Dynamic context-aware personalisation in a pervasive environment

Dynamic Context-Aware Personalisation in a Pervasive Environment

Sarah Gallacher, Eliza Papadopoulou, Yussuf Abu-Shaaban, Nick K. Taylor, Howard M. Williams

MACS Department, Heriot-Watt University, Riccarton, Edinburgh, UK
Tel. +44(0)131 451 4170
Fax. +44(0)131 451 3630
{ceesmm1, ceeep1, ya37, nkt, mhw}@macs.hw.ac.uk

Abstract. In the development of ubiquitous and pervasive systems, it is understood that mechanisms are required to take adequate account of user preferences. This paper presents several key challenges for personalisation in pervasive environments and introduces the Daidalos solution developed as part of a European research project, Daidalos. The Daidalos personalisation system architecture goes beyond customary simplistic preference management to provide two aspects of dynamicity: first in the establishment of user preferences, where learning mechanisms are used to refine and update preferences when the need arises; second during the application of preferences whenever the context of the user changes. The paper discusses how this system meets the outlined challenges and details how the system has been evaluated.

Keywords. Context, learning, personalisation, preferences

1 Introduction

During nearly two decades since Mark Weiser described his vision of future ubiquitous computing [1], much research has been focused on advancing towards this goal. Devices such as mobile phones, laptops and PDAs allow us to take computational technology with us wherever we roam, and network technologies such as wireless access allow us to remain networked far beyond the reach of physical cables. As the user navigates through their mobile environment they require ubiquitous access to an expanding array of surrounding services and resources. To meet this requirement, the notion of pervasive computing [2] has emerged. The result is a new and exciting set of challenges.

With a potentially large collection of resources and services vying for the user's attention and requiring some level of management, this presents one of the major challenges for pervasive computing. To meet this challenge, it is generally understood that mechanisms must be provided which will alleviate the user from some of the detailed interaction and decisions required in order to manage such a system. For this reason personalisation has become an important component in the management of pervasive resources and environments, helping to satisfy the current needs of individual users.

In a pervasive environment personalisation is the set of processes that adapt the environment and the services in it to fit the user's needs. This results in different user experiences for different users, or for the same user in different contexts. The level of personalisation provided is dependent on the user information held by the personalisation system.

Several guidelines exist for the management of user information, often referred to as the *user profile*. W3C provides CC/PP (Composite Capabilities/Preference Profiles) Structure and Vocabularies [3] outlining profile management guidelines within a web environment. Similarly ETSI provides guidelines [4] for user profile management within a mobile communications environment. As yet, no guidelines exist for user profile management within a pervasive environment. The more static approach to profiles and their management, described by W3C and ETSI, does not correspond with pervasive environments and their dynamic nature.

Of particular importance are user preferences that are invaluable when determining how pervasive environments and resources should be optimally adapted. If we have a richer and more accurate preference set it follows that personalisation can reflect more accurately the user's wishes. However, the

establishment and maintenance of a rich and accurate preference set is a nontrivial task and indeed several major challenges need to be addressed. A successful personalisation system should provide support for preference creation and maintenance, alleviating the user of such tasks where possible. Equally important are the mechanisms to apply personalisation at the correct time.

This paper describes the Personalisation System implemented within the Daidalos project, which aims to provide all round support for preference establishment, management and application in a pervasive computing environment. Daidalos [5] is a European research project, a major aim of which was to deliver dynamically adaptive, personalised services in a pervasive environment to static and mobile users. The project was divided into two phases. In the first phase a prototype was developed that successfully demonstrated basic personalisation based on static, manually entered user preferences [6]. Such personalisation was applied in the selection of services and their customisation during instantiation. The second phase focused on the implementation of a more ambitious Personalisation system, which provided enhanced personalisation based on dynamic preferences that are established and managed on behalf of the user through monitoring and learning techniques. In addition such preferences are applied dynamically (based on user context) to constantly tailor the environment to the user as they navigate through it. This paper describes the final architecture developed in this second phase.

The next section looks at the main challenges of providing preference management in pervasive environments and section 3 provides a brief overview of related work . Section 4 describes Daidalos preferences and section 5 goes on to introduce the Daidalos Personalisation system highlighting the two dynamic aspects of preference learning and preference application. Section 6 discusses how the Daidalos solution meets the challenges outlined in section 2. An evaluation of the Daidalos Personalisation System is presented in section 7 and finally section 8 provides a conclusion.

2 Challenges

Providing a universal and dynamic solution to pervasive personalisation is no trivial task. Numerous factors and issues must be considered. The following sections highlight several of the most taxing challenges.

2.1 Generic Personalisation

Within a general pervasive system, personalisation is required at two levels. Firstly it is required to tailor the internal services (and their behaviours) that are part of the pervasive system infrastructure – so-called "enabling services". Secondly, it is required to tailor external services – third party services – to behave in accordance with the user's preferences. Daidalos third party services can have a number of personalisable parameters (e.g. volume, font size) whose values need to be set in accordance with the user's *personalisable parameter preferences*.

In the case of the Daidalos platform, personalisation is required in the following internal processes:

(1) Service filtering and ranking. When a service request is made, relevant services are discovered and the optimum ones are selected to meet the user's needs. Discovered services may need to be filtered (i.e. services that do not satisfy the user's preferences are removed from the list) and then ranked (the list is ordered) based on criteria specified by the user's *filtering* and *ranking preferences*.
(2) Session adaptation. During service runtime, more suitable services may become available. If this occurs, the session may be re-composed to include the new service, based on the user's filtering and ranking preferences.
(3) Network selection. As with services, there can be a variety of different networks available e.g. LAN, WLAN, GPRS, Bluetooth, etc., and for a given type of network one may have different network providers. During network selection the available networks are ranked based on some criteria specified by the user's *network preferences*.
(4) Virtual Identity (VID) Selection. Virtual identities are used to protect the real identity of a user [7]. The user can masquerade behind any number of VIDs each with different access rights and attributes. To minimise user interaction the most appropriate VID can be selected on behalf of the user based on their *privacy preferences*.

One can see from the above that the personalisation system must be able to deal with preferences used in a range of different processes throughout Daidalos enabling services and third party services. Where possible this should be done in a generic way to reduce duplication of preference management processes.

2.2 Finding the Balance between Explicit and Implicit Personalisation

Personalisation is often referred to as being one of two types: explicit personalisation or implicit personalisation. The distinguishing feature is the process for establishing and maintaining user preferences. Explicit personalisation relies on user preferences that are created and managed manually by the user. In contrast, implicit personalisation provides mechanisms to automatically create and manage user preferences on behalf of the user. Both approaches have various strengths and weaknesses.

Explicit personalisation allows the user ultimate control over their preference set. The user can add, delete or edit preferences as required through provided GUIs. Therefore the user has a mental knowledge of what is contained within their preference set and what automatic behaviours to expect as a result. This means the user can also better understand how to manipulate their preference set to change automatic behaviours when required.

However, manual management of an entire preference set is a laborious and time consuming task. To facilitate optimum personalisation, preferences should exist that indicate how to adapt all resources (services, networks and devices) with which the user is in common contact in any regular contextual situation. The boundless possibilities of context situations and available resources can lead to a large preference set with potentially complex preferences. Maintaining such a preference set manually will require a large portion of the user's time and effort, detracting from the benefits that personalisation aims to provide. The average user may have difficulty in expressing their preferences and some user behaviours may be so implicit that users may not identify them as explicit preferences. For example, a user may not be aware that he/she always selects a service within a certain price range at a certain time of day.

Further, as the user encounters new resources or situations within their pervasive environment new preferences may be required or existing ones may need major amendments. For example, the user may start a new job exposing him/her to an array of new situations and resources each requiring representation in the preference set. As the preference set grows, the time and effort required to explicitly manage it mushrooms.

Implicit personalisation removes preference management burdens on the user. Instead monitoring mechanisms capture and store observed user behaviour and environmental data. Learning techniques process this raw data to produce preferences which can be used to update the user's preference set. This approach allows the user to enjoy the benefits of personalisation with minimal effort beyond their normal interactions with services, networks and devices.

Although this style of personalisation is more in line with the pervasive ideal, if not implemented appropriately it can lead to incorrect or unexpected system behaviours which may hinder rather than help the user. This could quickly lead to a loss of confidence in the system followed by its abandonment. Therefore it is essential that any implicitly maintained user information held within the system is accessible to the user in a human-understandable format. This would go some way to returning a feeling of control to the end user.

The benefits and drawbacks of both personalisation approaches complement one another and it seems a successful personalisation system would provide support for both explicit and implicit methods. A major challenge lies in finding the optimal balance between the two. As a general rule the aim should be to maximise user experience while minimising user interactions and distractions. The difficulty is that there is a linear relationship between these two sub-goals which does not naturally conform to this rule.

2.3 Implicit Preference Management

An obvious challenge is how to provide effective mechanisms and processes that will enable the implicit establishment and maintenance of the user's preference set. This process should be continuous to enable the refinement and updating of the user preference set throughout its lifetime taking into account changing context and user behaviours. There are several sub-issues to consider. Firstly, monitoring mechanisms are required to capture and archive user behaviour and context states for later use by learning algorithms. Consideration must be given to what contextual and behavioural data should be

monitored. Context is a broad term so perhaps some refinement is required in deciding what context attributes of an environment are most important. Equally it may not make sense to monitor every user behaviour, for example it may not be useful to monitor every keystroke as the user is typing. The purpose of the monitored data is a strong determining factor in both cases.

Secondly, the monitored data must be processed by appropriate learning techniques to extract accurate, context-dependent user preferences. With a plethora of learning algorithms to choose from an optimal one must be selected for the particular problem domain of user preference learning. However, things are not quite as simple as this as different types of user preference may be better suited to different learning techniques. Thus the ideal situation would be to choose the best learning algorithm for the particular type of user preference concerned. An obvious variable in this decision process is accuracy but other equally important factors such as output readability must be considered in this user centric domain. Reflecting on the previous challenge of a balance between implicit and explicit personalisation, a chosen preference learning algorithm should enable user-understandable output to be generated in order to allow the user to view and control their preference set.

Thirdly, an implicit personalisation system must be able to adapt swiftly to changes in user behaviour. Without the ability to provide rapid response to such changes the user will inevitably need to increase their interactions with the system to correct erroneous automatic behaviours and to update their preference set in line with their new requirements.

2.4 Dynamic Application of Personalisation

Another challenging issue is the question of when to apply personalisation. Context-dependent preferences are exactly that, dependent on the context of the user. Therefore, mechanisms must be provided to correctly evaluate the preference rules against the user's current context and apply the correct result. However, user context is always changing as the user moves through their environment. This can render particular preferences void through time. For example, user preferences may indicate that an enhanced mode of some news service (which charges a fee) is preferred when the user is at work while the standard (free) mode is preferred when the user is at home. If the user starts the service at work (in the preferred enhanced mode) and then goes home, the service should be automatically re-personalised (during service runtime) to the free mode when the user arrives home. This dynamic application of personalisation requires monitoring of important context attributes and re-evaluation of preferences based on context changes.

3 Related Work

Over the past decade many projects in academia and industry have attempted to provide personalisation within ubiquitous/pervasive environments with varying degrees of ambition and success.

Early projects such as Microsoft's Easy Living project [8] and MIT's Intelligent Room project [9] focussed on gathering information about the situation of the user but did not consider higher level user information such as preferences. Static internal rules dictated how the environment should adapt depending on context states. In doing so they provided context awareness but no personalisation, with the result that environments and resources did not adapt to the needs of individual users. Satyanarayanan et al. [10] argue that a higher level of user information is crucial for useful system decision-taking, adding that "otherwise it will be impossible to determine which system actions will help rather than hinder the user". The Intelligent Room Project considered the use of the semantic web to provide higher levels of information but again, this focused on environment information and not higher level user information such as preferences.

IBM's Blue Space [11] and UMA's Intelligent Home Project [12] identified this requirement and incorporated user preferences into system decision-making processes to provide some level of personalisation. However, user preferences required manual entry and management. Due to the level of human effort required, user information and preferences manually entered by the user were minimal and so the level of personalisation was basic. CMU's Project Aura [13] attempted to improve the quality of personalisation by requesting higher level information, such as the user's current task. However such an approach places a great burden on the user thereby discouraging system use.

In an attempt to overcome this challenge the scope of personalisation has expanded to include the gathering, processing and ongoing maintenance of user information as well as its application. Implicit

personalisation involving monitoring and machine learning mechanisms allows systems to unobtrusively gain and maintain preference sets as a basis for system personalisation. Projects such as the Adaptive House [14], MavHome [15], and Gaia [16] make implicit personalisation the driving force of their personalisation systems and see full automation as their key goal. User behaviour is monitored and offline machine learning techniques extract preferences from the raw data in batch mode. The extracted preferences are used to update the user's preference set. Although preference management burdens have been mitigated, the focus on autonomy removes all possibility for user control, i.e. the user cannot quickly change a preference or stop the occurrence of an automated process. In addition, batch learning algorithms are by their nature slow to respond to changes (due to the periods of inactivity between batch executions). Although they will eventually learn new preferences, the inevitable frustrating situations encountered could reduce user confidence in the system, perhaps discouraging system use altogether.

The Synapse Project [17] introduces a balance between automation and user control by operating in active and passive modes where active is autonomic and passive involves user input through pop-ups. The selected mode is dependent on internal confidence levels. It is unclear if the user can manually control mode settings or how long it may take the system to switch between modes (if the current mode is incorrect). Such an approach must strike a balance between the possibility of more accurate personalisation (due to user input) and the possibility of numerous pop-up messages burdening the user with input requests.

Ubisec [18], Mobilife [19] and Spice [20] all employ offline learning techniques to handle monitored user actions and online techniques to handle explicit user feedback. It is unclear whether such a distinction between inputs will allow for better personalisation. Both types of input could be equally important to both online and offline processes. The online techniques employed assimilate feedback immediately into the user's preference set allowing for quick adaptation. With such over-writing, care must be taken to avoid the problem of catastrophic forgetting. A change in user behaviour may be due to a change of mind or a change in the environment (that the system cannot capture) that might be temporary or long-term. When the change is temporary it is undesirable to over-write long established information.

The Daidalos Personalisation System provides mechanisms for gathering, processing and maintaining user preferences. User behaviour is monitored and offline learning algorithms continuously extract user preferences on behalf of the user to ensure an up-to-date preference set with minimal user effort. Users can control personalisation manually or provide input to system decision-making through various GUIs that aim to mitigate user interaction where possible. This allows the user control over the personalisation processes with minimal burden. Common issues related to implicit preference management such as preference conflicts and changing user behaviours are addressed. Finally, dynamic preference application ensures that all context-dependent personalisation is as accurate and up-to-date as possible.

4 Daidalos Preference Rules

Before introducing the Daidalos Personalisation system let's consider how Daidalos preferences are represented. A generic preference format is used for all personalisation tasks throughout the platform and third party services. This generic format is IF-THEN-ELSE rules. The BNF description of the preference rule format is shown in Figure 1.

Figure 1: BNF for Daidalos IF-THEN-ELSE preference format

Most preferences have two main parts; the condition and the outcome. The condition part of a preference can include tests on context attributes, special predicates or situations. The special predicates include tests on context attributes such as nearness. Situations are a number of context attributes and/or service actions that can sufficiently represent the state of a user at a certain time. These situations are described in an ontology and are maintained by an Ontology Manager. A preference may have an empty condition to allow for context-independent (i.e. static) preferences that always hold true. In Daidalos we are chiefly concerned with context-dependent preferences.

The term preference outcome is introduced to refer to the outcome of evaluating a preference, i.e. what is true when the related context condition is true. An outcome can take one of three forms:
(a) An assignment of a value to an attribute e.g. news_service_mode = STANDARD

(b) A method call, e.g. startService(*news*)
(c) Another nested preference

4.1 Preference Hierarchy

Preferences are indexed by the type of service and/or the identifier of the particular service to which they are relevant. Service type preferences are generic to a service type and used to personalise new services based on their service type. For example, there are stored preferences for service type "multimedia" and the user installs a new VoIP application on their device. The Personalisation System will personalise the new VoIP application using the service type preferences relevant to service type: "multimedia". By monitoring the use of the VoIP application, learning algorithms will learn service instance preferences specific to the VoIP service and store them. The next time the VoIP application is launched, the Personalisation System will personalise it using the newly learnt service instance preferences specific to the VoIP service. Another type of preference exists that is very generic, i.e. it is not specific to any service instance or service type. These preferences are used very rarely, in situations where the Personalisation System tries to personalise a newly installed service whose service type is also new and there are no generic service type preferences in the user's profile. In Daidalos, the Personalisation System uses the Context Management System for storage of all the preferences. The hierarchy of preferences is shown below.

Figure 2: Preference hierarchy showing relationship between general, service type and service instance preferences

4.2 Stereotypes

Establishing a preference set for a user is a laborious and time consuming task. Initially the preference set may be very sparse or even empty meaning that little or no personalisation will be possible. Over time the monitoring and learning techniques described in this paper will build and manage a substantial set of user preferences but there will still be an initial period of minimal personalisation provision. To overcome this issue the use of stereotypes provides a useful technique for creating an initial set of user preferences. A stereotype corresponds to a user who may have a typical pattern of behaviour and for which a typical set of preferences may apply. This enables the user to obtain an initial preference set which can be enhanced and refined over time by monitoring and learning processes. The use of stereotypes in Daidalos is fully described in [21].

5 The Daidalos Personalisation System

The overall architecture for the Daidalos system consists of two major platforms. At the upper level one has the Pervasive Service Platform (PSP) and beneath this the Daidalos Service Provisioning Platform (SPP). This is described in more detail in [22]. The Daidalos Personalisation System resides in the upper level platform, the PSP. Figure 3 illustrates the high level architecture of the Personalisation System.

Figure 3: The Daidalos Personalisation System architecture

The Personalisation System relies on the Context Management System (CMS) in two ways. Firstly, the CMS acts as a store for preferences and learning data. Secondly, the CMS supplies the Personalisation system with contextual information, notifying it when any changes occur in the user's context that may affect current personalisation. The Personalisation system communicates with both third party and enabling services to monitor user behaviour through the actions the user performs during service usage and to apply preferences.

Two main control threads operate continuously within the system; the learning thread and the application thread. The learning thread controls the process of monitoring user behaviour, extracting preferences and updating the user's preference set. The application thread controls the process of monitoring user context, re-evaluating preferences and re-personalising services as required. Both threads operate in parallel. The following sections describe both control threads.

5.1 The Learning Thread

Figure 4 shows the data flow in the learning thread. The main processes are explained below.

Figure 4: Flow of data in the learning thread

5.1.1 Monitoring User Behaviour

The first step in the learning thread is to monitor the user's behaviour as they interact with services in the pervasive environment. The Daidalos Personalisation System implements centralised, passive monitoring through the Action Handler (AH) component. This implies that the monitored service must pass important information to the AH when the user performs some action within the service. The reasons for this decision are twofold. Firstly, this approach removes any static dependencies between the AH and service instances as available services in a given pervasive environment may change while the user is using a service. Secondly, this approach passes control of the monitoring process back to the monitored service and perhaps also the user allowing them the final decision as to what should and should not be monitored. In Daidalos an action is defined as:

- ***action:*** an act performed by the user which changes the state of an entity where an entity may be a service, network or device.

The format of an action is a tuple with an attribute, operator and value (e.g. video_state = play). To perform successful context-dependent preference learning it is essential to store monitored actions with an appropriate *context snapshot*. The context snapshot should represent the context attributes that bear most influence on the action occurrence and is a subset of context tuples of the form attribute, operator, value (e.g. location = home). However, defining the context snapshot for an action is non-trivial. Context can be described as anything that characterises the situation of an entity where an entity can be a person, place or object [23]. Therefore, the list of possible context tuples is potentially very large. It is undesirable to store every context tuple with every action as the user's behaviour history would rapidly become too large and duplicated with high volumes of irrelevant data. To overcome this problem similar actions are grouped together using ontologies and static context snapshots are defined for the various action groups. This allows the most relevant context attributes to be stored with each action. All monitored data (actions and associated context snapshots) are then stored in the User Behaviour History (UBH) held in the context database until it is required by learning processes.

5.1.2 Storing Monitored Behaviour

Daidalos preference learning operates in a batch mode. Initially, the UBH database is empty and must be populated during some training period to create a dataset upon which learning algorithms can be executed. These batch learning algorithms are run periodically. Between runs the UBH continues to grow as new actions are performed by the user.

5.1.3 Extracting Preferences

When a learning execution occurs, the data in the UBH must be pre-processed ready for input to the learning algorithm. Firstly the list of actions is divided into subsets depending on the service from which they originated. Secondly the subsets are further divided depending on the action performed and then processed by the learning algorithms.

The Daidalos Learning Manager (LM) follows a library based design to accommodate multiple learning algorithms. The pluggable nature of the LM also allows for the addition or removal of algorithms as required. The learning algorithms used in Daidalos include Neural Network learning, Bayesian Network learning and decision tree learning. The decision tree learning algorithm is primarily used for preference learning due to its decision tree output which maps easily to the adopted IF-THEN-ELSE preference format. It will therefore be the focus of further discussion (although please see section 6.3 for further mention of the various algorithms used). The neural network approach will be the focus of a forth-coming paper and details of the Bayesian algorithm can be found here [24].

The decision tree learning algorithm itself is based on Quinlan's C4.5 decision tree learning algorithm [25] which uses Gain ratios instead of simple Gain (as in ID3) to combat problems arising from attributes

with multiple values (such as 'time' or 'date'). Once the tree building algorithm has been performed on all subsets the result is a set of decision trees, one for each subset. These trees can then be translated into the Daidalos IF-THEN-ELSE preference rule format for user readability. This is a relatively trivial operation using pre-order tree traversal. Every leaf in the tree is an action tuple with a common attribute and different value. Once translated into the preference format the leaves become the outcomes of the preference, the branches become the preference conditions and the attribute of the action tuple relates to the preference name.

5.1.4 Updating the Preference Set

Each new preference produced by the learning execution is passed to the Preference Manager (PM) for merging with the user's existing preference set. The PM acts as a guardian of the user's preference set. It is responsible for the storage, retrieval, evaluation and updating of preferences within the preference set. The PM component is a passive component in the Personalisation System. It does not initiate or trigger any actions by itself. It is a serving component reacting to requests of both third party and Daidalos enabling services

For each new preference received by the PM, the first step in the merging process is to check if an equivalent preference already exists in the user's preference set. If no such preference exists, the PM will store the new preference in the user's existing preference set. In the case where a preference already exists, a merging algorithm is used to merge the new preference with the existing preference. This process begins by breaking down each IF-THEN-ELSE rule (passed from the LM) into IF-THEN branches which are easier to merge with the existing preference. Each branch is then merged one at a time based on the merging rules identified in [26] where the reader can find a more detailed description of the preference merging algorithm.

5.2 The Application Thread

Figure 5 shows the data flow in the application thread. The main processes are explained below.

Figure 5: Flow of data through the application thread

5.2.1 Monitoring the Environment

Since Daidalos preferences are context-dependent their outcomes may change through time if the user's context changes. This means that personalisation applied in one context may become invalid when the user context changes. Further, updates to the user's preference set caused by learning may mean that personalisation applied before the update is no longer valid. Therefore, we must constantly monitor user preferences to capture when outcomes change and re-personalisation is required.

The Preference Condition Monitor (PCM) component is responsible for this [27]. It enhances personalisation support in a pervasive environment by providing a framework for monitoring changes in preferences. The PCM monitors context attributes specified in preference conditions, as well as responding to learning updates in the preference set. If a context change or a learning update occurs the PCM requests the re-evaluation of the relevant preferences and communicates the re-evaluations to the appropriate services, allowing them to re-personalise themselves.

Several challenges arise. In particular, there are numerous context changes that can occur. The PCM is only interested in context changes that can specifically affect preference outcomes of *active* services (it is pointless to request the re-personalisation of inactive services). Therefore, the PCM must be aware of what services are currently active and deduce which context changes can affect the outcomes of preferences related to those services.

To do this, the PCM manages two internal lists that cross-reference each other. The service list contains all the currently active services along with a list of preferences related to each service. The Condition List contains all the context attributes that occur in the condition parts of any such preferences. Figure 6 shows a logical view of the list structures and organisation.

Figure 6: Conceptual view of the PCMs internal service and condition lists including cross referencing

The PCM listens for service start/stop notifications from the Service Management System. When a service start notification is received, the newly active service is added to the service list. The PCM then queries the PM to retrieve a list of all preferences related to that service. The PM returns a list of the preference names and for each name an associated list of context attributes that occur in the condition part of that preference. For example, consider the following preference:

IF location = 'HOME' AND status = 'FREE'
THEN
news_service_mode = STANDARD
ELSE
news_service_mode = ENHANCED

The context attributes in the condition part of this preference are *location* and *status* and hence these would be returned along with the preference name 'news_service_mode'

For each context attribute the PCM checks the condition list to see if such an attribute already exists. If not, it is added and the appropriate context change event is registered for. If the attribute is already on the list it does not need to be added again and no context change registration is necessary as a registration will already exist.

Links are created between the preference names in the service list and the attributes in the condition list. Such links aid the PCM when removing services from the service list. When a service stops the service name and its related preference names must be removed from the service list. It is also crucial that any context attributes which become orphaned due to the removal of this service are removed from the condition list and that the PCM un-registers for the related context change events. The links make it easy to identify context attributes that no longer have any links to preference names on the service list. Any such attributes are removed from the condition list and the PCM un-registers for related context change events.

5.2.2 Evaluating Preferences

The PCM constantly listens for any context change events it has registered for (based on condition list). When a change of context event is received the PCM searches the condition list to locate the related context attribute. Now the PCM must determine what preferences are affected by the context change event. This is a relatively trivial task due to the cross-referencing links between condition and service lists. The PCM follows each link emanating from the attribute in the condition list to arrive at all preferences in the service list that are dependent on the attribute. Each preference name and its related service name are added to a *preference re-evaluation list* containing all the preferences which are affected by the context change. The preference re-evaluation list is sent to the PM where each preference on the list is re-evaluated and their outcomes returned.

Preference evaluation involves the comparison of the conditional part of the preference with the user's current context. The PM exposes two interfaces. One is used by enabling services (such as the PCM) and provides access to the full range of PM functionalities. The other is exposed to third party services allowing any third party service to request a preference outcome so it can personalise itself when required.

When a preference evaluation request is received (either by an enabling service or third party service) the PM must retrieve the appropriate preferences from the CMS where they are stored. Each preference can be uniquely referenced using the service identifier and preference name. The preference hierarchy also allows multiple preferences to be retrieved based on service identifier or service type. Once retrieval is complete the PM evaluates each preference and returns the preference outcomes to the requesting service.

If a preference has no condition (i.e. it is context-independent), the static outcome is simply returned. Such preferences can only be manually created by the user as learning will always create context-dependent preferences. When evaluating context-dependent preferences the PM queries the CMS

regarding the user's current context. The returned values are compared with the conditional parts of the preference to identify what preference outcome is true in the current situation. When the PCM requests the re-evaluation of a list of preferences the PM evaluates each preference in sequence and stores the re-evaluated outcome in a return list which is eventually returned to the PCM.

5.2.3 Personalising the Environment

On receipt of re-evaluated outcomes from the PM, the PCM communicates each re-evaluated outcome to the appropriate service. It is then up to the individual service to decide whether to implement this new outcome as internal factors which may affect the decision cannot be known by outside parties such as the PCM.

Through the above mentioned steps the PCM allows services to continually adapt at runtime to meet the user's current needs in their current context. In a mobile environment such dynamic personalisation is vital. This especially applies to long-running services. If such services are only personalised at service instantiation the validity of that personalisation will decrease rapidly.

6 Meeting the Challenges

6.1 Challenge 1 - Generic Personalisation

With a variety of different enabling and third party services requiring personalisation support it is necessary for the Daidalos Preference Management System to handle all requirements in a generic fashion. This is achieved through the use of a common IF-THEN-ELSE preference format for all preferences throughout the system regardless of how they are created or where they are to be used.

A number of different learning algorithms are employed within the platform for the purposes of preference learning. For consistency and to allow the user to view, understand and manipulate their entire preference set, internal learning algorithm formats are converted to the homogeneous IF-THEN-ELSE preference rule format. This homogeneous format not only allows for common management of all preferences but also means that preferences are more portable (e.g. with the appropriate ontology a preference for a personalisable parameter in service X can be used to personalise a new service Y with an equivalent personalisable parameter).

Of course it is understood that some learning algorithms hold data in complex structures and translation to IF-THEN-ELSE rules can be non-trivial. Overcoming this issue has been an interesting challenge and we are currently developing an adapted neural network structure that can be bi-directionally mapped to and from IF-THEN-ELSE rules. This work has been conceptually outlined in [28] but will be fully described in a forth-coming journal paper on the network design and application.

IF-THEN-ELSE rules provide a powerful format to represent context dependent user behaviours. Over the course of time the rules may become large and complex but the structure is rich enough to represent any situation. Indeed it was identified that large, complex rules can be optimised by careful management processes and cleanup operations. Instead the main issue tended to be the occurrence of conflicting rules (when a new learnt preference directly conflicts with the content of the existing preference set). There are two main reasons for this.

Firstly, the available context is not rich enough to distinguish between two conflicting behaviours. This is an issue faced by all context dependent preference systems. It is not possible to monitor all context related to the user and therefore it is inevitable that the determining context attribute for some behaviour will not be available. Secondly, the user can change their behaviour. The behaviour change may be due to a lifestyle change such as a new job or it may simply be a change of mind.

Daidalos uses several mechanisms to overcome conflicts at the point of merging new preferences into the existing preference set. It was noted that the system could not rely on user input for conflict resolution. This is because batch learning operations often run overnight or during periods of inactivity. In both situations the user is not always available and hence other mechanisms are exploited.

Confidence levels are utilised throughout the platform. When a new preference is created (either by the user or learning processes) it is assigned a confidence level. In the case of user created preferences the confidence level is initialised to 100% since it is based on explicit input from the user. In the case of learnt preferences, the confidence level is initialised based on the ratio of positive to negative examples of the action-context co-occurrences in the UBH. Adaptations have been made to the traditional C4.5 decision tree algorithm (and other algorithms in the LM) to allow for the calculation of confidence levels.

When a conflict occurs confidence levels are the first line of action in resolving conflicts with a higher confidence level giving priority over a lower one. If confidence levels are not sufficient to resolve a conflict the preference evaluation mechanisms can revert back to the learning processes where a 'deep' learning cycle will generate a comprehensive preference to subsume the two conflicting ones. See section 6.3 below for details.

6.2 Challenge 2 - Finding the Balance Between Explicit and Implicit Personalisation

In current computational environments, the user is the controller and environments act upon user commands. In contrast pervasive environments aim to mitigate the need for the user to explicitly control or command environments. This has major advantages as long as the automated adaptation is what the user required. In situations where the user decides to change behaviour for some (possibly unknown) reason, removing all possibility for explicit user control may lead to frustrating situations and possibly discourage system use. Therefore Daidalos provides several GUIs with the aim of returning control to the user if and when they require it.

The Preference GUI

The preference GUI allows the user to view all their preferences, add new preferences or edit/delete existing ones. This is made possible by the generic IF-THEN-ELSE preference format adopted by the system. Since all learning algorithms have a requirement to translate internal data into the preference rule format it is possible to display all preferences to the user.

The preference GUI has two modes of operation; a) with the help of a wizard, the user selects the conditions or situations (collection of conditions) that a preference will be dependent on and the action to take under these conditions, and b) using a free text editor, the advanced user can write the IF-THEN-ELSE preference including the conditions in the form of a combination of logical and relational expressions and the corresponding outcomes. In this case, the editor aids the user with coloured syntax, error indicators and automatic error checking to prevent syntactical errors. The preference GUI can be invoked as an independent application where the user will be able to view all of their preferences. It can also be invoked from within the GUI of a service to view the preferences relating to that particular service.

The Feedback GUI

The Feedback GUI gives the user the opportunity to provide input regarding automatic behaviours. This may be to provide extra knowledge to the system or to intervene during the application of automatic behaviours. Such feedback is essential for further refinement of internal system knowledge and returns control to the user. However, it is equally essential to reduce user distractions where possible by minimising the requirement for user input. Therefore the Daidalos Personalisation system Feedback GUI operates in two modes: *implicit feedback* or *explicit feedback.* The mode of operation is determined by the nature of the proposal and the preference confidence levels mentioned above.

This approach echoes the methodology adopted by the Synapse project where automatic behaviours and prompts are controlled based on the current mode of operation (determined by confidence levels). However, as well as performing behaviours automatically if confidence levels are high enough, the two modes of the Feedback GUI provide an extra level of prompting that greatly reduces user distractions in comparison with the Synapse 'passive mode' prompting.

If the proposal is based on a preference with a confidence level below some threshold or is of a sensitive nature (e.g. transferring a video to an un-trusted public device) specific authorisation will be required by the user. The Feedback GUI will operate in explicit feedback mode prompting the user with a modal ‘ok/cancel’ dialog box in the centre of the screen. The user must explicitly enter their decision.

Alternatively if the proposal is based on a preference with a confidence level above some threshold (but not high enough to warrant automatic implementation) and is not of a sensitive nature (e.g. transferring a

video to a trusted private device) it will not require explicit input from the user unless the proposal is unwanted. In this case the user will be prompted that the action is about to happen by a small modeless pop-up at the bottom right of their screen. If the user does not push the 'cancel' button within some timeout period it is inferred the user agreed with the proposal and positive feedback is gained implicitly. Therefore the user can ignore such prompts unless the proposal is undesirable.

In both situations the user's response is passed to the AH where it is treated as an action and stored in the UBH as input to learning. The Feedback GUI also returns the user's response to the PM indicating what the user response was (i.e. ack/nack) to the proposal. This allows the PM to refine the confidence levels of preferences throughout their lifetime.

6.3 Challenge 3 - Automatic Creation and Maintenance of the Preference Set

The user can perform many actions when interacting with services and devices in their pervasive environment. Depending on the learning requirements not all actions will be useful to implicit preference management. Section 5.1.1 detailed what actions were of interest for Daidalos learning processes as well as how a context snapshot is defined for each action. Determining the context snapshot to store with each action is no trivial task. Although the current solution involving ontologies and action groupings goes some way to overcoming the issue, enhancements could be made to achieve a less static solution. This may involve the processing of actions and context changes in real-time as they occur rather than creating large stores of historical behaviour and contextual data for periodic processing.

Such incremental approaches are not implemented in the Daidalos platform. Indeed, the focus was not on the development of new techniques but rather on the use of existing benchmark techniques. With this in mind a number of familiar batch learning algorithms were utilised for preference learning including a Neural Network, a Bayesian Network and C4.5 decision tree learning. Decision Tree or rule-based approaches are commonly used in other pervasive projects such as Mobilife and Spice. Their popularity perhaps stems from the ability of humans to understand their learnt output compared to the internal complexity of network approaches. However, network approaches such as Bayesian and Neural Networks are renowned for their high performance in classification and association tasks. Indeed both Neural and Bayesian networks are already widely used within pervasive projects such as Synapse and GAIA for context inference purposes. As yet, all three algorithms utilised in Daidalos have been developed and evaluated independently, hence we have not yet reached a stage where we can advise which algorithm is best suited to what tasks, however our investigations continue with planned user testing in the near future. Despite this, several observations are noted regarding the general usability of batch techniques for preference learning.

Across pervasive projects various batch algorithms (including those used in Daidalos) have been utilised widely for preference learning throughout pervasive systems. Perhaps this is due to their benchmark status and accessibility. However the batch, offline way in which they operate seems an unnatural choice for use in such a dynamic environment. New user behaviours can appear frequently and user information such as preferences must adapt rapidly to take account of such changes. Several other projects such as SPICE and Mobilife have identified this issue and provide mechanisms for rapid response support in their batch learning systems. A common approach utilises explicit user feedback to immediately update the preference set. As mentioned in section 3, this approach can be risky as changes in behaviour may be short term or long term. Therefore, perhaps a more considered approach is required to remove the risk of over-writing established information with short-term updates.

Daidalos implements a different solution for rapid response to changes in user behaviour. In addition to feedback mechanisms, a *dual store* structure is adopted by the UBH. As a dual store the UBH acts as both a short-term and long-term behaviour store [29]. Figure 7 below illustrates the content of each store.

Figure 7: Dual store user behaviour history showing separate short-term and long-term stores

The short-term store (STS) only holds actions which have occurred since the last execution of the batch learning algorithm at time t_i. The STS is where any new actions are initially stored and the periodic preference learning executions are performed on the data in the STS. After each learning execution at time t_j, the data in the STS is copied to the long-term store (LTS) and the STS is cleared.

The LTS stores all actions which have happened since the user started to use the system at time t_0 until the last learning execution occurred at time t_i. It acts as a backup containing the entire back catalogue of actions performed by the user.

This approach provides several benefits. Firstly, since learning is initially performed on the STS only, recent changes in user behaviour (e.g. since the last learning cycle) will be identified more rapidly as they will not be inhibited by much older behaviours. Secondly, the entire back catalogue of actions performed are retained in the LTS providing a complete dataset which can be called upon if required.

Retaining the LTS also provides benefit to the conflict resolution processes mentioned above that occur when merging new learnt preferences with the existing preference set. When it is not possible to resolve conflicts using preliminary methods (involving confidence levels) the PM can call on the LM to perform a 'deep' learning procedure. In this procedure the learning algorithms process the entire user behaviour history to return a complete preference based on all related actions stored in both the STS and the LTS. This new preference subsumes the two conflicting preferences and is stored in the user's preference set.

6.4 Challenge 4 - Applying Dynamic Personalisation

In such a dynamic environment it is essential that personalisation is also dynamic to continually meet the changing needs of the user. Section 5.2.1 described how the PCM supports dynamic personalisation by monitoring context which may change the outcome of active preferences. This functionality has been essential both within the platform as well as for external third party services. It has been used to drive re-personalisation of all personalisation tasks required by the Daidalos Preference Management System and has been successfully demonstrated in terms of third party service re-parameterisation and session adaptation.

The PCM has been designed to communicate re-personalisation information to services without forcing its implementation. Instead the service can decide whether to re-personalise after considering internal factors that are unavailable to the PCM. Services can also query the user if required by utilising the Feedback GUIs available through the platform. This has been a successful approach sharing control between the platform and the third party service developer.

7 System Evaluation and Discussion

The system has been evaluated through several means. A qualitative analysis was carried out within the Daidalos project [30]. Almost 100 people were surveyed to determine concerns regarding the system and views on its usefulness. Encouragingly only 11% objected outright to the system acting autonomously on their behalf, as long as privacy was respected. Indeed, most concerns regarded privacy with almost half the respondents objecting to the disclosure of location information. In terms of usefulness the most telling statistic showed that the majority of respondents aged 20 to 50 were prepared to pay for the conveniences the system provides. This percentage rose to 77% if subsidies were available.

In addition an in-house quantitative analysis has been carried out to evaluate the performance of the Personalisation system. A simulation modelling approach was adopted to investigate the accuracy of the system in terms of preference learning and dynamic application of preferences on behalf of a user. The C4.5 learning algorithm was used during the evaluation as this is the primary preference learning algorithm. The other algorithms will be evaluated separately with a view to providing a future comparison. Figure 8 illustrates the evaluation framework developed and used in the experiments performed.

Figure 5: The evaluation framework

A Mobile Phone Emulator Service (MPES) was developed as a third party service running on top of the Daidalos platform. The service workload comprising user actions and context was simulated using subsets of the Reality Mining dataset [31], which contains data of mobile phone usage for 100 persons spanning a total of 350,000 person hours across all persons. Data from the Reality Mining dataset is used to populate the usage history of the service and update the user's current context. The History Control component extracts history data from the dataset to populate the UBH in the CMS. User context updates

are controlled by the Context Control component which extracts such updates from the Reality Mining dataset. The Experiment Control component manages the experiments performed by firstly recording how the Personalisation system personalises the MPES service and then analysing the accuracy of the applied personalisation.

6.1 The Mobile Phone Emulator Service (MPES)

The MPES partially represents a subscriber in a GSM cellular network. It was developed as a third party service on top of the Daidalos framework. MPES emulates a call function which is specified using the following parameters:

- Call Id
- Start Time of the call.
- End time of the call.
- Type of call: Voice Call, SMS Message or Packet Data.
- Phone Number: of the other user involved in the call.

The service includes a Call Menu feature which simulates prompting the user with regards the call type option they want to use. This can be either: Voice Call, Short Message or Packet Data. The Daidalos Personalisation system re-arranges the entries in the menu based on the user's learnt preferences and current context.

6.2 The Reality Mining Dataset

The Reality Mining dataset [31] has been constructed from an experiment performed by MIT Media Lab involving 100 subjects subscribed to different cellular networks. It has records of subject profiles, calls and context spanning a total of 350,000 hours across all subjects. The dataset includes the following data:

- A record of every call a subject had been involved in. Each record contains the call details including: Id, call Start and End Time, Duration, Call Type, Direction, Phone Number Id, Contact Number and Status.
- Subject Profile: Table 1 lists a number of subject profile items included in the dataset.

Call records in the Reality Mining dataset were used in the experiments performed (as described in Section 6.3) to simulate:

- The MPES usage history which was stored in the CMS and used by the L M to generate preference rules. The history attributes are: Call Type, Call Start Time and Phone Number Id.
- User context updates which the PCM listens to in order to trigger a re-evaluation of the preferences. Any preference updates are sent to the MPES service. The context update attributes are: Call Start Time and Phone Number Id.

Table 1: The reality mining subject profile

6.3 Experimental Design

The aim of the experiments performed was to evaluate the accuracy of the Personalisation system described in Section 5. Parameters (and values that were used in the experiments) that affect the performance of the system are listed in Table 2 under four main categories. The first category includes parameters related to user history. Such parameters influence the preference rules generated by the LM. The second category is concerned with the learning method used to learn the preference rules. As described in Section 5.1.3, the C4.5 learning algorithm is used by the LM to generate the preference rules. The third category includes parameters related to the user's current context attributes and their values. As described in Section 5.2.2, the PCM component listens to changes in context attribute values to trigger the re-evaluation of preferences and hence the re-personalisation of the MPES service. The final category includes parameters related to the profile of the users interacting with the MPES service. Various user profile parameters were identified that can influence the performance of the Personalisation system. All the data used from the Reality Mining dataset in the experiments performed was for users with the listed values for such profile parameters.

Table 2: Parameters that can affect the performance of the Personalisation System.

Three sets of experiments were performed; U1_EXP, U2_EXP and U3_EXP, with each experiment set simulate a different, randomly selected, user U1, U2 and U3 interacting with the MPES service. The aim was to investigate the consistency of Personalisation system in accurately personalising the service for different users. As described above, U1, U2 and U3 history and context data was extracted from the Reality Mining dataset. For each user, the first 500 entries in the Reality Mining dataset were used in the experiment for that user. User data was then divided into 10 data parts as shown in Table 3. In each experiment, 10 runs were performed using nine user data parts as history data and one data part as test data (context updates). It was ensured that in each run, a different user data part was used as test data.

Table 3: The division of user data (history) into parts

The following steps were followed in all the experiments performed:

(1) The History Control component loads the history data into the CMS.
(2) Experiment Control sends a request to the LM of the Personalisation system to perform a learning cycle. New preference rules are generated.
(3) Context Control uses the current test data to simulate a series of context changes that occur discretely in time.
(4) Experiment Control records how the Call Menu in the MPES is re-personalised (which Call Type option is at the top of the menu list) after every context change.
(5) At the end of each run, the top item in the Call Menu after each context change is compared (by Experiment Control) with the actions associated with context change entries in the Reality Mining dataset to calculate the personalisation accuracy.

6.4 Experiment Results and Analysis

For each experiment run, the personalisation accuracy was calculated based on the number of times the system personalised the MPES service correctly. The Personalisation system accuracy achieved in U1_EXP for each of the 10 experiment runs performed is shown in Figure 8 in descending order. It can be seen that personalisation accuracy as high as 96% was achieved in one of the runs. As shown in Figure 11, an average accuracy of 84% was achieved across the 10 experiment runs performed.

For the U2_EXP set, better performance was achieved as shown in Figure 9. An accuracy of 98% was achieved with an average accuracy of 96.2% across runs as shown in Figure 11. The poorest results were achieved in U3_EXP as shown in Figure 10, with a highest accuracy of 92% and an average accuracy of 75.4% across runs. This lower average was due to poor accuracy results in two of the runs in U3_EXP where an accuracy of 42% and 52% were achieved reflecting a greater mismatch between the learnt preference rules (based on history data) and the test data parts used in those two runs. This could be seen as an indication of the unpredictable nature of U3.

It should be noted that user feedback was not utilised during the experimental testing phase. In real-world usage such mechanisms would enable the system to identify and adapt to inaccuracies much more rapidly, hence reducing the possibility of sustained personalisation errors. The provision of a preference management GUI can also provide support by enabling the user to manually update any inaccurate preferences.

Overall, it can be seen that the Personalisation system achieved acceptable personalisation accuracy levels in the experiments performed. It could also be noted that 100% personalisation accuracy was achieved in one instance where other subsets of the Reality Mining dataset were selected randomly to simulate user history and context.

Figure 6: Graph showing the accuracy of the Personalisation System for user U1 over 10 test runs

Figure 7: Graph showing the accuracy of the Personalisation System for user U2 over 10 test runs

Figure 8: Graph showing the accuracy of the Personalisation System for user U3 over 10 test runs

Figure 9: Graph showing the average accuracy of the Personalisation System across all 10 test runs for users U1, U2 and U3

Conclusion

As we continue towards the goal of true pervasiveness many challenges must be overcome including the management of pervasive environments and the ever increasing number of computational technologies therein. It is not only unrealistic to place this burden on the user but also contradictory to the pervasive idea. Therefore the boundaries of personalisation have expanded to include not only the application of user preferences but also their maintenance. This has raised some interesting challenges that a pervasive system offering personalisation must tackle. Some of the more taxing issues encountered through our research were described and discussed followed by an overview of related work in this research field.

The following sections introduced the Daidalos Personalisation system which aims to provide all round support for building and maintaining an up-to-date user preference set for use in a pervasive computing environment. The overall architecture of the system and its various components were described through the two parallel threads of control. The first thread controls the gathering and maintenance of the user preference set while the second thread controls the application of preferences.

The Daidalos Personalisation system addresses each of the challenges outlined in section X and these solutions were presented and discussed. A common rule format supports generic personalisation throughout the platform and third party services. It also helps in the balance of explicit and implicit processes allowing the user to understand and manipulate preferences stored within the system. The preference learning process was outlined including what user behaviours and context are monitored, the various algorithms used to process it and how new learnt information is incorporated into the preference set. Issues such as conflicts and rapid response are handled with minimal dependency on the user. Finally dynamic (re)personalisation is supported to ensure that in the ever-changing environment the user's current needs are met.

The Daidalos Personalisation system built upon a simplified architecture implemented in the first phase of the project. The enhanced architecture described in this paper is fully integrated into the project's second phase infrastructure which was demonstrated in Dec 2008. Further analysis and experimentation has shown that the system holds a good average accuracy with regard to service personalisation based on learnt user preferences. However, the unpredictable nature of users can often lead to reduced accuracy. In real-world usage, user feedback and preference management GUIs will mitigate such results.

Acknowledgements

This work was supported in part by the European Union under the FP6 programme (Daidalos project) and is being developed further under the FP7 project Persist. The authors also wish to thank MIT for use of their Reality Mining Dataset and all their colleagues in the Daidalos project developing the pervasive system. However, it should be noted that this paper expresses the authors' personal views, which are not necessarily those of MIT or the Daidalos consortium. Apart from funding the Daidalos project, the European Commission has no responsibility for the content of this paper.

References

[1] Weiser, M., 1991, "The Computer for the 21st Century", Scientific American, vol. 265(3), pp. 94-104.

[2] Saha, D., Mukherjee, A., 2003, "Pervasive Computing: A Paradigm for the 21st Century", Computer, vol. 36, issue 3, pp. 25-31.

[3] W3C, 2007, "Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 2.0, Available online: http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430/, Accessed on June 17th 2009.

[4] ETSI, 2005, Human factors (HF); User Profile Management, ETSI Guide, EG 202 325 v1.1.1, Available online: http://pda.etsi.org/exchangefolder/eg_202325v010101p.pdf, Accessed on June 17th, 2009.

[5] Daidalos, http://www.ist-daidalos.org

[6] Williams, M.H., Roussaki, I., Strimpakou, M., Yang, Y., MacKinnon, L., Dewar, R., Milyaev, N., Pils, C., Anagnostou, M., 2005, “Context-Awareness and Personalisation in the Daidalos Pervasive Environment”, International Conference on Pervasive Systems (ICPS 05), pp 98-107.
[7] Girao. J., Sarma, A., and Aguiar. R., 2006, “Virtual Identities - A Cross Layer Approach to Identity and Identity Management“, in Proc. 17th Wireless World Research Forum.
[8] Brummit, B., 2001, “Better Living Through Geometry”, Springer-Verlag, Personal and Ubiquitous Computing, Vol.5, No. 1, pp. 42-45.
[9] Peters, S., 2003, “Using Semantic Networks for Knowledge Representation in an Intelligent Environment”, Proceedings of the IEEE International Conference on Pervasive Computing and Communications, pp. 323-329.
[10] Satyanarayanan, M., 2001, “Pervasive Computing: Vision and Challenges”, IEEE Personal Communications, Vol. 8, Issue 4, pp. 10-17
[11] Yoshihama, S., 2003, “Managing Behaviour of Intelligent Environments”, Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom ’03), pp. 330-340.
[12] Lesser, V., 1999, “The Intelligent Home Testbed”, Proceedings of the Anatomy Control Software Workshop (Autonomous Agent Workshop), January 1999, pp.8.
[13] Sousa, J.P., 2006, “Task-based Adaptation for Ubiquitous Computing”, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Special Issue on Engineering Autonomic Systems, Vol. 36, No. 3
[14] Mozer, M.C., 2004, “Lessons from an Adaptive House”, In D. Cook & R. Das (Eds.), Smart Environments: Technologies, protocols and applications, pp. 273-294
[15] Youngblood, M.G., 2005, “Managing Adaptive Versatile Environments”, Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom ’05), pp.351-360.
[16] Ziebart, B.D., 2005, “Learning Automation Policies for Pervasive Computing Environments”, Proceedings of the 2nd International Conference on Autonomic Computing (ICAC ’05), pp. 193-203.
[17] Si, H.K.Y., 2005, “A Stochastic Approach for Creating Context-Aware Services on Context Histories in Smart Home”, ECHISE2005, Pervasive 2005, pp. 37-41
[18] Groppe, J., 2005, “Profile Management Technology for Smart Customisations in Private Home Applications”, Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA ’05)
[19] Strutterer, M., 2007, “Managing and Delivering Context-Dependent User Preferences in Ubiquitous Computing Environments”, Proceedings of the 2007 International Symposium on Applications and the Internet Workshops (SAINTW ’07)
[20] Cordier, C., 2006, “Addressing the Challenges of Beyond 3G Service Delivery: the SPICE Service Platform”, Workshop on Applications and Services in Wireless Networks (ASWN ’06)
[21] Papadopoulou, E., McBurney, S., Taylor, N., Williams, M.H., Lo Bello, G., 2008, “Adapting Stereotypes to Handle Dynamic User Profiles in a Pervasive System”, Proc. Fourth International Conference on Advances in Computer Science and Technology (ACST 2008), Malaysia, April 2008, pp. 7-12.
[22] Williams, M.H., Taylor, N., Roussaki, I., Robertson, P., Farshchian, B., Doolin, K., “Developing a Pervasive System for a Mobile Environment”, 2006, Proc. eChallenges - Exploiting the Knowledge Economy, pp. 1695-1702.
[23] Dey, A.K., 2001, “Understanding and Using Context”, Personal and Ubiquitous Computing Journal, vol. 2, pp. 139-172
[24] Frank, K., Rockl, M., Robertson, P., 2008, "The Bayslet Concept for Modular Context Inference", Proc. Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2008), Valencia, Oct 2008, pp. 96-101.
[25] Quinlan, J.R., 1993, “C4.5: Programs for Machine Learning”, Morgan Kaufman (1993).
[26] McBurney, S., Papadopoulou, E., Taylor N., Williams M.H., 2009, “Implicit Adaptation of User Preferences in Pervasive Systems”, Proc. Fourth International Conference on Systems (ICONS 09), pp. 56-62.
[27] Papadopoulou, E., McBurney, S., Taylor N., Williams M.H., 2008, “A Dynamic Approach to Dealing with User Preferences in a Pervasive System”, Proc. International Conference on Intelligent Pervasive Computing (IPC-08), Sydney, Dec 2008, pp. 409-416.
[28] McBurney, S., Papadopoulou, E., Taylor, N., Williams, M.H., 2009, "Giving the User Explicit Control over Implicit Personalisation", Proc. Persist Workshop on Intelligent Pervasive Environments (AISB '09), Edinburgh, April 2009, pp. 16-19.

[29] McBurney, S., Papadopoulou, E., Taylor, N., Williams, M.H., 2008, “Adapting Pervasive Environments through Machine Learning and Dynamic Personalisation”, International Conference on Intelligent Pervasive Computing (IPC-08), Sydney, Dec 2008, pp. 395-402.
[30] Taylor, N. K., Robertson, P., Farshchian, B. A., Doolin, K. J., Roussaki, I., Marshall, L., Mullins, R., Druesedow, S., Dolinar, K., 2010, "Pervasive Computing in Daidalos", To appear in IEEE Pervasive Computing, Vol. 9, Issue 2, Early Access available online at: http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=7756, accessed 14th April 2010.
[31] MIT Media Lab, The Reality Mining Dataset, Available online: http://reality.media.mit.edu/dataset.php, accessed on 5th Oct 2008.