Persona: A Portable Tool for Augmenting Proactive Applications with Multimodal Personalization Support

Fahim Kawsar
Department of Computer Science
Waseda University
Tokyo, Japan
fahim@dcl.info.waseda.ac.jp

Tatsuo Nakajima
Department of Computer Science
Waseda University
Tokyo, Japan
tatsuo@dcl.info.waseda.ac.jp

ABSTRACT

User centric personalization plays an important role for the adoption of proactive applications. However, stipulating system support to facilitate personalization features in proactive applications generically is still an open issue. In this paper we have addressed this particular issue and presented Persona, a tool that enables adding personalization features in proactive applications in a generic manner. A key feature of Persona is portability that allows it to be injected in various pervasive middlewares as a plug-in. Consequently, existing proactive applications can easily be extended with Persona for personalization support. We have discussed the design and implementation rationale behind Persona and shown it's direct implications with two different middlewares and several proactive applications.

Keywords

Personalization, Proactive Application, Pervasive Environment, Toolkit.

1. INTRODUCTION

When we talk about proactive systems, often the term personalization is misinterpreted. This is because of the presumption of the context aware characteristics of proactive applications. However, if the context aware characteristics conflict with users' preference, the applications' success ratios drop radically. Every user has own understanding and perspective towards an application and wants to personalize it regardless of its proactive behavior [1,2,15,17,22]. For the success of the application, it is essential to allow end users to personalize proactive applications. Here by personalization, we mean the active involvement of the end users to customize the adaptive behavior of the system. Current approaches implicitly associate personalization options with user contexts for proactive behavior. As a result, end users have minimal or no control over the pro-activity of the applications. In this paper, we have addressed this particular issue and presented *Persona,* a system tool that allows application developers to extend pervasive applications so that end users can personalize the pro-activity of the applications.

MUM'07, December 12-14, 2007, Oulu, Finland.

Considering the disappearing nature of pervasive environments, general approaches used in desktop computing [5,20] for preference management are not suitable for pervasive applications. Another impediment is filtering users' interactions that are meant for personalization. This is due to the inherent context modeling used in proactive applications. Persona has taken a unique approach to address these two impediments. It provides semantically rich data structure using which personalization options can easily be accommodated in proactive applications. Also, it supports several multi-modal interactions and has built-in support for two commonly used interaction paradigm in pervasive domains (Voice and Graphical User Interface). Users interactions are filtered out by Persona that are related to personalization utilizing the semantic data structure enabling an application to adapt users preferences. Furthermore, Persona can be used with various pervasive middlewares as a plug-in. So existing proactive applications atop different middlewares can be extended for incorporating user centric personalization features.

The rest of the paper is organized as follows: we place Persona against related works in section 2. The design principles of Persona are presented in section 3 followed by its technical detail in section 4. Then we proceed to the evaluation of Persona in section 5. We discuss some generic issues on section 6. Finally section 7 concludes the paper.

2. RELATED WORK

Most researches on personalization in proactive environment have taken a social perspective. 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 [10]. However their focus domain was only mobile phone services and the implication cannot be applicable to the proactive system, which 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 [8]. Brown and Jones have also defined the interactive and proactive systems where personalization activities fall into interactive systems [16]. In all three works, they have tried to furnish some levels of interactivity. However, we argue that a clear distinction between personalized and proactive systems is not appropriate, because all proactive systems needs to be personalized before hand or at runtime so that they match users mental model. It is not obvious that end users will welcome all

proactive behaviors. Each user has a different understanding and choice; the same proactive behavior cannot be applicable to all users because the pro-activity itself needs to be personalized by the user first. Our hypothesis in this paper is that personalization is an inherent part of proactive systems, which conflicts with some previous research proposition. We strongly argue that there cannot be a distinct borderline between personalization and pro-activity; on the contrary they are complementary to each other.

In traditional desktop computing usually graphical user interface is provided to personalize an application. This aspect has been well investigated in [5,20] and their implications are obviously not appropriate for the characteristics of pervasive applications. Dourish looked at the personalization aspect from system design point of view exploring a collaborative document management system. He used the term *appropriation to* denote a process by which people adopt and adapt technology [15]. Although, his work is very similar and influenced us significantly, he focused on pre-design consideration of group-ware for appropriation features rather than providing any system tool like ours to automate the process.

Sakai proposes a framework that focuses on the end user preferences of mobile phone applications [21]. But his approach cannot be applicable in generic context aware aspects. Furthermore, the framework is tightly coupled with the application considering their rigid focus on the mobile phone domain, thus making custom application development fairly complex. In [9] 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 these rules. In the user modeling and usability domain, a variety of studies have been conducted to provide toolkits or modeling language for assisting developers in designing effective interactive systems and modeling user activities through rigorous sensor data analysis [11,12]. However, none of them addresses the issue of a unified system support as we explore in this paper.

3. DESIGN PRINCIPLES

Persona is designed considering the following principles:

1) Syntactic Data Structure: Persona exploits a semantically rich data structure to represent the overall preference options of an application in a generic manner. Developers can use this data structure to construct application specific preference options. Persona internally uses this data structure to filter the real word events and provide the applications with specific events and data meant for personalizing an application.

2) Independent of Interaction Modality: Personalization is the direct impact of a subset of user's interaction with the application. Considering there are various interaction modalities (speech, gesture, digital ink, implicit controller, augmented reality, GUI etc.) and the best candidate depends on the application itself, Persona is built following a loosely structured design. Persona's core functionalities are independent of the underlying interaction modality. Any suitable interaction engine can be plugged into Persona without altering its primary functions or the applications running on top of it.

. ***3) Application Specific Conflict Resolution:*** Pervasive applications are usually deployed in multiple user environments where different users might have different personalization perspective towards the same system. Persona approaches this issue by separating the conflict resolution scheme from its core functionalities. Application specific policies can be fed into Persona for runtime conflict resolution.

4) Portability: Persona is sandwiched between pervasive middlewares and applications. It is independent of any external middleware support thus can easily be used with various middlewares without altering their functional features. Existing applications developed atop various middlewares can easily be extended for personalization features using Persona.

In the next section, the data structure with architectural building blocks of Persona and the programming model are explained.

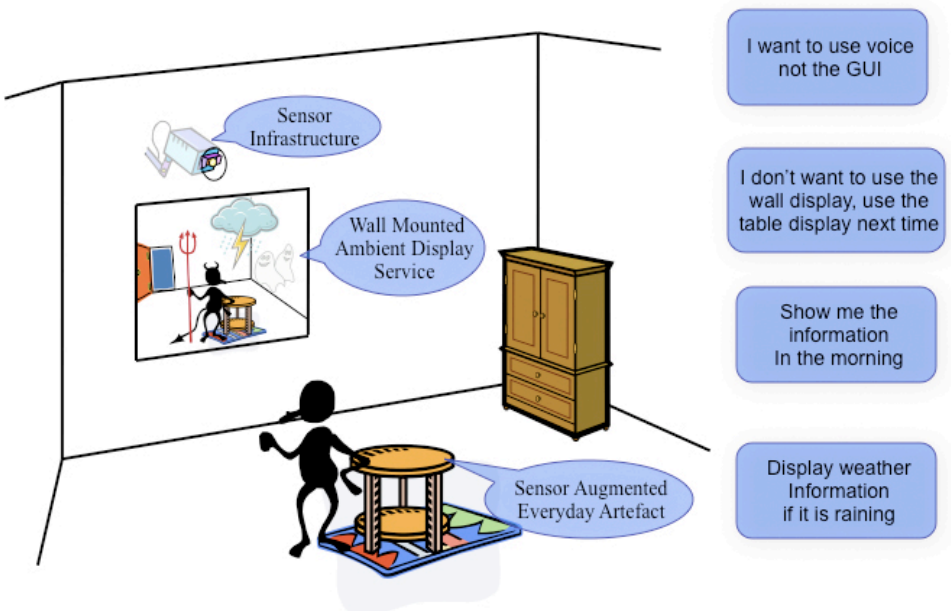

Figure 1. A Hypothetical Proactive Application

4. SYSTEM DESCRIPTION

Following the design guidelines presented in the previous section, Persona is built in a loosely structured manner where one core component plays the primary actor's role and provides interfaces for plugging other components. However, before discussing the technical details, we present the rationale that set the basis of Persona's underlying data structure and henceforth the architecture.

4.1 Data Structure

This unified data structure mentioned as one of the design goals, is just a logical carrier of the application specific preferences. Developers are free to build applications' preference options and grouping those options into some categories. Persona can then encapsulate these categories into a generic data structure. To further illustrate this aspect, lets consider Figure 1 that depicts a typical application scenario in pervasive environment. For this kind of applications, we may consider the following categorization of personalization options.

a) Artefact Preference: This category of personalization options is for enabling a user to select the participation of any artefact in the cooperative smart environment. For example, a user may want to use a wall-mounted display instead of a display-augmented table for ambient traffic information.

b) Action Preference: This category enables a user to set preferred actions. Usually a system consists of several actions that it actuates based on some conditions. Users can enable or disable actions using this class of preference information. For example, users can enable/disable the automatic/manual weather information display action on a hallway mirror.

c) Interaction Modality Preference: This class of options is to provide users with the flexibility to select their preferred interaction mechanism. For example, context-aware shopping assistant may have multiple user interfaces (like handwriting or voice for input and display or sound for output); users can select his/her preferred interaction modalities.

d) Timing Preference: This category enables users to associate an action of the proactive application with some contextual events like location, time, external presence etc. For example: a user may want the cell phone to automatically switch to silent mode when she is in the meeting room.

Albeit this classification is for an example purpose, it covers most of the pervasive applications. Once developers logically derive the categories, personalization options for each category can be added to Persona using some abstract APIs. These APIs logically convert the personalization options associated with categories into an internal data format that Persona uses for extraction, representation and processing of personalization features. To manipulate this data three operators are used:

a) Positive (P): This operator represents user's willingness to use the feature in context.

b) Negative (N): This operator represents user's unwillingness to use the feature in context.

c) Calculated (C): This operator is used by Persona internally to resolve conflict or to provide appropriate calculated decisions regarding preference feature based on a finite state engine.

4.2 Architecture

Persona is basically sandwiched between the proactive applications and the middleware used for that application. Figure 2 illustrates the basic building blocks of Persona that are described below.

1) Persona Core: This is the central player of Persona. It provides interfaces using which other components of Persona, i.e. *Application Interface, Input Interface, Preference Knowledge Base* and *Finite State Engine* are glued together. Basically upon receiving interaction events from the real world though *Input Interface* Persona Core filters out the preference option utilizing the *Preference Knowledge Base* and notifies the application using *Application Interface*. Also, when conflict arises among application actions, it consults the *Finite State Engine* for the appropriate resolutions.

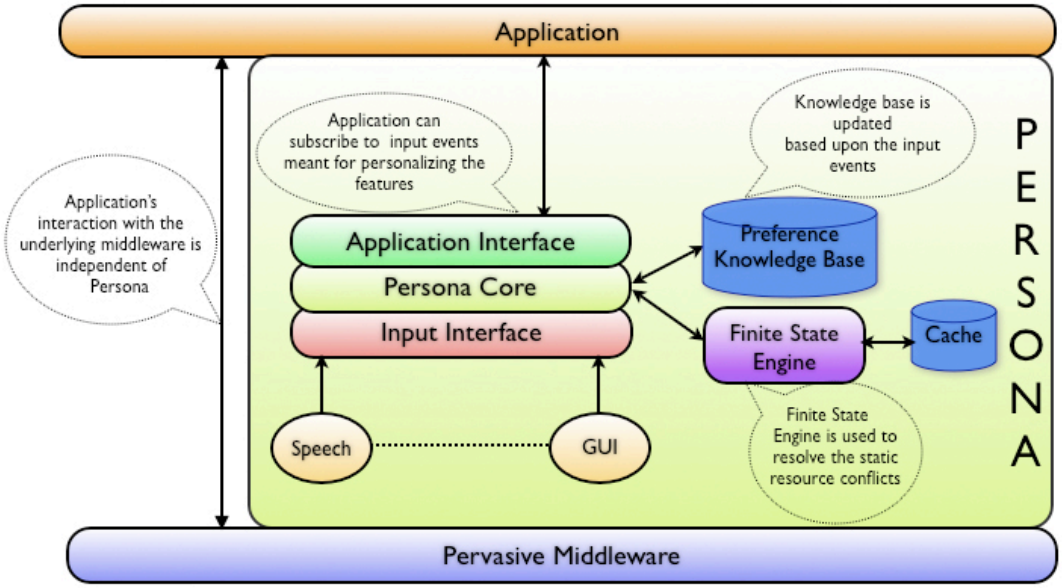

Figure 2. Architecture of Persona

2) Application Interface: This component is the access point for the application developers to use Persona. It provides an array of APIs that developers can use to define the personalization options of the application. Developers can create custom categorization of preference or can use the built-in ones presented in section 4.1. Furthermore, applications can subscribe to Persona to receive interaction events that are related to personalization or can poll periodically.

3) Input Interface: Persona is designed to host a diverse range of interaction techniques. This component offers a unified interface using which different interaction modality can be integrated into Persona. The rationale behind this unification is a set of generic interaction constructs that developers need to develop following Persona specific semantics. However in current prototype, we have deployed Persona with two built-in interaction modalities: speech and GUI.

a. Speech Engine: End users usually provide their preference in simple English language, like "*Do not turn off the light automatically*", *"Notify me every morning"*, etc. The speech engine in Persona is designed to handle such free from interactions. Developers provide a list of phrases and sentences that can be used for personalizing target application through APIs. Persona generates the corpus and the grammar file automatically which is later used by the recognizer [23]. This recognizer runs in the background when application starts. Upon recognizing a phrase it notifies the persona core. To enable Persona's speech engine the target application environment has to be equipped with on or multiple microphones.

b. Graphical User Interface (GUI) Engine: End users can provide their preference by manipulating GUI. Developers provide a list of options that can be used for personalizing target application through APIs. Persona automatically generates this GUI analyzing the options provided by the developers. Figure 3 shows a sample GUI that is automatically generated by Persona for the application presented in section 4.3 and coded in Figure 5

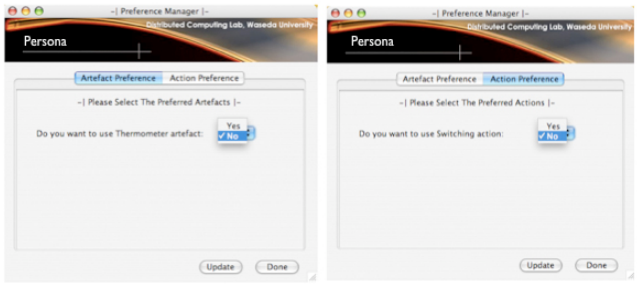

Figure 3. Automatically Generated GUI by Persona for a Sample Application

Because of the fair loose coupling, these engines can be replaced by other alternatives like gesture, handwriting etc. seamlessly. One interesting aspect is that, preference information is just another type of input information, thus these interaction engines also collect direct interaction data meant to control the application. It is Persona Core that internally filters the preference data from the control data and represents it to the application in a unified way using appropriate category.

4) Preference Knowledge Base: This is an XML file generated dynamically during the deployment time of the applications and contains application specific personalization options provided by the application developer using Persona's APIs. It also contains end users' preferences for each option and is updated at real time to reflect users' preference. When Persona receives external real world events, this knowledge base is consulted for filtering preference data and for decision-making.

5) Finite State Engine: In Persona conflict resolution is application specific and assumed to be provided by the application developers. The finite state engine is the interface that the developers can modify to provide their application specific policies. It internally maintains a small cache of past preference change and usage events, which can be exploited while defining resolution policy and to deduce calculated preference. This engine can also be used to recover from invalid conditions to maintain application flow. For illustrating the internals of this engine lets consider a simple application scenario: a smart mirror installed in the washroom that shows some ambient information related to a user when he/she brushes teeth in front of the mirror. The toothbrush is augmented to identify the user. The mirror can show information automatically whenever a user brushes his teeth in front of the mirror or the user can manually start the mirror stating his identity. Now if the toothbrush's preference is negative while automatic start up is preferred, then the application moves to an invalid state (because the system can not identify the user thus can not retrieve the information related to that user, like schedule etc.). When this conflict is identified by the application, it can use the finite state engine to receive calculated preference to maintain its workflow. In this scenario, depicted in Figure 4 the state engine can either alter the start up preference to manual, so that the user can manually state his/her identity. Alternatively if the toothbrush is being used, it can alter the toothbrush preference to positive.

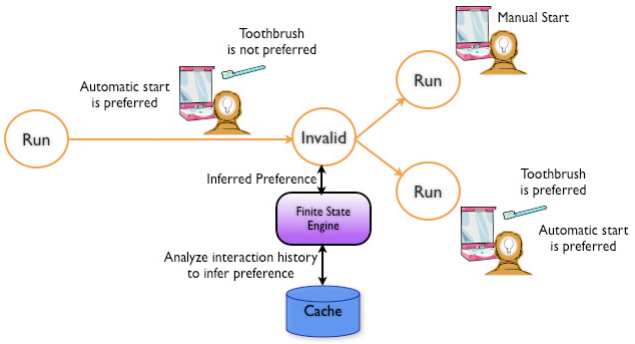

Figure 4. Example Operations of Finite State Engine

4.3 Programming Model: Integration of Persona into Applications

Essentially there are two types of users in Persona: the developers of the application and the end users of the application. Developers can use Persona for adding personalization support in application whereas end users can interact with the applications. Persona filters end users' interaction meant for personalization and accordingly modifies application's behavior. From developers' perspective, integrating Persona into applications includes following steps:

1. *Listing all the preference options of the applications.*

2. *Categorizing these preference options using taxonomy similar to the one presented in Section 4.1.*
3. *Generating the Positive and Negative statements for each of the preference options.*
4. *Listing these statements into Persona using APIs. A stand-alone library (Application Interface) is provided for the application developer. This list is used to generate the Preference Knowledge Base and corpus of the recognition engine.*
5. *Subscribing to Persona for personalization events.*
6. *Invoking the suitable interaction engine. In the current version GUI and Speech Recognizer are provided.*

Step 1-3 are design phase tasks where step 4-6 are development phase tasks. The code snippets in Figure 5 demonstrates the latter steps (4-6) utilizing the APIs presented in Table 1 for a very simple application composed of a thermometer and a cooler

Table 1. A Subset of Persona's Application Programming Interfaces (API)

API	Functionality
`public string addPreference(String name, String prefType)`	For adding a preference related to artefact, action, interaction modality and timing for which preference is necessary.
`public void addPositiveStmt(String id, String stmt)`	For adding a positive statement for the preference of an artefact, action, interaction modality and timing.
`public void addNegativeStmt(String id, String stmt)`	For adding a negative statement for the preference of an artefact, action, interaction modality and timing.
`public void subscribe(Object source, string callback)`	For subscribing to the preference manager for receiving preference data captured from real world interaction.
`public static float getPreference( String id, String type)`	For extracting preference from Preference Knowledge Base, return values include positive (1), negative (0) and calculated (0.1 ~ 0.99). The 2nd parameter specifies whether regular or calculated is required.

Application Code

```
1. Persona pm = new Persona("Speech Recognizer");
2. artefactID=pm.addPreference
                    ("Thermometer","Artefact");
3. pm.addPositiveStmt(artefactID,
              "I like to use the thermometer.");
4. pm.addNegativeStmt(artefactID,
         "Do not use the thermometer anymore.");
5. actionID=pm.addPreference("Switching","Action");
6. pm.addPositiveStmt(actionID,
          "Turn on the cooler automatically.");
7. pm.addNegativeStmt(actionID,
      "Never turn on the cooler automatically");
8. pm.subscribe(this,preferenceListener);
9. pm.start();
10. public void preferenceListener(Preference data){
11. String type=data.getType();
12. String stmt=data.getPhrase();
13. String id=data.getID();
14. float value=data.getPreference();
15. if(type.equals("action")){
16.    if(stmt.equalsIgnoreCase(
17.     "Never turn on the cooler automatically")){
17.      //do something in application specific way
18.    } } }
```

Preference Knowledge Base

```
19. <?xml version="1.0" encoding="UTF-8"?>
20. <application-preference>
21.  <artefact-preference>
22.  <artefact>
23.   <id>artefact-1</id>
24.   <name>Thermometer</name>
25.   <preference>preferred</preference>
26.   <positive-phrase>
27.    <phrase>I like to use the thermometer</phrase>
28.   </positive-phrase>
29.   <negative-phrase>
30.    <phrase>Do not use the thermometer anymore</phrase>
31.   </negative-phrase>
32.  </artefact>
33.  </artefact-preference>
34. <action-preference>
35.   <action>
36.    <id>action-1</id>
37.    <name>Switching</name>
38.    <preference>preferred</preference>
39.    <positive-phrase>
40.     <phrase>Turn on the cooler automatically</phrase>
41.    </positive-phrase>
42.    <negative-phrase>
43.     <phrase>Never turn on the cooler automatically</phrase>
44.    </negative-phrase>
45.   </action>
46.   </action-preference>
47. </application-preference>
```

Figure 5. Code Snippets and Preference Knowledge Base demonstrating Persona's usage in Applications

followed by the Preference Knowledge Base used for this application. The cooler is automatically turned on/off based on the sensed air temperature. Speech Engine is used as the interaction modality in this code example. The Preference Knowledge Base (line 19-47) is automatically created during application deployment time. In line 1 we have created a persona instance with speech engine. Then from line 2 to line 4 we have added the thermometer to persona, and added positive and negative statements based on the speech interaction engine constructs (Provided by the developers). These statements are used to generate the Preference Knowledge Base. For line 2, 3 and 4 in the application code, we have entries (line 22-32) in the Preference Knowledge Base. Similarly, for the switching action of the cooler we have added the action, positive and negative statements to persona, which cause the entries (line 35-45) in the Preference Knowledge Base for this action. In line 9 we start the persona core to capture real world events (speech). In this application we have used only two types of preference category: artefact preference and action preference. As depicted, due to the flexible design of this API we can accommodate other categories in the same manner. The speech recognizer engine runs in the background after deployment. Whenever the recognizer identifies a phrase, the persona core is notified. If the persona core finds a match for this phrase in one of the entries in the Preference Knowledge Base, it extracts the information for that phrase from the Preference Knowledge Base and sends it to the application. It also updates the Preference Knowledge Base, e.g. <preference> attribute to *Preferred*, *Not Preferred* or *Calculated* based on captured event. Similarly, for GUI engine when the GUI event is captured, it is sent back to application and the preference manager updates Preference Knowledge Base. As shown in line 10-38, the application uses preference callback to receive this information and it can utilize it in an application specific way. Application can also call explicitly `getPreference(id,type)` to get the regular or calculated preference from the Preference Knowledge Base, in case of calculated one, persona core uses the finite state engine to generate the preference. The developer can use this information in application specific way to reflect users preference in application's proactive behavior.

5. EVALUATION

In this section we will provide the evaluation of Persona. We have adopted a scenario based evaluation method introduced in [18] considering Persona closeness to end users.

5.1 Scenario Based Evaluation

Lest consider the following Scenario:

"*Joanna is a broker at the New York Stock Exchange. During her daily morning routine in the bathroom, while she is brushing her teeth and putting on her make-up, her mirror provides information she needs to start her day. During these activities she can watch her daily schedule and what the weather will be like, so she can dress fittingly. Furthermore she finds out if the subway is running properly. After arriving at the office she works non-stop for several hours contacting her clients, buying and selling on their accounts until her agent reminds her to take a coffee break and tells her not to forget her lunch appointment at 13:00 with one of her biggest clients. Later that afternoon she goes to the restaurant to meet her client. While she is waiting for her client, the table she is sitting at shows that tonight there are still tickets left for musical "Les Misérables" and that perfumes are on sale at Saks Fifth Avenue. After lunch she returns to the office, the computer on her desk informs her about some important memos she received during her absence. While she continues working, her desk lamp turns on automatically and the track "For Elise" from Best of Beethoven is being played as she starts responding a client's email."*

This scenario is implemented using three different proactive applications as shown in Figure. 6. All three applications are developed atop a middleware called Prottoy [3] and augmented with personalization support using Persona as mentioned in Table 2. All three applications were previously developed with personalization support and were reported in [4].

5.1.1 Observations

We are interested in several things from Persona's evaluation point of view:

Development Task: We have found that adding personalization options in applications was quite simple primarily because of the abstract APIs of Persona. For extending these applications developers need to analyze the applications and list the user centric personalization options into some categories. Since in all three applications voice based interaction is used, developers need to generate the statements that represent users preference towards specific options.

Code Complexity: The second important thing we have observed is that injecting Persona in these existing applications is pretty straightforward. Since, it is independent of the middleware and only application code need to be modified we could do that in a very short span of time and with the inclusion of about 270 lines of code for all three applications. Please note that, we have used a built-in speech recognizer. So to use other interaction paradigm we need to build custom engines which will increase the development time and cost.

End users' Impression: We had performed informal user trials involving 9 people (6 Male, Age Range: 21~32) to evaluate Persona's user-centric performance. Essentially, how end users feel like personalizing the behavior of proactive applications? We initially introduced them the applications and then asked them to use and personalize them. Each trial took about two hours followed by an interview. We have found, all the participants wants to personalize the application in their own way and interestingly the combination of all the personalization options for all three applications is unique for each participant. They explicitly mentioned, just like traditional desktop applications, they would definitely like to have the personalization options for physical world applications and effectively they would like to control everything. They do not want a smart place to be proactive rather to be reactive to their needs in their preferred ways. However, our current GUI and speech interaction are inadequate. Although GUI seemed acceptable to them in general, speech had received contrasting ratings. 6 of the participants found it to be annoying and not natural to converse with a space. Also, the speech recognizer used in the current prototype misinterpreted voices in some cases that caused frustrations among the participants.

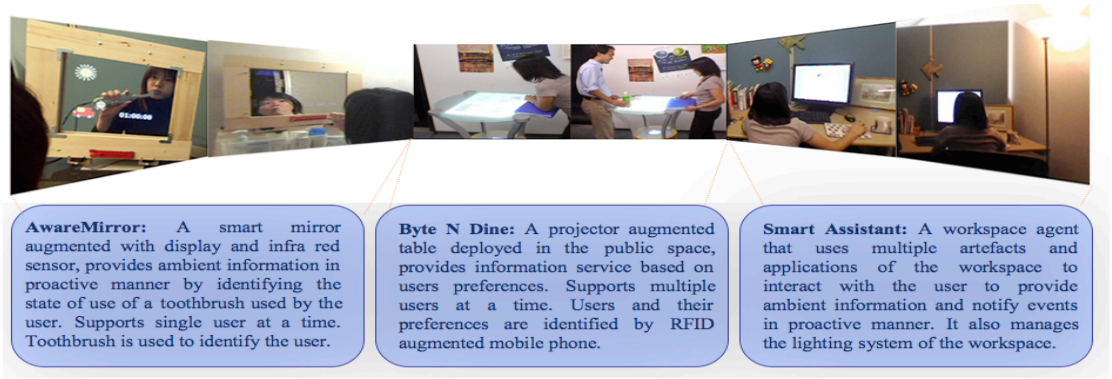

Figure 6: Applications implementing the scenario

Table 2. A Subset of Persona's Application Programming Interfaces (API)

	Functionality	**Augmented Artefacts**	**Preference Options**
Aware Mirror	Display useful information on the mirror	1. Mirror augmented with proximity sensors to detect user's presence 2. Toothbrush as an identifier of the user.	Artefact: *With /without toothbrush*
			Action: *Automatic/manual start/close*
			Interaction: *Tangible Button/Voice/GUI for navigation.*
			Timing: *Morning/Always*
Byte N Dine	Display preferred information on the table display	1. Table augmented with projector and RFID tag reader to identify users presence and preference.	Artefact: *None*
			Action: *Automatic/manual start/close*
			Interaction: *Touch Display/Voice/GUI for navigation*
			Timing: *Morning/Always, Alone/Always*
Smart Assistant	Suggesting user for a refreshment, providing just in time message and controlling lighting.	1. Sensor augmented chair and desk lamp to detect users presence and light sensitivity of the workspace. 2. Media player and email agent to play music and identifying email reception.	Artefact: *Yes/No Use of Lamp, Music Player*
			Action: *Yes/No Suggestion for break, Email notification, music play, automatic light.*
			Interaction: *None (All are proactive)*
			Timing: Always/On specific time

5.2 Portability Test

The next evaluation that we have performed is portability test. There are many pervasive middlewares in the literature. However, for this test we have picked two middlewares Context Toolkit [1] and Prottoy [3] based on the availability. Using these two middlewares, we have built a slightly variant version of the application DUMMBO [6] that captures the collaborative tasks like meeting etc. Instead of using iButton and a digital white board as in original application, we have used RFID tag and a touch panel display augmented table as shown in Figure 7. However, in this case we have facilitate preference options regarding capturing, i.e., only voice, only text on the display, both voice and text, timing duration etc. We have found using Persona with these middlewares was just adding another class file to accommodate the steps 3-6 mentioned in section 4.3 mimicking the code of Figure 5. Essentially persona can be used just as a library in the application space. We have used the speech recognizer engine for this application and a custom policy was implemented in the finite state engine to resolve conflicts.

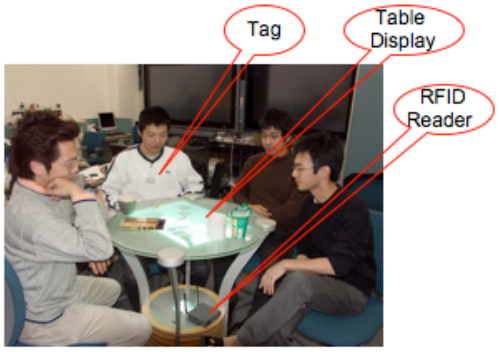

Figure 7: A variant of DUMMBO [6] Application

6. DISCUSSION

In this section, we will discuss some generic issues and put forth the avenue of our future work.

Preference Data Structure: Our major design concern was to provide a structured representation of the preference data that makes management of personalization easier. In section 4.3 we have shown, how to use the APIs to represent the personalization data. Furthermore, we have provided an exemplary classification scheme in section 4.1. But we do not claim that this categorization can handle all sorts of personalization requirements. However, this classification can be considered as a guideline for further derivations. Persona is flexible enough to accommodate further classes. For example, consider the revised lines of the scenario presented in section 5.

"….While she continues working, her desk lamp turns on automatically and, it dims into ***a pink*** *shade and the track …."*

In this case, our exemplary categories cannot handle this option. But Persona APIs allows us to easily accommodate this. For example, to support this option we can use

```
artefactID=pm.addPreference("Lamp","Generic-
                            Color-Preference");
pm.addPositiveStmt(artefactID,"I like to use
                                pink shade.");
```

This line will result following entries in the Preference Knowledge Base:

```
<generic-color-preference>
   <artefact>
     <id>artefact-2</id>
     <name>Lamp</name>
     <preference>preferred</preference>
     <positive-phrase>
       <phrase> I like to use pink shade </phrase>
     </positive-phrase>
   </artefact>
</generic-color-preference>
```

So, Persona will handle this category exactly in the same manner like other category. Because of this unified design, it is very easy to accommodate further categories of preferences.

Another important aspect is the operator used with the data structure. Currently we use two discrete operators (Positive and Negative, Yes/No options) and one continuous operator (Calculated) to represent users preferences. However, none of these operators are capable of handling semantically rich continuous values. For example: If a user wants to set the cooler at a *comfortable* level, current version of Persona cannot handle this action, unless the meaning of comfortable is specified in the application logic. Supporting these kind of semantically rich preference is an interesting topic and we are currently working on this issue.

Interaction Modality: Current speech interaction suffers from poor acceptability as we have found in the end user evaluation. Also, current Finite State Engine support to recover from erroneous states due to misinterpretation of voice is minimal as all dynamic situation is hard to predict during the design phase. We are working on a more loosely coupled speech interaction engine where semantics of the user statement is analyzed rather exact matching. Thus, future version of Persona will be more reliable. GUI and speech for collecting input might not be applicable to all systems. If we look at the loosely coupled design of Persona, it is visible that new interaction engine can be injected easily. So, if an application needs a gesture-based interaction, a gesture recognizer can replicate the operations of the speech recognizer and in that case the preference statement related APIs of persona core would consider the gesture primitives. The same is true for other input paradigms like handwriting or tag based interaction.

Multiple Users and Multiple Applications: In the current version, Persona does not have any component to identify the users, so maintaining profiles for multiple users is not supported. Though, we do not think identifying users is the primary responsibility of Persona, we are working on the implicit identification of users and integration of the technique in Persona to support multiple profiles. Currently, one global Preference Knowledge Base is maintained and altered by interaction events in single/multi user scenario. In the case of multiple applications, there should be no conflict as long as different applications use different phrases (GUI works without any conflict) for voice-based interaction. The same is true for gesture, handwriting, tag or computer vision based interactions.

Conflict Resolution: Usually pervasive applications are deployed in multi-user scenario and a very common issue is the conflict among users when multiple users try to personalize/control the same shared service simultaneously. Conflict resolution is an independent research problem and there are several researches that are exploring this problem for suitable solutions [14,19]. The most common techniques for resolving conflict are policies and rules with priority schemes among the entities. In our current finite state engine prototype, we have not used any resolution scheme that caused by the conflict among multiple users using a shared service. However, any suitable policy can be adopted in the finite state engine, or a more sophisticated resolver can be plugged into Persona. From this perspective, Persona does not solve the conflict resolution problem rather it just provides support for that.

Scalability in Large and Complex Environment: Some readers might argue about the applicability of Persona in large-scale environment. Considering the extensible and pluggable design and the previous issues in this section, we believe that scaling into a large environment has no affect at all on Persona. For example, in section 5 we have shown that three different applications with different requirements, interactions and functionalities worked smoothly.

No Generalization for Application Logic: Persona does not handle the application logic. It 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 an application specific way.

Deployment: Once applications are deployed, Persona is automatically deployed. However, it is necessary that the application environment possesses the appropriate tool for interactions; for example, in the current version a display and a microphone are needed for GUI and voice interactions respectively.

6.1 Future Work

For supporting multiple user profiles, we are now working on the integration of a user identification scheme within the persona core. Our voice based interaction is tightly coupled with the developer specific phrases. Currently we are working on synthesizing the sense of free form English language and applying this sense to Preference Knowledge Base. Also, during the

evaluation of our systems, we have found that some users want the system to identify their preference automatically from their previous activities. In this case history information and learning is very important. So, we would like to incorporate this adaptation support in future. Furthermore, we would like to develop more realistic applications from different domain using Persona.

7. CONCLUSION

Although several works emphasized the importance of personalization in proactive applications, unfortunately available pervasive middlewares have no support for it. Persona addresses this specific issue and enables developers to allow end users for personalizing applications in a unified way. The loosely structured design makes Persona extensible for accommodating additional features. Furthermore, Persona's portability enables existing applications atop various middlewares to support personalization in an intuitive way. The primary contribution of this paper is two fold. First, we have shown an intuitive design concept for structuring preference options in a proactive application to enable unified system support for preference management. Second contribution of the paper is a solid implementation of the concept in a portable toolkit with multiple built-in interaction techniques. From the pervasive computing point of view, we believe this work will seek major attention of the community, since we have approached the personalization aspect in a unique way minimizing the overhead of many system related issues considerably.

8. REFERENCES

[1] A. K. Dey. G. D. Abowd, D. Salber, "A Conceptual Framework and a toolkit for supporting the rapid prototyping of context-aware applications". Human-Computer Interaction, Vol-16 2001.

[2] D. A. Norman. "The Design of Everyday Things" NY:Basic Books.

[3] 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.

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

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

[6] J. Brotherton, G. D Abwod, K. Troung, "Supporting Capture and Access Interfaces for Informal and Opportunistic Meeting", GVU Technical Report GIT-GVU-99-06.

[7] K. Fujinami, T. Nakajima, "Towards System Software for Physical Space Applications", In Proc of the 20th ACM Symposium on Applied Computing (SAC) 2005, pp. 1613-1620, Santa Fe, USA, March 2005

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

[9] K. Nishigaki, K. Yasumoto, 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

[10] 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.

[11] M. Marinilli, A. Micarelli,, "Generative Programming Driven by User Models", The 10th International Conference on User Modeling 2005.

[12] M . S. McNee,S. K. Lam, J. A. Konstan and J. Riedl, "Interfaces for Eliciting New User Preferences in Recommender Systems" The 9th International Conference on User Modeling (UM'2003)

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

[14] N. Dunlop, J. Indulska, and K. Raymond, "Methods for Conflict Resolution in Policy-Based Management Systems",Proc. 7th IEEE International Enterprise Distributed Object Computing Conference, Brisbane, Sept 2003, pp 98-109.

[15] P. Dourish, "The Appropriation of Interactive Technologies: Some Lessons from Placeless Documents." Computer-Supported Cooperative Work: Special Issue on Evolving Use of Groupware, 12, 2003, 465-490.

[16] 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.

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

[18] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. IEEE Software, 13(6):47–55, November1996.

[19] S. Evi, S.W. Loke, and P. Stanski, "Methods for Policy Conflict Detection and Resolution in Pervasive Computing Environments", Policy Management for Web workshop in conjunction with WWW2005 Conference, Chiba, Japan, 10-14 May 2005

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

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

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

[23] Sphinx4 Speech Recognizer. http://cmusphinx.sourceforge.net/sphinx4.