Personalization and Context Aware Services: A Middleware Perspective

Fahim Kawsar[1], Kaori Fujinami[1], Susanna Pirttikangas[2], Tatsuo Nakajima[1]
[1] Department of Computer Science, Waseda University, Japan
[2] Department of Electrical and Information Engineering, University of Oulu, Finland
{fahim,fujinami,tatsuo}@dcl.info.waseda.ac.jp, msp@ee.oulu.fi

ABSTRACT
The basic goal of any context aware system is to provide some proactive services adopting users' context. However, often in real life proactive behaviors create complex problems. The end users of the system have an implicit understanding of the system. If the context aware behavior of the system conflicts with their understandings and reacts differently from users' expectation applications success ratio reduces radically. So, personalization is a crucial factor for the success of the proactive applications. In this paper we discuss this particular aspect from a middleware perspective. Initially we present a requirement analysis and propose a classification scheme for structural representation of preference information in proactive systems. Then we present a middleware, part of which exploits this classification to support application developers to facilitate the end users with the flexibility to personalize the context-aware services. This facility stems entirely from the middleware and is independent from the applications.

Keywords
Preference, Personalization, Middleware, Context Awareness.

1. INTRODUCTION
Providing proactive context aware services based on perceived users' context is one of the major focuses of ubiquitous computing. However, proactive systems involving multiple smart artefacts often create complex problems, if their behaviors are not inline with users' preferences and implicit understandings. Every user has own understanding and perspective towards a system and wants to personalize it in own way regardless of its proactive characteristics. For the success of the application, we argue every proactive context aware system should provide personalization facility.

Inspite of being an essential and recurring requirement of proactive application, there has been no attempt to generalize preference management support. This is because of the following two reasons:

1. There is no guideline for representing the preference information in a structured manner.
2. There is no middleware that provides a common base for supporting preference management features in proactive applications.

Both these reasons are complementary to each other and rooted for one particular issue that preference information is application specific. However, we argue that even though each application has different personalization perspectives, it is possible to accommodate them into some abstract common categories for generalizing preference management in application development.

In this paper, we have tried to validate our argument. First, we present our initial requirement analysis that guided us to approximate what user wants to personalize in a context aware system in general. Based on the analysis we propose a classification of preference types to represent the preference data in a structured way.

Then we present a component of a middleware that implements this taxonomy and assists the developers to handle personalization functionalities. For collecting users' preference information the middleware provides two interaction techniques: Graphical User Interface and Voice. These two techniques are selected based on the findings of our user study.

The novelty of our approach is that the middleware takes care of the identification, extraction, storage and representation of preference information. So the developers do not have to care about these issues rather they can only focus on how to utilize the information for smooth application flow that conform to users' preference. Furthermore, the taxonomy presented can act as a guideline for the proactive application developers to structure the preference support.

The rest of the paper is organized as follows: in section 2, we have presented the requirement study that includes three phases of interactions with the target user group to identify the personalization aspects. Section 3 presents the taxonomy of the preference information and other design issues. In section 4, the implementation of the middleware and the programming model are presented briefly with an illustrative application. Section 5 discusses on several issues where as section 6 compares the uniqueness of our approach with similar works. Finally, section 7 concludes the paper.

UbiPCMM06 September 18, 2006, California, USA

2. REQUIREMENT STUDY

Preference information in proactive applications is usually application specific. So, generalizing them into some categories for common base support is difficult. However, our hypothesis is that even the personalization options differ from application to application; there is a common pattern that can be generalized. We have approached towards this resolution through an infromal user study that can guide us to argue about the validity of our hypothesis. Our goal was to come up with some common categories of preference information that can be represented for possible system support. In this section we will describe the findings of our requirements study.

We have gone through three different phases (interview, user survey, brainstorm) to identify two issues:

1. What are the possible features that users want to personalize in a proactive system incorporating many physical artefacts?
2. What interaction mechanisms are well accepted for personalization functions?

The profiles of the participants are mentioned in table 1. We conducted three different phases of user interaction for rationalizing the quantative results through the observation of focus groups facial and immediate expressions. In all three phases, for clarifying to the target group what sort of systems we are talking about, we presented them 10 minutes video clip composed of various scenes from the movie "Minority Report". In the video clip based on the presence of a person many things in the home (TV, Music System, Door, Lighting System) are automatically started and configured perceiving user's context in a proactive way. All the interactions are voice controlled.

2.1 Guiding Topics

In all three phases the following topics were presented to the participants:

1. Impression: How do you like the system and do you want to use such smart artefacts and systems?
2. Preference Types: What sort of personalization do you require, i.e. what features do you want to personalize? Can you break it down to implicit cases?
3. Interaction Technique: How do you want to interact with the system for personalization? Options offered are: Voice, Gesture, Graphical User Interface, and Implicit Controller.

Table 1:Profile of the Participants

Phase	Participants	Age	Professions
Interview	1 Female 2 Male	24-31	Business Analyst, Graduate Student.
User Survey	4 Female 6 Male	22-42	Lawyer, Researcher, Graduate Student
Brainstorming	2 Female 3 Male	23-35	Researcher, Graduate Student

2.2 Interview

In the first phase we interviewed 3 people. After the video clip, their initial impression was "It is cool to live in such home" and mentioned many interesting points. It appeared that they want to personalize the proactive behavior completely that includes timing of action, selection of specific artefacts (like overhead lamp or desk side lamp etc.) or digital contents (like an audio clip), emotional mood based actuation, etc. One subject mentioned that she wants the system to know with whom she is and should reconfigure automatically. When being asked how they want the system to know their preference, one subject told it would be best if she does not need to do anything and the system will automatically learn from her past preferences but if she does like it she will change manually. The other two subjects mentioned that they want to put their preference manually. Regarding interaction technique, two of them preferred voice to other techniques. One subject told that he does not like the voice based interaction, because it is difficult to remember what to say to the system. Interesting finding was, all of them mentioned their reluctance towards gesture-based interaction for preference management or controlling the system because they feel, it is very ambiguous. The summary of the interview is presented in table 2.

Table 2: Summary of the Interview

Topic	Response
Impression	Like: 100% Dislike: 0%
Preference Type	Action, Timing, External Presence, Content, Emotional State, Artefact Participation.
Interaction Technique (Rank Based)	GUI (93.33%) Voice (86.67%) Controller (60%) Gesture (40%)

2.3 User Survey

In the second phase we conducted user survey of 10 people. Various questions were asked focusing on the guiding topics. The summary is presented in table 3.

Table 3: Summary of the User Survey

Topic	Response	Comment
Impression	Like: 80% Dislike: 20%	"The system will make me lazy."
Preference Type	Action, Timing, Emotional State, Artefact Selection, Control Mechanism	"I want to use specific artefact that I like." "I want to designate the role of each artefact"
Interaction Technique (Rank Based)	GUI (76%) Controller (66%) Voice (64%) Gesture (46%)	"I don't want to use voice because it is noisy to others."

2.4 Brainstorming

In this phase, we had a close discussion session for 2 hours focusing on the guiding topics. All the participants were the members of the author's lab including the author. We started the session with the video clip and identified various aspects. Then we focus on specific issues. The summary of the discussion session is presented in table 4.

Table 4: Summary of the Brainstorming Session

Topic	Response	Comment
Impression	Not Discussed	Not Applicable
Preference Type	Action, Timing, Content, Artefact Selection, Control Mechanism	"If preference is configured based on history it would be best."
Interaction Technique (Rank Based)	GUI (92%) Voice (72%) Gesture (68%) Controller (44%)	"If the gesture is simple I would like to use that." "Interaction feedback should provide a cue for decision making."

2.5 Summary of the Study

One of the interesting observations of our user study was the facial expression of the participants when the movie clip was being shown. It seemed obvious that if the proactive applications can be presented in the way they want, applications' success ratio will increase to a significant rate. However, as to our major concern, the derived statistics from the user study exhibits that, the users want to have personalized functions regardless of the functional advantages of the proactive behavior. 2nd row of table 2,3 and 4, clearly exhibit various personalization requirements of the users. Readers may argue about the scope of the applications being shown on the movie. We understand this issue and do not claim that our study is a complete one to validate our hypothesis. But, to some extent the extracted result gives a clear indication about the diversity of the personalization options in context aware systems. Also we have tried to identify the possible interaction techniques that users may want to use for providing the personalization information and the extracted statistics give us a good guideline. As to our next concern of providing system support for generating the personalization facilities, the findings of the user study has been used as the design guideline. In the next section, we present design issues where we have discussed how the extracted responses and statistics of the user study are used to generate the design decisions.

3. DESIGN DECISIONS

In this section, we will introduce the design concerns of the middleware component and the decisions that we have made for the implementation. Considering the summary of the user study, it is visible that, the user wants to have full personalization facilities even in a complete proactive system. Furthermore, we have seen that the required preference facilities depend on the functionalities of the applications. But, in general these preferences can be grouped into some generic classes to represent the preference information in a structured way.

3.1 Structured Representation of Preference Information

Structures representation is useful for the application developers to handle the preference functions in a unified way. So, based on the focus group input feedback, our prior experiences [5,6,7] and understanding, we have classified the preference attributes into following five generic types.

1. **Artefact Preference:** This attribute is for enabling the user to select the participation of any artefact in the cooperative smart environment. For example a user may wants to use the desk side light but not the overhead light in a smart lighting system. Using this attribute user can manipulate the artefacts' participation.
2. **Action Preference:** This attribute is to enable the user to set action according to their preference. Usually a system consists of several actions that it actuates based on some conditions. Users can enable or disable actions using this attribute. For example, user can enable/disable the automatic confirmation mail sending action of a system, that notifies user whenever system is reconfigured.
3. **Context Preference:** This attribute is to enable user to manipulate the participation of the context information in the system. Here the context information can be any information that the system developers consider as context that affects the systems behavior, like location, position, activity, emotional state etc. The system developers should define the overall systems functionality and integral context conditions. The users should use this attribute to provide their preference. For example, a user may specify that he/she does not want the music system to turn on automatically when he/she is with some one else.
4. **Control Preference:** This attribute is to provide user the flexibility to select their preferred control mechanism. For example, a smart display may have two modes: abstract and detail. To navigate from the abstract mode to detail mode the user can use voice or GUI or tangible interface. User can mention his/her preferred mechanism for this control action. Also, this attribute can be used in general to capture users interaction with the system, as we will see in the next section.
5. **Generic Preference**: Finally this attribute is for providing user the flexibility to provide their preference regarding the generic aspects of the system. For example if the system actuation is music, then which sound clip to play, if the system actuation is display then what should be the background color, font size, or timing of the display etc. In general, it should accommodate the preferences like content preference, timing preference or other generic features' preference of the application in context.

We anticipate that these five levels of preference attributes can accommodate most of the preference options for context aware systems in general. So, the middleware in concern should support this structured representation of the preference information.

3.2 Interaction Techniques

Our next concern is the interaction technique that the user should use for providing their preference information. The accumulated average of the user study on this issue indicates (as shown in figure 1) that, GUI has highest preference followed by Voice, Gesture and Controller. This result also matches the findings of [1] where speech was preferred as interface for controlling home appliances. Based on this statistics we have decided to provide two facilities in the middleware: GUI and Voice. That means, the end user can provide their preference to personalize the system using GUI and/or voice.

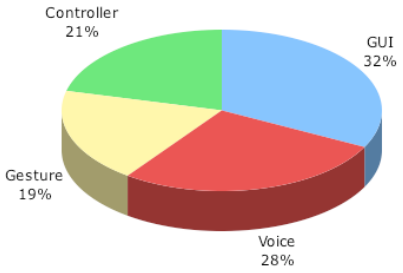

Figure 1: Preference of Interaction Technique

3.3 Required Features

Considering the two previous subsections, we require that the middleware to implement five types of preference information and to provide APIs to application developers to manage this preference information in a structured way. For manipulating the preference information we have decided to use two attributes:

1. **Positive:** This attribute represents user's willingness to use the options in context.
2. **Negative:** This attribute represents user's unwillingness to use the options in context.

Also the middleware should provide the two interaction mechanisms to collects preference information.

1. **Speech Recognition:** End user can provide their preference in natural English language phrase. The application developer provides the list of phrases that can be used for preference.
2. **Graphical User Interface (GUI):** End user can provide their preference by manipulating GUI that represents the applications.

Furthermore, the collected information should be classified (following the taxonomy of preference) internally by the middleware and should be represented to the application in a unified way. These supports should entirely be decoupled from the application meaning that the developers do not have to consider about the GUI generation or voice recognition/synthesis for collecting user input or to classify the input information into specific categories. The application developers' responsibilities are to:

1. Provide the application specific preference options in proper syntax using the middleware APIs.
2. Implement the application logic that handles the specific preference information when being delivered to the application.
3. Activate the desired interaction technique using the middleware APIs.

With these design concerns the component is implemented. In the next section we will discuss about the implementation.

4. IMPLEMENTATION

The component that handles the personalization aspects is basically part of a generalized middleware Prottoy [7] that is designed to extract, distribute, manage, model and represent the context information. The personalization component works on top the core components of this middleware as a pluggable component. For clarity here we will introduce Prottoy in a summarized manner. Prottoy is composed of three core components and three pluggable components.

The core components are:

- **Resource Manager:** Responsible for resource discovery, managing location information and reconfiguration of the underlying environment.
- **Artefact Wrapper:** Responsible for encapsulating artefacts and offering artefact service and context information to applications.
- **Virtual Artefact:** Responsible for providing unified interface to applications for interacting with the underlying layers.

The pluggable components are:

- **Interpreter:** Responsible for interpreting context information in application specific ways.
- **Preference Manager:** Responsible for managing preference and control information.
- **Artefact Proxy:** Responsible for providing context storage information.

For using Prottoy as a middleware for context aware applications developers usually create virtual artefact instances in their application that represent physical artefacts or other context sources. Using virtual artefact API developers can interact with the underlying physical artefacts or context sources. Furthermore, developers can use the pluggable components in the application based on application specific requirement. The preference manager component contributes to the issues we are focusing on this paper. For detail description of the other components please consult the reference [7]. The preference manager implements the taxonomy of preference mentioned in section 3.1 and the two types of input mechanism for capturing end user preference information.

4.1 Architecture of Preference Manager

Figure 2 shows the internal architecture of the preference manager. In the following the internal architecture is described.

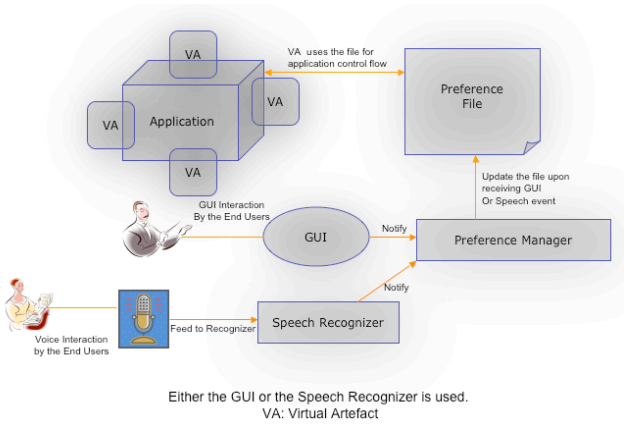

Figure 2: Internal Architecture of Preference Manager

1. **Preference File:** This file is the key player through which the virtual artefact component of Prottoy and the preference manager communicate with each other in a transparent

fashion. The virtual artefact internally uses this file to identify end users' preference regarding application behavior and adopt the preference at run time. This file is generated dynamically during the deployment time of the application by scanning application code. The file contains the application specific preference information that the application developers provide in the application code. Developers can provide the artefacts and actions that the application usage with corresponding preference options. These two features are manipulated by the positive and negative attributes as mentioned in the previous section. Also, they can provide context, control and generic preference options in the same way. However in case of these three preference types, the positive/negative attributes are not used. Figure 3 shows a snapshot of this file for the application we will mention in the programming model section.

2. **Preference Manager:** The preference manager is the central coordinator of the other components. When it receives GUI event or speech event from speech recognizer it looks for appropriate match in the preference file and if it finds a match it updates the file accordingly reflecting end users preference and notifies the application.
3. **Speech Recognizer:** This is the speech handler of the preference manager. When the application starts, it listens for any command or preference statement. When it recognizes a phrase it notifies the preference manager to update the preference file. For speech recognition we have used Sphinx [21].
4. **GUI Generator:** This is the GUI counter part of speech recognizer. It provides a GUI to the end user for providing the preference. It notifies the preference manager when preference is changed.

```
<?xml version="1.0" encoding="UTF-8"?>
<application-preference>
        <artefact-preference>
            <artefact>
                <id>artefact-1</id>
                <name>sentient lamp</name>
                <preference>preferred</preference>
                 <positive-phrase>
                        <phrase>I want to use light</phrase>
                 </positive-phrase>
                 <negative-phrase>
                        <phrase>I do not want to use light</phrase>
                 </negative-phrase>
            </artefact>
        </artefact-preference>
        <action-preference>
            <action>
                <id>action-1</id>
                <name>switching</name>
                <preference>preferred</preference>
                <positive-phrase>
                    <phrase>I like automatic lighting</phrase>
                 </positive-phrase>
                 <negative-phrase>
                    <phrase>I hate automatic lighting</phrase>
                 </negative-phrase>
            </action>
        </action-preference>
        <context-preference/>
        <control-preference/>
        <generic-preference/>
</application-preference>
```

Figure 3: Structure of the Preference File

4.2 Programming Model

The preference manager provides very simple APIs for the application developers to handle the tasks mentioned in section 3.3. Table 5 enumerates the major APIs and their functionalities.

Table 5: Preference Manager API

API	Functionality
`public String addArtefact(String name)`	For adding an artefact.
`public String addAction(String name)`	For adding an action.
`public void addPositivePhrase(String id, String phrase)`	For adding positive phrase for an artefact or action. this phrase represents users willingness to use a specific action or artefact.
`public void addNegativePhrase(String id, String phrase)`	For adding negative phrase for an artefact or action, this phrase represents users unwillingness to use a specific action or artefact.
`public string addPreference(String name, String prefType)`	For adding preference information.
`Void addPhrase (String id,String phrase)`	For adding generic phrase that reflects users preference.
`public void subscribe(Object source, string callback)`	For subscribing to callback. The callback receives preference and control information that developers can use.
`public Boolean checkPreference(String id)`	For checking artefact/action preference.

The code snippets in the following demonstrate the usage of some of these APIs in a very simple application that consists of a overhead light embedded with an ambient light sensor. The light is automatically turned on/off based on the sensed ambient light level of the surroundings.

```
1. PropertyList props = new PropertyList();
2. props.add("owner","Fahim");
3. VirtualArtefact sensor = new
4. VirtualArtefact("Environment-Light-
                        Level","Lambdax",prop);
5. VirtualArtefact lamp = new
       VirtualArtefact("Light","Lambdax",prop);
6.
7. PreferneManage pm = new PreferenceManager();
8. if(sensor.status){
9.     sensor.enablePreference(true);
10.    artefactID=pm.addArtefact(sensor.getName());
11.    pm.addPositivePhrase(artefactID,"I want to
                            use overhead light");
12.    pm.addNegativePhrase(artefactID,"I do not
                        want to use overhead light");
13.    sensor.subscribe(this,sensorListener);
14. }
15. if(lamp.status){
16.    lamp.enablePreference(true);
17.    actionID=pm.addAction("switching");
18.    pm.addPositivePhrase(actionID,"I like
```

```
                              automatic lighting");
19.   pm.addNegativePhrase(actionID,"I hate
                              automatic lighting");
20. }
21. pm.subscribe(this,preferenceListener);
22. pm.startRecognizer(); // or pm.stratGUI();
23.
24. // virtual artefact call back
25. public void sensorListener(Context data){
26. String context = data.getContextData();
27. //do something
28. }
29. // preference callback
30. public void preferenceListener(Preference
                                        data){
31. String type=data.getType();
32. String stmt=data.getPhrase();
33. if(type.equals("action")){
34.    if(stmt.equalsIgnoreCase("I like automatic
                                      lighting")){
35. //do something
36. }
37. }
38. }
```

In first 6 lines we have created two artefact instances for this application using Prottoy's virtual artefact component. Please see reference for detail of these APIs [7]. In line 7 we have created a preference manager instance. Then from line 9 to line 20 we have added the light artefact to the preference manager, enable the preference for this artefact and added one positive phrase and one negative phrase. Similarly for the switching action we have added the action, positive phrase and negative phrase to the preference manager. Then in line 21 and 26 we have subscribed to sensor data using virtual artefact API and subscribed to preference manager for preference and control data. Then we started the recognizer. Finally we have shown in line 30-38, how the preference callback can be used to extract preference information that can be utilized by the application developer according to application specific way.

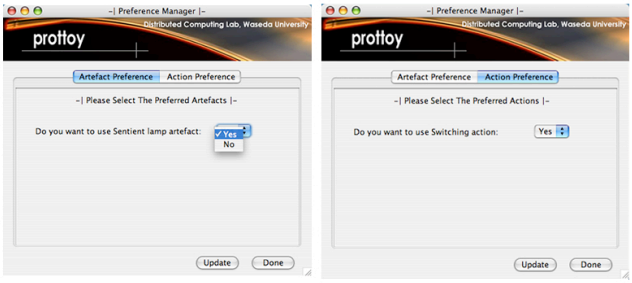

Figure 4: Automatically Generated GUI

In this application we have used only two types of preference attributes: artefact preference and action preference. However the other three types of attributes can be used in the similar way. After application deployment preference manager capture end user preference either by speech recognition or by GUI events, and modifies the preference file accordingly and notifies the application using the callback. For example, in figure 4, we have illustrated a snapshot of this component's GUI for a simple application presented in the code listing. The GUI is automatically deployed without the application developers' effort. Since in this application context preference, generic preference and control preference are not used, the generated GUI does not include them. However, if the input mechanism is speech then a speech recognizer will run in the background. When the preference manager receives a phrase from recognizer that matches the predefined phrases provided by the developers it is sent to the application using the callback.

One interesting issue is that, if we add several commands as control preference we can actually get the command in the callback and can change the applications' control flow. Thus the preference manager also provides the controllability of the system besides preference reflection.

4.3 Sample Application

We have developed several context aware applications on top of Prottoy for evaluating the performances of its various components. In this section we will introduce one application "RoonRoon" and how preference and controllability are managed in the applications using the preference manager component of Prottoy. As mentioned before, preference manager has two types of users:

1. **Application Developers:** They provide the application specific preference options and the phrases that users need to use for providing their preference information.
2. **Application Users:** They are users of the application. They have to use the developer defined phrases to interact with the application. The interaction includes personalization and control facility.

4.3.1 RoonRoon

RoonRoon is a small wearable pet, (a physical embodiment artefact, that can monitor users movement). It is very active pet. It can talk to you, can listen to you, can monitor your activity and do some gestures to notify you about its state of mind. Basically RoonRoon has three state of mind: Happy, Normal and Sad. RoonRoon becomes happy when you do healthy physical movement, it is sad when you don't do any physical activities at all and RoonRoon is normal when your movement is in between. To notify its sate of mind to user it can vibrate, it can talk and can play a music clip. RoonRoon is built with small sensor node that contains a 2D accelerometer and a vibrator. The sensor node communicates with the host using Bluetooth. The sensor node monitors the physical movement and notifies the RoonRoon's host system. A Bluetooth headset with microphone is used to communicate with RoonRoon. Figure 5 shows a user wearing RoonRoon. For personalizing and controlling RoonRoon activity several facilities are provided as shown in table 6.

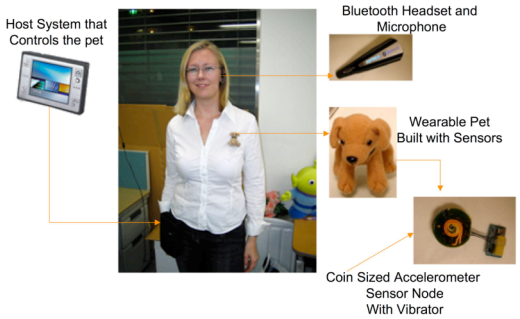

Figure 5: User Wearing RoonRoon and Application Component

User can personalize RoonRoon's functionality completely using voice or GUI (in the host machine). Also user can control its behavior by saying explicit commands or can change the preference at runtime. All these facilities stem from the preference manager component.

Table 6: Preference Option in RoonRoon

Preference Types	Options
Action Preference	Setting action: What action to actuate when it is happy, sad and normal. Setting action for later actuation: User can instruct RoonRoon to activate later.
Context preference	Whether to consider user movement state during actuation: If user is on move what RoonRoon should do when it is active.
Control Preference	How to activate RoonRoon (Manually | Automatic)
Generic Preference	Timing

4.3.2 Observation

From developers point of view we have found preference manager to be very useful, since we only need to provide the preference options/phrases and implement the callback. In RoonRoon for preference facilities we need to write 97 lines of code respectively. We do not need to consider about the speech recognition and GUI generation. Furthermore, the structured representation of the preference information made the implementation of the callback very simple.

From end user point of view, if they use voice based interaction they need to know the appropriate phrases for using the system and using those phrases they can personalize and control the application. From the evaluation, we have found that when the users use the GUI, the system works nice. But sometimes the speech recognizer could not identify the phrases correctly; as a result the some preference options changes automatically while using speech recognizer. This makes the system unstable. However, users have found the facilities useful, particularly they were convinced with the way preference manager changes their preference at runtime.

5. DISCUSSION

It has been mentioned in several works [3,4,14,16,20] that the most important criteria for the success of a context aware system depends how well it copes with humans' implicit understanding of the systems. Unfortunately available context-aware middlewares in the literature do not consider this as a concern for middleware although they emphasize the perspective of personalization. Prottoy has taken a unique approach in this perspective. The preference manager component of Prottoy provides end users with the facility to control the system from their preference point of view. We have shown how the speech and/or GUI facilities are generated independently from application development codes in Prottoy for providing the preference support. We believe this approach attempts to provide the flexibility to support the mental model of end users to some extent. Of course it is impossible to exactly identify users understanding but Prottoy's options can be seen as a tool dedicated for reducing the gap of humans implicit understanding of system and context aware behavior of the system. We believe the major contributions of this work are:

- A guideline and taxonomy for structured representation of preference information.
- A middleware component, that takes care of the acquisition and representation mechanism of preference information with a loosely coupled architecture.

5.1 Focus on Specific Issues

There are few issues that should be explained in detail. In the following subsections these issues are considered.

5.1.1 No Generalization for Application Logic

Our approach does not attempt to handle the application logic. The preference manager receives the information from the environment and presents it to the application in a structured way using the preference attributes. It is the responsibility of the developer to utilize this information in application specific way. For example: In the RoonRoon application, if the application receives that user does not want to receive any event when he/she is walking, we have implemented the logic in the application that handles the behavior of RoonRoon when it is active and the user is walking.

5.1.2 Controllability

As we have mentioned in section 4 that the "Control Preference" attribute contributes to support controllability aspect. Basically our approach collects the control command from the user and presents it to the application so that developers can handle it in application specific way. Since the GUI is automatically generated, an indirect contribution of our approach is automatic user interface generation from structured control preference attribute.

5.1.3 Preference Classification

Based on the user study, we have grouped the preference attributes. But we do not claim that this grouping is enough and can handle all sorts of applications. However we believe our taxonomy can be considered as a guideline for further classification. In the user study we have mentioned some users want the system to identify their preference automatically from their previous activities. In such case history information is very important. In fact in Prottoy we have Artefact Proxy component that provides storage information to the applications. This information can be utilized to configure the system analyzing users interaction pattern with the system. We are focusing on this issue with great interest and hope to come with some interesting results soon.

5.1.4 Interaction Paradigm

Our GUI and voice provision for collecting users input may not be always applicable to all systems. However if we look at the architecture of the preference components, it is clearly visible that any new interaction mechanism can easily be accommodated because of the loose coupling among the components. So if some applications need a gesture-based interaction, one gesture recognizer can replicate the operations of the speech recognizer. Similarly a RFID tag-based interaction can easily be implemented.

5.2 Performance

From the performance point of view, we have found in the sample applications that the Preference Manager works pretty well. It takes care of extracting and representing preference information independently. The APIs we have shown in the implementation section demonstrates the simplicity of the approach. The automatic GUI generation and voice recognition minimizes the developers' task considerably. The structured representation of the preference/control information gives the developer more control to handle the flow of the proactive application in users' preferred way. Also the number of lines that developers need to write for personalization facilities is significantly small. Considering the functionalities of the application we argue such reduction of codes minimizes developers' task considerably.

However, sometimes the speech recognition is not correct and the system preference changes that does not conform to users preferences. We do not consider it as a major drawback since a better recognizer may handle this issue in a better way. Also since we do not identify/authenticate the voice of the user, anyone can personalize and control a system. So in terms of ownership issue it creates a complex problem. Currently we are focusing on these two issues to come up with a better resolution.

6. RELATED WORK

Although personalization in proactive context awareness is a very important issue, there has been very little work in this area. Most of the works basically presents their research results based on some case study that has been conducted on target users. Barkhuus and Dey presented an interesting case study on some hypothetical mobile phone services and have shown that users prefer proactive services to personalized ones [13]. However their focus domain was only mobile phone services and the implication cannot be applicable to the context aware system that involves many physical artefacts. Some researches that precede Barkhuus's work also argued whether information should be pushed towards the user or should be pulled by the user for customization of the context aware systems [11]. Brown and Jones have also defined the interactive and proactive systems where, where personalization activities fall into the interactive systems [15]. In all three works, they have tried to drawn some level of interactivity. However in all cases, we argue that a clear distinction between personalized and proactive system is not perfectly applicable, because all proactive systems needs to be personalized before hand or at runtime so that it matches users mental model. Our target user study also confirms this argument. The video clip shown to them contains several proactive behavior of the system, but we have found each user wants to personalize this behavior. Each user has a different understanding and choice; a same proactive behavior cannot be applicable to all users because the proactive ness itself needs to personalize by the user using the system. We have mentioned and shown in this paper that personalization is an inherent part of proactive systems, which conflicts with some previous research proposition. We strongly argue that there cannot be distinct borderline between personalization and proactive ness rather they are complement of each other. Our implication is that personalization is an important conjugal part for successful deployment of proactive systems.

In traditional desktop computing for providing personalization, usually graphical user interface is provided using which user can personalize a system/application. This aspect has been well investigated in desktop computing paradigm [9,10,17,19]. Considering proactive applications in pervasive environment, there are not many woks in the literature that focuses on this aspect. Prottoy is a unique middleware because it handles personalization and controllability aspects besides discovery, extraction, distribution, management and representation of context information. Sakai et. al proposes a framework that focus on explicitly on end user preferences on the mobile phone applications [18]. But their approach cannot be applicable in boarder context aware aspects. Also they do not provide any voice based interaction technique to control and personalize the system. Furthermore, the framework is tightly coupled with the application considering their rigid focus on mobile phone domain, thus making custom application development fairly complex. In [12] a rule based approach has been proposed to control and configure information appliances. However, their approach does not cover how to personalize the system using such rules. Also we believe uttering specific phrases as in our approach for controlling appliances are easier for the end users than generating rules for context aware behavior. Voice and GUI based interaction for controlling smart spaces has been investigated in various projects like Odisea [8], EasyLiving [2] etc. However since in our approach speech or GUI has been used just as preferred input mechanism based on user study, we believe any standard mechanism that has been used in these projects for supporting voice or GUI based interaction is applicable in Prottoy.

7. CONCLUSION

In this paper we have focused on personalization aspect for context aware computing. Initially we have introduced a user study and then we have implemented the findings in a component of a middleware called Prottoy. From the context aware computing point of view we believe this work has a major contribution to the context aware community, since Prottoy approaches the personalization aspect in a unique way while reducing application developers burden considerably.

8. REFERENCES

[1] B. L. Brumittet and J. Cadiz. "Let there be light: Examining interfaces for homes of the future"; In Interact 2001.

[2] B. L. Brumittet, B. Meyers, J. Krumm, A. Kern, and S. Shafer. "Easy living: technologies for intelligent environments."; In 2nd International Symposium on Handheld and Ubiquitous Computing, 2000.

[3] D. A. Norman. The Design of Everyday Things. NY: Basic Books, 2002.

[4] D. A. Norman. Emotional Design. NY: Basic Books, 2004.

[5] F. Kawsar, K. Fujinami, and T. Nakajima. Augmenting Everyday Life with Augmented Artefacts. In Smart Object and Ambient Intelligence Conference, 2005.

[6] F. Kawsar, K. Fujinami, and T. Nakajima. Experiences with Developing Context-Aware Applications with Augmented Artefacts. In 1st International Workshop on Personalized Context Modeling and Management for UbiComp Applications, A Workshop in conjunction with UbiComp2005, the 7th International Conference on Ubiquitous Computing, 2005.

[7] F. Kawsar, K. Fujinami, and T. Nakajima. Prottoy: A Middleware for Sentient Environment. In The 2005 IFIP

International Conference on Embedded And Ubiquitous Computing, 2005.

[8] G. Montoro, X. Alamn, and P. A.Haya. Spoken interaction in intelligent environments: a working system. advances in pervasive computing. Austrian Computer Society, 2004.

[9] G. Rossi, D. Schwabe, and R. Guimares. Designing personalized web applications. In The tenth international conference on World Wide Web, 2001.

[10] H. K. O. Stiermerling and V. Wulf. How to make software softer - designing tailorable applications. In Symposium on Designing Interactive Systems, 1997.

[11] K. Cheverst, K. Mitchell, and N. Davies. "Investigating context-aware information push vs. information pull to tourists."; In Mobile HCI 2001.

[12] K. Nishigaki, K. Yasumoto, N. Shibata, M. Ito, and T. Higashino. Framework and Rule-Based Language for Facilitating Context-Aware Computing Using Information Appliances. In First International Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet, 2005

[13] L. Barkhuus and A. Dey. "Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined. "; In The 5th International Conference on Ubiquitous Computing, 2003.

[14] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. IEEE Personal Communications, 2001.

[15] P. J. Brown and G. J. F. Jones. "Context-aware retrieval: Exploring a new environment for information retrieval and information Itering."; Personal and Ubiquitous Computing, 5(4), 1997.

[16] R. H. Harper. Why People Do and Don't Wear Active Badges: A Case Study. In Computer Supported Cooperative Work, 1996.

[17] S. Farrell, V. Buchmann, C. S. Campbell, and P. P. Maglio. "Information programming for personal user interfaces."; In Intelligent User Interfaces, 2002.

[18] S. H., Y. Murakami, and T. Nakatsuru. Personalized Smart Suggestions for Context-aware Human activity Support by Ubiquitous Computing Networks. NTT Technical Report, 2-2, 2004.

[19] T. Rist and P. Brandmeier. Customizing graphics for tiny displays of mobile devices. Personal and Ubiquitous Computing, 6(4), 2002.

[20] V. Bellotti and K. Edwards. "Intelligibility and Accountability: Human Considerations in Context-Aware Systems.", Human-Computer Interaction, 16,2-4, 2001.

[21] Website of Sphinx4 Speech Recognizer. http://cmusphinx.sourceforge.net/sphinx4.