Multimedia Tools and Applications manuscript No.
(will be inserted by the editor)

A Portable Toolkit for Supporting End-User Personalization and Control in Context-Aware Applications

Fahim Kawsar · Kaori Fujinami · Tatsuo Nakajima · Jong Hyuk Park · Sang-Soo Yeo

Received: date / Accepted: date

Abstract A context-aware application in the pervasive computing environment provides intuitive user centric services using implicit context cues. Personalization and control are important issues for this class of application as they enable end-users to understand and configure the behavior of an application. However most development efforts for building context-aware applications focus on the sensor fusion and machine learning algorithms to generate and distribute context cues that drive the application with little emphasis on user-centric issues. We argue that, to elevate user experiences with context-aware applications, it is very important to address these personalization and control issues at the system interface level in parallel to context centric design. Towards this direction, we present *Persona*, a toolkit that provides support for extending context-aware applications with end-user personalization and control features. Specifically, Persona exposes a few application programming interfaces that abstract end-user customization and control mechanisms and enables developers to integrate these user-centric aspects with rest of the application seamlessly. There are two primary advantages of Persona. First, it can be used with various existing middlewares as a ready-to-use plug-in to build customizable and controllable context-aware applications. Second, existing context-aware applications can easily be augmented to provide end-user personalization and control support. In this paper, we discuss the design and implementa-

Fahim Kawsar
Computing Department, Lancaster University, UK
E-mail: fahim.kawsar@comp.lancs.ac.uk

Kaori Fujinami
Department of Computer, Information and Communication Sciences
Tokyo University of Agriculture and Technology, Tokyo, Japan
E-mail: fujinami@cc.tuat.ac.jp

Tatsuo Nakajima
Department of Computer Science, Waseda University, Tokyo, Japan
E-mail: tasuo@dcl.info.waseda.ac.jp

Jong Hyuk Park
Department of Computer Science and Engineering, Kyungnam University, Korea
E-mail: parkjonghyuk1@hotmail.com

Sang-Soo Yeo
Division of Computer Engineering, Mokwon University, Korea
E-mail: ssyeo@msn.com

tion of Persona and demonstrate its usefulness through the development and augmentation of a range of common context-aware applications.

Keywords End-user · Personalization · Context-Aware Application · Toolkit

1 Introduction

A context-aware application in the pervasive computing environment typically runs atop distributed smart objects embedded with awareness technologies (sensors, actuators and perception algorithms) where application uses these objects to collect context information or to perform some services that cause changes in the real world (e.g., adjusting the air-conditioner based on sensed temperature) including adaptation of its own behavior [10, 21, 33]. Usually, these actuation and self-adaptation are implicit, i.e., application autonomously performs these actions based upon the context cues. Due to this autonomy end-users have limited control over the application behavior which cause confusions as they struggle to understand why an application is behaving in a certain way (e.g., why the nearby mirror turned into a display showing today's weather forecasting, etc.) and how to change/configure that behavior. Eventually, these confusions hinder the adoption of context-aware applications [3, 5]. Previous studies reported that without bringing users at the center of the control, it is very difficult to increase the adoption of context-aware applications in pervasive computing environment [13, 17]. These studies suggest that supporting user-centric personalization and control will significantly impact the adoption of context-aware applications.

In this paper, we address this end-user personalization and control issues from a system perspective. Here by personalization, we refer to the active involvement of end-users to customize the adaptive behavior of the application. These issues raise two requirements:

- An appropriate user interface using which end-users can control and personalize context-aware applications.
- An appropriate system component that translates end-user interactions into corresponding system events enabling other components to adapt their behavior to reflect end-users' preferences.

Although, there have been numerous efforts to systemize the gathering and distribution of context cues to build context-aware applications [2, 10, 20, 23], these user-centric requirements were met at the application level by ad-hoc implementations. Our objective is to systemize this ad-hoc practice through a suitable system component and to seamlessly integrate it with existing middlewares.

Our contributions in this work are two-fold.

1. We present a stand-alone toolkit *Persona* that provides suitable system support for augmenting context-aware applications and can be used with various middlewares. Specifically, the toolkit offers:
   - An interface engine to generate multi-modal user interfaces for providing end-user personalization and control support. The engine has built-in support for graphical user interface and speech interface.
   - A range of application programming interfaces that enable developers to capture end-users' interactions with context-aware applications.
   - A document centric approach to provide the control of the application to end-users in such a way that an application's default runtime behavior (triggered by dynamic context information) is overlaid by end-users' preferences with a finite state engine.

2. We show the effectiveness of Persona by augmenting three existing context-aware applications with multi-modal personalization and control support.

The rest of the paper is organized as follows: in the next section we discuss the background of our work and place it against related works. The design aspects of Persona are explained in Section 3 followed by its architectural detail in Section 4. After that in Section 5, we discuss the programming model offered by Persona. Then in Section 6, we demonstrate Persona's ability to augment context aware applications with personalization and control support by illustrating three applications. Before concluding we put forth a few interesting issues in Section 7 for further discussions.

2 Background

A context-aware application that run atop or within smart environment usually consists of the following components:

– **Basic Application Component:** This component takes care of the basic application behavior, e.g., application logic, interfaces, etc.
– **Communication Component:** This component manages the access to smart objects and utilization of their services. Typical functions performed by this component include locating, managing, configuring and interacting with the smart objects. Different smart objects may have different communication and data protocols. It is the communication component that handles this heterogeneity and provides application with an unified access.
– **Perception and Adaptation Component:** This is where application handles the context data derived from the smart objects by applying application specific context reasoning and modeling policies. Applications use this context information to adapt application behavior accordingly and to further interact with the smart objects via communication component to actuate smart object services.

The last two components are recurrent and usually supported by a suitable infrastructure [2, 10, 23, 32]. Typically, such infrastructure handles the access issues by providing a discovery mechanism and provides Application Programming Interface (API) to interact with the smart objects transparently, and there by tries to separate the application from the underlying environment. However, the end-user centric issues are left entirely open at the first component and often completely missing with the presumption that application can correctly adapt its behavior autonomously by sensing surrounding contexts. We argue that such assumption considerably degrades the acceptability of the context-aware systems. By taking control and personalization away from the users, these applications typically makes end-users life harder, as they keep wondering why the system is behaving in a certain way and how to configure it applying their own preferences and styles [5, 31]. This work puts forth the argument that end-users should be the center of control and should have the ability to personalize a context-aware system. Rogers [31] and Bell et. al. [4] concurred similar views on their works with human-centric computing.

End-user personalization in context-aware applications is not a new topic. There are works that looked at these issue from socio-technological perspectives. 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 [3]. 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 [6]. In all three works, they have tried to furnish some levels of autonomous interactivity. In line with these works, in this paper we have tried to systemize this end-user interactivity. Our prime motivation is to offer system support to developers that enable them to bring end-users at the center of the control and give end-users the flexibilities to personalize a context-aware system. We consider, theses issues are typically common to all proactive applications, thus providing a systematic and structured support will make context-aware application development simpler , less complicated and more close to end-users.

Our approach basically driven by the separation-of-concern principle as we have taken the end-user interaction away from the basic application component (mentioned earlier). From an abstract point of view the proposed toolkit, Persona provides two levels of support for the application developers. First, by providing a range of APIs and well defined amendable data structure through a plug-in module, it enables developers to augment an application with personalization support. Application developers can provide this application specific control and preference management in a systematic fashion. Second, by providing an interface engine, Persona simplifies the end-user interface development as any suitable interaction modality can easily be integrated with an application to let end-users personalize and control the application.The interface engine has built-in support for graphical user interface and speech interface. End-users' interactions regardless of the interaction modality are presented to the application in a unified fashion through well defined system events enabling developers to integrate them in the application logic. As a consequence, developers can solely focus on the application logic to define the context-awareness of the system, while providing end-users with the fullest control to personalize the system.

2.1 Related Work

Considering the focus of this paper is more on system support for personalization rather than social perspectives, we would like to discuss some relevant research projects. Most of the personalized systems that we found in the pervasive literature usually built on the concept of *Personal Infromation Cloud* where users preferred information and services follow the users. There are two ways for providing such personal clouds: either using portable device or infra-structured embedded system. One project that combined both these alternatives is Personal Interaction Point (PIP) system [18]. Intel's Personal Server utilizes small and powerful wearable hardware that user can use to carry all their personalized content and services (to some extent). Users can use environment displays to access the content via wireless technologies [35]. Another work that focused on dedicated hardware for personalized services is Wearable Key by Matsushita et al. [26]. Their TouchNet system consists of a tranmitter and a receiver that can be connected via human body conductor and this idea is used to identify the user to provide his/her personalized contents. Although these projects and similar initiatives provide fair solutions from the end-user perspective to provide personalized services, none of them had approached a toolkit for the content developer, i.e., application developer. The work presented in this paper primarily targets the application developers to provide the personalized content. Our approach enables application developers to use any of these technologies mentioned above to be integrated into application environment to support personalization.

Rule based tools like iCAP [12], Stick-e-notes [30], Alfred [16] provide visual tool, or sound macros to the end-users to define conditional rules based on the context to connect input and output events. Similarly recognition tools, or more formally Programming by Demonstration systems like CAPpella [11] uses machine learning techniques to allow end-users to associate personalized rules with real world events.These approaches are valid for rapid prototyping and also to personalize the pro-active behavior of the applications. However, they do not provide any general guideline regarding application development to systemize these supports. Two notable recent works are Situation [9] and PersonisAD [1]. In Situation, Dey etl al. extended their well known Context Toolkit [10] with a new situation component that exposes the internals of a context-aware system thus enabling developers and designers to build systems with intelligible and controllable user interfaces that are more close to end-users [9]. In PersonisAD [1], a query response of context information is augmented with meta context information that can be used to provide users with feedback of the system's internal state. However, our work differs from them as we have tried to generalize the interface development mechanism for personalization and control support, while multiplexing it with the internals of the applications through structured APIs.

Sakai proposes a framework that focuses on the end-user preferences of mobile phone applications [19]. 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 [29] 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. 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. Speech and GUI-based interaction for controlling smart spaces has been investigated in various projects like Odisea [28], EasyLiving [7] etc. Especially the EasyLiving project highlights the suitability of natural language based speech interface for smart spaces. Persona has built in support for voice and GUI based interaction modalities adhering to these projects suggestions. In addition the loosely coupled design of Persona and the plug-gable input interface component (see Section 4.2) enables Persona to accommodate any suitable interaction modality into the application environment.

In traditional desktop computing graphical user interface is provided to personalize an application. This aspect has been well investigated in [14, 34] 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 [13]. Although his work is very similar and influenced us significantly, he focused on pre-design considerations of group-ware for appropriation features in stead of providing a system tool like ours to automate the process. 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 [25, 27]. However, none of them addresses the issue of a unified system support as we explore in this paper.

3 Design Issues

Figure 1 depicts a hypothetical context-aware application scenario in pervasive computing environment; a smart space populated with a range of smart objects and sensing infrastruc-

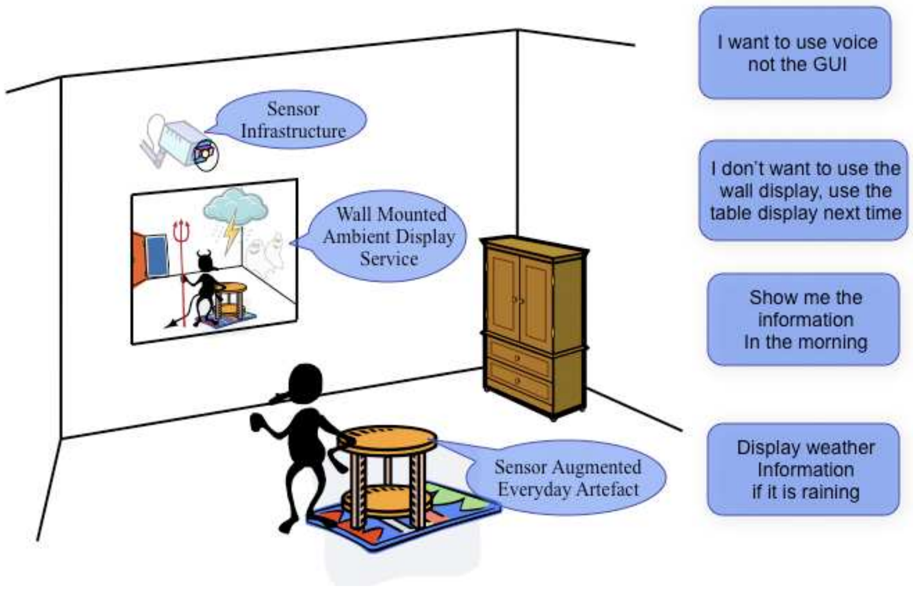

Sensor Infrastructure
Wall Mounted Ambient Display Service
Sensor Augmented Everyday Artefact
I want to use voice not the GUI
I don't want to use the wall display, use the table display next time
Show me the information In the morning
Display weather Information if it is raining

Fig. 1 A Hypothetical Multimodal Context-Aware Application involving sensors and smart objects

ture that provides some proactive services. As we have mentioned earlier, bringing end-users at the center of the control of such applications requires a suitable user interface that allows them to control and configure the system as well as a system component that can translate these interactions into appropriate system events that can be multiplexed with the rest of the application logic. Systemizing these processes leads to following two requirements:

- A system component is necessary that can generate intelligible and controllable user interface. Ideally, this interface should be independent from the underlying application logic.
- A system component that can capture users interactions and represents these interactions into the underlying application in a way that can be systematically modeled during development time. This will enable an application to reflect end-users preferences dynamically at runtime.

The first requirement basically asks for a system component that can generate user interfaces meant for personalization and control regardless of the application logic. Personalization is the direct impact of a subset of users 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, this requirement is met by building an interface engine with bi-directional plugs. That is, at one side, this engine enables developers to specify the required interaction by offering them with a collection of APIs. One the other side any multimodal interface can be plugged into this engine that enables end-users to interact with the system.

The second requirements is more focused on the interaction translation and integration issues. First of all, it is important to understand how personalization and control features can be represented in a unified manner. To discuss this, lets consider Fig. 1 again, the hypothetical scenario gives the impression that there are multiple categories of preferences and controls such as:

- **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.
- **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.
- **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.
- **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 he/she is in the meeting room.

Albeit this listing is for an example purpose, it covers most of the pervasive applications. So to abstract these preference and control options in a unified manner it is important to derive a syntactically well defined data structure that can be used as the logical carrier for the preference features. In Persona, we have designed such a data structure that enables developers to define these application specific preference and control options. Developers are free to build applications preference options and grouping those options into some categories as above. Persona can then encapsulate these categories into a generic data structure.

The next issues is how the end-users' interactions converge with preference and control features. In Persona, this is done by following a document centric approach. Persona is offered to the developers as an application plug-in, The plug-in basically provides a collection of APIs using which application developers can add control and personalization features to an application and can select appropriate interaction interface for those features. Persona then takes care of the rest by generating the appropriate user interfaces following the preference and control specifications provided by the developers. Persona converts this specifications into an XML formatted preference document. In addition to provide the interface specification, this document is also used by Persona to overwrite an application's default behavior to reflect end-user's preferences. Figure 2 shows this simple design methodology.

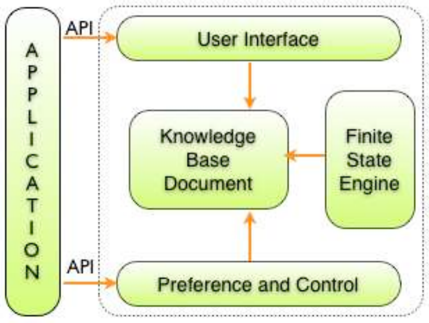


Fig. 2 Basic Building Blocks of Persona

Another design issue is how the end-users behavior will be merged with the default system behaviors. Typically, a context-aware application is driven by a set of predefined rules that are triggered when particular contexts are matched. However, Persona's design methodology implies that the application rules will be independent from end-users' runtime preferences. Accordingly, it is necessary to resolve the behavior of the system at runtime. Persona assumes that the end-users always have the highest priority and any system behavior can be written off by Persona if it conflicts with end-users' preferences. To provide this runtime resolution Persona employs a finite state engine, that operates on the document (mentioned earlier) to define the actual runtime behavior of the system.

To conclude this section, here we are summarizing the design principles that are followed in Persona

1. Providing an application independent interface engine capable of plugging different interaction techniques.
2. Providing a unified data structure backed up by well defined APIs to enable developers to provide preference and control specification and to capture end-users' interactions.
3. Providing a document centric approach to connect the various part of the system.
4. Providing a Finite State Engine to enable dynamic time resolution to reflect end-users' preferences nullifying systems' default behavior triggered by context sensing (when appropriate).

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

4 Architecture of Persona

Following the design guidelines presented in the previous section, Persona is built in a loosely structured manner where one core component plays the primary actors role and provides interfaces for plugging other components. Persona is basically sandwiched between a context aware application and its corresponding middleware (used to capture and distribute context and actuate events). Figure 3 illustrates the basic building blocks of Persona that are described below.

4.1 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 and control options for the application in context. Developers can create custom categorization of preferences or can use the built-in ones (discussed in Section 3). Furthermore, applications can subscribe to Persona to receive interaction events that are related to personalization and control or can poll periodically. Currently this interface is implemented in java and offered to the developers as a library package. In the later part of this section, we will discuss how developers can use this component to support personalization and control in their applications.

4.2 Input Interface

As we discussed in the previous section, one of the requirements to support end-user personalization and control is to provide appropriate user interfaces. These interfaces typically

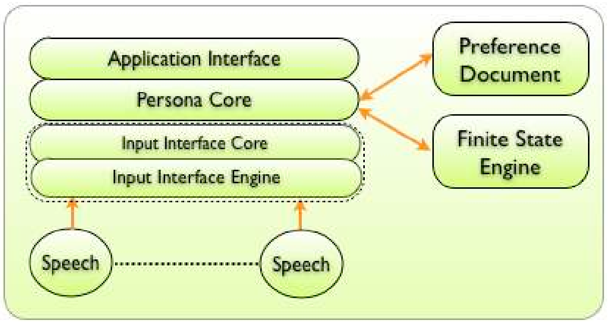
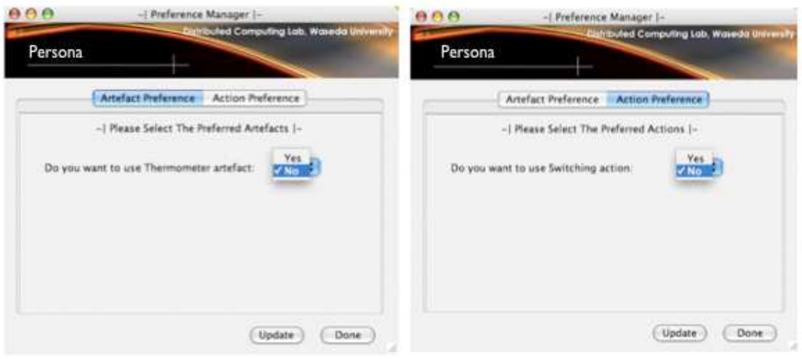


Fig. 3 Internal Architecture of Persona

vary from application to application, and employ a multitude of interaction modalities, e.g., graphical user interfaces, speech, projected interface, tangible interface, speech interface, gesture based interfaces, etc. However, regardless of the interaction modalities, end-users' interactions with context aware applications have the identical meanings to the application. For example, consider a smart lamp that turns itself on automatically and adjusts the brightness adapting ambient light level. However, a user may want the lamp to be turned on manually but to adapt the brightness autonomously and can do this customization by saying a particular phrase (speech), or by pressing a specific button (tangible), or by selecting a specific option from the GUI panel, etc. However, to the underlying application that controls the lamp, these actions have the identical meaning. The above rationale leads to a 2-layer input interface architecture.

1. Input Interface Core: This core acts as a wrapper for the underlying interaction modality (speech, GUI, gesture, etc.). It provides a plug-in architecture and offers a range of APIs to developers using which a developer can wrap any specific multimodal interaction interface. This core converts users' interactions from varying user interfaces into a unified interaction primitives and passes them to underlying Persona Core for further processing.
2. Input Interface Engine: This engine is interface specific and enables an application to exploit a variety of multimodal interfaces. In the current prototype two interaction modalities are integrated.
   – **Speech Interface:** End-users usually provide their preferences in simple english language, like "Do not turn off the light automatically", "Notify me every morning", etc. The speech interface 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 [1]. This recognizer runs in the background when an application starts. To enable this speech engine the target application environment has to be equipped with one or multiple microphones.
   – **Graphical User Interface (GUI):** 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 4 shows a sample GUI that is automatically generated by Persona for the application presented in Section 5.

[1] http://cmusphinx.sourceforge.net/


-| Preference Manager |-
Distributed Computing Lab, Waseda University
Persona
Artefact Preference
Action Preference
-| Please Select The Preferred Artefacts |-
Update
Done
-| Preference Manager |-
Distributed Computing Lab, Waseda University
Persona
Artefact Preference
Action Preference
-| Please Select The Preferred Actions |-
Update
Done

Do you want to use Thermometer artefact: Yes [✓ No]

Do you want to use Switching action: Yes [✓ No]

Fig. 4 Automatically Generated GUI by Persona for a Sample Application

Due to this clear separation and unified representation of interaction primitives, it is possible to support a range of multi-modal interfaces in Persona.

4.3 Persona Core

This is the central component of Persona and connects other components (i.e., Application Interface, Input Interface, and Finite State Engine) of Persona with each other exploiting the Preference Document (described in Section 4.4). Upon receiving interaction events from the real world though Input Interface, Persona Core filters out the preference options utilizing the Preference Document and notifies the application using Application Interface. Also, when conflict arises among application states, it consults the Finite State Engine (described in Section 4.5). for the appropriate resolutions.

4.4 Preference Document

This is an XML file generated dynamically during the deployment time of an application that uses Persona for personalization and control support. It contains application specific personalization options provided by the application developer using Personas APIs. Each of these personalization and control options is parameterized by a number of interaction possibilities and based on end-users' interactions the preference and control options are properly set. When Persona receives external real world events, this document is consulted for filtering preference data and for decision-making.

4.5 Finite State Engine:

In context-aware application, typically several actions are triggered based on the sensed context and these triggering rules are defined by the developers during development time. Most of the time, these rules do not take users preferences into account, thus when end-users' are brought into the center of control and customization often there could be conflict between the default application behavior and users preferred behavior. This eventually might cause an application to move into invalid states. The finite state engine component of Persona

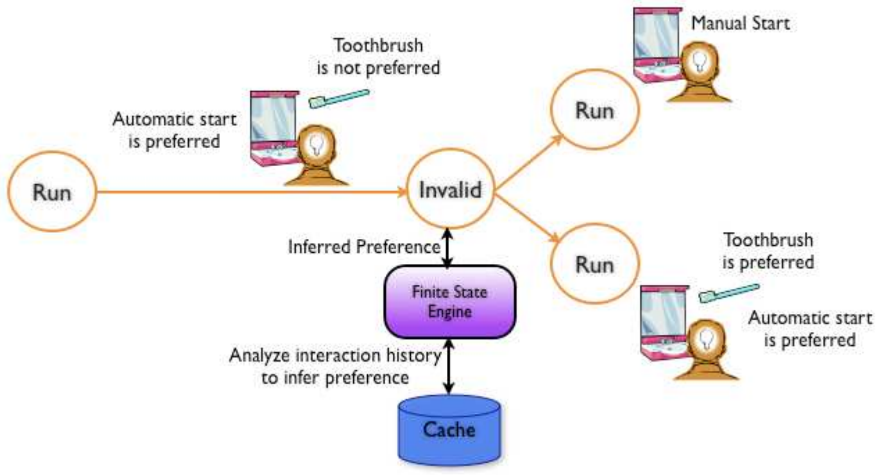


Fig. 5 Example Operations of Finite State Engine

is designed to handle these situations. It internally maintains a small cache of past preference change and usage events, which can be exploited to recover from invalid conditions to maintain application flow. To understand the working mechanism of this component, lets consider a simple application scenario: a smart mirror installed in the washroom that shows some personalized ambient information related to a user when he/she brushes teeth in front of the mirror. The toothbrush is augmented to identify the user [15]. 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/her identity. Now if the user does not want to use the toothbrush but still wants the mirror to trigger its display functionality automatically, then the application moves to an invalid state (because the system can not identify the user thus cannot retrieve the personalized information related to that user like schedule etc.). When this conflict is identified by the application, it can use the finite state engine to maintain its workflow. In this scenario, depicted in Fig. 5, the finite 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.

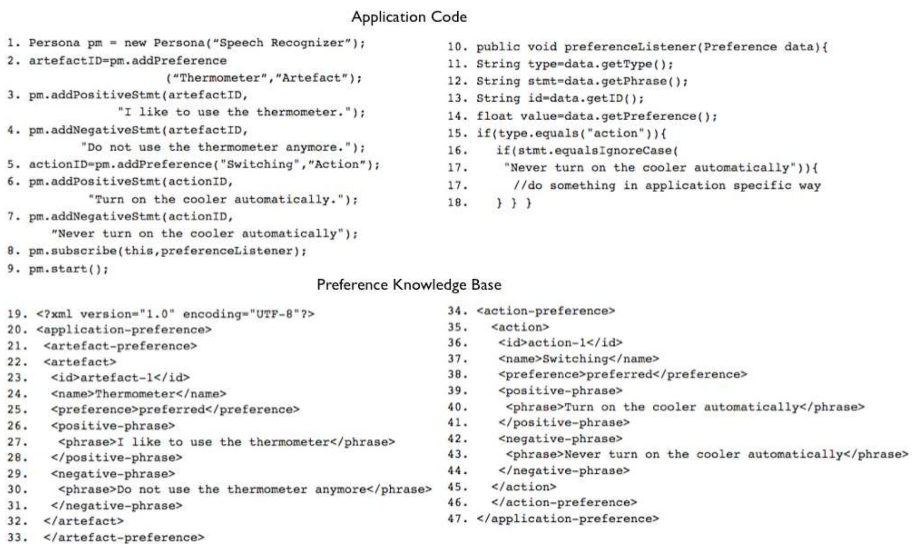
5 Programming Model: Integration of Persona into Applications

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 and control support in an application whereas end-users can interact with the applications. Persona filters end-users interaction meant for personalization and accordingly modifies applications behavior. From developers perspective, integrating Persona into applications requires following steps:

1. Listing all the preference and control options of the applications.
2. Categorizing these options using taxonomy similar to the one presented in Section 3.
3. Generating the Positive and Negative statements for each of the preference options.

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.

Fig. 6 Persona APIs Available to the Developers



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>
```

Fig. 7 Code Snippets and Preference Knowledge Base demonstrating Personas usage in Applications

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 Document and corpus of the speech 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.

Steps 1-3 are design phase tasks where step 4-6 are development phase tasks. The code snippets in Fig. 7 demonstrates the latter steps (4-6) utilizing the APIs presented in Fig. 6 for a very simple application composed of a thermometer and a cooler followed by the Preference Document generated for this application.

In 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 document (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 document. For lines 2, 3 and 4 in the application code, we have entries (line 22-32) in the preference document. 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 document. 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 document, it extracts the information for that phrase from the preference document and sends it to the application. It also updates the preference document e.g. ¡preference¿ attribute to Preferred or Not Preferred based on captured event. Similarly, for GUI engine when the GUI event is captured, it is sent back to application and the persona core updates preference document. 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 preference from the preference document.

6 Evaluation through Application Development

We have adopted a scenario based evaluation method introduced in [24] for evaluating Persona's support in providing applications with construct for end-user personalization and control. Let us 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 makeup, 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 accordingly. Furthermore she can find 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 Misrables 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 clients email.

This scenario is implemented using following three different context-aware applications:

6.1 AwareMirror Application

AwareMirror is a smart mirror installed in the washroom as shown in Fig. 8. In addition to its primary task of reflecting someones image it can also provide some useful information about the person who is using the mirror. Before deploying this application we have carried out two potential users surveys. About 50 people aged 20-50 participated in the surveys.

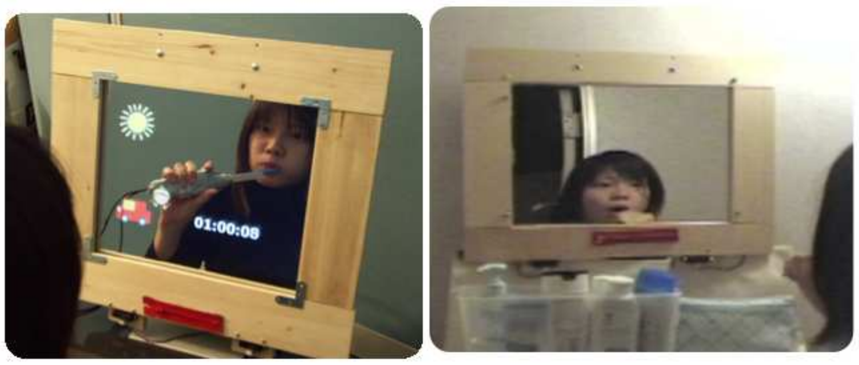


Fig. 8 AwareMirror Application in Operation

rephrase Prior to deploying this application we performed two surveys on potential users. The first survey was to find out the information categories that are preferred by the user to be displayed and second survey was to figure out the best sensing technologies. Based on the survey result, we have selected three categories of information to be presented to the users; these are i) Schedule ii) Transportation Information, iii) Weather Forecasting. The sensing technology preferred by the participants is implicit sensing via everyday objects rather than any vision based explicit sensing. The artefacts and the preference options are explained below.

Physical Artefacts

1. Mirror: A regular mirror augmented with LCD display to show personalized information proactively and proximity sensors to detect users presence.
2. Toothbrush: A regular toothbrush augmented with accelerometer sensor used to identify a user

Preference Options

– Artefact: With /without toothbrush
– Action: Automatic/manual start/close
– Interaction: Tangible Button/Voice/GUI for navigation.
– Timing: Morning/Always

6.2 ByteNDine Application

This application shown in Fig. 9 is designed for a public/private dining space scenario. The goal of the application is to provide the latest news to the user while dining. A lot of people prefer to read newspaper, magazine, books in a caf or a restaurant. We tried to capture this practice by providing information on the dining table, which means the table acts as an ambient display. We have assumed that people will carry a tag/token that will represent his/her preferred topic. Accordingly the system provides that topics latest news to the user in an unobtrusive manner. User can browse through the news and look for detailed information or can simply close the display. The functional components and the preference options are

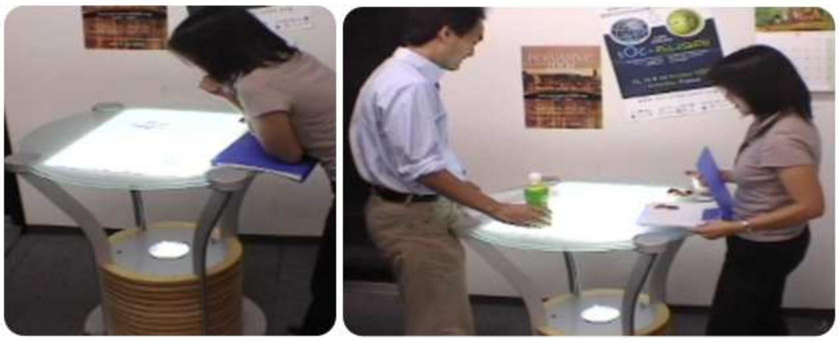


Fig. 9 ByteNDine Application in Operation

explained below.

Physical Artefacts

1. Table augmented with projector and RFID tag reader to identify users presence and preference.

Preference Options

– Artefact: None
– Action: Automatic/manual start/close
– Interaction: Touch Display/Voice/GUI for navigation
– Timing: Morning/Always, Alone/Always

6.3 Smart Assistant Application

This application (Fig. 10) is designed for a workspace running on the users desktop. It is a simple media level context aware application that can track users activities by the usage of artefacts populated in the workspace. To be specific, the application uses a chair, a desk lamp, a tray and a few mugs and jars for sensing users contexts. Based on the state of these artefacts the application can track if a user is in the workspace and whether he/she is working for a long time. If so the application suggests the user to take a refreshment and can provide the user with some predefined schedule notification. Also the application can control workspace lighting based on the users presence and surrounding environments brightness. It can also play music using systems media player. The system uses an animated chatting agent to interact with its user. The user can additionally chat with this agent during leisure time. Though the chatting agent is not very smart, it can entertain the user for a while. The artefacts and the preference options are explained below.

Physical Artefacts

1. Chair and Lamp: Sensor augmented chair and desk lamp to detect users presence and light sensitivity of the workspace.

Fig. 10 Smart Assistant Application in Operation

2. Media Player: A simple agent to play music.

Preference Options

– Artefact: Yes/No Use of Lamp, Music Player
– Action: Yes/No Suggestion for break, music play, automatic light.
– Interaction: None (All are proactive)
– Timing: Always/On specific time

All three applications are developed atop a middleware called Prottoy [22] and augmented with personalization support using Persona. All three applications were previously developed with personalization support and were reported in [21].

6.4 Observations

We were interested in several things from Personas evaluation point of view:

1. **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.
2. **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.
3. **End-users Impression:** We had performed informal user trials involving 9 people (6 Male, Age Range: 21 32) to evaluate Personas 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 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.

7 Discussions

Our major design concern was to provide a structured representation of the preference data that makes management of personalization easier. In Section 5 we have shown how to use the APIs to represent the personalization data. Furthermore, we have provided an exemplary classification scheme in Section 3. 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 6.

”While she continues working, her desk lamp turns on automatically and, it dims into a pink shade and the track . In this case, the built-in 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 as any 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) 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, the 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.

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 situations are 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. Given the loosely coupled nature of Persona, it can be easily seen that a new interaction engine can be injected seamlessly into other applications. 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.

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 6 we have shown that three different applications with different requirements, interactions and functionalities worked smoothly. 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. 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.

8 Conclusion

Although several works emphasized the importance of supporting end-user personalization in context-aware applications, unfortunately available contex-aware middlewares do not have adequate support for that. Persona addresses this specific issue and enables developers to allow end-users for personalizing context-aware applications in a unified way. In this paper we have discussed the background and design rationales behind Persona and explained the resulting system architecture in detail. We have also shown how Persona can provide support to develop and extend context-aware application by discussing a range of context-aware applications. The primary contributions of this paper are two fold. First, it allows a range of context-aware applications to support end-user personalization and control in a systematic fashion with multiplexing it with application code explicitly. Second, it can be used with different context-aware middlewares and thus can be used to extend existing context-aware applications with personalization and control support. We consider, out work is useful for pervasive computing community, and specially for applications that are context-aware.

References

1. M. Assad, D. J. Carmichael, J. Kay, and B. Kummerfield. Personisad: Distributed, active, scrutable model framework for context-aware services. In *Fifth International Conference on Pervasive Computing*

(Pervasive 2007), pages 55–72, 2007.

2. J. E. Bardram. The java context awareness framework - a service infrastructure and programming framework for context-aware applications. In *The 3rd International Conference on Pervasive Computing (Pervasive 2005)*, pages 98–115, 2005.
3. L. Barkhuus and A. Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. In *5th International Conference on Ubiquitous Computing*, pages 150–156, 2003.
4. G. Bell and P. Dourish. Yesterday's tomorrows: Notes on ubiquitous computing's dominant vision. In *Personal and Ubiquitous Computing*, volume 11(2), pages 133–143, 2007.
5. V. Bellotti and K. Edwards. Intelligibility and accountability: Human considerations in context-aware systems. *Human-Computer Interaction*, 16(2-4), pages 193–212, 2001.
6. 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), pages 153–263, 1997.
7. B. L. Brumittet, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: Technologies for intelligent environments. In *2nd International Symposium on Handheld and Ubiquitous Computing (HUC 2000)*, pages 12–29, 2000.
8. K. Cheverst, K. Mitchell, and N. Davies. Investigating context-aware information push vs. information pull to tourists. In *Mobile HCI*, pages 1–6, 2001.
9. A. Dey and A. Newberger. Support for cotext-aware intelligibility and control. In *ACM Conference on Human Factors in Computing Systems (CHI 2009)*, pages 859–868, 2009.
10. A. K. Dey, G. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2-4):97–166, 2001.
11. A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. a cappella: Programming by demonstration of context-aware applications. In *ACM Conference on Human Factors in Computing Systems (CHI 2004)*, pages 33 – 40, 2004.
12. A. K. Dey, T. Shon, S. Streng, and J. Kodama. icap: Interactive prototyping of context-aware applications. In *4th International Conference on Pervasive Computing (Pervasive 2006)*, pages 254–271, 2006.
13. P. Dourish. The appropriation of interactive technologies: Some lessons from placeless documents. In *Computer-Supported Cooperative Work: Special Issue on Evolving Use of Groupware*, pages 465–490, 2003.
14. S. Farrell, V. Buchmann, C. S. Campbell, and P. P. Maglio. Information programming for personal user interfaces. In *Intelligent User Interfaces*, pages 190–191, 2002.
15. K. Fujinami, F. Kawsar, and T. Nakajima. Awaremirror: A personalized display using a mirror. In *3rd Third International Conference on Pervasive Computing (Pervasive 2005)*, pages 315–332, 2005.
16. K. Gajos, H. Fox, and H. Shrobe. End user empowerment in human centered pervasive computing. In *International Conference on Pervasive Computing (Pervasive 2002)*, pages 1–7, 2002.
17. R. H. Harper. Why people do and don't wear active badges: A case study. In *Computer Supported Cooperative Work*, pages 297–318, 1996.
18. D. Hilbert and J. Trevor. Personalizing shared ubiquitous devices. *ACM Interactions Magazine*, pages 34–43, 2004.
19. S. Hiroshi, Y. Murakami, and T. Nakatsuru. Personalized smart suggestions for context-aware human activity support by ubiquitous computing networks. In *NTT Technical Report*, pages 77–84, 2004.
20. J. I. Hong and J. A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *The Second International Conference on Mobile Systems, Applications, and Services (Mobisys 2004)*, pages 177–189, 2004.
21. F. Kawsar, K. Fujinami, and T. Nakajima. Augmenting everyday life with sentient artefacts. In *2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies (sOc-EUSAI 2005)*, pages 141–146, 2005.
22. F. Kawsar, K. Fujinami, and T. Nakajima. Prottoy: A middleware for sentient environment. In *International Conference on Embedded and Ubiquitous Computing (EUC 2005)*, pages 1165–1176, 2005.
23. F. Kawsar, K. Fujinami, and T. Nakajima. Deploy spontaneously: Supporting end-users in building and enhancing a smart home. In *The Tenth International Conference on Ubiquitous Computing (Ubicomp 2008)*, pages 282–292, 2008.
24. R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, 13(6), pages 47–55, 1996.
25. M. Marinilli and A. Micarelli. Generative programming driven by user models. In *10th International Conference on User Modeling*, pages 30–39, 2005.
26. N. Matsushita, S. Tajima, Y. Ayatsuka, and J. Rekimoto. Wearable key: Device for personalizing nearby environment. In *Proceedings of the Fourth International Symposium on Wearable Computers (ISWC'00)*, pages 119–126, 2000.

27. M. . S. McNee, S. K. Lam, J. A. Konstan, and J. Riedl. Interfaces for eliciting new user preferences in recommender systems. In *9th International Conference on User Modeling*, pages 178–187, 2003.
28. G. Montoro, X. Alamn, and P. A.Haya. Spoken interaction in intelligent environments: a working system. *Advances in pervasive computing. Austrian Computer Society*, pages 7–12, 2004.
29. 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*, pages 345–351, 2005.
30. J. Pascoe. The stick-e note architecture: Extending the interface beyond the user. In *2nd international conference on Intelligent user interfaces (IUI 1997)*, pages 261 – 264, 1997.
31. Y. Rogers. Moving on from weiser's vision of calm computing: Engaging ubicomp experiences. In *The Eighth International Conference on Ubiquitous Computing (Ubicomp 2006)*, pages 404–421, 2006.
32. M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.
33. A. Schmidt. *Ubiquitous Computing-Computing in Context*. PhD thesis, Lancaster University, 2002.
34. O. Stiermerling, H. Kahler, and V. Wulf. How to make software softer - designing tailorable applications. In *Designing Interactive Systems (DIS)*, pages 365–376, 1997.
35. R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. The personal server: Changing the way we think about ubiquitous computing. In *Fourth International Conference on Ubiquitous Computing (Ubicomp 2002)*, pages 194–209, 2002.

Authors

Fahim Kawsar is a PostDoc in the Embedded Interactive System group of Lancaster University,UK. He received his Ph.D. and M. Engg. at the Distributed Computing Lab of Waseda University in 2009 and 2006 respectively. His research interests evolve around ubiquitous computing with specific interest in smart object systems, human-centric system infrastructures and tangible interfaces. He has published in the areas of distributed middleware, smart objects, personalization, and physical interfaces. He was a recipient of 2006-08 Microsoft Research (Asia) fellowship.

Kaori Fujinami is an associate professor in the department of computer, information and communication sciences at Tokyo University of Agriculture and Technology. He received a MS in Electrical Engineering and a Ph.D. in Computer Science from Waseda University in 1995 and 2005, respectively. He has been working on context recognition and representation through the utilization of daily objects since 2002.

Tatsuo Nakajima is a professor in Department of Computer Science and Engineering of Waseda University. His research topics are operating systems, distributed systems, real-time systems, ubiquitous computing, and interaction design. Currently, he is is leading two projects: an operating system for future multicore based information appliances and persuasive technologies for motivating desirable lifestyle.

Jong Hyuk Park received his Ph.D. degree in the Graduate School of Information Security from Korea University, Korea. He is now a professor at the Department of Computer Science and Engineering, Kyungnam University, Korea. He has published about 100 research papers in international journals and conferences. He has been serving as chairs, program committee, or organizing committee chair for many international conferences and workshops. He was editor-in-chief of the International Journal of Multimedia and Ubiquitous Engineering

(IJMUE), the managing editor of the International Journal of Smart Home (IJSH). He is Associate Editor / Editor of 14 international journals including 8 journals indexed by SCI(E). In addition, he has been serving as a Guest Editor for international journals by some publishers: Springer, Elsevier, John Wiley, Oxford Univ. press, Hindawi, Emerald, Inderscience. His research interests include security and digital forensics, ubiquitous and pervasive computing, context awareness, multimedia services, etc. He got the best paper award in ISA-08 conference, April, 2008.

Sang-Soo Yeo received his bachelor's, master's and Ph.D. degrees in Computer Science from Chung-Ang University, Seoul, Korea. He previously taught at Dankook University, Seoul, Korea. He has joined Kyushu University in Japan as a visiting scholar at the Graduate School of Information Science and Electrical Engineering (ISEE). And then he came back to Korea and he worked for BTWorks, Inc. as a General Manager and he was involved in Hannam University as a visiting professor at the same period. Now he is a professor at Division of Computer Engineering, Mokwon University, Korea. Dr. Yeo has been serving as Chairs for a number of conferences and workshops; MUE 2007, IPC-07, FBIT 2007, FGCN 2007, WPS 2008, SH'07, MUE 2008, ISA 2008, CSA 2008, UMC 2008, BSBT 2008, FGCN 2008, ASEA 2008, SecTech 2008, and ICUT 2009. He is an associate editor of the International Journal of Multimedia and Ubiquitous Engineering (IJMUE) and he has been served as a Guest Editor for the International Journal of Security and Its Applications (IJSIA), the International Journal of Smart Home (IJSH), and the International Journal of Autonomous and Adaptive Communications Systems (IJAACS). Dr. Yeo's research interests include Security, Ubiquitous Computing, Multimedia Service, Embedded System, and Bioinformatics.