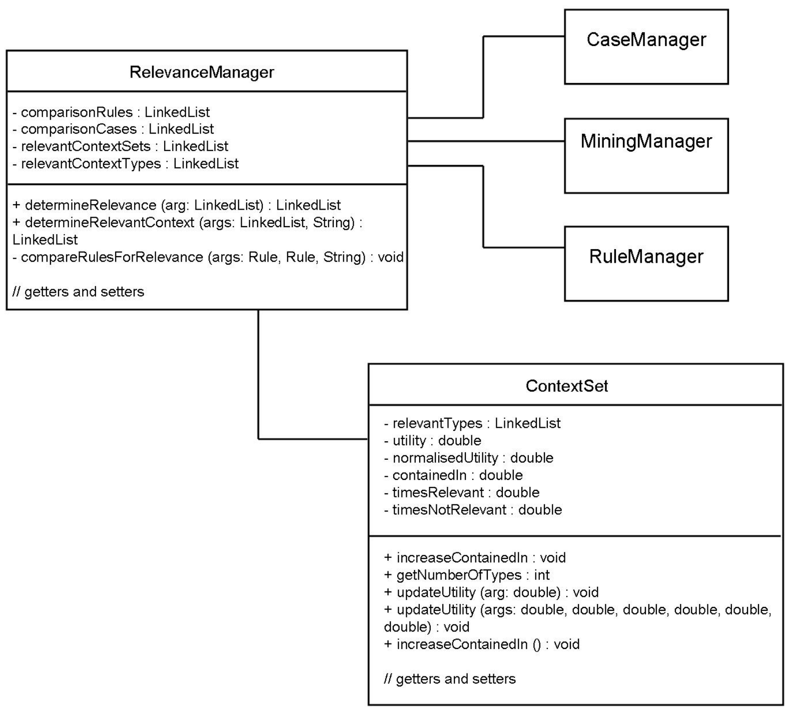

RelevanceManager
- comparisonRules : LinkedList
- comparisonCases : LinkedList
- relevantContextSets : LinkedList
- relevantContextTypes : LinkedList
+ determineRelevance (arg: LinkedList) : LinkedList
+ determineRelevantContext (args: LinkedList, String) : LinkedList
- compareRulesForRelevance (args: Rule, Rule, String) : void
// getters and setters
CaseManager
MiningManager
RuleManager
ContextSet
- relevantTypes : LinkedList
- utility : double
- normalisedUtility : double
- containedIn : double
- timesRelevant : double
- timesNotRelevant : double
+ increaseContainedIn : void
+ getNumberOfTypes : int
+ updateUtility (arg: double) : void
+ updateUtility (args: double, double, double, double, double, double) : void
+ increaseContainedIn () : void
// getters and setters

Figure 4.11: Information Selection Classes

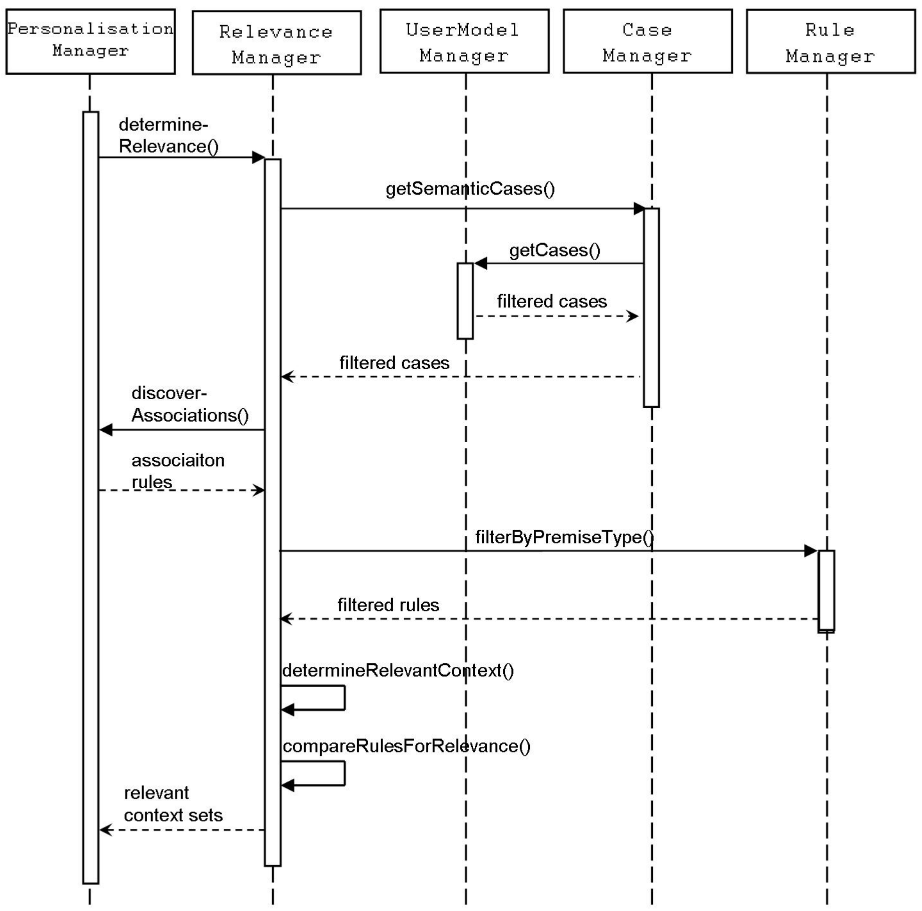


Figure 4.12: Information Selection - Sequence of Operations

Determination in the `startRecommendationProcess()` method by invoking `determineRelevance()` in `RelevanceManager`. Information Selection begins like Implicit Preference Determination with `RelevanceManager` collaborating with `CaseManager` to acquire a set of filtered user model cases from `UserModelManager`. Specifically, cases are filtered semantically by problem type using the `getSemanticCases()` method, rejecting those cases that do not match the current problem type. Cases at this stage are not filtered by context as the aim is to determine the set of context types that in general are important for a given type of recommendation problem. Retained cases are subsequently mined by `MiningManager` (as described in Section 4.5), and a set of discovered association rules are returned. Rules are filtered by `RuleManager` so that rules of the form *context* ⇒ *solution* are retained. Rules of this form facilitate the determination of context relevance as described in the design of Information Selection (Section 3.2.6). Remaining rules are prioritised by strength and the N strongest (i.e., most accurate) rules, as specified as part of personalisation configuration, are compared. The `determineRelevantContext()` command iteratively selects pairs of rules and invokes the `compareRulesForRelevance()`. `compareRulesForRelevance()` provides the set of operations that compares each pair of rules to determine the set of context types that are relevant for the current recommendation problem using the four relevance rules outlined in Section 3.2.6. Listing 4.13 illustrates the `compareRulesForRelevance()` method. The set of operations compare the solution and the set of context types and values in each rule and determines if the context is relevant. The four relevance rules outlined in Section 3.2.6 are implemented as conditionals in Lines 5, 9, 13 and 17.

Listing 4.13: Determining Relevant and Irrelevant Context

```
1  // for each context type in the rule, compare context and solution values:
2  // check to determine that the two rules have context type that can be
       compared
3  if(comparisonRule.hasPremiseOfType(baseType)) {
4          // same context, same action
5          if (basePremise.equals(comparisonPremise) && baseOutput.equals(
               comparisonOuput))
6                  // mark as relevant
7
8          // same context, different action
9          if (basePremise.equals(comparisonPremise) && !baseOutput.equals(
               comparisonOuput))
10                 // mark as irrelevant
```

```

        // different context, same action
        if (!basePremise.equals(comparisonPremise) && baseOutput.equals(
            comparisonOuput))
                // mark as irrelevant

        // different context, different action
        if (!basePremise.equals(comparisonPremise) && !baseOutput.equals(
            comparisonOuput))
                // mark as relevant
[...]
// set of context types relevant are stored as a relevant context set
relevantContextSets.add(relevantSet);
irrelevantContextSets.add(irrelevantSet);
```

This iterative process continues until all rules have been compared. Relevant context types or sets of types are stored as `ContextSet` objects (Figure 4.11). This object is designed to facilitate the storage of multiple relevant data types and the utility of this data in recommendation decisions. As described in Section 3.2.6, the storage of multiple types together supports dependencies between context, whereby certain context types are relevant depending on the type and value of other context. Note that the number of times a context set was determined relevant or not relevant is also stored as `timesRelevant` or `timesNotRelevant` attributes in `ContextSet` - a value that is used later as part of Utility Assignment. When all rules have been compared, the set of inferred relevant context and non relevant context sets can be accessed from the `RelevanceManager` via `PersonalisationManager`.

4.8 Utility Assignment

Utility Assignment, which determines the appropriate utility to assign to relevant context sets in a recommendation decision, begins when Information Selection returns to `PersonalisationManager`. Figure 4.13 illustrates `UtilityManager` class. Cases and rules used at the Information Selection stage along with the relevant context sets and their associated consistency values (determined by Information Selection) are retrieved from `RelevanceManager`. A utility value for each relevant context set is then evaluated using *consistency*, *precision*, *recall*, *confidence*, *support*, and *lift* metrics as described in Section 3.2.7.

Figure 4.14 illustrates the sequence of operations that are executed during Utility Assignment,

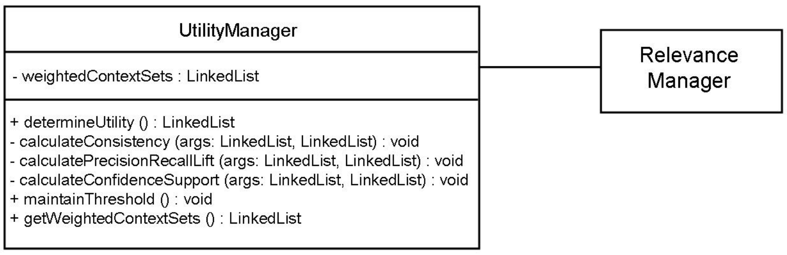

UtilityManager
- weightedContextSets : LinkedList
+ determineUtility () : LinkedList
- calculateConsistency (args: LinkedList, LinkedList) : void
- calculatePrecisionRecallLift (args: LinkedList, LinkedList) : void
- calculateConfidenceSupport (args: LinkedList, LinkedList) : void
+ maintainThreshold () : void
+ getWeightedContextSets () : LinkedList
Relevance Manager

Figure 4.13: Utility Assignment Classes

which begin when `PersonalisationManager` invokes `UtilityManager.determineUtility()`. Context sets evaluated as relevant as well as user model cases and rules used during Information Selection are acquired and passed as input to the `UtilityManager` with this invocation. `PersonalisationManager` uses the `getContextSets()`, `getComparisonCases()` and `getComparisonRules()` operations in `RelevanceManager` to retrieve this information. The `determineUtility()` method call provides the operations to determine the utility of relevant context. Three methods are executed for each rule as part of this operation and their outputs are iteratively combined to determine the appropriate weight to assign to each relevant context set - `calculateConsistency()`, `calculatePrecisionRecall`(), and `calculateConfidenceSupportLift()`. Listing 4.14 illustrates the `calculateTypeUtility()` method. The `updateUtility()` command (Line 11) maintains the utility of each context set as rules are evaluated. The `updateUtility()` command as shown in Listing 4.15, illustrates how the utility of a context set is computed.

Listing 4.14: Calculating Context Utility

```
1  // for all rules and for all context sets:
2  // if rule contains  context set in its premise
3  if (rule.containsContextSetProportion(set) != 0) {
4          // retrieve values for rule consistency, confidence, support,
               precision, and recall
5          consistency = rule.getConsistency();
6          confidence = rule.getConfidence();
7          lift = rule.getLift();
8          support = rule.getSupport();
9          precision = rule.getPrecision();
10         recall = rule.getRecall();
```

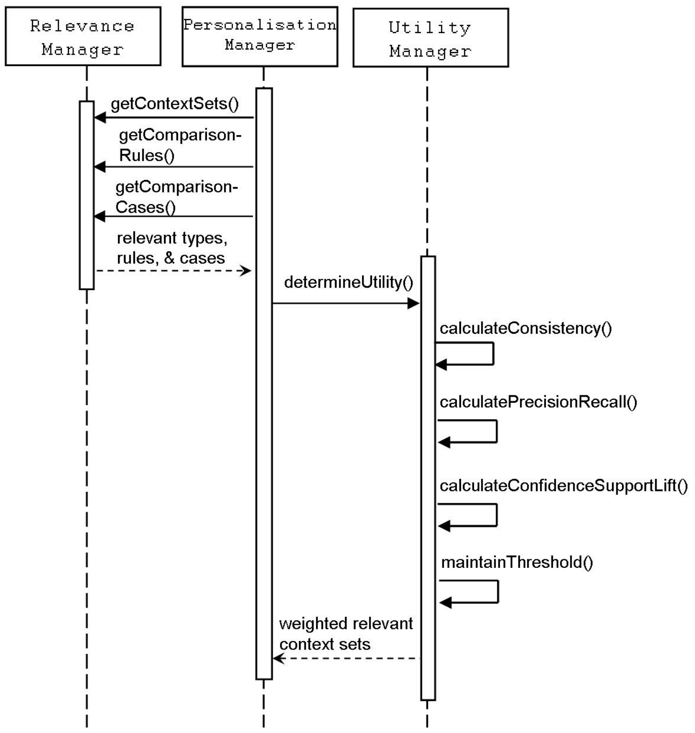


Figure 4.14: Utility Assignment - Sequence of Operations

```
        set.updateUtility(consistency, confidence, support, lift, precision,
            recall);
}
```

Listing 4.15: The `updateUtility()` method in `ContextSet`

```
public void updateUtility(double consitency, double confidence, double
    support, double lift, double precision, double recall) {
        utility += consistency * (confidence*support*lift) * (precision*
            recall);
}
```

`calculateConsistency()`determines the consistency value of each context set by subtracting its `timesNotRelevant` attribute value from its `timesRelevant` value and normalising this value to the range of 1. `calculatePrecisionRecall()` determines the precision and recall values for each rule and assigns the corresponding value to each context set contained in that rule. Section 3.2.7 outlined the formulas for calculating precision and recall. The *relevant items retrieved* value (used in the formulas for precision and recall) is calculated by iterating through rules and counting the number of relevant context types contained in the rule using the `hasPremiseOfType()` command. Listing 4.16 illustrates the implementation of the precision and recall formulas. `calculateConfidenceSupportLift()` also iterates over the set of rules and relevant context contained in the rule are assigned a confidence, support, and lift value corresponding to the value of that rule.

Listing 4.16: Calculating Precision and Recall

```
double precision = relevantItemsRetrieved/relevantContextTypes.size();
double recall = relevantItemsRetrieved/rule.getPremises().size();
```

The `maintainThreshold()` call extracts the weights of context sets from past relevant user model cases with the aim of using knowledge about past recommendation decisions. Weights stored in past cases, which are retrieved by invoking `RelevanceManager.getComparisonCases()` are combined, averaged, and compared to the utility of context sets evaluated as part of the current execution of Utility Assignment. Utility values for matching context sets are compared to determine if they exceed a weight threshold value as specified in the personalisation configuration file. If the threshold is exceeded for any context set, then its utility value is incremented or decremented so that the difference no longer exceeds this threshold value. The final utility value for each context set, which represents its utility for the current recommendation problem and context, is evaluated, stored, and returned to the `PersonalisationManager` and used to rank candidate recommendation choices.

4.9 Recommendation Generation and Ranking

Recommendation Generation and Ranking is the final stage of the multi-stage recommendation process. Figure 4.15 illustrates the classes involved at this stage. The problem type, context, implicitly determined user preferences, and the utility of relevant context types are passed to `RankingManager`, which is responsible for determining the set of candidate behaviour recommendations and ranking them in order according to inferred user preferences.

Figure 4.16 illustrates the sequence of operations that occur during recommendation generation and ranking. The process is initiated by invoking the `startGenerationAndRanking()` method in `RankingManager`. Inputs necessary for recommendation generation and ranking are passed in as

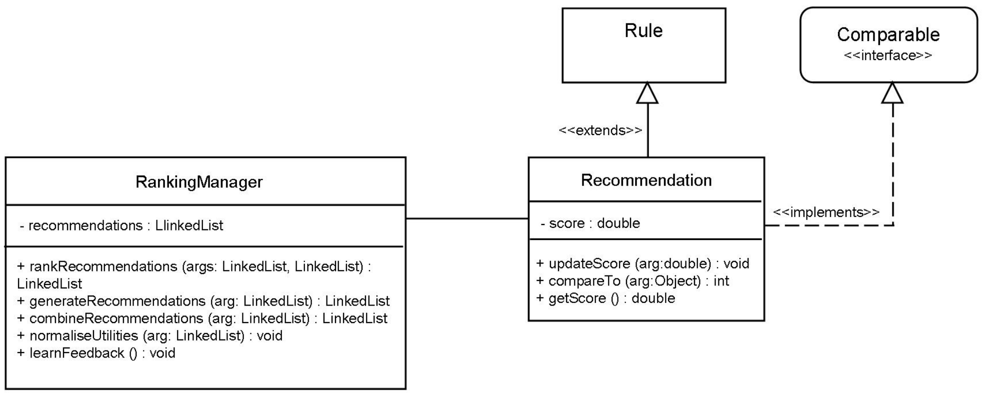

Rule
Comparable
<<interface>>
<<extends>>
<<implements>>
RankingManager
- recommendations : LlinkedList
+ rankRecommendations (args: LinkedList, LinkedList) : LinkedList
+ generateRecommendations (arg: LinkedList) : LinkedList
+ combineRecommendations (arg: LinkedList) : LinkedList
+ normaliseUtilities (arg: LinkedList) : void
+ learnFeedback () : void
Recommendation
- score : double
+ updateScore (arg:double) : void
+ compareTo (arg:Object) : int
+ getScore () : double

Figure 4.15: Recommendation Generation and Ranking Classes

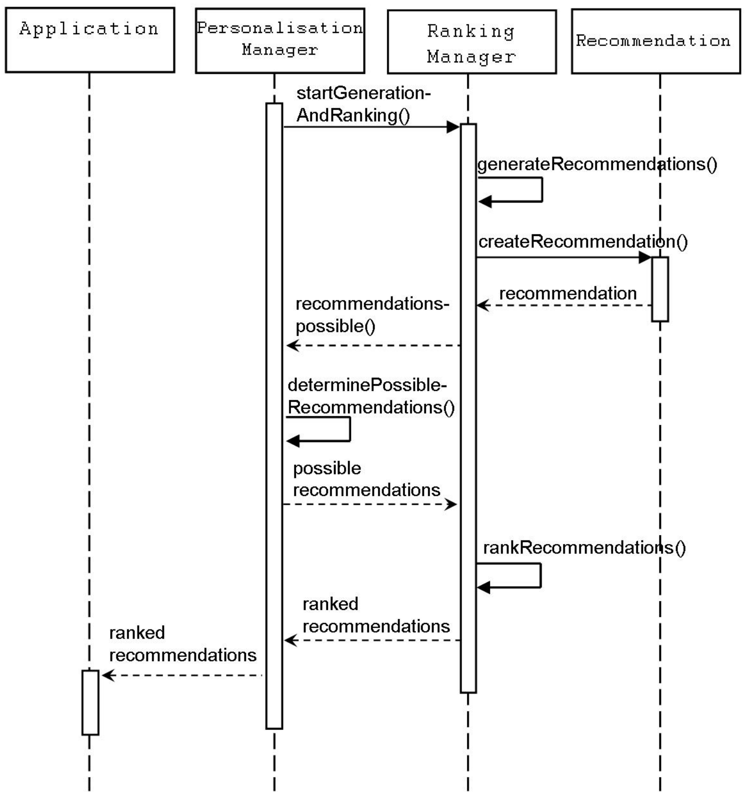


Figure 4.16: Recommendation Generation and Ranking - Sequence of Operations

parameters with this call. Two distinct operations are executed as part of this method - `generateRecommendations()` and `rankRecommendations()`. `generateRecommendations()` evaluates the set of user preferences (inferred by Implicit Preference Determination) and determines the set of candidate choices. That is, N number of rules for each solution or solution feature are selected to be ranked. The value for N is the number of recommendations to be presented to the user and is specified as part of the personalisation configuration file. The `determinePossibleRecommendations()` command provides operations that ensures each candidate choice is actually possible (e.g., determining if a candidate restaurant choice is available). For each possible choice, a `Recommendation` object is constructed (Figure 4.15). `rankRecommendations()` scores each `Recommendation` instance. Listing 4.17 shows the implementation of the `rankRecommendations()` method, illustrating how the set of candidate choices are scored according to the utility of relevant context sets using the MAUT approach as described in Section 3.2.8. Utilities are first normalised around 1 as required by a MAUT approach. A score for each candidate recommendation is computed by determining the proportion of applicable context sets it contains (i.e., context values that match the context of the current problem) and multiplying this value by the context set's normalised utility value to compute a score. The implementation of the `java.lang.Comparable` interface and the `compareTo()` method enables different recommendations to be ranked according to their score.

Listing 4.17: Ranking Candidate Recommendations with `rankRecommendations()`

```java
normaliseUtilities(sets);
// for all recommendations
for (int a = 0; a < recommendations.size(); a++) {
        score = 0;
        Recommendation rec = (Recommendation)recommendations.get(a);
        // for all context sets
        for (int j = 0; j < sets.size(); j++) {
                ContextSet set = (ContextSet)sets.get(j);
                proportion = rec.containsContextSetProportion(set);
                if (proportion != 0) {
                        score = proportion * set.getNormalisedUtility();
                        rec.updateScore(score);
                }
        }
}
```

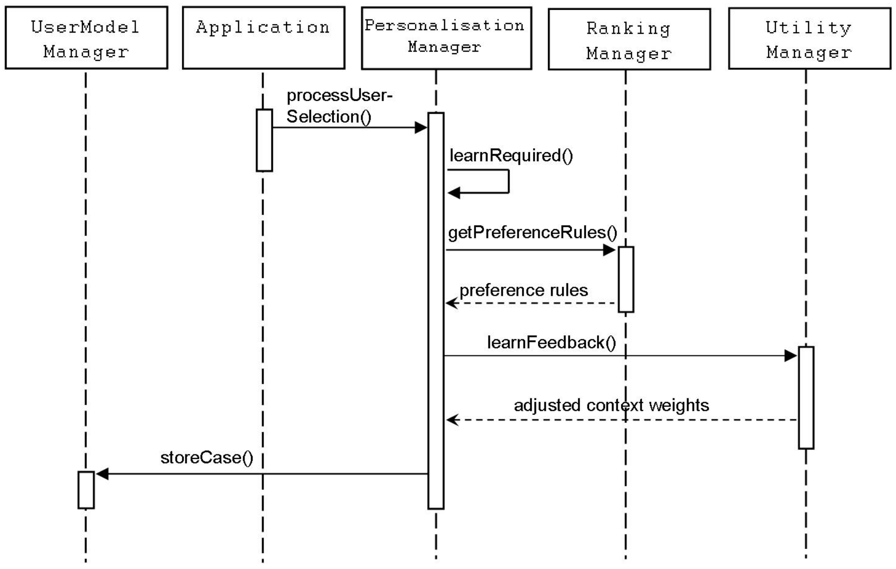


Figure 4.17: Learning from Implicit User Feedback - Sequence of Operations

When a set of recommendations is presented to the user, the application waits for a user selection so that this recommendation problem can be added as a new case to the user model. When a user selection arrives, the application first determines if any learning is required before adding the case to the user model. This ensures that the correct context utility is stored in the case, providing accurate knowledge that aids future recommendation decisions. Figure 4.17 illustrates the sequence of operations that execute when a user selects their preferred recommendation behaviour.

Listing 4.18: Learning from Implicit User Feedback

```
// retrieve selected rule and recommended rule
// for all context sets:
// learning possible if context set in one rule but not the other
if(selectedRule.hasPremiseSet(set) && !recommendedRule.hasPremiseSet(set)
        || !selectedRule.hasPremiseSet.hasPremiseSet(set) && recommendedRule
            .hasPremiseSet(set)) {
        set.updateUtility(INCREMENT_VALUE);
        rankRecommendations();
}
```

`PersonalisationManager` processes the selected recommendation to determine if learning is required. As described in Section 3.2.8 learning is only necessary if the user selects a recommendation that is not the highest ranked. If learning is necessary, the set of preferences rules are retrieved from `RankingManager` and the `UtilityManager.learnFeedback()` command is invoked to adjust context utilities. Listing 4.18 shows the implementation of the learning algorithm as implemented in the `learnFeedback()` method. The conditional statement (Line 4) determines that learning is possible (i.e., rules have different context sets) and the `updateUtility()` command increases utility values of context set's by a pre-specified increment value and rules are re-ranked. This process is repeated until the recommendation selected by the user is ranked highest or no more learning is possible.

4.10 Chapter Summary

This chapter has described the implementation details of the personalisation approach proposed in this thesis. Specifically, the chapter describes how the techniques and algorithms designed for different stages of the multi-stage process are implemented. The implementations of techniques and algorithms are described in detail along with the illustration of relevant classes, sequences diagrams, configuration data, and source code excerpts. Each stage of the personalisation process and its operations are designed and implemented in such a way as to minimise the need for user input regarding preference details and expert knowledge and input regarding relationships between different context types, preferences, and recommendation behaviour. The approach supports the evaluation of recommendations of different problem and context types, which is easily extended by defining or extending `UserModel-Case` with new types and values. Behaviour at different stages of the process can also be customised external to source code through the use of a personalisation configuration file.

The following chapter evaluates the personalisation approach through the development of two user case study applications, designed to assess the accuracy of recommendations against actual user choices. The accuracy of Information Selection and Utility Assignment techniques are also evaluated against actual user choices. In addition, a set of simulated experiments are conducted to evaluate the effect of different user model sizes and different numbers of supported context types on recommendation accuracy.

Chapter 5

Evaluation

" the ability of a recommendation system to identify user interests correctly and make proper recommendations is critical to the success of the system" [114]

The evaluation process described in this chapter is focused on measuring the accuracy of recommendation decisions. As outlined in the research question investigated in this thesis, the main objective of developed personalisation techniques and algorithms is to support context-aware applications in making *accurate* personalised recommendations to users (i.e., a recommendation that matches the preferred choice of a user). The previous chapters have described the design and implementation of a multi-stage recommendation process that includes techniques and algorithms for supporting applications in providing personalised recommendations to application users in different context. The accuracy of recommendation decisions is predicated on the personalisation approach effectively addressing the requirements outlined in Section 1.2. Other measures associated with recommendation decisions such as trust (i.e., whether users perceive recommendations made by the application to be reliable), transparency (i.e., whether users adequately understand the reasons for application recommendations), ability of users to modify recommendations, and response time at which recommendations are returned are also important [131, 177, 169, 65], but as the personalisation approach has not been designed to facilitate these measures, they are not the focus of this evaluation. Future studies could extend the evaluation to assess the personalisation approach under these measures.

This chapter discusses the evaluation of the personalisation approach described in this thesis and assesses the ability of its algorithms and techniques to support applications in providing accurate personalised recommendations to users. The chapter begins with a description of the evaluation set-up including the set of user-studies and simulated experiments that were conducted, the approaches taken to measure accuracy, and the set of statistics used for analysing results. The remainder of the chapter describes the set of evaluation tests completed for testing recommendation accuracy. The two recommendation strategies designed as part of the personalisation approach (described in Section 3.2.1.1) are evaluated against each other and against several of the most relevant recommendation approaches used in state of the art personalised systems (random recommendations, rule-bases, neural networks, user preference models, and case-based reasoning). The accuracy of Information Selection and Utility Assignment algorithms is also analysed. In addition, the results of tests conducted to evaluate how the personalisation approach performs for different user model sizes and for different number of supported context types are presented. Each section describes the strategy or algorithm being tested, the set of test data that is used, and the accuracy results for that strategy or algorithm when compared against user selections.

5.1 Set-up

The main aim of the evaluation is to assess the accuracy of our personalisation approach in making recommendations that match the preferred choice of users. Accuracy is measured by comparing recommendations made by the application under different context with actual user choices. The overall accuracy of a recommendation approach for a user is calculated by counting the number of times the approach provides recommendations that match that of the user for a set of recommendation problems. The motivation for comparing recommendations with actual user choices is to determine if the application is able to make decisions in different context settings that reflect those a user would make given the same set of surrounding context information. The aim is to facilitate applications to aid users with recommendation problems, in situations where the user is not fully aware of their surrounding environmental context, by recommending actions that best reflect the ones they would choose if they were better aware of their context.

As part of this evaluation two case studies involving real users and a set of simulated experiments have been conducted. User studies, for two different application scenarios, queried users for their preferred behaviour for a set of recommendation problems. Each problem consists of a different set of context types and values, therefore enabling information about the preferred behaviour of users for different combinations of context values to be gathered. In the second case study, users were also asked about the set of context types they considered relevant and the relative importance of each type when deciding on their preferred outcome. Evaluation tests compare recommendations, relevant context, and context utility generated by the personalisation approach with actual user responses to determine an accuracy value. Comparisons between application recommendations and user choices are conducted offline after all case study data is collected. Section 5.1.2 discusses the reasons for not evaluating the approach with a live deployment.

The set of simulated experiments that were designed to assess how the personalisation approach performs when making recommendations with different values for user model size and number of context types considered in recommendation decisions. User choices are generated in these simulations using artificial preference models designed to represent user preferences under different context.

The accuracy of each recommendation strategy or algorithm is shown as a series of graphs. Graphs illustrate the accuracy for each user and an overall average accuracy for the set of users for a given strategy or algorithm. The different recommendation strategies and state of the art recommendation approaches in each application scenario are evaluated using the same data. It is therefore fair to compare the results of their evaluation directly. Each user is referred to by a numeric identifier, which is consistent within each application scenario, enabling the recommendation accuracies of each user to be compared directly.

5.1.1 Personalisation Configuration

Listing 5.1 illustrates the personalisation configuration file, which shows how the personalisation approach described in this thesis was configured when gathering the evaluation results described in this chapter. The same configuration data is used to facilitate the fair comparison of accuracy results gathered for the personalisation approach across the set of evaluation tests.

Listing 5.1: Personalisation Configuration File

```
# PERSONALISATION CONFIGURATION PROPERTIES
association.rulesToGenerate=2000
association.minSupport=1
association.confidenceThreshold=1
association.supportThreshold=1
filtering.caseContextMatches=1
filtering.ruleContextMatches=1
relevance.rulesToCompare=100
utility.weightDifferenceThreshold=0.8
learning.learningRate=0.025
ranking.rulesToRecommend=3
```

It was discussed in Section 4.3 that values for properties in the personalisation configuration file are determined according to the requirements of the application and the characteristics of user model data (e.g., user model size, number of information types in user model cases). Given that the set of available user model cases in the user study evaluations is relatively small (i.e., 15 cases for the first study and 40 cases for the second), the values are set in order to minimise the loss of relevant decision making knowledge that is discovered. For example, threshold values for filtering user cases and association rules for Implicit Preference Determination and Information Selection are set to their minimum possible values. Other values are set based on several test executions of the personalisation approach on a user model that was randomly selected from the gathered user responses. These values were found to provide accurate result while minimising the amount of processing required (e.g., the generation of 3000 association rules on the same user model data did not provide more accurate recommendations)[1]. The final property, representing the number of recommendations to be presented to the user, is set according to the requirements of an application or preferences of the user. A value greater than one is selected in this case so that learning from incorrect recommendation decisions is facilitated (as described in Section 3.2.8.1).

The values for configuration properties are also selected as to ensure recommendations are evaluated and presented to users within an acceptable time frame. An acceptable threshold on execution time is defined according to a user's tolerance for delay in an application [140] and applications that return results outside this threshold risks frustrating the user [162]. Application response times varying from two seconds [130] to ten seconds [142] have been proposed as acceptable. As a guideline, this thesis adopts a response time bound of 0-12 seconds, which is determined from analysis of responsiveness requirements defined for application services of the Hermes application framework [64] (i.e., configuration properties have been selected to ensure personalised recommendations are presented to users within 12 seconds of their recommendation request). Note that although a threshold value of 12 seconds is selected, improvements can be made to the recommendation process to improve its response time (e.g., selecting an association discovery algorithm that is more efficient than Apriori). A more in depth analysis of the computational efficiency/overhead of the recommendation process, including possible algorithm improvements and detailed evaluation, is future work (Section 6.2.1).

As mentioned previously (Section 4.3), future work will also investigate the dynamic and implicit adaptation of these values to different user problems and context (Section 6.2.2).

[1]Note that these tests provided an indicator as to suitable configuration values and were not statistically evaluated. Further tests would be necessary to statistically validate whether a given set of configuration values performs better than another different set.

5.1.2 Issues with Live Deployment

The decision to not evaluate the personalisation approach by deploying it on mobile devices and giving it to users to use over time was taken to ensure the validity and fair comparison of acquired results. There are a number of factors associated with deploying to real users that may affect the validity and fair comparison of results.

Deploying to mobile devices relies on connected sensor devices to acquire the set of context data that facilitates the personalisation approach to make recommendation decisions. However, sensor data is inherently uncertain [128, 56, 166, 152] and the approach is not currently designed to support uncertainty associated with acquired context (though this is planned for future work - Section 6.2.4). The difficulty with effectively measuring uncertainty associated with context and the effect of this uncertainty on recommendation decisions means that the accuracy of the personalisation approach cannot be measured fairly. For a fair evaluation to be made in the presence of uncertain data, this uncertain data and the effect of uncertainty on recommendation decisions would have to be accurately quantified.

Current sensor devices are also limited to providing context of certain types (e.g., location, time). Many other context types that may influence a user's behaviour decision (e.g., weather, who they are with) cannot be easily or implicitly acquired. However, the personalisation approach has been designed to ensure that relationships between multiple different context types are supported. The lack of sensors available to provide applications with context of different types limits the number of types an application supports and consequently limits the evaluation of the personalisation approach to a limited number of context types and relationships. A related issue is that users may make recommendations based on context information not supported by the application. A fair comparison between application decisions and user decisions therefore cannot be made as the context information available to each is not equivalent.

In addition, giving devices to users does not facilitate the acquisition of data about the relevance and utility of different context types that are considered by users in recommendation decisions. The form factor of mobile devices makes inserting information about recommendation decisions (i.e., relevance and utility) difficult and often intrusive to users.

A "non-live" (or offline) evaluation minimises these problems and is designed to ensure that evaluation data is gathered in a controlled manner so that user selections and outputs generated by the personalisation approach can be compared fairly. Explicitly querying users about their preferred actions in different context, where users are explicitly informed about the context types and values that can be considered, is taken to ensure that the effect of data uncertainty and other information that may influence or bias a user's decision is minimised. Informing users about the context types that can be considered in recommendations also facilitates the evaluation of multiple different context types and values.

The non-live evaluation is also designed so that the evaluation can focus directly on acquiring those user responses that are required for comparing user selections with outputs generated by the personalisation approach (i.e., recommendation selections, relevant context types, and context utility). A non-live evaluation also avoids delays related to hardware failure and failures related to recording all necessary context and user decision making data during application execution - deployment issues that are not related to the development and evaluation of the personalisation approach. Issues related to the cost and availability of resources and devices required for deploying context-aware applications in real world environments would also have restricted the number of user evaluations possible.

5.1.3 Measuring Accuracy

Two different offline approaches were adopted for assessing the accuracy of the personalisation approach proposed in this thesis: *separate* test cases (also referred to as a *holdout* approach [103]) and *leave one out* test cases [103, 59].

A separate test cases approach was used to assess recommendation accuracy in our first user study evaluation. With this approach responses from users about recommendation problems are divided in two subsets - those that are stored in the user model and those that are not. This facilitates the evaluation of the personalisation approach along two separate dimensions. That is, the approach is tested on problems that are stored in the user model and on problems that are not. This facilitates the evaluation to assess the accuracy of the recommendation approach for recommendation problems that were posed in the past and also its accuracy when generalising user model knowledge to new (previously unencountered) problems. This evaluation approach is executed as follows: each case (both those in the user model and those that are omitted) is evaluated in turn with the context values stored in the case representing the inputs to the recommendation problem (i.e., what is the user's preferred behaviour in this context). Recommendations are generated using user model data and the accuracy of recommendations is calculated by comparing the recommended outputs with user's selection stored in the test cases. The overall accuracy of a recommendation approach for a given user is the average of the test case results.

On analysis of the results of the first study, the main issue with a separate test cases approach is that because the approach is tested with problems that are already contained in the user model and therefore has knowledge about, results may be unfairly biased. This is further discussed in our analysis of results gathered using a separate test case approach. Consequently, subsequent evaluations followed a *leave one out* process. In contrast to a separate test case approach, leave one out does not purposely include test cases in the user model. Instead, a user model is generated with all cases that have been gathered except for the one being tested. That is, a test case is removed from the overall set of cases and the remaining cases make up the user model. This user model is then used to determine a recommendation for the test case problem that was removed. Therefore, each case is tested as if all other cases preceded it. As before (with a separate test case approach) an accuracy measure is calculated by comparing the applications recommendation with the user's selection stored in each test case and the overall accuracy for a given user is the average of the set of test case results.

5.1.4 Evaluation Statistics

An important feature of real data is that it exhibits a certain degree of variation [135]. Statistical tests are therefore important as they aid in distinguishing between chance variation and systematic effects [135]. To evaluate the effectiveness of the personalisation approach described in this thesis, we compare the accuracy of its recommendation strategies and algorithms (in making recommendations that match a user's preferred choice) against a baseline accuracy and against the accuracy of other recommendation approaches. The results are statistically analysed to determine if differences in accuracy are *statistically significant* (i.e. the difference is systematic and is unlikely to have occurred by chance). Two sets of tests were used for this evaluation: Student's t-tests and the F-test [135, 138].

Student's t-distribution [75] provides the first set of statistics used for assessing recommendation accuracy in this thesis. This distribution is effective in estimating the mean of normally distributed data when the sample size is small[2]. Students t-test, a statistical significance test based on Student's t-distribution, is used to assess the accuracy of various recommendation strategies and algorithms and to test for statistical significant differences between them. t-tests require specifying a null and alternative hypothesis. A t-value calculated from a test statistic and a number of degrees of freedom determine if the null hypothesis is accepted or rejected [138]. The thesis uses two types of Student's t-tests for evaluation: *one sample t-test* and *paired t-test*. The one sample t-test is used to compare the accuracy of recommendation algorithms against a baseline accuracy value while the paired t-test compares the accuracy of algorithms against each other.

The one-sample t-test is used to compare a sample mean to a specific value. The hypotheses of a one sample t-test are specified as:

[2]Appendix A.2 presents the normal plots of the data presented in this chapter.

$$H_0: \mu = specified\,value$$
$$H_1: \mu \neq specified\,value$$

where H_0 is the null hypothesis (i.e., the average accuracy of a recommendation approach equals 50%), H_1 is the alternative hypothesis (i.e., the average accuracy of a recommendation approach does not equal 50%), and μ is the sample mean (i.e., the average accuracy of a recommendation approach, e.g., 50%). The null hypothesis represents the view that, in the long-run, the average accuracy of the recommendation algorithm is equal to the specified baseline accuracy value. The test statistic for a one sample test, which measures departure from the null hypothesis $\mu = \mu_0$ where $\mu_0 = specified\,value$ is:

$$t = \frac{(\bar{x} - \mu_0)}{s/\sqrt{n}}$$

$\bar{x}$ is the mean (or average) accuracy of a recommendation algorithm, μ_0 is the baseline value we compare recommendation accuracy to, s is the standard deviation of the recommendation accuracy values, and n is the number of users. The value of t relative to critical values, defined by a significance level and number of degrees of freedom, determines whether the null hypothesis is accepted or rejected with a certain confidence level. A confidence level of 99% (i.e. a significance level $\alpha = 0.01$) is chosen. If the null hypothesis is accepted then it is concluded, with 99% confidence, that there is no statistical evidence to support the view that the accuracy of the recommendation algorithm is better than the specified baseline accuracy value. In contrast, if the null hypothesis is rejected, there is 99% confidence that the accuracy of the recommendation algorithm is better than the specified baseline accuracy value. In this evaluation, the *specified value* is set to *50,* which is a widely adopted comparison value in statistical analysis [138, 135] and represents the accuracy of a *random* recommendation. Therefore the one sample t-test determines if the recommendation accuracy of the personalisation approach is, in the long run, better than a random recommendation and not simply due to chance variation.

In contrast to the one sample t-test, the paired t-test is used to compare accuracy of recommendation algorithms against each other. The paired test differs from a one-sample test as it examines the *differences* between the results of two algorithms. The hypotheses of a paired t-test are specified as:

$$H_0: \mu_1 - \mu_2 = 0$$
$$H_1: \mu_1 - \mu_2 \neq 0$$

where H_0 is the null hypothesis (i.e., the difference between the average accuracy of recommendation approach A and the average accuracy of a recommendation approach B is equal to 0), H_1 is the alternative hypothesis (i.e., the difference between the average accuracies of two recommendation approaches does not equal to 0) , and μ_1 is the mean (or average) accuracy of one recommendation algorithm and μ_2 is the mean (or average) accuracy of a second recommendation algorithm. The null hypothesis represents the view that if the difference in accuracy between two recommendation algorithms is not statistically significant, then the difference between their long-run means is zero. The test statistic for a paired t-test is:

$$t = \frac{\bar{d}}{{}^{s}/_{\sqrt{n}}}$$

$\bar{d}$ is the mean, s is the standard deviation of the differences between the accuracy results of the two algorithms for each user, and n is the number of users. Similar to a one sample t-test, the value of t relative to critical values, defined by a significance level and number of degrees of freedom, determines whether the null hypothesis is true with a certain confidence level. A significance level $\alpha = 0.01$ is chosen, meaning that it can be determined, with 99% confidence, whether one recommendation algorithm is more accurate than another.

In addition to the Students t-test, the F-ratio statistic test based on the F-distribution is used to compare accuracy of multiple data sets together. Fisher's *Least Significant Difference* (*LSD*) [138] facilitates the comparison of multiple data sets at once to determine if they are statistically significantly different. This supports the analysis of results related to determining how the accuracy of the personalisation approach is affected given a set of multiple different test constraints. Specifically, the F-ratio test is used to compare accuracy of recommendations for a set of different user model sizes and different number of supported context types described later in Section 5.4 and Section 5.5. The null hypothesis tests that there is no difference in the long run mean accuracy for a set of samples. The hypotheses for this test is:

$$\begin{aligned} H_0 &: \mu_1 - \mu_2 - \mu_3 - ...\mu_n = 0 \\ H_1 &: \mu_1 - \mu_2 - \mu_3 - ...\mu_n \neq 0 \end{aligned}$$

The LSD value is calculated using:

$$LSD = t\sqrt{\frac{2MSE}{J}}$$

t is the test statistic for a given significant level ($\alpha = 0.01$ in this thesis), *MSE* is the *mean square*

Context Types	Meal (3), Day (2), Company (2), Transport (2), Occasion (2), Time Constraint (2)
Restaurant Properties	Cuisine, Cost, Food Quality, Service/Atmosphere, Distance

Table 5.1: Restaurant Recommender Context and Solution Features

error, which is the average of the variances in the sample mean, and J is the number of users. If the value of two sample means differs more than this LSD value, then it is 99% certain that they are statistically significantly different.

5.2 Context-aware Restaurant Recommender System

The first evaluation test that was performed was the evaluation of the original recommendation strategy. As discussed in Section 3.2.1.1, this original strategy generates recommendations using the techniques for Implicit Preferences Determination and Recommendation Generation and Ranking. That is, user model cases are filtered, useful associations between context and user choices are identified, these association rules are filtered, and recommendations represented in these rules are ranked based on their applicability and strength. Information Selection and Utility Assignment techniques are *not* supported as part of this strategy and consequently information relevance and utility is not dynamically adjusted for different recommendation problems or context. Instead, all context types are considered to be relevant and of equal utility.

5.2.1 Application Scenario and Set-up

To assess the accuracy of the original recommendation strategy, a personalised context-aware restaurant recommender system introduced in Section 3.1.2 was developed. This application was developed to provide restaurant recommendations to users taking into account their current context. A user study was developed to gather user model and test case data from real users. As part of the study, *10* participants were questioned on *15* different recommendation problems. Each problem queried the user for their preferred choice regarding a set of restaurant properties for a given context[3]. Table 5.1 summarises the set of context types and restaurant properties supported by the application. Values in brackets indicate the number of possible values for each information item. 96 possible distinct context relationships are possible given the set of context types and number of possible values (i.e., $3 \times 2 \times 2 \times 2 \times 2 \times 2 = 96$).

[3]An example recommendation problem is shown in Appendix B.1.

5.2.2 Analysis

As previously outlined, a separate test cases evaluation (Section 5.1.3) was used to assess the accuracy of recommendations for restaurant properties as compared with actual user choices. Responses for 10 of the 15 cases gathered from the user were included in a user model while the 5 remaining cases acted as new cases and were omitted from the model. This proportion was determined given the lack of data available and the need to provide the user model with a sufficient number of cases that represent a set of different user preferences. On analysis of user responses, it was evident that the preferred behaviour of users changed depending on context, highlighting the presence of a dependency relationship between context and user preferences.

The evaluation of recommendation accuracy proceeded as follows: the problem type and context features of a user model case is read in by the application. This set of information provides the application with a description of the current problem and context. A recommendation is then generated by the application using user model cases including the case being tested. The accuracy of the recommendation is determined by counting the number of recommended restaurant properties that match those selected by the user. Each user model case is tested in the same way. The evaluation process continues by testing new cases (i.e., cases that are not already contained in the user model). The process proceeded in a similar manner to testing user model cases with one exception - the *new* case being tested is not included in the user model. That is, the application is generating recommendations for a set of context values that have not been previously considered. The accuracy, like previously, is determined by counting the number of recommended restaurant properties that match actual user choices. The results for both user model test cases and new cases are collated and averaged to give an overall recommendation accuracy for the strategy. Figure 5.1 illustrates the recommendation accuracy for each user using the original recommendation strategy. The average recommendation accuracy of all users is also shown in the right-most column (64.9%).

To assess the usefulness of this recommendation strategy, these results are compared with a random recommendation (as described in Section 5.1.4). Table 5.2 shows the statistics used to compare this strategy with a random recommendation. The calculated t-value of 7.01 is greater than the 99 percentile Students t distribution ($t_{.99}$) for 9 degrees of freedom (3.25). In addition, the hypothesised mean accuracy value of 50 does not fall within the 99% confidence interval values. Therefore, the null hypothesis is rejected and it can be stated that this recommendation strategy outperforms a random recommendation with 99% confidence.

As recommendations are generated from knowledge about user recommendation decisions, the results, as expected, indicate that the approach is more accurate than a random recommendation. This

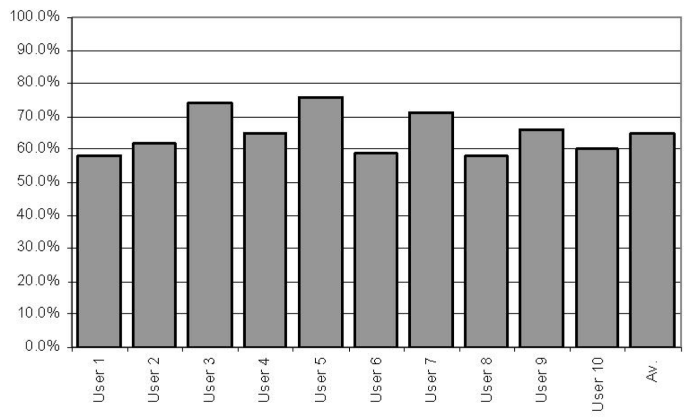


Figure 5.1: Original Strategy Accuracy

Statistic	Value
Null hypothesis	$\bar{x} = 50$
Number of Samples (n)	10
Degrees of Freedom (d)	9
Sample Mean ($\bar{x}$)	64.9%
Standard Deviation (s)	6.7%
Students t-value (t)	7.01
Critical Value ($t_{.99}$)	3.25
99% Confidence Intervals	57.9%, 71.8%

Table 5.2: Summary Statistics for Original Strategy Accuracy

difference is likely (with 99% confidence) to be between approximately 8% and 29%. The confidence intervals indicate (with 99% confidence) that the approach is most likely to make recommendations that match a users preferred action for different context 58% to 71% of the time. The standard deviation of 6.7% shows the recommendation accuracy on average varies around the mean by approximately 10%. Given the small sample size, this standard deviation is seen as low and indicates that the recommendation approach provides a relatively consistent level of accuracy for the set of users that were tested.

As stated in Section 5.1.3, closer analysis of results indicate that the separate test case approach may have unfairly biased results. Figure 5.2 illustrates the results separated for user model cases and new cases and Table 5.6 summarises the test statistics. The results show that the personalisation approach on average provides more accurate results when tested with user model cases (by approximately 16%), increasing the overall value of accuracy of the approach. Greater recommendation accuracy was

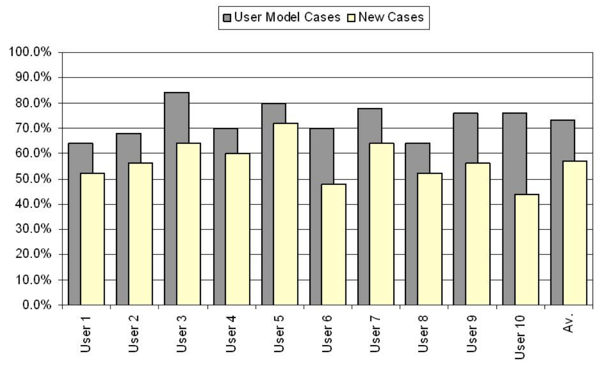


Figure 5.2: User Model Cases versus New Cases

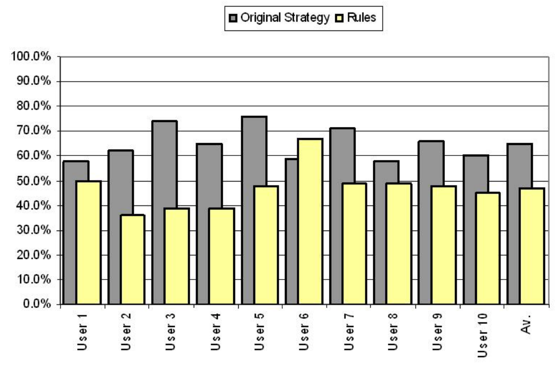
Accuracy	Average	Standard Deviation
User Model Cases	73.0%	6.4%
New Cases	56.8%	8.4%

Table 5.3: User Model Cases versus New Cases Summary Statistics

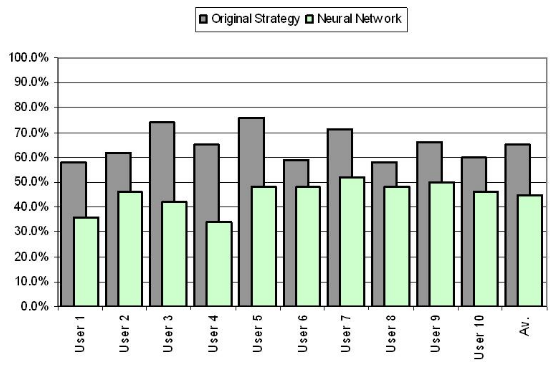
also achieved for all users. This result is not surprising as knowledge about behaviour that matches that of a user is already stored in the user model and indicates that the personalisation approach is more accurate for recommendation problems that have been previously solved. The results also show that the accuracy of the approach varies by a lower amount (approximately 2%) when evaluated for problems already stored in the user model. This indicates that the personalisation approach is more consistent at providing recommendations that match those preferred by the user for problems that have been previously solved. A possible cause of this bias is that user model cases and new cases were not evenly divided (i.e., 10 user model cases versus 5 new cases). This separation proportion was decided given the small number of gathered cases available and the need to provide the user model with an adequate number of cases about different user preferences. Further studies are required to determine if more evenly distributed cases or increases in user model size would affect the accuracy of results in each category.

5.2.3 Comparisons

As part of this first evaluation study, the accuracy of the original recommendation strategy was also compared against accuracy values for rule based and neural network recommendation approaches. A rules-based approach was selected for comparison due to its popularity and support in many existing context-aware applications and frameworks. Neural networks were also chosen as they represent a



(a) Rules



(b) Neural Networks

Figure 5.3: Rules and Neural Network Accuracy

history-based alternative to making recommendations by learning from past user behaviour [163, 98].

To construct a rule-based version of the restaurant recommender application, the 10 study participants also acted as developers. All developers were of M.Sc., Ph.D., or postdoctoral level in computer/engineering related disciplines. Each developer was given the set of context and restaurant properties supported by the application as well as user model cases, which provided knowledge about the preferences of a user. The aim for developers was to construct a set of rules that generate recommendations that match the preferred choice of a user. Rules were implemented in Java using conditional statements. The *EasyNN-plus tool*[4] was used to generate neural networks. A different network was generated for each user using user model cases as training data.

The accuracy of both the rule-based and the neural network approach is determined in the same manner as used for the original recommendation strategy. The context of each test case (user model and new cases) provide the input into both the rule base and neural network systems, and values for restaurant properties that are outputted are compared with actual user selections. The accuracy results for each user for the 15 test cases are averaged to give an accuracy value for each user, and these values are subsequently averaged to give an overall accuracy for both rule-based and neural network approaches. Figure 5.3 illustrates the recommendation accuracy for each user using rule-based and neural network approaches. For comparison, the accuracy of the original strategy is also illustrated. The average recommendation accuracy of all users for each approach is shown in the right-most column in each graph.

A paired t-test was used to compare the original strategy with rule-based and neural network

[4] *http://www.easynn.com/*

Statistic	Value	
	Rules	Neural Network
Null Hypothesis	$\bar{x}$-$\bar{y}$ = 0	$\bar{x}$-$\bar{y}$ = 0
Number of Samples (n)	10	10
Degrees of Freedom (d)	9	9
Original Strategy Sample Mean ($\bar{x}$)	64.9%	64.9%
Rule/Neural Network Sample Mean ($\bar{y}$)	47.0%	45.0%
Sample Mean Difference	17.9%	19.9%
Standard Deviation of Difference (s)	12.4%	8.0%
Paired Students t-value (t)	4.54	7.82
Critical Value ($t_{.99}$)	3.25	3.25
99% Confidence Intervals	5.0%, 30.71%	11.6%, 28.17%

Table 5.4: Summary Statistics for Rule-based and Neural Network Comparison

approaches. Table 5.4 summarises the statistics used for this test. The calculated t-values (4.54 and 7.82) are greater than the 99 percentile Students t distribution ($t_{.99}$) for 9 degrees of freedom (3.25). In addition, the hypothesised accuracy difference of 0 between the approaches does not fall within the 99% confidence intervals. Therefore, the null hypothesis is rejected and we can say with 99% confidence that the original strategy outperforms both a rule-based and neural network approach for providing accurate personalised recommendations in different context.

As shown by these tests, the personalisation approach provides more accurate recommendations than both rule-based and neural network approaches (with 99% confidence). When compared with a rule-based approach, the original strategy is shown to provide recommendations that on average are approximately 18% more accurate. For the set of users that were evaluated, the average difference between accuracies ranged from 5% to 30.71%, which indicates that the original strategy is significantly more accurate than developed rules in some instances but this difference is significantly less in other instances. For example, the personalisation approach was significantly more accurate than rules for User 3. The only instance where a rules-based approach outperformed the original strategy was for User 6. This was found to be the case because the original strategy is shown to have performed poorly (i.e., accuracy of 48%) for new cases for this user, therefore bringing down the overall average accuracy. This indicates the information stored in the user model of this user was not representative of the preferences of the user for new problems. The variance in difference between the accuracy of the original strategy and developed rules is also partly attributed to the ability of developers to effectively implement a set of recommendation rules that captured the preferences of users in different context. To further determine the reasoning for the poor performance of a rule-based approach developers were questioned post implementation[5]. Questions consisted of a set of statements regarding rule-based

[5]illustrated in Appendix B.3.

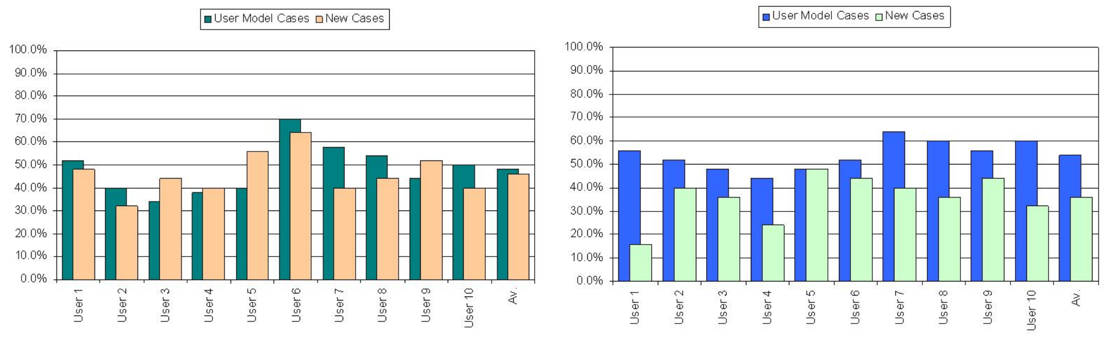
Statements	Likert Scale
It proved difficult to identify the permutations of rules required	4
The system was overall difficult to design and implement	3.7
The system will cater for new cases	1.5
The system captures the requirements of the user	3.4
The system will produce accurate outputs	2.8

Table 5.5: Rule-based Implementation Opinions

implementation using the Likert (five) scale system, where 1 corresponds to *Strongly Disagree* and 5 to *Strongly Agree*. Table 5.5 summarises developer responses. The most relevant findings indicate that developers found it difficult to identify and generalise rules to the context of different recommendation problems and to implement rules that will cater for new problems.

The evaluation results have also shown that the original strategy of the personalisation approach outperformed a neural network approach for all users. On average neural networks were outperformed by between 11% and 28%, and on average by almost 20%. The poorer performance of neural networks is attributed to the problem of over fitting due to the small number of cases available to train each neural network [163]. The problem of over-fitting is reduced by the design of the personalisation approach described in this thesis as cases are filtered by context (as described in Section 3.2.5) to ensure that knowledge discovered during association mining is tailored for the context of the recommendation problem.

Figure 5.4 and Table 5.6 expand on the evaluation results with an illustration of how rules and neural networks performed for user model cases and new cases for each user (following the separate test case evaluation approach). The similar average accuracy of rules for user model and new cases supports the finding that developers aimed to, but found it difficult to, generalise knowledge about user preferences so that they could be applied to the set of different recommendation problems. This finding is also supported by similar standard deviations, which show that the accuracy of rules approach vary by similar amounts when testing the two sets of problems. The problem of bias as a result of having information about user selections does not apply for rules as developers attempted to generalise solutions as opposed to only focusing on user model problems. As neural networks were trained with user model data, it is not surprising that tests with user model cases are more accurate than those of new cases, and therefore increasing the overall average of accuracy values. As shown in Figure 5.3, the biggest difference in accuracy between the original strategy and neural networks is with User 4. This is attributed to the poor performance of neural networks in new cases for this user. Returning to Figure 5.4, neural networks provided the same accuracy values for user model and new problems for User 5. The original strategy also scored highly for User 5 indicating that



(a) Rules (b) Neural Networks

Figure 5.4: Rules and Neural Networks: User Model Cases versus New Cases

Accuracy	Average		Standard Deviation	
	Rules	Neural Networks	Rules	Neural Networks
User Model Cases	48.0%	54.0%	10.9%	6.3%
New Cases	46.0%	36.0%	9.2%	9.7%

Table 5.6: User Model Cases versus New Cases Summary Statistics

the preferences for User 5 are relatively consistent for different recommendation problems. The large difference between user model and new problems for User 1 is unclear given such a difference was not exhibited for both the original strategy and for rules. The black box nature of a neural network approach also makes it difficult to analyse. Further tests for User 1, with increased user model data, would be required to determine the reason for this outcome.

5.3 Context-aware Travel Recommender System

It was described in the Section 3.2.1.1 that the original strategy omits Information Selection and Utility Assignment when evaluating recommendations. Its limitation therefore, is its assumption that users use the same set of context types and place the same relative importance to different context types when making behaviour decisions. The extended recommendation strategy is designed to address this limitation by including Information and Utility Assignment, which provide the functionality that dynamically adjusts the relevance and utility of information to reflect the preferences of the user for different recommendation problems. It is hypothesised that by performing these operations, recommendation accuracy will increase.

This section describes the evaluation of the extended recommendation strategy, which consists of a number of tests designed to assess the accuracy of its generated outputs. Specifically, the accuracy results of the extended strategy are compared with the original strategy, enabling the effectiveness of Information Selection and Utility Assignment techniques and algorithms to be measured. This "before and after" process is commonly used to determine if newly developed algorithms are actually effective in improving recommendation accuracy [118]. The algorithms for Information Selection and Utility Assignment are also individually evaluated by comparing their outputs with user responses.

5.3.1 Application Scenario and Set-up

To assess the accuracy of the extended recommendation strategy, a different, extended user study was conducted, which unlike the previous study, also queried users about information relevance and utility. The personalised context-aware travel recommender system introduced in Section 3.1.2 was developed as the application for this study. This application was developed to make personalised travel (i.e., route and transport) recommendations for users in different context. The implementation of this application also highlights the ability of the personalisation approach to make recommendations for different types of problems using the same process.

As part of the study, 20 participants[6] were asked about 20 different travel problems. Users were queried about the type of route and transport they would take between two locations for a given context[7]. For each problem, users were also queried for the context types they considered relevant and each type's relative importance in their decision. Locations and available routes were selected using an area of Dublin city as shown in Figure 5.5. Four different travel routes (based on time, distance, number of turns, and number of junctions) and three different modes of transport (walking, via bus, via car) were available to users for each case. For each problem, users were also asked if they would change the type of route and transport they had selected if the value of a context type or set of types changed. This provided a total number of 40 cases for each user. Six independent context types are supported, each type had five possible values, equating to 5^6 (i.e., 15625) possible context combinations.

5.3.2 Analysis of Original Recommendation Strategy

The first evaluation performed for this study assessed the accuracy of the original recommendation strategy for the set of travel recommendation problems. This is conducted to provide a baseline accuracy from which the results of the extended strategy can be compared. Unlike the previous study, a

[6]A sample size of 20 was determined using the results of initial users as shown in Appendix A.1.
[7]A example recommendation problem is shown in Appendix B.4.

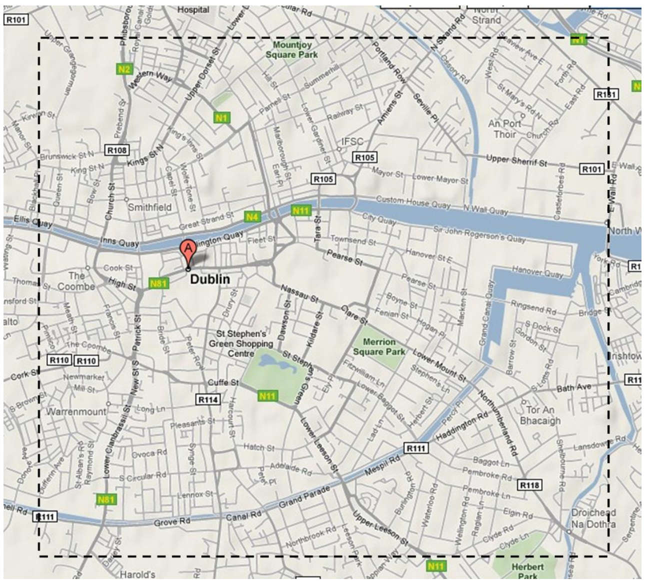
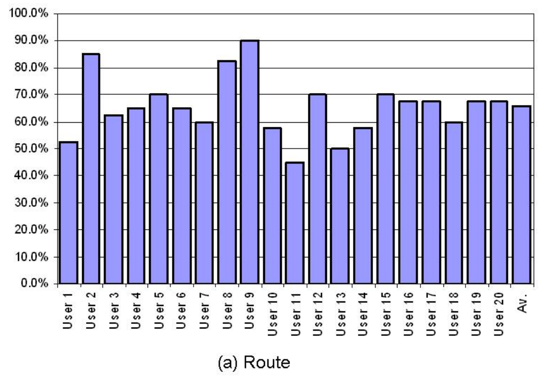


Figure 5.5: Dublin City Coverage

leave one out approach (Section 5.1.3) is adopted here to evaluate recommendation accuracy. Accuracy measures for both route and transport recommendations are acquired and analysed. The evaluation proceeded as follows: the case being tested is removed from the user model and a recommendation is generated by the application using all other cases. Recommendation accuracy is determined by comparing recommended actions with actual user selections stored in the user model. Each case is tested in the same way and accuracies are averaged to determine an overall user accuracy. Figure 5.6 illustrates the accuracy for each user for route and transport recommendations using the original strategy. The average recommendation accuracy of all users for each problem type is also shown in the right-most column.

To assess the usefulness of the strategy, the results are compared against a random recommendation as performed in the previous study (Section 5.2). Table 5.7 shows the statistics used to compare this strategy with a random recommendation. The calculated t-value of 6.25 for route recommendations and 4.09 for transport recommendations is greater than the 99 percentile Students t distribution ($t_{.99}$) for 19 degrees of freedom (2.86). In addition, the hypothesised mean accuracy value of 50 does not fall within the 99% confidence interval values. Therefore, the null hypothesis is rejected and it can be stated that this original recommendation strategy outperforms a random recommendation for both personalised route and transport recommendations with more than 99% confidence.

These results show that the original strategy provides similar levels of accuracy using a leave one out approach as with a separate test case approach in the first user study (i.e., 64.9% previously versus



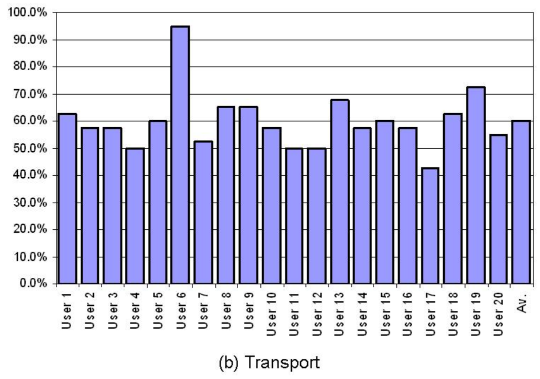


Figure 5.6: Accuracy of Original Strategy for Route and Transport Recommendations

65.6% and 59.8%). However, the results of this study are clearly more accurate than those provided by the original strategy in the first for *new cases* (i.e., those not stored in the user model), which was 56.8%. On analysis of individual user results, it was determined that several high value outliers have very high levels of accuracy which increase the overall average accuracy (i.e., Users 2, User 8, and User 9 for route and User 6 for transport). When these values are removed an average accuracies of 52.8% (for route) and 55.3% (for transport) are calculated, which are more in line with the accuracy for the original strategy for new cases in the first user study and with what was expected. This is because this study considers a larger number of context relationships than the first study (i.e., 15625 versus 96) and as a result there are a larger number of combination of factors that are likely to affect the preferred choice of the user to different degrees. Capturing this knowledge in the user model with a limited number of cases is difficult and limits the application to generalise user model knowledge to the larger number of context combinations that are possible.

The standard deviation is larger than expected at 17% (for route) and 19% (for transport) of the mean value. This result is attributed to the variances that exist relating to the consistency of user decisions. That is, some users (e.g., User 9 for routes and User 6 for transport) are largely consistent with the decisions they make and are likely to choose the same actions for similar problems. This enables the personalisation approach to effectively learn about their behaviour patterns. However, other users (e.g., User 11 for routes and User 17 for transport) are largely inconsistent about their preferred actions, therefore making inferences about their preferred behaviour more difficult. The differences in accuracy values for these types of users are seen to cause the relatively large standard deviation that is exhibited.

Statistic	Value	
	Route	Transport
Null hypothesis	$\bar{x} = 50$	$\bar{x} = 50$
Number of Samples (n)	20	20
Degrees of Freedom (d)	19	19
Sample Mean ($\bar{x}$)	65.6%	59.8%
Standard Deviation (s)	11.1%	10.8%
Students t-value (t)	6.25	4.09
Critical Value ($t_{.99}$)	2.86	2.86
99% Confidence Intervals	58.4%, 72.7%	52.9%, 66.7%

Table 5.7: Summary Statistics for Original Strategy Accuracy

5.3.3 Analysis of Extended Recommendation Strategy

After establishing a baseline accuracy value represented by the results of the original strategy, evaluation was performed to assess the accuracy of the extended recommendation strategy. As described in Section 3.2.1.1, the extended strategy aims to improve on the original strategy by including Information Selection and Utility Assignment techniques and algorithms. The aim is to improve the accuracy of recommendations by providing functionality that adapts the set of relevant context and its utility according to the user problem and context.

The evaluation of the extended recommendation strategy proceeded using the same leave one out process as used for evaluating the original strategy. The one main difference is that recommendations are made with Information Selection and Utility Assignment included in the recommendation process. Figure 5.7 illustrates the accuracy for each user for route and transport recommendations using the extended strategy. The results acquired previously for the original strategy are also included in the illustration. The average recommendation accuracy of all users is also shown in the right-most column. It is shown that for all users, the extended strategy either matches or outperforms the original strategy. The overall average of the difference is 7.7% for route and 4.5% for transport, shown in the right-most column in each graph.

The results of the original and extended strategies are directly compared to statistically assess the effectiveness of the extended strategy. Table 5.8 illustrates the statistics used for the paired t-test. The calculated t-values (5.7 and 3.85) for route and transport recommendations are both greater than the 99 percentile Students t distribution ($t_{.99}$) for 19 degrees of freedom (2.86). In addition, the hypothesised accuracy difference of 0 between the two approaches does not fall within the 99% confidence intervals. Therefore, the null hypothesis is rejected and it can be concluded with 99% confidence that the extended recommendation strategy outperforms the original strategy in providing personalised recommendations to users in different context. This indicates that there is a statistically

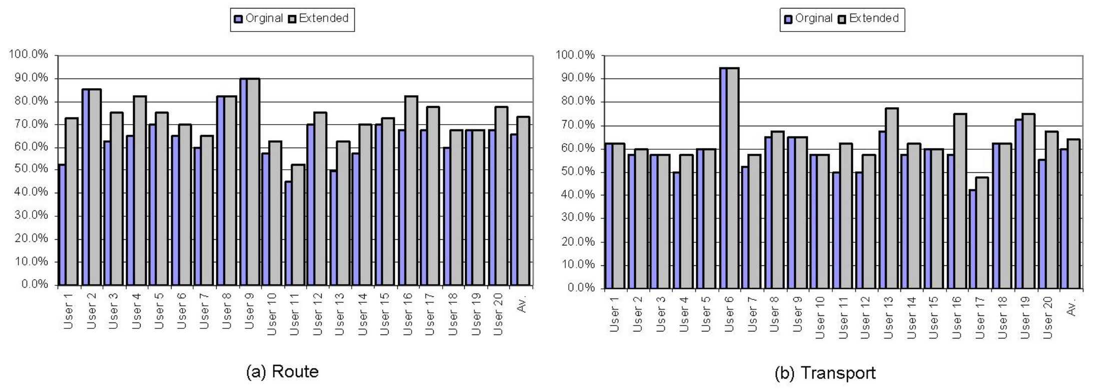


Figure 5.7: Extended Strategy vs Original Strategy Accuracy

significant benefit to using Information Selection and Utility Assignment when making personalised recommendations to context-aware application users.

The results show that the extended strategy outperforms the original strategy by 7.6% for route and by 4.5% for transport recommendations. The standard deviation of the extended strategy is also slightly less for both problem types than the original strategy indicating that the approach is overall slightly more consistent (approximately 1%) at tailoring recommendation decisions to user preferences. The reduced standard deviation and the increase in accuracy (in the majority of users) is attributed to the techniques and algorithms added to the extended approach that facilitate the identification of context relevance and utility that are important in user decisions in contrast to considering the entire set of information with equal utility. This enables recommendations to be made using information that is specifically tailored to the user problem and context, and therefore improves accuracy. These results also further support the finding that users do vary the relevance and utility they attribute to different context types in decisions, and that by facilitating these differences, more accurate recommendations result. For some users (e.g., User 8 and User 9 for route, and User 5 and User 6 for transport), the accuracy for the two strategies is equal. This occurs when users do not attribute different relevance or utility to decisions, and therefore techniques that facilitate the adjustment of relevance and utility have no effect. Consequently, the results indicate that extended approach is more effective for some users than it is for others. Further studies would be required to assess the properties associated with the decision making behaviour of individual users to determine if including functionality for Information Selection and Utility Assignment is beneficial. An investigation of the trade off between the accuracy provided by including these functions and the extra processing resources required for executing these functions would also provide an insight into deciding when Information Selection and

Statistic	Value	
	Route	Transport
Null Hypothesis	$\bar{x}$-$\bar{y} = 0$	$\bar{x}$-$\bar{y} = 0$
Number of Samples (n)	20	20
Degrees of Freedom (d)	19	19
Extended Strategy Sample Mean ($\bar{x}$)	73.2%	64.3%
Extended Strategy Standard Deviation	8.9%	10.1%
Original Strategy Sample Mean ($\bar{y}$)	65.6%	59.8%
Sample Mean Difference	7.6%	4.5%
Standard Deviation of Difference (s)	5.9%	5.2%
Paired Students t-value (t)	5.7	3.85
Critical Value ($t_{.99}$)	2.86	2.86
99% Confidence Intervals	3.8%, 11.45%	1.1%, 7.8%

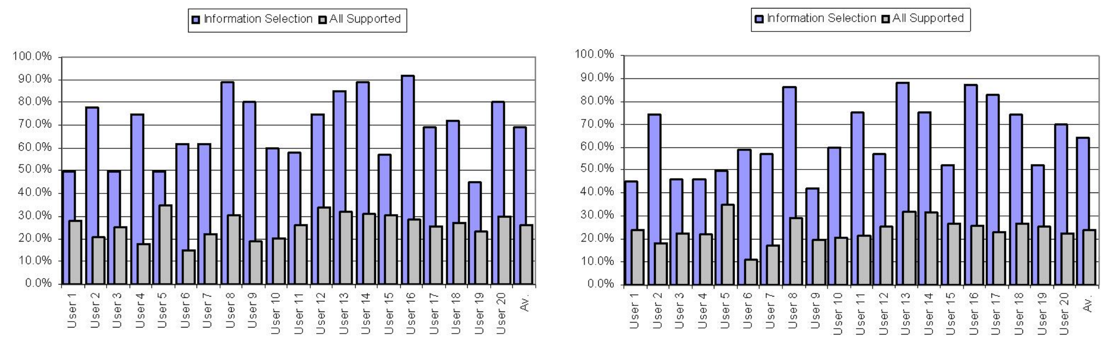
Table 5.8: Summary Statistics for Comparing Extended and Original Strategy Accuracy

Utility Assignment is appropriate. The following two sections (Section 5.3.4 and Section 5.3.5) are designed to further analyse the effect of adjusting information relevance and utility on the accuracy of recommendation decisions.

5.3.4 Analysis of Information Selection

This section describes the evaluation of the Information Selection technique developed as part of the recommendation process proposed in this thesis. As described in Section 3.2.6, this technique is designed to determine the set of context types that should be considered for a given recommendation problem and context. Like other tests, the accuracy of Information Selection is assessed by comparing context types evaluated as relevant by Information Selection with those types users explicitly specified as being relevant for each recommendation problem. User responses to travel recommendation problems highlight that for different context, users consider different context types when making decisions.

The evaluation of Information Selection proceeded as follows: the case being tested is removed from the user model and a recommendation is generated by the application using all other cases. Accuracy is determined by comparing the set of context types in the recommended rule that are evaluated as relevant with those types explicitly specified as relevant by the user. Note that context types not specified as relevant by the user but evaluated as relevant by the application are classed as being incorrect. Each case is tested in the same way and accuracies are averaged to determine an overall user accuracy. Figure 5.8 illustrates the accuracy of Information Selection for each user for both route and transport recommendations. The average recommendation accuracy of all users is also shown in the right-most column.



(a) Route (b) Transport

Figure 5.8: Information Selection Accuracy

Statistic	Value	
	Route	Transport
Null Hypothesis	$\bar{x}$-$\bar{y} = 0$	$\bar{x}$-$\bar{y} = 0$
Number of Samples (n)	20	20
Degrees of Freedom (d)	19	19
Information Selection Sample Mean ($\bar{x}$)	68.9%	63.9%
All Context Relevant Sample Mean ($\bar{y}$)	26.0%	24.0%
Sample Mean Difference	42.8%	39.9%
Standard Deviation of Difference (s)	14.8%	15.0%
Paired Students t-value (t)	12.91	11.85
Critical Value ($t_{.99}$)	2.86	2.86
99% Confidence Intervals	33.3%, 52.3%	30.2%, 49.5%

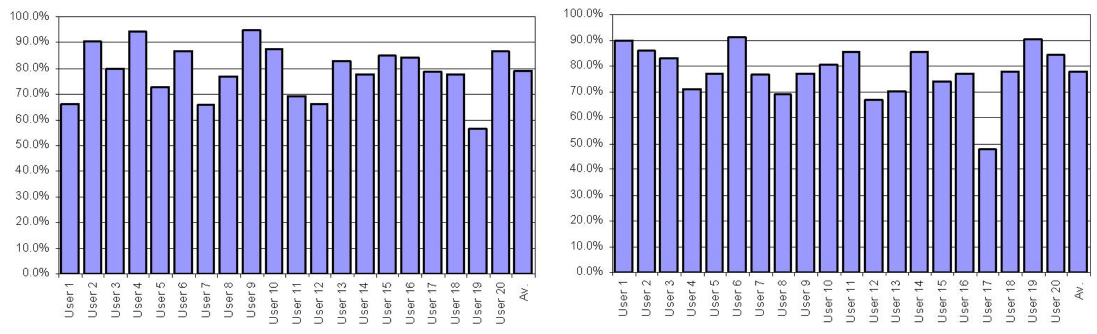
Table 5.9: Summary Statistics for Comparing the Accuracy of Information Selection with All Context Types Relevant

To assess the usefulness of the strategy, the results are compared against an approach which considers all supported context types as relevant. Figure 5.8 illustrates the accuracy values if all supported context types were considered relevant. A paired t-test was used to compare the accuracy of Information Selection with the accuracy if all context types were considered relevant. Table 5.9 illustrates the statistics used for this test. The calculated t-values (12.91 and 11.85) for route and transport recommendations are both greater than the 99 percentile Students t distribution ($t_{.99}$) for 19 degrees of freedom (2.86). In addition, the hypothesised accuracy difference of 0 between the two approaches does not fall within the 99% confidence intervals. Therefore, the null hypothesis is rejected and we can say with 99% confidence that the Information Selection techniques and algorithms proposed in this thesis are more accurate in determining relevant context information than when all supported context types are considered relevant.

The low accuracy results related to an approach that considers all context types as being relevant further illustrates that users do not consider all possible context types when making decisions. The mean value for this approach for the evaluated users, suggests that these users use only about 25% of the context types presented to them when making decisions. This finding applies to all users as shown by the low results associated with this approach. The Information Selection technique developed as part of the personalisation approach is shown to be significantly more accurate at determining context relevance. Specifically, the average improvement of accuracy as a result of adjusting context relevance is around 40%. The relatively large standard deviation of the difference shows that there is significantly large variance in the difference in accuracy of the two approaches indicating that Information Selection was significantly better for some users (e.g., User 8 in route), but the difference is less significant for others (e.g., User 5 in transport). This suggests that some users were consistent in the information they deemed relevant for similar recommendation problems, while others users were less so when making the inference of information relevance more difficult, and therefore less accurate. One issue that warrants further investigation is that for some users, the correlation between the accuracy results of information selection and the accuracy of the extended strategy is unclear. Most obviously is User 6 in transport where the recommendation accuracy is over 90% but the accuracy for Information Selection is about 60%. A possible reason for this that should be investigated is the reliability of responses of certain users. That is, some users may not have based their recommendation choices on the context information they stated. This is consistent with several previous studies [25, 141, 116, 159] as discussed in Chapter 1 (Section 1.2) - that users can be inaccurate and unreliable when explicitly providing information about their preferences. An extension to this evaluation could investigate approaches for accurately determining the context information users considered relevant in decisions without the need for their explicit input before evaluation comparisons with Information Selection are made.

5.3.5 Analysis of Utility Assignment

This section describes the evaluation of the Utility Assignment technique developed as part of the recommendation process proposed in this thesis. As described in Section 3.2.7, this technique is designed to determine the relative importance of context types in recommendation decisions. Accuracy is again measured by comparing utility values inferred by the personalisation approach with utility values gathered from user responses for the set of recommendation problems. In a similar way to Information Selection, user responses showed that users consider different context types with varying levels of importance for different recommendation problems and context.



(a) Route (b) Transport

Figure 5.9: Utility Assignment Accuracy

The evaluation of Utility Assignment proceeded in a similar manner to the evaluation of Information Selection: the case being tested is removed from the user model and a recommendation is generated by the application using all other cases. The utility of relevant context types specified by the user are compared with the utility of corresponding context as evaluated by Utility Assignment to determine a Utility Assignment accuracy. Each case is tested in the same way and accuracies are averaged to determine a user accuracy. Figure 5.9 illustrates the accuracy of Utility Assignment for each user for route and transport recommendations. The average recommendation accuracy of all users is also shown in the right-most column.

As users assigned different values of utility for relevant context types in each problem, comparison with an approach that considered all context types to have equal utility does not provide any meaningful results (as the accuracy of an approach with equal utility, which does to capture the differences in the relative importance of different context types, would equate to 0). Therefore, to assess the usefulness of Utility Assignment, the results are compared against a random assignment of utility. Table 5.10 shows the statistics used to compare Utility Assignment with a random recommendation. The calculated t-value of 12.53 and 12.35 for route and transport recommendations is greater than $t_{.99}$ (2.86) and the hypothesised mean accuracy value of 50 does not fall within the 99% confidence interval values. Therefore, the null hypothesis is rejected and it we conclude that it is 99% certain that Utility Assignment provided by the personalisation approach described in this thesis is more accurate in determining the relative importance of context types than an approach that assigns random utility to context types.

The results show that the Utility Assigned technique developed for the personalisation approach described in this thesis is effective in determining the utility (i.e., around 79%). As indicated by

Statistic	Value	
	Route	Transport
Null hypothesis	$\bar{x} = 50$	$\bar{x} = 50$
Number of Samples (n)	20	20
Degrees of Freedom (d)	19	19
Sample Mean ($\bar{x}$)	79.0%	78.0%
Standard Deviation (s)	10.3%	10.1%
Students t-value (t)	12.53	12.35
Critical Value ($t_{.99}$)	2.86	2.86
99% Confidence Intervals	72.4%, 85.6%	71.5%, 84.5%

Table 5.10: Summary Statistics for Utility Assignment Accuracy

confidence intervals, accuracy as high as 85% is also possible. The standard deviation of around 10% is expected given the standard deviation of previous results. Most users provide a similar level of Utility Assignment accuracy but two users seem to break this trend (User 17 for transport and User 19 for route). As before with Information Selection, this low accuracy is likely to be a result of either high variance associated with user responses about utility or the unreliability of user responses. The low accuracy of User 17 for transport recommendations explains the low accuracy of the extended strategy for that user, despite User 17 having a high accuracy for relevance. This either indicates that decisions about relevance and utility both have to be sufficiently accurate in order for accurate recommendations to be made or that the user was unreliable with their responses about information relevance. The low accuracy of User 19 for route is also interesting because the low utility accuracy coupled with the low relevance accuracy of this user has not significantly affected the accuracy of recommendations made with the extended strategy. Further studies are therefore required to investigate the reliability of user responses about relevance and utility information for this user.

5.3.6 Comparisons

To further assess the effectiveness of the extended recommendation strategy, the evaluation also compares its accuracy against the accuracy of similar alternative recommendation approaches that are currently adopted in the state of the art. The first evaluation compared the original strategy with approaches supported in context-aware applications. The comparison here in contrast, selects two widely used approaches in state of the art recommendation systems for comparison: preference models and case-based reasoning (CBR). These two approaches are analogous to the vector space and history based models used to represent user preferences discussed in Chapter 2. As the user model developed as part of the personalisation approach described in this thesis borrows properties from both vector space and history based models (Section 3.2.3), a comparison with a preference model and a CBR approach was deemed relevant. Given the design of the personalisation approach to include techniques and algorithms dynamically adapt user preferences, information relevance, and information utility to different user problems and context, it is assumed that this approach will provide more accurate results than a preference model and a CBR approach.

A preference model based on the shortest distance was generated for each user for route recommendations, which is the commonly adopted policy for existing route recommendation applications (e.g., Satellite/GPS navigation devices). The preferred transport mode specified by each user at the start of the study represented the preferred mode of transport for each user. For CBR, a similarity function was defined, consisting of all context types supported by the application, with each type assigned equal utility. A learning step that filters or prioritises decision making information is not supported by this CBR implementation. To generate recommendations, each case in the user model, not including the test case, is assessed to determine the case most similar to the current problem using an exact match comparison. The action associated with the most similar case is recommended. If CBR retrieves more than one case with competing candidate choices, then a random selection is taken from those retrieved cases.

The accuracy of both the user's preference model and CBR is determined in a similar manner as other tests. The context of each test case is read and represents the context of the current recommendation problem. Recommendations made using each approach is then compared with actual user selections and a recommendation accuracy for each user and overall average is calculated. Figure 5.10 and Figure 5.11 illustrate the accuracy for each user for route and transport recommendations using the preference model and the CBR approach respectively. The accuracy of the extended recommendation strategy is also shown in each.

The results of the preference model and the CBR approach are directly compared with the extended strategy to statistically determine, using the paired t-test, if there is any difference in their effectiveness for providing accurate personalised recommendations. Table 5.11 illustrates the statistics used for comparing the preference model approach with the extended recommendation strategy proposed in this thesis. The calculated t-values (6.38 and 4.22) for route and transport recommendations are both greater than the 99 percentile Students t distribution ($t_{.99}$) for 19 degrees of freedom (2.86). In addition, the hypothesised accuracy difference of 0 between the two approaches does not fall within the 99% confidence intervals. Therefore, the null hypothesis is rejected and it can be concluded with 99% confidence that the extended recommendation strategy outperforms the preference model approach for providing personalised recommendations to users in different context.

Similarly, Table 5.12 illustrates the statistics used for comparing CBR with the extended recom-

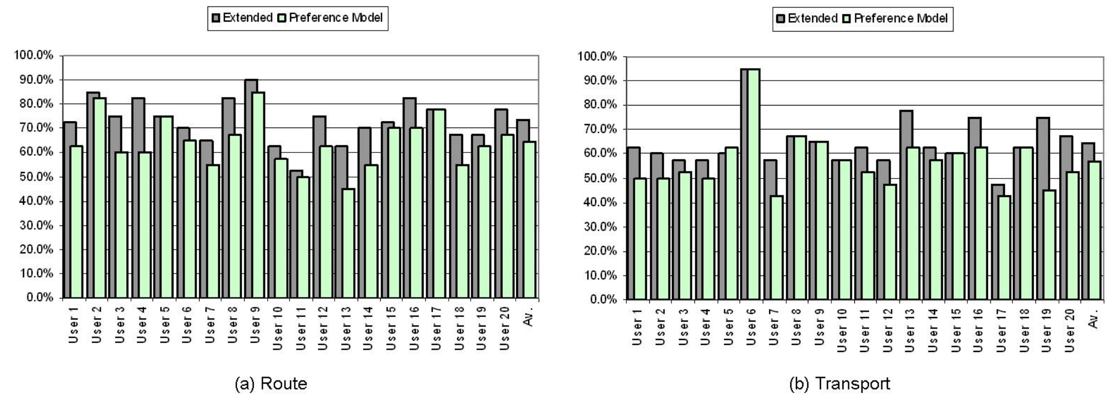


Figure 5.10: Extended Strategy vs Preference Model Accuracy

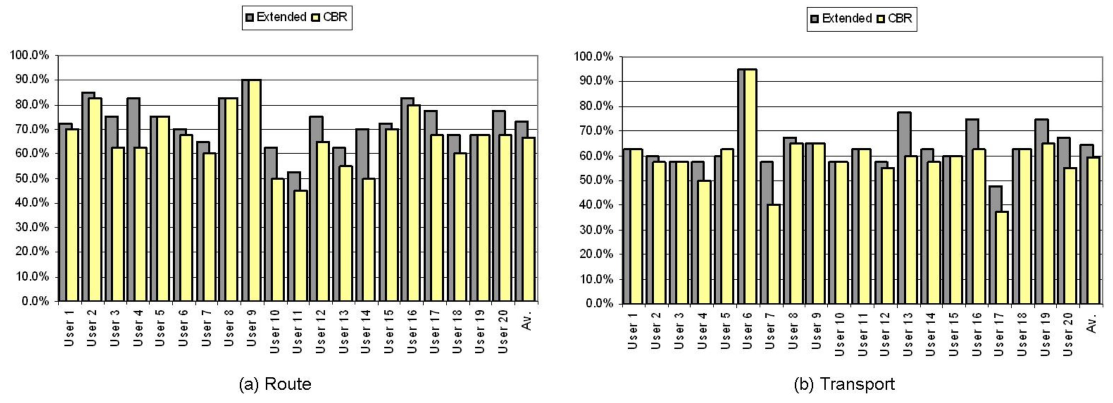


Figure 5.11: Extended Strategy vs CBR Accuracy

mendation strategy. Again, the calculated t-values (4.88 and 3.46) for route and transport recommendations are both greater than $t_{.99}$ (2.86) and the hypothesised accuracy difference of 0 between the two approaches also does not fall within the 99% confidence intervals. We can therefore conclude with 99% confidence that the extended strategy also outperforms a CBR approach for making personalised recommendations to users in different context. As discussed in the analysis of the extended recommendation strategy, recommendation accuracy was significantly improved by adjusting the relevant and utility of context types in recommendation decisions. It is therefore reasoned that CBR did not perform as well as the extended recommendation strategy as it considered all context types with equal utility.

As shown by these tests, the accuracy provided by the extended strategy outperformed that of both preference models and CBR. When compared with preference models, the extended strategy

Statistic	Value	
	Route	Transport
Null Hypothesis	$\bar{x}$-$\bar{y}$ = 0	$\bar{x}$-$\bar{y}$ = 0
Number of Samples (n)	20	20
Degrees of Freedom (d)	19	19
Extended Strategy Sample Mean ($\bar{x}$)	73.2%	64.3%
Preference Model Sample Mean ($\bar{y}$)	64.2%	56.8%
Sample Mean Difference	9.0%	7.5%
Standard Deviation of Difference (s)	6.3%	7.9%
Paired Students t-value (t)	6.38	4.22
Critical Value ($t_{.99}$)	2.86	2.86
99% Confidence Intervals	4.9%, 13.0%	2.4%, 12.5%

Table 5.11: Summary Statistics for Comparing the Accuracy of Preference Models with the Extended Strategy

Statistic	Value	
	Route	Transport
Null Hypothesis	$\bar{x}$-$\bar{y}$ = 0	$\bar{x}$-$\bar{y}$ = 0
Number of Samples (n)	20	20
Degrees of Freedom (d)	19	19
Extended Strategy Sample Mean ($\bar{x}$)	73.2%	64.3%
CBR Sample Mean ($\bar{y}$)	66.5%	59.5%
Sample Mean Difference	6.7%	4.8%
Standard Deviation of Difference (s)	6.1%	6.3%
Paired Students t-value (t)	4.88	3.46
Critical Value ($t_{.99}$)	2.86	2.86
99% Confidence Intervals	2.7%, 10.7%	0.8%, 8.9%

Table 5.12: Summary Statistics for Comparing the Accuracy of CBR with the Extended Strategy

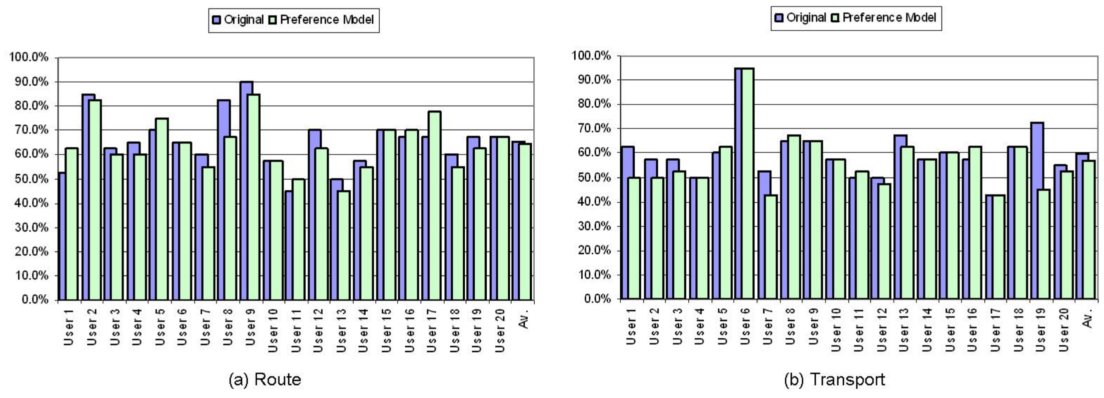


Figure 5.12: Original Strategy vs Preference Model Accuracy

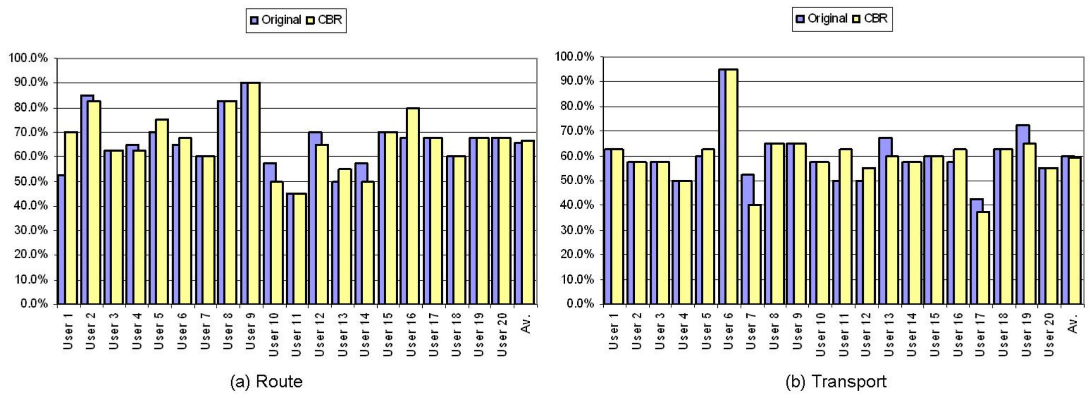


Figure 5.13: Original Strategy vs CBR Accuracy

improves recommendations on average by 9% for routes and 7.5% for transport. The improvement over CBR is slightly less at 6.7% and 4.8% respectively. The standard deviation of the differences in both comparisons are as expected and are in line with previous results, and attributed to the variance in the user responses about information relevance and utility.

The set of accuracy values for preference models and CBR are similar to those provided by the original strategy (Figure 5.12 and Figure 5.13). In particular, the results of CBR are very similar to those for the original strategy, and both these two approaches provided slightly more accurate results than preference models (approximately 3% more accurate). When statistically analysed the accuracy of the original strategy versus both preference models and CBR are not statistically significantly different. When comparing the original strategy with preference models, paired t-values of 1.04 (for route) and 1.85 (for transport) were computed. Similarly, when comparing the original strategy with CBR, paired t-values of -0.67 (for route) and 0.33 (for transport) were calculated. All paired t-values are less that critical value of 2.86 indicating with 99% confidence that there is no statistically significant difference in accuracy between the original strategy and preference models and between the original strategy and CBR. This finding is not surprising as the original strategy and CBR are both based on user model cases and both do not adapt information relevance and utility when making recommendations. They also both perform slightly better than preference models as preferences models, as well as not adapting relevance and utility, also do not adapt recommendations to different context. As a result, the difference in accuracy between the extended strategy when compared with preference models and CBR are as expected. The reasons for the difference are similar to those described when comparing the extended strategy with the original strategy (Section 5.3.3) - the extended strategy facilitates the adjustment of relevance and utility for different recommendation problems. In addition, the results show that the extended strategy provides more accurate recommendations for some users than it does for others. This, as previously described, is a consequence of how different users make decisions. Specifically, users who vary the relevance and utility of information in decisions are more likely to benefit in contrast to users who consider the same set of information with the same utility for different problems. Further studies are required to effectively assess the decision making patterns of individual users to determine when Information Selection and Utility Assignment would be beneficial and the trade off between any improved accuracy provided versus the extra processing resources required.

5.4 Analysis of Different User Model Sizes

A well-known problem associated with history based models is the lack of user model data, which limits the ability of applications to generalise to new recommendation problems [118, 103, 10]. The lack of data also means that algorithms for Information Selection and Utility Assignment may tend to over-fit the data [118]. Although over-fitting is unavoidable due to the lack of data, our previous evaluation results indicate that dynamically determining information relevance and utility using these two techniques can improve the accuracy of recommendations. A set of experiments are conducted to test if recommendation accuracy can be improved by increasing user model size, which would reduce the effects of the lack of data in user models (i.e., problems with generalisation and over-fitting) on recommendation accuracy.

To generate user models of different sizes, cases are simulated for a set of users by defining artificial preference models that represent their preferences. A separate preference model is created for each unique context combination, simulating a users preference for that context. The context aware travel recommender system for making route recommendations was reused for this evaluation. The section of Dublin city, as shown in Figure 5.5 was implemented as a state space model where each state represents a main road junction and links between states represent route segments (i.e., roads) between junctions. Links also contain information about traversal time and distance. A users preference model for each context is then defined by assigning random weights to route segments. The cost of the route segment is computed by the following cost function:

$$Cost(segment) = length(segment) * weight(segment)$$

The cost indicates the inverse desirability of a particular route segment for a particular user (i.e., the higher the cost of a route segment, the less desirable it is to a user). Consequently, route segments with high weighting will result in a higher cost and therefore a lower desirability as compared to route segments with a low weight. The overall desirability of a route is made up of the cost of each individual route segment included in that route. Using this approach, user models, for each individual user, can be generated using a specific cost function generated for that user along with the A* algorithm. A user model is generated using the A* algorithm and cost function, ensuring that generated routes will minimise the user's cost function, and therefore represent the preferred routes of that user. Routes are mapped to a particular type (i.e., time, distance, number of turns, and number of junctions) using information about alternative routes, traversal time, and route distances stored in the domain model. A similar approach to artificially generating dummy user preference models is adopted by Smyth and McGinty [170] and by Rogers and Langley [159] amongst others. The approach described here is distinguished from those previous as a separate preference model is generated for each user for each unique combination of context values.

Using this user preference model, a set of travel route recommendation problems can be generated. This process proceeded as follows: A random set of values is generated for the set of supported context types (as outlined in Section 5.3.1). If no preference model currently exist for that combination of context values, then a new preference model is generated by randomly assigning values to each route segment in the domain model. A random start and end location is then generated, and the preferred user route is evaluated using the generated preference model and A* algorithm. The combination of context values as represented by the preference model and the route type evaluated for each start and end location are stored as a case in the user model. This process is repeated depending on the number of cases to include in the user model.

As part of this evaluation, the accuracy of the extended recommendation strategy for user model sizes of 50, 100, 200, 500 and 1000 cases were assessed. For each user model size category, a user model for 20 different users was generated. The recommendation accuracy for each user and their user model was measured in the same manner as previous leave one out evaluations. That is, for each user the case being tested is removed from the user model and a recommendation is generated by the application using all other cases. Recommendation accuracy is determined by comparing recommended actions with generated user selections stored in each user model case. Each case is tested in the same way and accuracies are averaged to determine a user accuracy.

Figure 5.14 and Table 5.13 illustrates the summarised accuracy values and statistics for each user model size category[8]. The results shown suggest that recommendation accuracy increases as user model size increases. The variability in accuracy is also reduced with increases in user model size.

To statistically assess the effect of user model size on accuracy, the LSD value based on the F-distribution as described in Section 5.1.4 is used. Using the formula described in Section 5.1.4, an LSD value of 9.6 was calculated on the accuracy results. The LSD value shows with 99% certainty that a user model size of 50 is statistically less accurate than other user model sizes (as their difference is greater than the LSD value of 9.6). However, despite the increases in accuracy for other user model sizes, these increases are concluded as being not statistically significant.

The initial, statistically significant, increase in accuracy when user model size increased from 50 to 100 indicates that user model size does affect recommendation accuracy. The results also indicate that there is a steady increase in accuracy as user model size increases, and that the variability in accuracy is reduced with each increase. These results support our initial theory that increasing user model size provides applications with more knowledge so that they can generalise to new problems and reduce the problem of over fitting. However, statistically significant differences in accuracy with other user model size increases were expected. The steady increase in accuracy as user model size increases does suggest that at some point the difference will become statistically significant, although the improvements in accuracy are not as significant as expected. A possible reason for this relates to the personalisation configuration settings described in Section 4.3 and 5.1.1. As previously discussed, these values are configured depending on the properties of user model data. Less than expected improvements could be a result of ineffectively setting these values. Specifically, the current configuration settings may not facilitate the discovery of relevant knowledge in larger user model sizes. For example, the generation of more associations rules and more restricted filtering settings may be required to ensure that more knowledge is generated and more effectively identified for each recommendation problem.

[8]Appendix A.3 illustrates the accuracy values of individual users in each category.

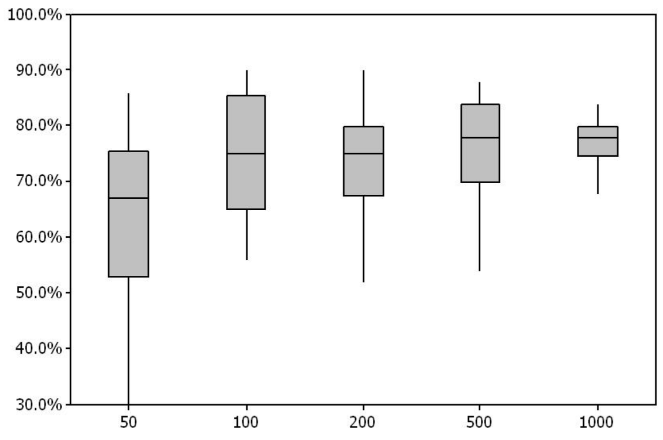


Figure 5.14: User Model Sizes Accuracy

User Model Size	50	100	200	500	1000
Mean ($\bar{x}$)	64.2%	73.4%	74.8%	76.5%	77.6%
Standard Deviation (s)	15.7%	15.3%	9.7%	8.8%	5.3%
Sample Size	20	20	20	20	20

Table 5.13: Summary Statistics for the Accuracy of Different User Model Sizes

Investigations into the adjustment of these values and their effect on different user model sizes would provide relevant knowledge as to the appropriate configuration settings for different user model sizes. An approach to dynamically adjusting these values depending on the context and preferences of users could also be investigated (and planned for future work - Section 6.2.2). These investigations would focus on determining whether more accurately tailored settings would be more effective in facilitating the discovery and identification of relevant user preference and decision making behaviour stored in larger user models.

5.5 Analysis of a Different Number of Supported Context Relationships

A number of simulated experiments were also conducted to assess the accuracy of the extended recommendation strategy for different numbers of supported context relationships. The accuracy of the recommendation strategy for 10, 15, 20, 25, and 30 independent types, each with 5 possible values is evaluated. Table 5.14 illustrates the number of distinct context relationships possible for the

# Context Types	Possible Context Combinations
10	5^{10}
15	5^{15}
20	5^{20}
25	5^{25}
30	5^{30}

Table 5.14: Context Types and Number of Possible Combinations

different values of context types supported. A user model containing 100 test cases are generated for each user in each category using the same process (using generated user preference models as described for analysing different user model sizes- Section 5.4) to ensure that results can be compared fairly. The main difference here is that the user model size remains the same, with the number of context types included in user model cases changing.

Figure 5.15 illustrates the summarised accuracy values for each user model size category[9]. The box plot diagram illustrated in Figure 5.15 shows that recommendation accuracy decreases as the number of context types and context relationships supported increases. To statistically assess the effect of the number of supported context types on accuracy, an LSD value of 6.72 was calculated on the accuracy results. Table 5.15 summarises the statistics for the accuracy for the different number of context types supported. The mean values for each category illustrated in Table 5.15 suggests that recommendation accuracy decreases as the number of context types increases. However, when assessed using the calculated LSD value, it can be stated with 99% confidence that the difference in accuracy for the different number of context types supported are not statistically significantly different. That is, the current data suggests that the personalisation approach described in this thesis is as accurate for making recommendations with 10 context types, each with 5 possible values, as it is for making recommendations with 30 different context types.

Although differences in accuracy are not analysed as being statistically significant, the steady decrease in accuracy as the number of context types increases indicates that further increases to the number of context types supported will have a statistically significant effect on recommendation accuracy. Given that the user model size remains constant for each category for this set of tests, these results were expected. As the user model size remains constant, the personalisation approach is unable to capture knowledge about the increasing number of context types that may affect recommendation decisions. Consequently, the effectiveness of generalising to accommodate problems with large numbers of possible context combinations diminishes, resulting in reduced accuracy. There are two possible solutions worthy of further investigation that may facilitate the recommendation approach to maintain

[9]Appendix A.4 illustrates the accuracy values of individual users in each category.

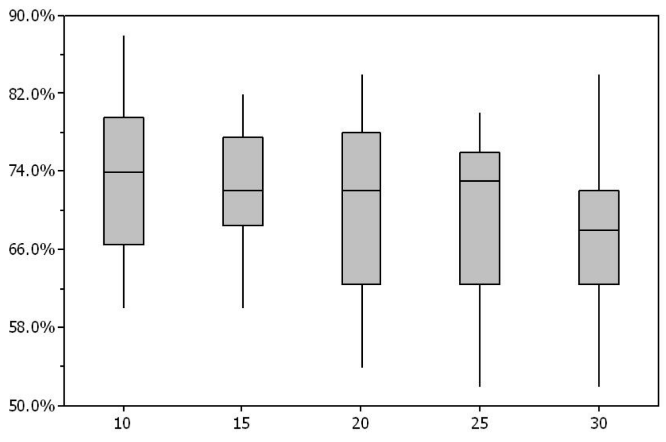


Figure 5.15: Context Types Supported Accuracy

# Context Types Supported	10	15	20	25	30
Mean ($\bar{x}$)	73.3%	73.0%	70.7%	70.3%	67.1%
Standard Deviation (s)	7.4%	7.3%	8.6%	7.7%	9.1%
User Model Size	100	100	100	100	100
Sample Size	20	20	20	20	20

Table 5.15: Summary Statistics for the Accuracy of Different Numbers of Supported Context Types

recommendation accuracy as the number of context types supported increases. One possible solution is to increase user model size as the number of context types increases. It is reasoned from the results in Section 5.4 that increases in user model size can lead to improved accuracy as larger user models are more likely to contain more knowledge about the different possible context combinations and corresponding user preference. A second possible solution is to alter the configuration values outlined in Section 5.1.1. Specifically, stricter settings for filtering user model cases and association rules so that only those cases or rules that match a larger number of context values, are retained (the current filtering configuration value is 1, which informs applications to retain any cases and rules that match *one* of the values of the current context). Given the larger number of possible context combinations as the number of context types supported increases, stricter filtering settings would facilitate applications to be more focused on those context types that are relevant.

5.6 Evaluation Summary

This chapter has described the evaluation of various aspects of the personalisation approach described in this thesis. Two case studies, consisting of real users and different problem domains, and a set of simulated experiments were conducted to assess the accuracy of techniques and algorithms developed in this thesis for making personalised recommendations in different context. Student t-tests and the F-ratio test were used to statistically analyse results.

The evaluation of the original recommendation strategy illustrates that its accuracy ranges from 59%-66% for the set of recommendation problems tested. When statistically compared against random, rule-based, and neural network approaches, the original strategy is statistically more accurate in providing personalised recommendations to users in a context-aware domain. Issues regarding the separate test case evaluation, that may have biased results, were highlighted and reasons for the poorer performance of rules and neural networks were discussed. Results for the original strategy in the second user study using a leave one out approach was shown to have similar results as recommendations made on new cases in the first study. When compared with the extended recommendation strategy, the extended strategy was shown to provide more accurate recommendations averaging 73.2% (for routes) and 64.3% (for transport) for the collection of evaluated users. This shows that overall, techniques for Information Selection and Utility assignment that adapt the relevance and utility of information for different problems can lead to more accurate results. Analysis showed that the extended strategy was more beneficial for some users and had no affect on the accuracy of others indicating that users vary in the degree in which they attribute different relevance or utility to decisions. Further investigations are necessary to analyse individual user behaviour patterns to determine when the dynamic adjustment of relevance and utility is beneficial. Specific tests focused on Information Selection and Utility Assignment showed that Information Selection is statistically significantly better than considering all context types in recommendation decisions with an average accuracy difference of 39% (for routes) and 43% (for transport) and that Utility Assignment is statistically more accurate than randomly assigning utility values to context types. The results of these tests for some users were difficult to reason about due to their inconsistency in relation to the recommendation accuracy values gathered for those users and further investigation is required to quantify the accuracy and reliability of explicit user responses about relevance and utility. Evaluation of preference models and CBR show that they are also less accurate than the extended strategy for the majority of users. The results for these two approaches were not statistically different for those acquired for the original strategy, indicating that the extended strategy is overall more accurate as it dynamically adjusts context relevance and utility. Finally, simulated experiments showed that recommendation accuracy increases as user model sizes increase and decreases as the number of context types supported increases. These results are as expected and various approaches to further improve recommendation accuracy for these two evaluation categories are suggested for future investigation (such as the adjustment of configuration settings).

The following chapter concludes with a summary of the most significant contributions to the state of the art of this thesis and discusses research issues that remain open for future work.

Chapter 6

Conclusions and Future Work

This thesis describes the design and implementation of a novel approach to supporting accurate personalised recommendations in context-aware applications. The approach is developed as a multi-stage recommendation process, which consists of techniques and algorithms that are designed to minimise the need for developers and users to explicitly define and maintain recommendation decision making information. Specifically, the approach supports the dynamic and implicit identification of relevant relationships between context and user preferences, which provides the knowledge from which user preferences, relevant information, information utility, and the ranking of candidate choices for different problems and context is determined. This chapter summarises the achievements of the work and its contributions to the state of the art, and concludes with a discussion of the potential areas for future work.

6.1 Achievements

An analysis of the state of the art personalisation techniques in context-aware applications and frameworks highlighted two main limitations that motivated the work presented in this thesis. Firstly, these techniques rely on human input and knowledge to identify, relate, and prioritise relationships between different context and user preferences to ensure correct personalised behaviour - a task which is time-consuming, complex, and error-prone particularly when large numbers of possible relationships exist between data. Secondly, existing techniques rely on the definition of algorithms that are inherently static, and therefore rely on explicit updates to ensure that the accuracy of recommendations is maintained despite the presence of new or changing information relationships, relevance, or utility. While existing personalisation techniques discussed in Chapter 2 can be used to develop personalised systems in different domains, these limitations restrict applications in effectively making personalised recommendations to users in a context-aware environment, where the user's surrounding information, on which their preferred behaviour is dependant on, is constantly changing. To take advantage of the opportunities presented by the emergence of more sophisticated sensor devices and the greater availability of context data, a new personalisation approach that can dynamically identify and evolve relevant relationships between context and user preferences, while minimising the reliance on human expertise and input is necessary.

This thesis presented a personalisation approach designed to address the issues with state of the art personalisation techniques discussed in Chapter 2. A primary challenge of this work is to provide techniques and algorithms that support applications in dynamically and implicitly discovering relevant relationships between context and user preferences. The ability to accurately identify relevant relationships facilitates applications with making accurate personalised recommendations to users for different recommendation problems and context without the need for explicit human input. Chapter 3 described how this approach was designed using the *Three Examples* design methodology. The incorporation of the approach with the Hermes application framework supports the acquisition of up-to-date context and user interaction data, which are used to maintain a model of the user's environment and their preferences. Using this acquired data, user preferences can be implicitly inferred at run-time for different problems and context using a form of association mining that both discovers relationships between context and user actions and filters these relationships to determine those that are most relevant and accurate for the current problem. This approach also facilitates the implementation of applications that support a large number of context and the identification of changing relationships that may occur over time. The approach also consisted of techniques designed to dynamically adapt information relevance and information utility, which facilitates applications to automatically tailor relevant decision making information depending on the problem and context of the user. Techniques and algorithms designed for these two functions use relevant relationships between context and user actions as input and are easily adaptable to identify relevant information for different types of recommendation problems. Candidate recommendations generated from inferred user preferences are ranked based on knowledge about information relevance and utility. The ranking technique that has been adopted facilitates candidate choices to be ranked along multiple dimensions that have varying levels of relative importance. The combination of techniques and algorithms developed as part of the personalisation approach proposed in this thesis has provided greater support for personalisation in context-aware applications. Specifically, the reliance on explicit human knowledge and input is minimised, facilitating the development of personalised context-aware applications that can consider large number of relationships between context and user preferences and can automatically adapt to identify new and changing relationships that exist between this data.

The implementation of this personalisation approach was described in Chapter 4. The set of techniques and algorithms of the approach were implemented in a manner that facilitates both extensibility, to support new types of recommendation problems and context, and customisability to tailor the run-time execution of the approach for different applications. This facilitates the approach to be re-used and configured for different recommendation problems and different sets of context data depending on the requirements of the application and domain.

The evaluation of the personalisation approach was described in Chapter 5. Two user case studies were conducted and the proposed approach was compared with actual user responses for different types of recommendation problems. Two evaluation strategies were assessed. The *original* strategy accuracy provided an average accuracy ranging from 59% - 66% (for different recommendation problems) and was shown to outperform rule-based and neural network approaches by between 8% - 12%. A post-implementation study showed that improved accuracy over a rule-based approach was due to the difficulty developers experienced when attempting to effectively implement a set of recommendation rules that captured the preferences of users in different context. The poorer performance of neural networks was due to the small number of cases available to train each neural network. The *extended* strategy overall outperformed this original strategy by an average of between 4% - 7% and provided a mean accuracy value ranging from 64% to 73%. Preference model and CBR approaches provided accuracy values that were not statistically different to those for the original strategy, and both provided less accurate results than the extended strategy. The overall higher accuracy of the extended strategy for making personalised recommendations shows the advantage of supporting techniques that adapt relevant information and information utility for different user problems and context. However, the results did show that the extended strategy provided more accurate recommendations for some users than it did for others. This shows that some users vary the relevance and utility of context information they use when making decisions more than other users and further investigation is required to analyse the decision making behaviour of individual users to determine exactly when adjusting relevance and utility will lead to improved recommendation accuracy. Specific tests focused on Information Selection and Utility Assignment showed that Information Selection is statistically more accurate than considering all context types in recommendation decisions with an average accuracy difference of 39% (for routes) and 43% (for transport) and that Utility Assignment is statistically more accurate than randomly assigning utility values to context types. These results also highlighted that users do not consider all context information provided to them when making decisions nor do they attribute equal relative importance to each. The evaluation also investigated the effect of different user models sizes and different number of context types supported by an application on recommendation accuracy, which showed that recommendation accuracy increases as user model sizes increase, but decreases as the number of supported context types increases. Several possible approaches aimed at maintaining or improving the accuracy of recommendations when the number of supported context types increase were suggested.

In summary, the research presented in this thesis has focused on investigating the provision of techniques and algorithms that support context-aware applications in making accurate personalised recommendations to users while minimising the need to explicitly pre-define information about: relationships between context, user preferences, and behaviour; relevant decision making information for different problems and context; and the utility of relevant information.

The main contributions of this thesis are summarised as:

- An overview of context-aware personalised applications with respect to the techniques they use to evaluate candidate recommendation choices. Systems are evaluated with particular focus on their provisions for determining the effect of different values of context on user preferences, adjusting the relevance and utility of information for different recommendation problems, ranking candidate choices using multiple decision making features, and supporting large numbers of context relationships.

- A customisable multi-stage recommendation approach that provides a set of techniques and algorithms, which eliminates the dependency on developers to explicitly specify relevant relationships between context and user preferences. Two developed recommendation strategies, which differ in the set of techniques and algorithms they include, show that this approach provides personalised recommendations for different user problems and context with an average accuracy of between 59% - 66% for the original strategy and 64% to 73% for the extended strategy.

- A set of techniques and algorithms to support applications with *specific* personalisation functions. Specific techniques and algorithms for adjusting information relevance and utility to specific problems and context were shown to be statistically significantly better at determining relevant information (by an average of between 39-43%) when compared against an approach that considers all context types as relevant and also statistically significantly better at determining the utility of context than randomly assigning utility values to different context.

- Two case studies involving real users showed that adjusting the relevance and utility of information provided more accurate recommendations for some users than it did for others, highlighting that different users vary in the relevance and relative importance they attribute to different context to different degrees. This also shows that techniques and algorithms for adjusting relevance and utility are beneficial for some users but are less so or of no benefit for others.

- A comparison of the personalisation proposed in this thesis with several of the most relevant recommendation approaches (random, rules, preference models, case-based reasoning, and neural networks) showed the comparative performance of each when providing recommendations that match the preferred choice of users for different context. Difficulties associated with identifying and generalising context relationships caused the poorer performance of rule-bases, while the lack of training data affected the accuracy of neural networks. Preference models and CBR has similar levels of accuracy as the original strategy and provided less accurate results than the extended strategy for the majority of users as they do not support the adjustment of context relevance and utility for different recommendation problems.

6.2 Future Work

Throughout the development of the personalisation approach described in this thesis, a number of issues worthy of further investigation were identified. This section outlines the key areas identified for future work. This work relates to: extending the evaluation, dynamic adaptation of configuration data; utility, temporal, and sequential data; context uncertainty; and user-application scrutiny.

6.2.1 Evaluation Extensions

The evaluation described in the previous chapter highlighted several findings that are worthy of further investigation. Firstly, it was shown that the extended strategy of the personalisation approach, which adapts information relevance and utility, provides more accurate results for some users than it did for others. An extension of the evaluation would investigate the cause of this, such as an in-depth analysis of the decision making behaviour of individual users to determine when the extended strategy of the personalisation approach is beneficial. Secondly, the results of several users across the set of tests did not directly correlate and further tests to assess the accuracy and reliability of the responses from these users about relevance and utility is planned. Other possible evaluation extensions could assess the performance of the personalisation approach for other quality measures as outlined in the previous chapter such as trust, transparency, and efficiency/latency of decision making.

6.2.2 Dynamic Adaptation of Configuration Data

The personalisation approach described in this thesis enables developers to explicitly configure the execution of its techniques according to the requirements of the application and the user (Section 4.3). However, these properties are currently required to be statically defined and therefore do not adapt to different user problems or context. An interesting direction for future work would be to investigate a means that dynamically adapts this configuration information in order to optimise the effectiveness of the personalisation approach for different situations and further minimise the need for explicit human input. As an example, the number of rules generated as part of Implicit Preference Determination could be automatically adjusted depending on the state of the user's device. Also for different types of problems, the application may automatically learn an optimal number of rules to compare in order to accurately determine relevant context or automatically customise the number of recommendations presented to the user depending on their context. Further research is necessary to determine if dynamic adaptation of personalisation configuration data would lead to improved recommendation decisions.

6.2.3 Utility, Temporal, and Sequential Data

The personalisation approach described in this thesis, utilises relationships between context and user choices, discovered from past user interactions, to make behaviour recommendations tailored to the user's current problem and context. As discussed in Section 3.2.7, the approach is designed to adjust the utility value inferred for different sets of context using information about utility values in past user decisions stored as weights in the user model. However, the current design only the adjustment of utilities based on a pre-defined threshold value (i.e., if the difference between the inferred utility and utility stored in past cases is greater than this threshold value, then the inferred value is modified so that the difference no longer exceeds this threshold). An interesting direction for future work would be to investigate techniques that make greater use of utility values stored in the user model with the aim of improving the accuracy of generated recommendations. Specifically, a process for mining utility values associated with different context types or context sets could be developed using the existing association discovery technique to determine if useful context utility knowledge, that would improve the accuracy of recommendation decisions, can be inferred.

The design of the personalisation approach also does not currently support the identification of temporal or sequential information associated with past user interactions. Temporal and sequential data are seen as useful as they not only capture what action a user takes, but also when those actions are taken and in what order [153, 57, 133]. Therefore, an interesting extension to the personalisation approach would be to incorporate techniques that support temporal and sequential data, which will facilitate applications to proactively make recommendations (using temporal knowledge about what the users is likely to request next), and improve the accuracy of recommendations that occur in sequence (using sequential data about the order in which users perform tasks). An investigation into suitable techniques for storing and analysing sequential and temporal data (such as sequential data mining [3]) is future work.

6.2.4 Context Uncertainty

The personalisation approach presented in this thesis has been shown to provide accurate personalisation recommendations to users by providing techniques and algorithms that identify a user's preferred behaviour depending on their surrounding environmental context. Context data acquired from the Hermes framework, which are used in the various personalisation techniques and algorithms in the approach, are assumed to be accurate. As a result of this assumption, uncertainties that may exist in acquired context information are not considered as part of the recommendation process. Further research is required to accurately determine the effect of context uncertainty on user preferences and to investigate how the personalisation approach can be extended to mitigate the effect of context uncertainty on recommendation decisions. A number of possible extensions are seen as worthy of future investigation. Firstly, uncertainty associated with context could be incorporated with the developed Information Selection technique so that only information above a specified level of certainty is retained. Secondly, information about uncertainty (e.g., confidence values) could be integrated with Utility Assignment so that less certain information is given less weight when ranking candidate choices. A third approach could be to discover patterns between different types of context using a similar association determination process as described in Section 3.2.4 to determine those types and values that are likely or unlikely to occur together, with context readings that do not conform to discovered patterns considered as unreliable. An investigation into the effectiveness of these approaches may lead to knowledge that would maintain or improve the accuracy of a recommendation in the presence of uncertain data.

6.2.5 User-Application Scrutiny

This thesis provided an approach that supports applications in generating, evaluating, and presenting personalised recommendations to users for different problems and context. As part of the current design, user-device interaction is restricted to users making recommendation requests and selecting their preferred choice from a recommended set. However, research has shown that an important element of user-based applications is the need for users to be able to query or scrutinise the decision making process [27, 180, 16, 34], which facilitates trust and acceptance of recommended behaviour. The extension of the personalisation approach to include techniques that support users to query applications about their decision making behaviour is therefore desirable.

The personalisation approach was designed using techniques that generate and evaluate user preference knowledge in a rules format with the intention that knowledge in rule form would facilitate the integration of developed personalisation techniques and algorithms with new techniques that support user-application scrutiny. Future investigations could focus on possible solutions that analyse and aggregate inferred association rules with the aim of providing an extension to the current design that enables users to query applications for decision making knowledge such as information relevance, utility, and ranking of behaviours.

6.3 Chapter Summary

This chapter summarised the motivation for the research work and the most significant achievements of the work presented in this thesis. In particular, it outlined how this work contributed to the state of the art in personalised, context-aware computing by providing a personalisation approach that supports the dynamic and implicit determination of user preference, relevant information, information utility, and ranking of candidate behaviour recommendations using information about past context and user interactions. The provisions of techniques and algorithms that minimises the need for explicit user and developer knowledge and input about user preferences eases the deployment of personalised context-aware applications and supports applications with utilising the growing availability of information in context-aware environments. User studies and computer simulations conducted during evaluation showed that the techniques and algorithms developed for this approach are effective in making accurate personalised recommendation to users for different problems and context. The chapter concluded with suggestions for future work arising from the research undertaken in relation to this thesis.

Appendix A

Additional Evaluation Results

A.1 Sample Size

We calculate the appropriate sample size using results data gathered from initial users of the context-aware travel recommender system. Using the sample size table [138], the appropriate sample size for our user study evaluation was computed. A combined standard deviation estimate is caculated by averaging the variances in accuracy of two sample responses (1.85 and 2.81). These variances are pooled:

$$S^2 = \frac{S_1^2 + S_2^2}{2}$$

giving a pooled value of 2.38. We increase the standard deviation to 4 to give a more conservative estimate of what our sample size should be. A D value to be read from the sample size table is computed using:

$$D = \frac{\delta}{\sigma}$$

Using a 5% δvalue representing the yied shift we want to detect, th D value is calculated to be 1.2. Reference to the sample size table with this value and 0.05 values for $\alpha$and β (represnting the risk of type one and type two errors), a sample size of *20* was calculated.

A.2 Normal Plots for Evaluation Results

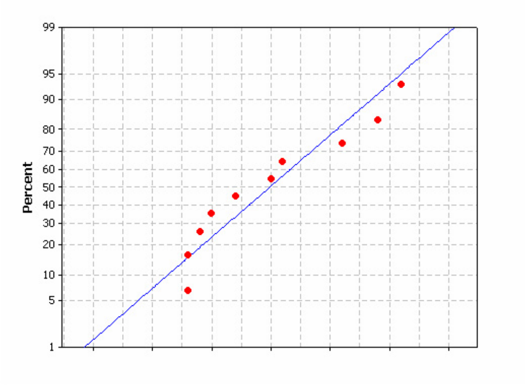


Figure A.1: Restaurant Recommender Original Strategy Accuracy

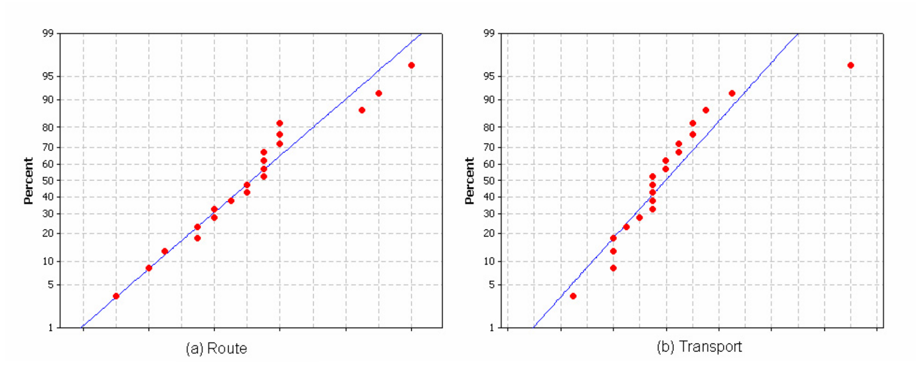


Figure A.2: Travel Recommender Original Strategy Accuracy

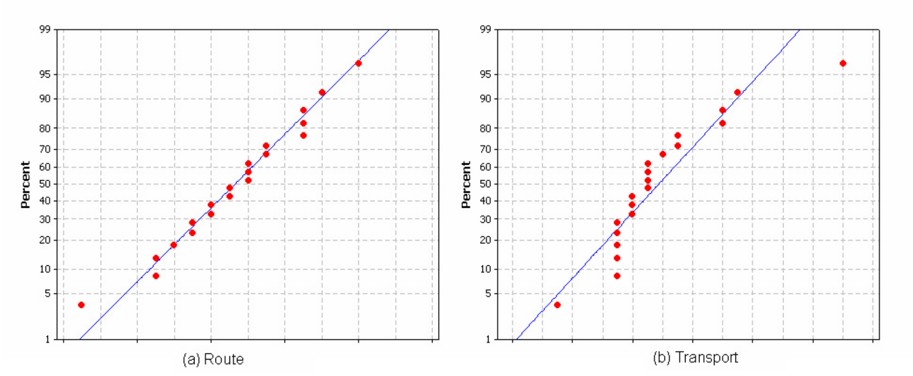


Figure A.3: Extended Strategy Accuracy

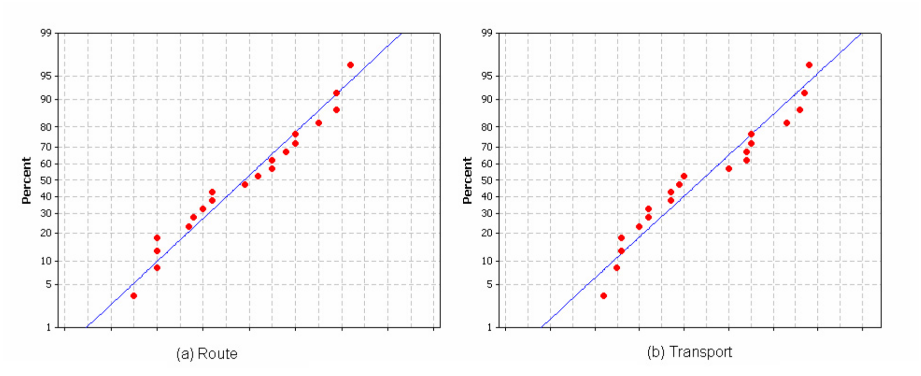


Figure A.4: Information Selection Accuracy

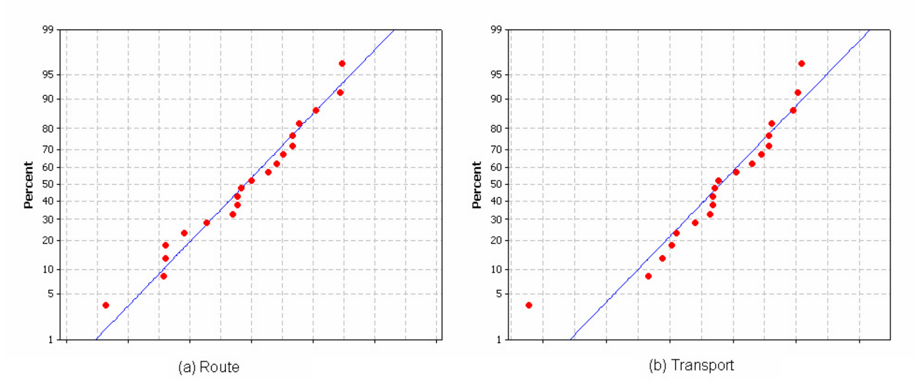


Figure A.5: Utility Assignment Accuracy

A.3 User Model Size Analysis Results

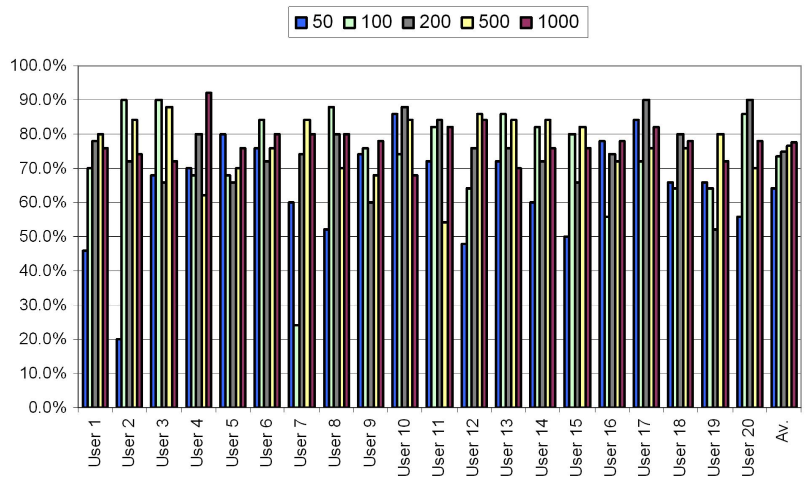


Figure A.6: Accuracy for Each User in Each User Model Size Category

A.4 Number of Supported Context Relationships Analysis Results

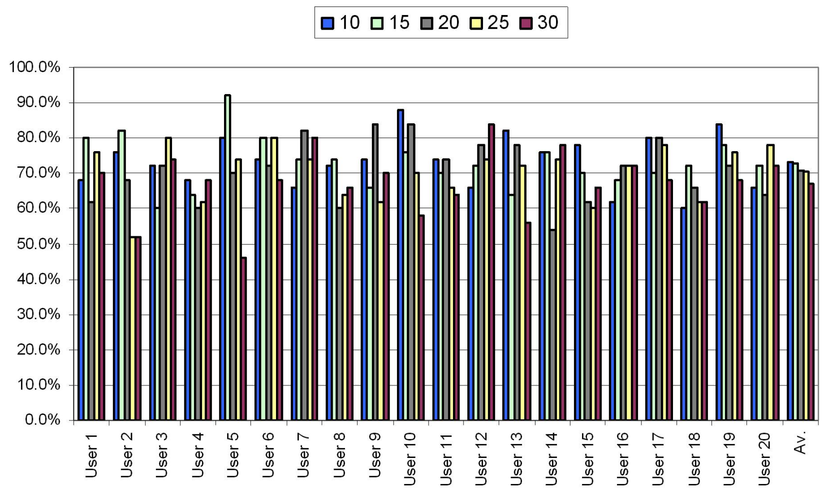
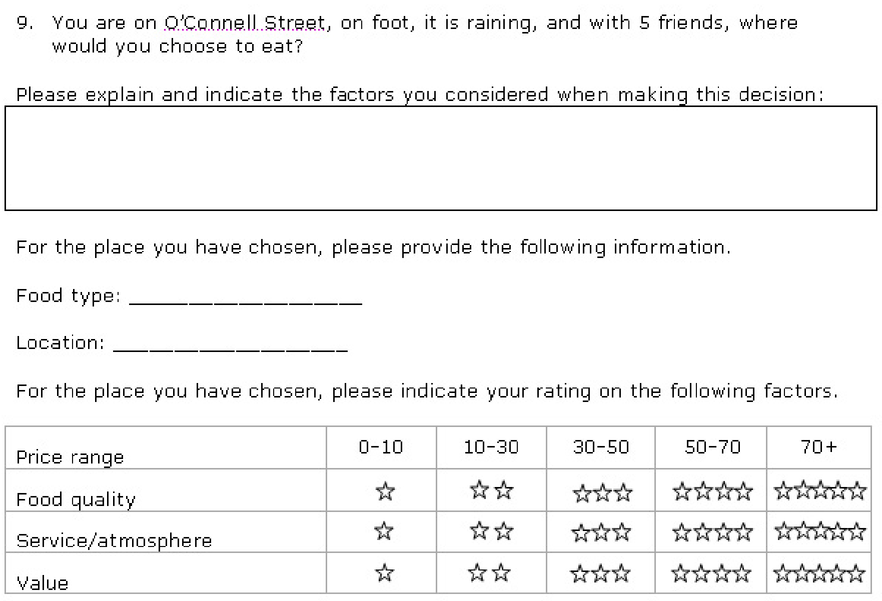


Figure A.7: Accuracy for Each User in Each Supported Context Number Category

Appendix B

Additional User Case Study Data

B.1 Restaurant Recommender Example User Problem



9. You are on O'Connell Street, on foot, it is raining, and with 5 friends, where would you choose to eat?

Please explain and indicate the factors you considered when making this decision:

For the place you have chosen, please provide the following information.

Food type: ___________________

Location: ___________________

For the place you have chosen, please indicate your rating on the following factors.

Price range	0-10	10-30	30-50	50-70	70+
Food quality	☆	☆☆	☆☆☆	☆☆☆☆	☆☆☆☆☆
Service/atmosphere	☆	☆☆	☆☆☆	☆☆☆☆	☆☆☆☆☆
Value	☆	☆☆	☆☆☆	☆☆☆☆	☆☆☆☆☆

B.2 Rules-based System Outline Implementation

Listing B.1: Context-aware Restaurant Recommender Java Class

```
import java.util.Vector;

/**
 * Restaurant Recommender application. The application will recommend
 * restaurants to a user based on their preferences and current situation (
    context)
 *
 * @author <insert name>
 *
 */
public class RestaurantRecommender {
        /* Outputs */
        // restaurant properties
        String cuisine;
        String cost;
        String foodQuality;
        String serviceAndAtmosphere;
        String distance;
        /* Inputs */
        // user context
        String company;
        String transport;
        // other context
        String meal;
        String day;
        String occasion;
        String timeConstraint;

        /**
         * Constructor. Sets the current context (update to set context to
            the one
         * you are testing for).
```

```
     * Recommends an appropriate type of restaurant to the user based on
         their preferences
     * and on their situation.
     *
     */
    public RestaurantRecommender() {
        /** context set to default. Change when testing different
            cases **/
        setContext(Constant.FRIENDS, Constant.ON_FOOT, Constant.EMPTY
            , Constant.EMPTY, Constant.EMPTY, Constant.EMPTY);
        recommendRestaurantExample();
        recommendRestaurant();
    }

    /**
     * Example recommendRestaurant method
     */
    private void recommendRestaurantExample() {
        Vector restaurantProperties = new Vector();
        /** Example rule **/
        if(company.equals(Constant.FRIENDS) && transport.equals(
            Constant.ON_FOOT)) {
                cuisine = Constant.CHINESE;
                cost = Constant.CHEAP;
                foodQuality = Constant.GOOD;
                serviceAndAtmosphere = Constant.TWO_STAR;
                distance = Constant.CLOSE;
                restaurantProperties.add(cuisine);
                restaurantProperties.add(cost);
                restaurantProperties.add(foodQuality);
                restaurantProperties.add(serviceAndAtmosphere);
                restaurantProperties.add(distance);
        }
        System.out.println("Example - Restaurant type to recommend:
            " + restaurantProperties.toString());
    }
```

```

	/**
	 * Recommends a type of restaurant to the user based on their
	    preferences and current situation
	 */
	private void recommendRestaurant() {
		Vector restaurantProperties = new Vector();
		/** INSERT BEHAVIOUR RULES HERE **/
		/**************************/


		/**************************/
		System.out.println("Restaurant type to recommend: " +
		    restaurantProperties.toString());
	}

	/**
	 * Application start
	 *
	 * @param args
	 */
	public static void main(String[] args) {
		new RestaurantRecommender();
	}

	/**
	 * Sets the current context of the user.
	 * @param company
	 * @param transport
	 * @param meal
	 * @param day
	 * @param occasion
	 * @param timeConstraint
	 */
	public void setContext(String company, String transport, String meal
	    , String day, String occasion, String timeConstraint) {
```

```
                        this.company = company;
                        this.transport = transport;
                        this.meal = meal;
                        this.day = day;
                        this.occasion = occasion;
                        this.timeConstraint = timeConstraint;
            }
}
```

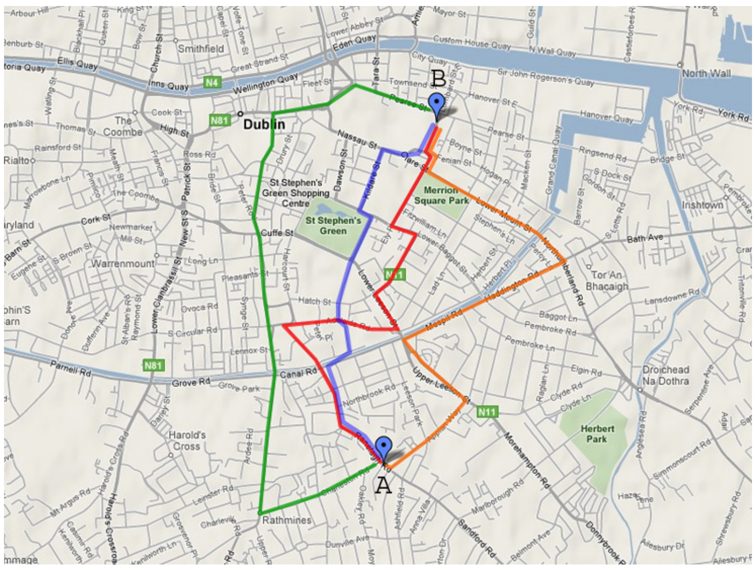
B.3 Developer Post-Implementation Questionnaire

Please give your opinion to the following set of statements:	Strongly Disagree	Dis-agree	50/50	Agree	Strongly Agree
I found it difficult to identify the information required for the application.	1	2	3	4	5
I found it difficult to categorise the information required for the application.	1	2	3	4	5
I found it difficult to identify relationships (associations/dependencies) between information.	1	2	3	4	5
I found it difficult to identify the different user situations.	1	2	3	4	5
I found it difficult to relate information to user situations.	1	2	3	4	5
I found it difficult to represent all possible user situations.	1	2	3	4	5
I found that in different situations, users want different recommendations.	1	2	3	4	5
I found that in different user situations different information is used for recommendation decisions.	1	2	3	4	5
I found that in different user situations different information associations/dependencies exist.	1	2	3	4	5
I found it difficult to identify to include in recommendation decisions.	1	2	3	4	5
I found that relationships between information have an affect in recommendation decisions.	1	2	3	4	5
I found that relationships between information changed depending on the user situation.	1	2	3	4	5
I found it difficult to identify the information relationships that affect recommendation decisions.	1	2	3	4	5
I found it difficult to represent all possible associations/dependencies between information.	1	2	3	4	5
Overall I thought the application was difficult to design and develop.	1	2	3	4	5

B.4 Travel Recommender Example User Problem

Please examine the map showing part of Dublin city and the description of the current situation and answer the subsequent questions that follow.

You are travelling from A: Ranelagh to B: Pearse Street. It is a weekday, the time is 9.00am, it is sunny, you are alone, and you are in a hurry.



1. Please indicate the route you would mostly likely choose to take given the current situation.

☐ Red ☐ Blue ☐ Green ☐ Orange

2. Please check the appropriate box(es) to indicate the information you considered when making this route decision.

☐ Day ☐ Time ☐ Weather ☐ Distance ☐ Company ☐ Time Available

3. Please rank the information you considered for this route decision (i.e. the checked boxes in Q2.) starting at 1 for the most important.

Day ◯ Time ◯ Weather ◯ Distance ◯ Company ◯ Time Available ◯

4. Given that the transport options of walking, taking the bus, or driving are available, please indicate your preferred choice for the above problem.

☐ Walk ☐ Bus ☐ Car

5. Please check the appropriate box(es) to indicate the information you considered when making this transport decision.

☐ Day ☐ Time ☐ Weather ☐ Distance ☐ Company ☐ Time Available

6. Please rank the information you considered for this transport decision (i.e. the checked boxes in Q5.) starting at 1 for the most important.

Day ◯ Time ◯ Weather ◯ Distance ◯ Company◯ Time Available ◯

7. Given that the weather changed from sunny to raining, would you change the route you would choose to take? If yes, please specify.

☐ Yes ☐ No

☐ Red ☐ Blue ☐ Green ☐ Orange

8. Similarly, given that the weather changed from sunny to raining, would you change the transport option you chose? If yes, please specify.

☐ Yes ☐ No

☐ Walk ☐ Bus ☐ Car

Any other comments?

Bibliography

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(a):39–52, 1994.

[2] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999.

[3] Jean-Marc Adamo. *Data mining for association rules and sequential patterns: sequential and parallel algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[4] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems*, 23(1):103–145, 2005.

[5] Gediminas Adomavicius and Alexander Tuzhilin. Using data mining methods to build customer profiles. *IEEE Computer*, 34(2):74–82, 2001.

[6] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions in Knowledge and Data Engineering*, 17(6):734–749, 2005.

[7] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 1993.

[8] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri

Verkamo. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, pages 307–328, 1996.

[9] David W. Aha. Editorial. *Artificial Intelligence Review*, 11:7–10, 1997.

[10] Fahd Albinali, Nigel Davies, and Adrian Friday. Structural learning of activities from sparse datasets. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 221–228, Washington, DC, USA, 2007.

[11] Robert B. Allen. User models: theory, method, and practice. *International Journal of Man-Machine Studies*, 32(5):511–543, 1990.

[12] Christos Anagnostopoulos, Athanasios Tsounis, and Stathes Hadjiefthymiades. Context awareness in mobile computing: A survey. In *Mobile HCI '04, Mobile and Ubiquitous Information Access*, September 2004.

[13] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

[14] Mafruz Zaman Ashrafi, David Taniar, and Kate A. Smith. A new approach of eliminating redundant association rules. In *Database and Expert Systems Applications (DEXA)*, pages 465–474, 2004.

[15] Mafruz Zaman Ashrafi, David Taniar, and Kate A. Smith. Redundant association rules reduction techniques. *International Journal of Business Intelligence and Data Mining (IJBIDM)*, 2(1):29–63, 2007.

[16] Mark Assad, David J. Carmichael, Judy Kay, and Bob Kummerfeld. Personisad: Distributed, active, scrutable model framework for context-aware services. In *Pervasive*, pages 55–72, 2007.

[17] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* New York: ACM Press, Addison-Wesley, 1999.

[18] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263 –277, 2006.

[19] Evelyn Balfe and Barry Smyth. Case-based collaborative web search. In *Advances in Case-Based Reasoning, 7th European Conference (ECCBR)*, pages 489–503, 2004.

[20] Jakob E. Bardram. The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications. In Hans-Werner Gellersen, Roy Want, and Albrecht Schmidt, editors, *Pervasive*, volume 3468 of *Lecture Notes in Computer Science*, pages 98–115, 2005.

[21] Louise Barkhuus and Anind K. Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. In *Ubicomp*, pages 149–156, 2003.

[22] Leon Barnard, Ji Soo Yi, Julie A. Jacko, and Andrew Sears. Capturing the effects of context on human performance in mobile computing systems. *Personal Ubiquitous Computing*, 11(2):81–96, 2007.

[23] Patrick Baudisch. *Dynamic Information Filtering*. PhD thesis, GMD Forschungszentrum Informationstechnik GmbH, Sankt Augustin, Germany, 2001.

[24] Russell Beale and Peter Lonsdale. Mobile context aware systems: The intelligence to support tasks and effectively utilise resources. In *Mobile Mobile Human-Computer Interaction*, pages 240–251, 2004.

[25] Joseph E. Beck and Kai min Chang. Identifiability: A fundamental problem of student modeling. In *User Modeling*, pages 137–146, 2007.

[26] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval:two sides of the same coin? *Communications of the ACM*, 35(12):29–37, 1992.

[27] Victoria Bellotti and Keith Edwards. Intelligibility and accountability: human considerations in context aware systems. *Human Computer Interaction*, 16:193–212, 2001.

[28] Shlomo Berkovsky, Lora Aroyo, Dominik Heckmann, Geert-Jan Houben, Alexander Kröner, Tsvi Kuflik, and Francesco Ricci. Providing context-aware personalization through cross-context reasoning of user modeling data. In *in proceedings of the Workshop on Decentralized and Ubiquitous User Modeling (UbiDeUM), at the International Conference on User Modeling (UM)*, 2007.

[29] Gregory Biegel and Vinny Cahill. A framework for developing mobile, context-aware applications. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 361, Washington DC, USA, 2004.

[30] Mustafa Bilgic and Raymond J Mooney. Explaining recommendations: Satisfaction vs promotion. In *Beyond Personalization: A Workshop on the Next Stage of Recommender Systems Research at the International Conference on Intelligent User Interfaces*, 2005.

[31] Jan Blom. Personalization: a taxonomy. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 313–314. ACM, 2000.

[32] 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.

[33] Keith Bradley and Barry Smyth. An architecture for case-based personalised search. In *Advances in Case-Based Reasoning, 7th European Conference*, pages 518–532, 2004.

[34] Nicholas A. Bradley and Mark D. Dunlop. Towards a multidisciplinary model of context to support context-aware computing. *Journal of Human-Computer Interaction*, 20(4):403–446, 2005.

[35] Peter J. Brown. The stick-e document: A framework for creating context-aware applications. In *In Proceedings of Electronic Publishing*, volume 8, pages 259–272, June – September 1996.

[36] Peter J. Brown and Gareth J. F. Jones. Context-aware retrieval: Exploring a new environment for information retrieval and information filtering. *Personal Ubiquitous Comput.*, 5(4):253–263, 2001.

[37] Robin D. Burke, Kristian J. Hammond, and Benjamin C. Young. The findme approach to assisted browsing. *IEEE Expert: Intelligent Systems and Their Applications*, 12(4):32–40, 1997.

[38] Hee Eon Byun and Keith Cheverst. Exploiting user models and context-awareness to support personal daily activities. In *Workshop in UM2001 on User Modelling for Context-Aware Applications*, 2001.

[39] Hee Eon Byun and Keith Cheverst. Harnessing context to support proactive behaviors. *AIMS 2002 (AI in Mobile Systems)*, 2002.

[40] CDNow. Online `http://www.cdnow.com`, 2008.

[41] Datong Chen, Albrecht Schmidt, and Hans-W. Gellesen. An architecture for multi-sensor fusion in mobile environments. In *Proceedings International Conference on Information Fusion, Sunnyvale, CA, USA*, July 1999.

[42] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA, 2000.

[43] Harry Chen, Tim Finin, and Anupam Joshi. An intelligent broker architecture for context-aware systems. In *Adjunct Proceedings of Ubicomp 2003*, Seattle, Washington, USA, October 2003.

[44] Li Chen and Pearl Pu. Preference-based organization interfaces: Aiding user critiques in recommender systems. In *User Modeling*, pages 77–86, 2007.

[45] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.

[46] Zhixiang Chen, Xiannong Meng, Richard H. Fowler, and Binhai Zhu. Websail: From on-line learning to web search. In *WISE '00: Proceedings of the First International Conference on Web Information Systems Engineering (WISE'00)-Volume 1*, page 206, Washington, DC, USA, 2000.

[47] James Cheng, Yiping Ke, and Wilfred Ng. Effective elimination of redundant association rules. *Data Mining and Knowledge Discovery*, 16(2):221–249, 2008.

[48] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Experiences of developing and deploying a context-aware tourist guide: the guide project. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 20–31. ACM, 2000.

[49] Keith Cheverst, Keith Mitchell, and Nigel Davies. Exploring context-aware information push. *Personal Ubiquitous Computing*, 6(4):276–281, 2002.

[50] Siobhán Clarke and Cormac Driver. Context-aware trails. *Computer*, 37(8):97–99, 2004.

[51] Google Code. Android - an open handset alliance project. Online `http://code.google.com/android/`, October 2008.

[52] Paul Cotter and Barry Smyth. Ptv: Intelligent personalised tv guides. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 957–964, 2000.

[53] Olivier Coutand, Sandra Haseloff, Sian Lun Lau, and Klaus David. A cbr approach for personalizing location-aware services. In *1st Workshop on Case-based Reasoning and Context Awareness (CACOA)*, 2006.

[54] Lorcan Coyle. *Making Personalised Flight Recommendations using Implicit Feedback.* PhD thesis, Trinity College Dublin, 2004.

[55] Lorcan Coyle, Padraig Cunningham, and Conor Hayes. A case-based personal travel assistant for elaborating user requirements and assessing offers. In *ECCBR*, pages 505–518, 2002.

[56] Waltenegus Dargie. The role of probabilistic schemes in multisensor context-awareness. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 27–32, Washington, DC, USA, 2007.

[57] A. Philip Dawid and Vanessa Didelez. Identifying optimal sequential decisions. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI 2008)*, pages 113–120, 2008.

[58] Ramon Lopez de Mantaras and Eva Armengol. Machine learning from examples: inductive and lazy methods. *Data Knowledge Engineering*, 25(1-2):99–123, 1998.

[59] Pierre A. Devijver and Josef Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, London, 1982.

[60] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. Technical report, Georgia Inistitute of Technology, College of Computing, 1999.

[61] Anind K. Dey and Gregory D. Abowd. Cybreminder: A context-aware system for supporting reminders. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 172–186. Springer-Verlag, 2000.

[62] 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.

[63] Paul Dourish. What we talk about when we talk about context. *Personal Ubiquitous Computing*, 8(1):19–30, 2004.

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

[65] Cormac Driver and Siobhán Clarke. An application framework for mobile, context-aware trails. *Pervasive and Mobile Computing*, 4(5):719–736, 2008.

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

[67] Jennifer G. Dy and Carla E. Brodley. Feature selection for unsupervised learning. *J. Mach. Learn. Res.*, 5:845–889, 2004.

[68] Ali F. Farhoomand and Donald H. Drury. Managing information overload. *Communications of the ACM*, 45(10):127–131, 2002.

[69] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[70] Peter C. Fishburn. Preference structures and their numerical representations. *Theoretical Computur Science*, 217(2):359–383, 1999.

[71] Enrique Frias-Martinez, Sherry Y. Chen, and Xiaohui Liu. Survey of data mining approaches to user modeling for adaptive hypermedia. *IEEE transactions on systems, man and cybernetics*, 36:734–749, 2006.

[72] Avelino J. Gonzalez and Douglas D. Dankel. *The Engineering of Knowledge-Based Systems, Theory and Practice*. Prentice Hall, 1993.

[73] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 439–446, CA, USA, 1999.

[74] Michele Gorgoglione, Cosimo Palmisano, and Alex Tuzhilin. Personalization in context: Does context matter when building personalized customer models? In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 222–231, Washington, DC, USA, 2006.

[75] William Sealy Gosset. The probable error of a mean. *Biometrika*, 6(1):1–25, March 1908. Originally published under the pseudonym “Student”.

[76] Sabine Graf, Kathryn MacCallum, Tzu-Chien Liu, Maiga Chang, Dunwei Wen, Qing Tan, Jon Dron, Fuhua Lin, Nian-Shing Chen, Rory McGreal, and Kinshuk. An infrastructure for developing pervasive learning environments. In *Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 389–394, 2008.

[77] Mary-Liz Grisé and R. Brent Gallupe. Information overload: addressing the productivity paradox in face-to-face electronic meetings. *Journal of Management Information Systems*, 16(3):157–185, 1999.

[78] Tao Gu, H. Pung, and D. Zhang. A middleware for building context-aware mobile services. In *IEEE Vehicular Technology Conference*, Milan, Italy, May 2004.

[79] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining Knowledge Discovery*, 15(1):55–86, 2007.

[80] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, 2001.

[81] Trevor Hastie, Robert Tibshirani, and Robert Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2003.

[82] Conor Hayes and Padraig Cunningham. Smart radio - community based music radio. *Knowledge Based Systems*, 14(3-4):197–201, 2001.

[83] Dominik Heckmann, Eric Schwarzkopf, Junichiro Mori, Dietmar Dengler, and Alexander Kroner. The user model and context ontology gumo revisited for future web 2.0 extensions. In *Proceedings of the International Workshop on Contexts and Ontologies: Representation and Reasoning Collocated with the 6th International and Interdisciplinary Conference on Modelling and Using Context*, 2007.

[84] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *2nd IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 77–86, 2004.

[85] Karen Henricksen and Jadwiga Indulska. Personalising context-aware applications. In *OTM Workshop on Context-Aware Mobile Systems (CAMS)*, volume 3762, pages 122–131. Springer, 2005.

[86] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Using context and preferences to implement self-adapting pervasive computing applications: Experiences with auto-adaptive and reconfigurable systems. *Software - Practice & Experience*, 36(11-12):1307–1330, 2006.

[87] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 292.1, Washington, DC, USA, 2003.

[88] Jason I. Hong and James A. Landay. An infrastructure approach to context-aware computing. In *In Human-Computer Interaction*, volume 16, pages 287–303, 2001.

[89] Jason I. Hong and James A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189, NY, USA, 2004.

[90] Derek Hao Hu and Qiang Yang. Cigar: Concurrent and interleaving goal and activity recognition. In *AAAI Conference on Artificial Intelligence*, pages 1363–1368, 2008.

[91] Peizhao Hu, Jadwiga Indulska, and Ricky Robinson. An autonomic context management system for pervasive computing. In *Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008)*, pages 213–223, 2008.

[92] Anthony Jameson. Modeling both the context and the user. *Personal and Ubiquitous Computing*, 5(1):29–33, 2001.

[93] Gareth J. F. Jones, David J. Quested, and Katherine E. Thomson. Personalised delivery of news articles from multiple sources. In *ECDL '00: Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries*, pages 340–343, London, UK, 2000.

[94] Martin Jonsson, Patrik Werle, and Carl Gustaf Jansson. Context shadow: An infrastructure for context aware computing. In *Proceedings of Artificial Intelligence in Mobile System*, 2003.

[95] Ivar Jørstad, Do Van Thanh, and Schahram Dustdar. Personalisation of next generation mobile services. In *Ubiquitous Mobile Information and Collaboration Systems (UMICS)*, volume 242 of *CEUR Workshop Proceedings*, 2006.

[96] Judy Kay, Bob Kummerfeld, and Piers Lauder. Personis: A server for user models. In *AH '02: Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 203–212, London, UK, 2002.

[97] Kevin Keenoy and Mark Levene. Personalisation of web search. In *Intelligent Techniques for Web Personalization*, pages 201–228, 2003.

[98] Eungyeong Kim, Hyogun Yoon, Malrey Lee, and Thomas M. Gatton. The user preference learning for multi-agent based on neural network in ubiquitous computing environment. *Agent and Multi-Agent Systems: Technologies and Applications*, 4953/2008:547–556, 2008.

[99] Alfred Kobsa. Supporting user interfaces for all through user modeling. In *Sixth International Conference on Human-Computer Interaction*, pages 155–157, Yokohama, Japan, 1995.

[100] Alfred Kobsa. Generic user modeling systems. *User Model. User-Adapt. Interact.*, 11(1-2):49–63, 2001.

[101] Anders Kofod-Petersen and Agnar Aamodt. Contextualised ambient intelligence through case-based reasoning. In *Advances in Case-Based Reasoning, 8th European Conference, ECCBR 2006*, pages 211–225, 2006.

[102] Anders Kofod-Petersen and Marius Mikalsen. Context: Representation and reasoning – representing and reasoning about context in a mobile environment. *Revue d'Intelligence Artificielle*, 19(3):479–498, 2005.

[103] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence 2*, volume 12, pages 1137–1143. Morgan Kaufmann, 1995.

[104] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2):273–324, 1997.

[105] Joost N. Kok and Walter A. Kosters. Natural data mining techniques. *Current Trends in Theoretical Computer Science*, pages 603–613, 2001.

[106] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[107] Panu Korpipää, Esko-Juhani Malm, Ilkka Salminen, Tapani Rantakokko, Vesa Kyllönen, and Ilkka Känsälä. Context management for end user development of context-aware applications. In *MDM '05: Proceedings of the 6th international conference on Mobile data management*, pages 304–308, NY, USA, 2005.

[108] Sotiris Kotsiantis and Dimitris Kanellopoulos. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science Engineering*, 32(1):71–82, 2006.

[109] Ronny Kramer, Marko Modsching, and Klaus ten Hagen. Development and Evaluation of a Context-driven, Mobile Tourist Guide. *International Journal of Pervasive Computing and Communication*, 1(1), March 2005.

[110] Andreas Krause. Context-aware mobile computing: Learning context-dependent personal preferences from a wearable sensor array. *IEEE Transactions on Mobile Computing*, 5(2):113–127, 2006.

[111] Bob Kummerfeld, Aaron Quigley, Chris Johnson, and Rene Hexel. Merino: Towards an intelligent environment architecture for multi-granularity context description. Pittsburgh, USA, June 2003. Workshop on User Modeling for Ubiquitous Computing.

[112] Stan Kurkovsky and Karthik Harihar. Using ubiquitous computing in interactive mobile marketing. *Personal Ubiquitous Computing*, 10(4):227–240, 2006.

[113] Wai Lam, Snehasis Mukhopadhyay, Javed Mostafa, and Mathew Palakal. Detection of interest shifts for personalized information filtering. In *Detection of interest shifts for personalized information filtering*, pages 317–324, 1996.

[114] Ting-Peng Liang, Hung-Jen Lai, and Yi-Cheng Ku. Personalized content recommendation and user satisfaction: Theoretical synthesis and empirical findings. *Journal of Management Information Systems*, 23(3):45–70, 2007.

[115] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 924–929, 1995.

[116] Greg Linden and Brent Smith. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Intelligent Systems*, 7(1):76–80, 2003.

[117] Eamonn Linehan, Shiu Lun Tsang, and Siobhán Clarke. Supporting context-awareness: A taxonomic review. Technical report, (TCD-CS-2008-37), Department of Computer Science, Trinity College Dublin, http://www.cs.tcd.ie/publications/tech-reports/reports.08/TCD-CS-2008-37.pdf, 2008.

[118] Huan Liu and Hiroshi Motoda. *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series).* Chapman & Hall CRC Press, 2007.

[119] Yunhao Liu, Lei Chen, Jian Pei, Qiuxia Chen, and Yiyang Zhao. Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications (PerCom 2007)*, pages 37–46, Washington, DC, USA, 2007.

[120] Huang Liusheng, Chen Huaping, Wang Xun, and Chen Guoliang. A fast algorithm for mining association rules. *Journal of Computer Science and Technology*, 15(6):619–624, 2000.

[121] Robert Lokaiczyk, Andreas Faatz, Arne Beckhaus, and Manuel Görtz. Enhancing just-in-time e-learning through machine learning on desktop context sensors. In *CONTEXT*, volume 4635 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2007.

[122] Tinghuai Ma, Yong-Deak Kim, Qiang Ma, Meili Tang, and Weican Zhou. Context-aware implementation based on cbr for smart home. In *Wireless And Mobile Computing, Networking And Communications, 2005. (WiMob'2005), IEEE International Conference on*, volume 4, pages 112–115, 2005.

[123] Pattie Maes. Agents that reduce work and information overload. *Communications ACM*, 37(7):30–40, 1994.

[124] Gary Marchionini. Information seeking in electronic environments. *Cambridge MA: Cambridge Univ. Press (Cambridge series on humancomputer-interaction).*, 1995.

[125] Atsushi Maruyama, Naoki Shibata, Yoshihiro Murata, Keiichi Yasumoto, and Minoru Ito. P-tour: A personal navigation system for tourism. In *In proceedings of the 11th World Congress on Intelligent Transport Systems*, volume 2, pages 18–21, 2004.

[126] Sarah McBurney, M. Howard Williams, Nick K. Taylor, and Eliza Papadopoulou. Managing user preferences for personalization in a pervasive service environment. In *AICT '07: Proceedings of the The Third Advanced International Conference on Telecommunications*, page 32, Washington, DC, USA, 2007.

[127] Lorraine McGinty and Barry Smyth. Turas: A personalised route planning system. In *Topics in Artificial Intelligence, 6th Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, page 791, 2000.

[128] Susan McKeever, Juan Ye, Lorcan Coyle, and Simon Dobson. A multilayered uncertainty model for context aware systems. pages 1–4, Sydney, Australia, 2008.

[129] Microsoft. Easy living. Online `http://research.microsoft.com/easyliving/`.

[130] Robert B. Miller. Response time in man-computer conversational transactions. In *Fall Joint Computer Conference 33 (part 1)*, pages 267–277. AFIPS Press, 1968.

[131] Barnshad Mobasher. Data mining for web personalization. *The Adaptive Web: Methods and Strategies of Web Personalization*, 4321:90–135, 2007.

[132] Martin Modahl, Bikash Agarwalla, Scott Saponas, Gregory Abowd, and Umakishore Ramachandran. Ubiqstack: A taxonomy for a ubiquitous computing software stack. volume 10, pages 21–27, London, UK, 2005.

[133] Joseph Modayil, Tongxin Bai, and Henry Kautz. Improving the recognition of interleaved activities. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 40–43, New York, NY, USA, 2008.

[134] Miquel Montaner, Beatriz López, and Josep Lluís De La Rosa. A taxonomy of recommender agents on the internet. *Artificial Intelligence Review*, 19(4):285–330, 2003.

[135] David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics.* W.H. Freeman and Company, fifth edition edition, 2003.

[136] Yaser Mowafi and Dongsong Zhang. A user-centered approach to context-awareness in mobile computing. In *MobiQuitous*, pages 1–3, 2007.

[137] Bruce Moxon. Defining data mining. *DBMS Data Warehouse Supplement: The Hows and Whys of Data Mining, and How It Differs From Other Analytical Techniques*, 9(9), 1996.

[138] Eamonn Mullins. *Statistics for the Quality Control Chemistry Laboratory.* The Royal Society if Chemistry, 2003.

[139] Tatsuo Nakajima and Ichiro Satoh. A software infrastructure for supporting spontaneous and personalized interaction in home computing environments. *Personal Ubiquitous Computing*, 10(6):379–391, 2006.

[140] Jakob Nielsen. *Usability Engineering.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[141] Jakob Nielsen. Personalization is over-rated. Online `http://www.useit.com/alertbox/981004.html`, 1998.

[142] Jakob Nielsen. Response times: The three important limits. Online, October 2006. `http://www.useit.com/papers/responsetime.html`.

[143] Donald A. Norman. The invisible computer: Why good products can fail, the personal computer is so complex, and information appliances are the solution. *MIT Press*, 1998.

[144] Antti Oulasvirta and Jan Blom. Motivations in personalisation behaviour. *Interact. Comput.*, 20(1):1–16, 2008.

[145] Cosimo Palmisano, Alexander Tuzhilin, and Michele Gorgoglione. User profiling with hierarchical context: An e-retailer case study. In *6th International and Interdisciplinary Conference (CONTEXT)*, pages 369–383, 2007.

[146] Jason Pascoe. The stick-e note architecture: Extending the interface beyond the user. In *Intelligent User Interfaces*, pages 261–264, 1997.

[147] Martha E. Pollack. Intelligent assistive technology: The present and the future. In *User Modeling*, pages 5–6, 2007.

[148] Stefan Poslad, Heimo Laamanen, Rainer Malaka, Achim Nick, Phil Buckle, and Alexander Zipf. Crumpet: creation of user-friendly mobile services personalised for tourism. In *Proceedings of the 2nd International Conference on 3G Mobile Communication Technologies (3G 2001)*, pages 28–32, 2001.

[149] Jim Prentzas and Ioannis Hatzilygeroudis. Categorizing approaches combining rule-based and case-based reasoning. *Expert Systems*, 24(2):97–122, May 2007.

[150] Amigo Project. Online `http://www.amigo-project.org`, 2008.

[151] Anand Ranganathan and Roy H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *ACM/IFIP/USENIX International Middleware Conference*, pages 143–161, Rio de Janeiro, Brazil, 2003.

[152] Anand Ranganathan and Roy H. Campbell. Provably correct pervasive computing environments. In *Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 160–169, 2008.

[153] Chedy Raíssi, Toon Calders, and Pascal Poncelet. Mining conjunctive sequential patterns. *Data Mining and Knowledge Discovery*, 17(1):77–93, 2008.

[154] Roland Reichle, Michael Wagner, Mohammad Ullah Khan, Kurt Geihs, Massimo Valla, Cristina Fra, Nearchos Paspallis, and George A. Papadopoulos. A context query language for pervasive computing environments. In *PerComSixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008)*, pages 434–440, 2008.

[155] James Reilly, Jiyong Zhang, Lorraine McGinty, Pearl Pu, and Barry Smyth. Evaluating compound critiquing recommenders: a real-user study. In *ACM Conference on Electronic Commerce*, pages 114–123, 2007.

[156] Jukka Riekki, Oleg Davidyuk, Jari Forstadius, Junzhao Sun, and Jaakko Sauvola. Enabling context-aware services for mobile users. In *IADIS Distributed and Parallel Systems and Architectures conference as part of IADIS Multi Conference on Computer Science and Information Systems*, pages 11–29. ACM Press, 2005.

[157] Thomas Rist and Patrick Brandmeier. Customizing graphics for tiny displays of mobile devices. *Personal Ubiquitous Comput.*, 6(4):260–268, 2002.

[158] 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.

[159] Seth Rogers, Claude-Nicolas Fiechter, and Pat Langley. An adaptive interactive agent for route advice. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 198–205. ACM, 1999.

[160] Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[161] Victor Rosenberg. Factors affecting the preferences of industrial personnel for information gathering methods. *Information Storage and Retrieval*, 3(3):119–127, 1967.

[162] Dario Rossi, Marco Mellia, and Claudio Casetti. User patience and the web: a hands-on investigation. *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, 7:4163–4168, 2003.

[163] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, second international edition edition, 2003.

[164] Norman M. Sadeh, Enoch Chan, and Linh Van. Mycampus: An agent-based environment for context-aware mobile services. In *In M. Gini (ed.), Proceedings of AAMAS02 Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices.* ACM, 2002.

[165] Alfons H. Salden, Remco Poortinga, Makram Bouzid, Olaf Drogehorn, Jerome Picault, Michael Sutterer, R. Kernchen, C. Rack, Mateusz Radziszewski, and Petteri Nurmi. Contextual personalization of a mobile multimodal application. In *International Conference on Internet Computing*, 2005.

[166] Mahadev Satyanarayanan. Coping with uncertainty. *IEEE Pervasive Computing*, 2(3):2, 2003.

[167] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *1st International Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[168] Amit Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–42, 2001.

[169] Rashmi Sinha and Kirsten Swearingen. The role of transparency in recommender systems. pages 830–831, 2002.

[170] Barry Smyth and Lorraine McGinty. The route to personalization: A case-based reasoning perspective. *Expert Update, The British Computer Society Specialist Group on Artificial Intelligence (BCS-SGAI)*, 5(2):27–36, 2002.

[171] Irene Stadnyk and Robert Kass. Modeling users' interests in information filters. *Communications of the ACM*, 35(12):49–50, 1992.

[172] Kostas Stefanidis, Evaggelia Pitoura, and Panos Vassiliadis. A context-aware preference database system. *Journal of Pervasive Computing & Communications*, 1, 2005.

[173] Graeme Stevenson, Lorcan Coyle, Steve Neely, Simon Dobson, and Paddy Nixon. Construct — a decentralised context infrastructure for ubiquitous computing environments. In *IT&T Annual Conference, Cork Institute of Technology, Ireland*, 2005.

[174] Oliver Stiemerling, Helge Kahler, and Volker Wulf. How to make software softer-designing tailorable applications. In *DIS '97: Proceedings of the 2nd conference on Designing interactive systems*, pages 365–376. ACM, 1997.

[175] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[176] Michael Sutterer, Olivier Coutand, Olaf Droegehorn, Klaus David, and Kees van der Sluijs. Managing and delivering context-dependent user preferences in ubiquitous computing environments. In *SAINT-W '07: Proceedings of the 2007 International Symposium on Applications and the Internet Workshops*, page 4. IEEE Computer Society, 2007.

[177] Kirsten Swearingen and Rashmi Sinha. Beyond algorithms: An hci perspective on recommender systems. *ACM SIGIR 2001 Workshop on Recommender Systems*, 2001.

[178] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. Feature-weighted user model for recommender systems. In *User Modeling*, pages 97–106, 2007.

[179] Georgia Tech. Aware home. Online `http://awarehome.imtc.gatech.edu/`, 2008.

[180] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *International Conference on Data Engineering Workshops*, pages 801–810, 2007.

[181] Alexander Tuzhilin. Report on the kdd2000 panel personalization and data mining: Exploring the synergies. *SIGKDD Explorations*, 2(2):115–116, 2000.

[182] Elena Vildjiounaite, Otilia Kocsis, Vesa Kyllonen, and Basilis Kladis. Context-dependent user modelling for smart homes. In *User Modeling*, pages 345–349, 2007.

[183] Detlof von Winterfeld and Ward Edwards. *Decision Analysis and Behavioral Research.* Cambridge UniversityPress., 1986.

[184] Chuan Wang and Christos Tjortjis. Prices: An efficient algorithm for mining association rules. *Intelligent Data Engineering and Automated Learning*, 3177:352–358, 2004.

[185] Torben Weis, Marcus Handte, Mirko Knoll, and Christian Becker. Customizable pervasive applications. In *International Conference on Pervasive Computing and Communications (PERCOM)*, pages 239–244, 2006.

[186] M.H. Williams, I. Roussaki, M. Strimpakou, Y. Yang, L. MacKinnon, R. Dewar, N. Milyaev, C. Pils, and M. Anagnostou. Context-awareness and personalisation in the daidalos pervasive environment. *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*, pages 98–107, July 2005.

[187] Patrick Wilson. Unused relevant information in research and development. *Journal of the American Society for Information Science*, 46(1):45–51, 1995.

[188] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005.

[189] Xindong Wu, Chengqi Zhang, and Shichao Zhang. Efficient mining of both positive and negative association rules. *ACM Transactions on Information Systems*, 22(3):381–405, 2004.

[190] Kaijian Xu, Manli Zhu, Daqing Zhang, and Tao Gu. Context-aware content filtering & presentation for pervasive & mobile information systems. In *Ambi-Sys '08: Proceedings of the 1st international conference on Ambient media and systems*, pages 1–8, ICST, Brussels, Belgium, 2008.

[191] Yahoo! Music match jukebox. Online `http://www.musicmatch.com`, 2007.

[192] Yuping Yang. Provisioning of personalized pervasive services: Daidalos personalization functions. *Pervasive Computing and Applications, 2006 1st International Symposium on*, pages 110–115, Aug. 2006.

[193] Lei Yu and Huan Liu. Efficient feature selection via analysis of relevance and redundancy. *J. Mach. Learn. Res.*, 5:1205–1224, 2004.

[194] Zhiwen Yu, Xingshe Zhou, Daqing Zhang, Chung-Yau Chin, Xiaohang Wang, and Ji Men. Supporting context-aware media recommendations for smart phones. *IEEE Pervasive Computing*, 5(3):68–75, 2006.

[195] Mohammed J. Zaki. Mining non-redundant association rules. *Data Min. Knowl. Discov.*, 9(3):223–248, 2004.

[196] Massimo Zancanaro, Tsvi Kuflik, Zvi Boger, Dina Goren-Bar, and Dan Goldwasser. Analyzing museum visitors' behavior patterns. In *User Modeling*, pages 238–246, 2007.

[197] Jianpei Zhang, Zhongwei Li, and Jing Yang. A divisional incremental training algorithm of support vector machine. *Mechatronics and Automation, 2005 IEEE International Conference*, 2:853–856, 2005.

[198] Jiyong Zhang and Pearl Pu. A comparative study of compound critique generation in conversational recommender systems. In *Adaptive Hypermedia and Adaptive Web-Based Systems, 4th International Conference*, pages 234–243, 2006.

[199] Andreas Zimmermann. Context-awareness in user modelling: Requirements analysis for a case-based reasoning application. In *Case-based reasoning research and development : 5th International Conference on Case-Based Reasoning (ICCBR)*, 2003.

[200] Andreas Zimmermann and Andreas Lorenz. Listen: a user-adaptive audio-augmented museum guide. *User Modeling and User-Adapted Interaction*, 18(5):389–416, 2008.

[201] Andreas Zimmermann, Andreas Lorenz, and Reinhard Oppermann. An operational definition of context. In *Sixth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 07)*, pages 558–571, 2007.

[202] George Kingsley Zipf. *Human Behaviour and the Principle of Least Effort: An Introduction to Human Ecology*. Hafner Pub. Co, 1949.