Enabling user-oriented management for ubiquitous computing: The meta-design approach

Bin Guoa,b,⇑, Daqing Zhanga, Michita Imaib

a Telecommunication Network and Services Department, Institut TELECOM SudParis, 9, rue Charles Fourier, 91011 Evry Cedex, France
b Department of Information & Computer Science, Keio University, Japan

Abstract

This paper presents iPlumber, a user-oriented management system for ubiquitous computing environments. Different from previous low-benefit “zero-configuration” systems or high cognitive-cost “end user programming” tools, our work attempts to attain a better balance between user benefits and cost by exploring the meta-design approach. A set of typical management activities in ubicomp environments are supported, from basic-level software sharing, foraging, and low-cost software configuration to advanced-level cooperative software co-design and error handling. These activities are elaborated through a smart home control scenario. The usability of our system is validated through an initial user study with a total of 33 subjects to test the management activities from an open exhibition environment and a controlled university environment.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

With the development of wireless sensor networks, ubiquitous computing (ubicomp), a post-desktop model that extends the computing domain from traditional computers to sensor-augmented everyday objects (i.e., the so-called smart objects, such as information appliances, smart toolboxes [1], and smart cups [2]), is moving closer to our lives. By analyzing the data from the attached sensors, smart objects can sense the context change in the ubicomp environment and adapt its behavior to assist users. Many context-aware applications have been developed to manifest how ubicomp benefits our daily lives. As ubicomp systems will finally be used by end users, we must consider how they can be managed and controlled by end users. This paper proposes a new approach that allows users to cooperatively manage their ubicomp spaces.

One of the important challenges for user-oriented management in ubicomp environments is in regards to the type and degree of control that should be provided to users. In desktop computing environments, most end users find themselves becoming system administrators, for they have to concern themselves with chores that would seem familiar to a mainframe system operator: performing software installation and removal, handling errors, and managing security [3]. Things become more complex in the coming ubicomp era. Taking software management for example: when installing a context-aware application, end user managers often have to configure it to meet their particular physical configuration and mental needs, such as selecting a preferred output device for a reminder application. This is mainly due to different hardware deployment issues (different sensors and devices deployed), diverse user preferences, and ever-changing user needs. They may also have to handle possible hardware and software failures during the long-term maintenance process.
Thinking about a method to benefit user needs while lowering user costs becomes a substantial challenge for ubicomp system designers.

There are two quite distinct approaches to user-oriented management for ubicomp environments (refer to Section 2 for details): one is automation, which considers that ubicomp systems are complex to non-technical users and that they should support zero-configuration; the other is end user programming (EUP), which considers that ubicomp applications cannot cater to all potential needs of different users and aims at providing ways that empower end users to create applications to meet their idiosyncratic needs. Analyzing the two approaches indicates that different system-design methods influence the activities in ubicomp management. From a software engineering perspective, the automation approach provides a pre-designed, closed system, which costs little to end users; however, the benefits obtained are also limited since the diversity and changeability of both user requirements and environment settings often cause breakdowns and failures in the system. The EUP method allows users to exert control over their environment through a “design-in-use” method which brings additional benefits to cater to user needs; however, it also imposes a certain cognition load to use the application creation tools. Thus, there is no effective way to balance the benefits and costs of user-oriented ubicomp management.

To circumvent these issues, we have developed the iPlumber (I, Plumber) system. As the name suggests, our system is designed with the purpose of enabling users to act as software-plumbers, which can install and maintain systems in the coming ubicomp era. It also follows the “design-in-use” design paradigm, but in order to lower the learning burden to users, we have introduced the meta-design method [4]. Unlike previous systems, the meta-design method provides a collaborative environment that allows users to share and reuse the context-aware applications they have created as well as their management experiences. The main aim of iPlumber is to attain a “high-benefit, low-cost” ubicomp management system:

- **High-benefit**: Rather than restricting users to being passive consumers and presenting them with closed systems (manifested in the “automation” method), meta-design provides them with opportunities and tools to modify or extend the system to fit their needs. In addition, it offers social collaboration means for handling errors and breakdowns in the long-term system maintenance process.

- **Low-cost**: A vast majority of user requirements for task management can be met by searching among the shared applications and performing a simple configuration of those that are interesting. For the needs that still cannot be met from shared applications, end user creation is also supported, but the learning burden and cost is reduced by allowing them to share and reuse programming resources and samples (e.g., inference rules and applications created by others).

Various ubicomp management activities, ranging from software foraging and configuration to system co-design and error handling, are facilitated under the meta-design method. In the following sections, we will present the iPlumber system by identifying the requirements from a smart home scenario, followed by explanations on how our system benefits these management activities. The effectiveness of our system is validated through a user study, where 33 subjects (including 18 average users from an exhibition and 15 educated users from a university) were recruited to test the management activities provided.

### 2. Related work

The development of a cooperative ubicomp management system relates to several research topics. In the following, we give a summary of previous work and clarify our contributions.

#### 2.1. Management of ubicomp systems

During the last few years, a number of researchers have been working on system management in ubicomp environments. We can classify them by users’ sense of control of the system.

Considering that ubicomp environments consist of many complex technical objects that are to be handled by non-technical users, some studies believe that a ubicomp system should support “zero-configuration” or “automation.” The AMUSE project proposes the cell pattern to support self-configuring of an e-health system [5]. The Comanche project collects user preferences in a knowledge base and builds relevant specifications for system configuration [6]. The Personal Digital Secretary system uses machine learning methods to derive user models that can be leveraged to support proactive behaviors [7]. The automation method has lessened some of the burden of human labor and has made human tasks more efficient, yet automation cannot be an answer to all the problems that people have. As investigated in references [3,8], there are several reasons that people simply resist automation. First, people often desire to perform tasks that they enjoy or tasks that are important in constructing positive self-image. For example, though people may want to give full control to the system like an air conditioning system, they may want to exert control on the style and the content that should be taken by a reminder application. Second, auto-collection of user preferences is good but not enough [9], as user preferences often change and breakdowns often occur, leading to the requirement of user-management over the original system to make it adapt to these changes. In summary, the biggest challenge for designing an automatic ubicomp system is the need to carefully consider an appropriate level of automation to employ.

To avoid the issues found in the automation approach, the *end user programming* (EUP) approach is introduced. The EUP approach stems from the perspective that it does not seem possible in ubicomp applications to cater to all of the potential needs of all categories of users. Instead, the aim is to provide a set of methods and toolkits that allow non-expert users to create or modify a software system
to meet their idiosyncratic needs. The state-of-the-art EUP tools for ubicomp environments always provide a certain level of programming abstraction. According to the approach used, these tools broadly fall into two groups: visual programming and physical programming:

(1) **Visual programming.** Many novices struggle with syntax problems when programming using texts [10]. Visual programming tools attempt to provide alternative input mechanisms to bypass these problems. Two major methods were used previously. Some systems use graphical objects to represent elements of a program such as action commands and control structures. These objects can be moved around and combined in different ways to form programs. For example, iCAP allows end users to exert control over a sensing environment through a visual, rule-based programming tool [11]. TCE is an ontology-based ubiquitous task management system that allows end users to compose Semantic Web services using a web-based graphical interface [12]. After analyzing the trade-off between autonomy and user control in a smart car-breaking system, Cheverst et al. proposed a simple user interface for end user control [13]. Utilizing explicit, easy-to-understand metaphors is another way to teach beginners how to program, and it has been exploited in a few systems. The Jigsaw project uses a jigsaw-puzzle metaphor-based interface to allow end user configuration of smart devices [14]. A magnetic-poetry metaphor-based EUP system for context-aware applications is described in CAMP [15].

(2) **Physical Programming.** Unlike systems in the prior group that ask users to work with programming techniques on the screen, systems in this group attempt to create an entirely physical or tangible programming environment. As defined by Montemayor et al. [16], physical programming is “the creation of computer programs by physically manipulating computationally augmented objects in a smart environment”. StoryRoom [16] and AutoHan [17] are two good examples of this group.

Although they open up more space for end user control, there are also several problems with existing EUP tools. First, the EUP approach imposes a learning burden to non-expert users. As depicted in previous studies, most of them are based on the structured programming metaphor, which is mainly familiar to programmers. Second, these EUP tools are designed for a single group of users, whereas the sharing and reuse of applications and management experiences is almost neglected. The previous investigation has shown, however, that there is substantial overlap among users’ needs with ubicomp applications [11].

Our system differs from and perhaps outperforms previous ones in the following respects. To better balance between “benefits from control” and “cost of control”, we propose the meta-design method. By allowing users to share context-aware applications and the inference rules (for context reasoning) they create, other people can reuse them by simply configuring some parameters to meet their particular needs, or composing/altering these shared resources to create new applications. A collaborative environment also provides benefits for user experience exchange in various ubicomp management activities, such as software foraging and error handling.

### 2.2. Error handling in ubicomp systems

Maintaining robust ubicomp applications is important for end users to accept and use them. There are several reasons that may influence the robustness of ubicomp systems, such as fallible sensors, incorrect configurations (in hardware or software), and inconsistent contexts (or uncertainty). There have been limited studies on how to handle failures in ubicomp systems. Chen et al. [18] presented an assumption-based reasoning method for resolving context inconsistency. The Gaia project allowed context-aware services to reason about uncertainty using Bayesian networks [19]. CYSMN surveyed the causes of failures in a location game and presented several strategies to deal with them [20]. Our earlier work provides a rule-based infrastructure that allows developers to deal with possible sensor failures in context-aware systems by exploring the semantic-rich relationships (e.g., location relations) among physical objects [21]. These systems exploited different reasons that may cause failures to ubicomp systems, and explored non-monotonic or rule-based techniques to deal with them; however, all of them seek to find a solution in terms of system development, and none of them study the problem from an end user perspective. Our work surveys the failures that may happen in the maintenance process, and proposes a hybrid method, a combination of automatic error detection and collaborative error handling, to tackle this issue.

### 2.3. User collaboration and knowledge sharing

User collaboration and knowledge sharing can happen in different stages of the software lifecycle: the development process, distribution process, and maintenance process.

In the development process, users cooperate to develop applications. They also share and reuse resources (e.g., source code, software APIs) to speed up the development process. For developers, open source, where developers can access source code and make improvements in a public manner, is the most successful method for collaborative software development. SourceForge (http://sourceforge.net) is the largest repository of open source projects available on the Internet. Google Code (http://code.google.com) is Google’s site for developer tools, APIs, and technical resources. In the ubicomp domain, to facilitate knowledge sharing and reuse among developers, a number of studies have explored the Semantic Web technologies to model and reason contexts, as reported in Semantic Space [22] and CoBrA [18]. There are also collaborative development platforms designed for end users. Yahoo Pipes (http://pipes.yahoo.com), for example, is a web application that provides a graphical user interface for building data mashups that aggregate RSS feeds to create web-based apps. User cooperation and knowledge sharing facilitate the development of applications, which is also explored in
our system: however, the biggest difference between our system and previous ones is that we consider the cooperation between developers and end users (developers provide meta-applications and end users can customize them), which is not exploited in previous ones.

There are often many applications with similar functions that make it difficult for users to choose. In the distribution process, user cooperation has been explored in numerous systems for application recommendation. For example, in mobile application stores like the Apple Store (http://store.apple.com), users can give scores to the applications they have experienced, and the scoring result from many users can help new users to choose. We import this method into our system to facilitate foraging of high-quality user-created applications.

In the maintenance process, users need to learn many things in order to get involved with using an application. They also seek expert help when encountering breakdowns with an application. In open source culture, official support often is lacking and peer users become important learning sources, which we call the "collaborative learning" manner. As reported in previous studies [23,24], discussion forums provide an important manner for knowledge transfer and problem solving among peer users in open source projects. After investigating the Apache voluntary support system, Lakhani and Hippe find that it functions effectively [23]. They also analyze the reasons as to "why information providers are willing to perform the question-answering task for free." In the STeP_IN project [25], the "ask expert" function was developed to allow a user to post a question to other users who have expertise on relevant Java library usage. The biggest difference between ubicomp applications and traditional desktop applications is that a breakdown in a ubicomp application can be caused by not only software problems, but hardware ones (e.g., the sensor is broken, or the sensor is not used in the right way). Failures may also be brought in when allowing end users to alter or create applications in an EUP system like iPlumber. However, previous practice and research on discussion forums has focused on developers of desktop applications. Our system differs from them in two aspects: First, we introduce the conventional discussion forum method into the ubicomp domain. Second, the collaborative error handling in our system incorporates the participation of both developers and end users, and we have made a preliminary experiment to validate whether end users themselves can be the "plumber" of their ubicomp environment with the aid of collaborative learning.

3. Meta-design and scenario analysis

Having witnessed the quickly disappearing distinction between users and developers, as in the cases of Wikipedia (http://www.wikipedia.org) and Yahoo Pipes, with the meta-design approach we concentrated on establishing a new software development methodology by viewing software systems as continuously evolving socio-technical systems. This is achieved by reformulating a continuum of different degrees of ubicomp design and management activities for both developers and end users.

In the ubicomp domain, developers include a wide range of entities, such as sensor makers, service providers, and software design companies, as well as users with certain expertise. Our system provides tools for them to quickly prototype application templates (or meta-applications) that can be shared on the web. End users can perform simple configurations of these meta-applications to meet their ever-changing needs. Shared resources, like meta-applications, inference rules, and user experiences, also lower the barriers for end users to become system co-designers and error handlers. In the following, we manifest the meta-design approach through a smart home (an important ubicomp domain) scenario and then analyze the requirements for designing our system.

The Green family is composed of the father Bob, the mother Alice, and their son Tony. Bob is an engineer and he works in a big company. Due to his busy lifestyle, he often forgets to take something needed (such as his wallet or an important file) when he leaves home for work every day. Bob wants to find an application that can help him. He comes to the web-sharing center and inputs the keywords "reminder" and "home". Five relevant applications are listed and ordered by score ranked by their users. He reads the application description and selects the one (named AlertMan1.0) with the highest score that can meet his requirement; however, after checking the configuration of his home, the server tells him that this application cannot be installed in his house. He then chooses and installs another application called SmartReminder (developed by Kevin from a software company), which also has a high ranked score.

Before using the application, Bob has to configure it according to his preferences and the configuration of his house. There are two configuration items: the first one provides two possible ways to detect his leaving home status and asks the user to configure one of them; the second one asks him what he wants to be reminded of as well as the reminder style (such as sending an SMS message, playing a music clip, etc.) Bob tries to run the application after configuring it, but finds that it does not work well. He then seeks help on the discussion forum for the application and also sends an e-mail to the developer for possible reasons. After examining several received suggestions, he finds that the problem is caused by the object (such as his wallet) of which he wants to be reminded of. The wallet is placed under a book, which means the ultrasonic tag (U3D) it is equipped with cannot be detected by the ultrasonic location system installed in his house. Thus, the application cannot determine whether this object has already been taken by Bob. In response to a suggestion from one of his friends, he installs the UltrasonicMonitor application to automatically detect possible failures in the ultrasonic location system.

Having learned that objects equipped with ultrasonic tags cannot be placed in concealed places, he thinks this can help realize a sensor-augmented treasure game to play with his child. Several objects are hidden in the house, and once an object is found, a pre-defined multimedia action can be conducted by a smart device (like a projector) to transmit relevant information. He tries to create the game by himself. After checking the web-sharing center, he finds that there are two shared rules that can implicitly detect the hidden/ found status of an ultrasonic-tag equipped object. He imports
the rules and constructs the game. For sharing his creation with other users, he specifies the configuration items for the game through the configuration-interface-design widget, and then publishes it on the web-sharing center.

Building up a management system for ubicomp environments relies on many techniques. Here we mainly focus on ubicomp software management, and several key issues are derived from the scenario.

(A) **Software foraging and automatic installation support.** As found on the Yahoo Pipes website, one of the biggest problems with software sharing on social websites is information overload. There may be many applications created by different users with similar functions, yet different qualities. Therefore, it is important to find a reasonable mechanism to recommend the most reliable applications to users. An application under the ubicomp environment has a lot of particular requirements for its running environment and device capabilities. Due to the heterogeneous configuration of different families, not all user-selected applications can be installed. Users do not have the expertise for this, and we should support automatic checking of the local environment to determine whether an application can be installed. Moreover, in a ubicomp environment, an ubicomp application consumes contexts derived from different data sources (e.g., a reminder application fetches data from several smart objects and the user), but there are several smart devices (such as computers and smart phones) that can install the application. Therefore, the portability or mobility of the application becomes another challenge to tackle.

(B) **Software configuration management.** To meet diverse environments and user needs, in the meta-design approach, the shared application is designed in the form of a meta-version, which should be configured to adapt particular needs. As shown in the previous scenario, Bob has to select the object(s) that he should be reminded to take, and specify the prompting style, such as sending an SMS message, and so on. There should be an easy-to-use interface that can help the user quickly perform this activity.

(C) **Co-design and information sharing support.** End users sometimes want to design their own applications because the shared applications sometimes still cannot meet user needs or desires for creativity, like the treasure game. We should provide ways that can ease user effort as a system co-designer and allow them to share and reuse their creations and knowledge.

(D) **Error handling and fault tolerance.** An application sometimes does not work well after installation or during the long-term maintenance process. In these cases, we should help them diagnose and handle possible errors with it.

4. The iPlumber system

Based on the identified requirements, we developed the iPlumber system for ubicomp management. As shown in Fig. 1, it consists of several collaborating modules: physical
integration, local context manager, management toolkits and the web sharing center.

The physical integration module provides context abstraction to separate the low-level context-sensing implementation from context manipulation, which is achieved by introducing the wrappers. The wrappers are self-configuring components that can communicate with various sensors and actuators, such as abstracting raw contexts from them or adjusting the behaviors of actuators (e.g., commanding the smart projector to display a video clip) according to the decisions made by applications. Decoupling wrappers from the actual implementation of the sensors/actuators ensures openness and makes it possible to include newly available sensors (e.g., OSGi [26]).

The local context manager is a middleware layer that can separate context providing and context usage, which is based on our prior work [21,22]. Several components are integrated:

- The individualized ontology server maintains the individualized context ontology extended from the domain ontology shared through the web sharing module. The domain ontology, called SS-ONT, is built using OWL, the Semantic Web language. The SS-ONT domain ontology defines a set of basic concepts and relationships that exist in a typical ubicomp domain, such as humans and smart objects in the smart home domain. The individualized ontology contains both the basic information defined in a domain ontology and the particular context instances in the local environment, such as human profile and ownership of objects.
- The context aggregator gathers context make-ups from wrappers and inserts them into the individualized ontology server. The context query engine provides an abstract interface for applications to extract desired contexts from the ontology server. By executing inference rules, the context reasoner infers abstract, higher-level contexts from raw sensed contexts. The context subscription component allows an application to subscribe to some contexts it requires from the individualized ontology server, which enables automatic notification when such contexts are generated.
- Upon acquiring new contexts by subscription or querying, context-aware applications can trigger the related action in response to the current situation. The action can be made by adapting the behavior of the application, or using actuators to affect the behaviors of devices.

The management module spans a range of system management activities, from basic activities like software foraging, installation, and configuration, to advanced ones like software co-design and error handling. We present them in Section 5.

The web sharing center is a global layer where users can publish, share, and reuse resources such as meta-applications, inference rules and system maintenance experiences. On one hand, the components in this module provide common template building blocks for ubicomp management. On the other hand, the management toolkits generate new components (e.g., meta-applications, user experiences) for this module.

5. User-oriented ubicomp management

Here, we present how ubicomp environment management is handled by end users in iPlumber. In particular, we discuss how the identified requirements in Section 3 are implemented.

5.1. Software foraging and installation

In iPlumber, a lot of applications with similar functions but in different qualities are shared in the web sharing center. We make the following efforts to facilitate software foraging and installation.

Software foraging and recommendation. To deal with information overload problem in a social contribution environment, we provide a collaborative recommendation method. The recommendation is reflected through two kinds of survey data:

- **User rating**: iPlumber allows users to give a score to the application they experienced, and the rating information can help others make decisions. For example, when Bob searches on keywords “reminder, home” and related software from the management center page (left of Fig. 2), five applications are listed (right of Fig. 2). The search result is ordered according to user ratings (demo data is used).

---

![Home Explorer](image)

**Fig. 2.** Screenshots of the service management center page (left) and service browsing page (right).
Number of users of an application: An application seems to be more useful and reliable when it has more users. This information is also helpful for user selection.

Software installation support. As mentioned in Section 3, software installation in ubicomp environments faces problems like heterogeneous hardware/software configuration and alternative installation devices. To understand how iPlumber facilitates ubicomp software installation, we need to first learn the construction of a meta-application in our system.

To support context-based behavior change, a typical ubicomp application consists of a set of IF–THEN templates (or action-setting rules), within which the IF part consists of several contexts to depict a particular situation that can be derived by using inference rules (e.g., a rule to determine that a person leaves home) and the THEN part is specified by desired actions (e.g., sending him a reminder to take important items). The meta-version of the SmartReminder application mentioned in the scenario depicted in Section 3 is illustrated at the top of Fig. 3. It consists of two optional inference rules for detecting the “leaving home” situation and an action-setting rule to specify the action to be taken when detecting that the person leaves home without taking an important item. In the following, we explain how our system supports the two problems faced during ubicomp software installation.

Configuration requirement checking. We can see that the execution of a context-aware application depends on a number of contexts that can be detected by the deployment sensors in a smart home. Due to the heterogeneous nature of hardware configuration in smart homes, support for an application is not always found in a particular smart home. For example, the meta-application shown in Fig. 3 requires the ubicomp environment to install the indoor-location system to detect user and object locations. To facilitate automatic configuration requirement matching, all of the requirements in the form of abstracted contexts, such as ‘hasLocation’ context, will be specified in the application profile, which can then be used to match the individualized ontology specification of this house to see if it can be supported.

Application mobility. In an ubicomp environment, there are various computing devices, such as computers and smart phones, which can run context-aware applications. To support the mobility of an application from one device to another, we adopted the platform-independent programming language Java, and each device

Fig. 3. Configuration of the meta-application.

Fig. 4. Execution of an application on a smart device.
should install the Java runtime environment (JRE) and a famous Java rule engine Jess (http://www.jess-rules.com) to run the application. Fig. 4 illustrates the working process of an application on a smart device. When the application starts, it will send commands to the local context manager to load the inference rules of this application. In addition, it loads its action-setting rules into the Jess rule engine of the device on which it is installed. The context subscription component of the local context manager can notify the smart device when a context inferred from this application’s inference rules is generated, and then the relevant action-setting rule is triggered in response to the current situation.

5.2. Low-cost software configuration

The meta-application is an application template that should be configured by end users. To support user configuration of a meta-application, its developer can specify several parameters whose values can be altered by end users. To minimize the learning burden of end users, a user-friendly configuration interface is also required. In the following, we will describe how low-cost software configuration is achieved in iPlumber.

(1) Meta-application parameters. The parameters usually come from different parts of an application and thereby lead to distinct tailoring methods.

- **Individualizing a variable**: The developer can specify a variable of a rule (inference or action-setting) as a configurable parameter. End users are allowed to choose among alternative instances within the scope of this variable (determined by OWL axioms) and replace it with a preferred instance. In the example shown in Fig. 3, ‘y’ is a parameter that is replaced with the ‘BriefcaseA’ instance by Bob.
- **Specifying constant values**: The developer can also specify a constant used in a rule as a parameter. End users can reset its value within its scope. For example, in Fig. 3, ‘Entrance’ is an instance of class ‘Door’ and can be replaced by another instance ‘HouseDoor’, the name of the entrance door of Bob’s house.
- **Selecting a device for action output**: For action-setting rules, end users can specify a smart device (e.g., actuator) for action output when an action-setting rule is triggered. In Fig. 3, the user selects the house’s CDPlayer as the output device.
- **Specifying action type and contents**: After selecting an action output device, the relevant action types (e.g., play, speak, and display) supported by this device are listed and the user can select a preferred action type. At the same time, he can also specify the action contents, such as selecting a music clip or video clip to be played. As shown

![Image of configuration interfaces to the meta-application]

Fig. 5. Configuration interfaces to the meta-application.
in Fig. 3, Bob specifies an audio clip to be played by the CDPlayer.

(2) **Configuration interface design.** Although the configuration parameters in the meta-application are clearly specified, it is still hard for end users to read text-based rules and configure the parameters directly. In order to not place a high cognitive load on end users, we follow a simple preference-setting approach applied by many websites (e.g., the city setting in Yahoo Weather), where the parameterization tasks are mapped to simple word-based form-filling questions (see Fig. 5b). This kind of configuration interface is designed by application developers through the configuration-interface-design widget, as shown in Fig. 5a. Taking the **SmartReminder** application for example, the configuration interface consists of two form-filling questions:

- **Q1** is a multiple-choice question that corresponds to the two inference rules listed at the top-left of Fig. 3. This question allows the customizer to choose one way to detect his leaving home activity. Existing rules that were previously configured by the customizer for leaving home activity detection are also allowed to be chosen.
- **Q2** corresponds to the action-setting rule shown at the bottom-left of Fig. 3, which allows users to specify the output device and multimedia contents (see Fig. 5 b–d).

5.3. **Software co-design and information sharing**

In some cases, users want to design an application because either (1) the shared applications cannot meet their particular management requirements, or (2) the user wants to compose his experience, knowledge, and creativity in a new application. As a collaborative platform, our system facilitates user creation and knowledge sharing through the following activities:

- **Mashup activity.** The term “mashup” was originally used in Web development, which indicates a Web application that combines data or functionality from two or more sources to create new services. In iPlumber, this activity allows users to compose ubicomp applications by selecting and modifying the shared rules (via the web sharing module). This is a relatively low-complexity creation method.
- **Mashup plus activity.** As the name suggests, this activity is more powerful and thus more complex than the above one. It allows experienced users to create applications by not only using existing rules, but also creating new rules.
- **Resource publishing activity.** Newly created rules and applications can be published to the web sharing center and be shared by others. This facilitates knowledge and experience sharing and reuse among users.

The scenario described in Section 3 demonstrates how these activities enable users to be co-designers in ubicomp environments. In that scenario, Bob finds that the **SmartReminder** application he configured could not work well. After examining the possible reasons collected from peer users, Bob learns that ultrasonic tags cannot be placed in a hidden environment, which may make the relevant object an undetectable one (detailed discussion about this can be found in Section 5.4). Bob thinks that this capacity can help himself implement a sensor-augmented treasure game to play with his child. The traditional treasure game hides several objects (including a “treasure box” and other imagina-
tive roles, like “magic wand”, “fake treasure”, and “bomb”) in the physical environment and asks children to hunt the “treasure box” from them. With the support of smart objects in the house, Bob thinks that the game can be augmented by automatically detecting the “object found” event and presenting relevant multimedia information to the player through a presentation device like a projector.

As a new developer, Bob first searches the web sharing center to see if there already exist rules to detect the hidden/found status of an ultrasonic-tag equipped object. Although there are no rules that can directly report these situations, he finds two rules related to U3D: the **U3D_NotUpdated** rule; and the **U3D_Updated** rule. From the description of the two rules, he learns that they are designed to determine if the data from an ultrasonic tag can be received by the location system, which implicitly reflects the undetectable/detectable (i.e., hidden/found) status of the relevant object. He imports the two rules and constructs the game application via a mashup activity. The **U3D_Updated** rule is defined as an action-setting rule that allows users to specify the multimedia action (e.g., displaying a picture) to be executed when an object-played role (e.g., the treasure) is found. The game created by Bob can be published to the web sharing center for sharing with others. The two rules are formulated as (1) and (2) using the Jess rule language:

\[
(\text{SmartObject (name>?x)) (U3D (name>?y))(hasSensor ?x ?y) (hasUpdateTime ?y ?tcr : (intervalGreaterThan ?t Int : 2))} \\
\Rightarrow (assert (hasStatus ?x notU3DUpdated))
\]

\[
(\text{SmartObject (name>?x)) (U3D (name>?y))(hasSensor ?x ?y) (hasUpdateTime ?y ?tcr : (intervalLessThan ?t Int : 1))} \\
\Rightarrow (assert(hasStatus ?x U3DUpdated))
\]

5.4. **Failure detection and error handling**

One of the challenges relevant to ubiquitous computing is the problem of dealing with errors and failures during system development, installation and maintenance processes. In the following, we present our initial effort in iPlumber to deal with these issues.

(1) **Enhancing the quality of developed applications.** Similar to the development of traditional computer applications, ubicomp application developers should test the service they created and remove any errors in it before it is published. There are two ways provided for application evaluation:
• **Simulation mode.** This method allows the developer to quickly specify some simulation values (e.g., sensing inputs) and test whether a single rule or an application he created works. The simulation values are specified in the individualized context ontology through an ontology editor. Before testing a rule or an application, the developer should first configure it from a meta-version to a workable version. For example, to test the second Leaving Home rule defined in the SmartReminder application, the developer, Kevin, first configures the premise of the rule: *Kevin is approaching the entrance of his house with his briefcase.* To trigger the configured rule, the following three facts should be true: *Kevin is near his briefcase, the briefcase is moving, and Kevin is walking to the entrance.* The developer can simulate these facts by predefining the related sensor inputs, such as specifying the values of both Kevin and the briefcase’s ultrasonic tag to make them logically ‘near’ each other, or by simply defining the relevant property of an instance, such as defining the ‘status’ property of the briefcase to ‘move’. Afterwards, the developer can execute the rule in simulation mode to see whether it works properly.

• **Real sensing mode.** This is a real context-sensing mode that acquires inputs from the real world. It can help developers find out possible problems that may occur during the real-world deployment and practice, and take actions to avoid them in the development or help them generate a ‘help’ file that can assist users in dealing with possible errors during software installation and maintenance.

(2) **Collaborative error handling.** When installing in a particular ubicomp environment, an application sometimes cannot work well because of incorrect software or hardware configuration settings. As a collaborative environment, the user can seek help from the communication channels supported by the web sharing center. For example, in the scenario from Section 3, if Bob finds that the application he configured cannot send out a reminder when he leaves home, he can seek help on the discussion forum of the application he customized or directly send an e-mail to Kevin, the developer. The feedback shows that there might be several reasons for this:

(A) The location sensor tag equipped on the object is not turned on.

(B) The ultrasonic location system in the house does not work well.

(C) The object to be reminded is placed in a concealed place.

Bob is sure that the sensor tags are all turned on. There is also no problem with the location system in his house, because the indoor positioning system, another application based on the location system works correctly. He finally discovers that the problem is caused by the last suggested reason, because his wallet, the object he wants to be reminded, is placed under a book, which is a concealed place. When the ultrasonic tag is placed in a concealed place, the signal it transmits cannot be received by the location system and thus the action-setting rule (see Fig. 3) cannot be triggered.

(3) **Automatic failure detection.** During the long-term maintenance process, there might be various reasons, such as a sensor is broken or out of battery, sensor noise, or communication errors, that may cause failures in a ubicomp system. It is very convenient for users if the system can automatically detect failures when they occur. iPlumber has made an initial attempt at handling this, by seeking to use rules to automatically detect sensor failures. As described in the scenario, one friend suggests to Bob that he install the UltrasonicMonitor application to detect possible failures caused by the ultrasonic location system. The application explores potential failure objects in the house by leveraging the physical relations among objects. As shown on the left of Fig. 6, there are various physical relations among physical objects, and one example that can detect the failure caused by “concealing” is shown on the right of Fig. 6.

In this example, Book-C is concealed and cannot be detected by the location system. The following logic is used
for detecting this failure: By analyzing the physical relation between smart object Book-A and Table-B, we can infer that Book-A is not placed directly on the desktop because there is a height difference between them. A knowledge of physics indicates that there must be something between Book-A and the desktop, and thus Book-C is detected. Based on this analysis we can conclude two things about the “failed” object: first, that it is present; and second, its approximate location. In other words, rule-based reasoning can help us to automatically detect possible failures in ubicomp environments. More detailed information on this can be found in our prior work [21].

6. System implementation

We have built a prototype implementation of our system. It uses a series of hardware and software techniques, as described in the following sections.

6.1. A prototype ubicomp environment

Corresponding to the scenario depicted in Section 3, we created numerous prototypical smart objects by attaching sensors to everyday objects. As shown in Fig. 7a, two types of sensors were used: ultrasonic location tags (U3D) to localize objects; and MICA2 Mote sensors (http://www.xbow.com) to detect object status.

Two smart devices were used to present multimedia actions to players, as shown in Fig. 7b. The smart projector displays information in the form of animations or images on the wall that the player is currently facing, and the smart ultrasonic speaker generates related voices that make players feel that the voices are coming from the walls (please refer to [27] for technical details). A snapshot of the treasure game implementation is shown in Fig. 7c, where the player is looking for the object (top of Fig. 7c) and watching the multimedia action feedback displayed on the wall of the playing field (bottom of Fig. 7c).

6.2. Software infrastructure

We adopted Java 2 as the development platform. The Protégé-OWL editor is used for defining the domain ontology and for simulation testing as mentioned in Section 5.4. SPARQL (http://www.w3.org/TR/rdf-sparql-query), the Semantic Web query language, was used as the context querying language. Jess, a forward-chaining inference engine, is used to execute user-defined rules. The programming-level interaction between the OWL ontology (SS-ONT) and the inference engine is implemented through the Protégé-OWL API (http://protege.stanford.edu/). The context subscription function is implemented using Apache ActiveMQ (http://activemq.apache.org). The web interfaces of our system are implemented using Java Server Pages (JSP).

7. Evaluation and discussion

We conducted a series of experiments to evaluate the usability of our system. The performance of our ontology-based context infrastructure has been previously reported [21]. The main purpose here is to validate whether users can effectively exert control over the ubicomp environment through the management activities we provide.

We set out to evaluate the software foraging and configuration activity during the 2008 Keio Techno-Mall (an exhibition held in the Tokyo International Forum), because there are more people of different occupations and different ages, which offers us a better chance to gather feedback and suggestions from average users. During the one-day exhibition, more than 70 attendees visited our demo sessions, and 18 of them, ranging in age from 18 to 65, experienced the aforementioned management activities provided by our system. Besides evaluation during the exhibition, the software configuration activity was also tested in a controlled environment, where 15 students, ranging in age from 21 to 34, from Keio University were recruited by e-mail for the study. The recruited subjects

Fig. 7. Prototypical smart artifacts and action devices.
varied in gender, discipline (87% of the subjects were not computer science students, and one third of them are from non-technical majors like management, economics, and medicine) and programming ability (20% of them had programming knowledge, while two-thirds of them had none). The software co-design activity was also evaluated within the controlled environment, as there seems to be more experienced users in a university environment. We also tested the collaborative error-handling method by asking subjects to detect a sensor failure.

7.1. Software foraging

The software foraging activity supports users in finding the right application to meet their requirements. To validate the usability of our software foraging interface (see Fig. 2), the subjects during the exhibition were asked to find an application that can “remind me to take something that I forget to take when I leave home.” The subjects were allowed to input the keywords and write down the name of a selected application from the search results.

Different keywords were used by the subjects, including single words like remind and forget, multi-words like “remind, home”, and phrases like “leave home reminder”. There were three applications that could meet the requirement, and 14 out of 18 subjects chose the AlertMan1.0 application, which has a higher rank score and more users. Another four subjects chose the SmartReminder application. No one chose the AlertMan Demo application, the beta-version of AlertMan application whose rank score is relatively low compared to the two other applications. The questionnaire result on user feedback about the usability of the software foraging interface indicates that about 83% of them (see Fig. 8) believed that our system could effectively help them find the application in need.

7.2. Software configuration and co-design

In our system, user controllability and creativity in ubicomp environments is mainly reflected and attained through two management activities: software configuration and co-design. This section describes our experiments on validating them.

(1) Software configuration. As mentioned previously, this activity was evaluated in two different occasions: one was held during an exhibition, where 18 average users were asked to configure the simple SmartReminder application; the other was conducted in a controlled environment, where 15 university students (educated users) were asked to exert an in-depth configuration of the treasure game described in Section 5.3. Before the configuration sessions during the two evaluation occasions, we gave the subjects an initial impression of a ubicomp environment by first demonstrating the application to be configured by them.

During the test sessions in the exhibition, the subjects were asked to configure SmartReminder through its configuration interface. During the test sessions, we observed that most testers could perform this task with little or no guidance, although the time taken to accomplish this task varied from subject to subject (ranging from three to eight minutes). Five testers chose the first way to detect his leaving home activity, since the mobile phone they used could change networks from 3G to home Wi-Fi. More testers chose the second way, and various settings were made to “the object they usually take when leaving home” (including iPod, wallet, coat, etc.) and “the object to be reminded to take” (including watch, mobile phone, change pocket, etc.). As shown in Fig. 8, almost 78% of the subjects felt that our system provided an easy way for them to configure ubicomp applications.

During the test sessions in the university, we asked the subjects to firstly customize the shared treasure game. After that, they were asked to arrange the game environment according to their configuration. The configuration time ranged from 8 to 17 min. The number of object-based roles (i.e., the objects to be hidden) created by the subjects ranged from three to seven. In one of the subject-configured games shown in Table 1, four object-based roles were created. The subjects defined many interesting scenes. For

![Fig. 8. Evaluation results for software foraging, software configuration, and co-design.](image-url)
Table 1
One of the subject-configured treasure games.

<table>
<thead>
<tr>
<th>Event</th>
<th>Object (role)</th>
<th>Video setting</th>
<th>Audio setting</th>
<th>Hidden place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find an object</td>
<td>Room-key (Treasure-box-key)</td>
<td>FLASH</td>
<td>“Treasure key is found”</td>
<td>Under a book</td>
</tr>
<tr>
<td>Find an object</td>
<td>Japanese book (Guide)</td>
<td>JPEG</td>
<td>“A hint for you”</td>
<td>In a drawer</td>
</tr>
<tr>
<td>Find an object</td>
<td>Yellow-box (Treasure-box)</td>
<td>FLASH</td>
<td>“Treasure box is found”</td>
<td>On the shelf</td>
</tr>
<tr>
<td>Find an object</td>
<td>Snoopy-box (Bonus giver)</td>
<td>GIF</td>
<td>“The Bonus”</td>
<td>Under the chair</td>
</tr>
<tr>
<td>Game wins</td>
<td>Room-key, Yellow-box</td>
<td>FLASH</td>
<td>“Congratulations, you are a rich guy now!”</td>
<td>–</td>
</tr>
</tbody>
</table>

instance, the game example shown in Table 1 introduced an interesting ‘bonus giver’ role, which could give the player an added bonus when it was found. This result indicates that the meta-design approach enables the subjects to modify the behaviors of meta-applications in terms of their self-image. The data shown in Fig. 8 clearly reveals that most subjects (87%) found it easy to configure a ubi-comp application, which has a relatively high ranked score compared to the test result from the exhibition.

Despite high ratings, the experiments also revealed several issues. One subject in the exhibition pointed out that there was a high diversity of human habits and considerations, so it was too difficult to implement a customization system that could meet all requirements from users. It is true that since all of the configuration parameters of a shared application are defined by its developer, he may sometimes overlook releasing some important parameters to other users; however, our system is founded on the open collaborative concept. Thus, different users can communicate directly with each other to express their requirements or ideas utilizing web communication channels, which can assist application designers in improving their applications.

(2) Software co-design. In order to evaluate whether users can create new applications when existing applications cannot meet their needs, we tested the two end user creation activities mentioned in Section 5.3 with the fifteen subjects from Keio University.

The mashup activity is evaluated first. The subjects were asked to construct an application by selecting three rules from the web sharing center. This was a game application called UbiToy, where a smart object (e.g., a cup, a doll) could perform relevant actions according to different interaction behaviors from the player. Three interaction behaviors were defined, including (1) the player approaches the toy, (2) the toy is tilted, and (3) the toy is picked up, and the subjects were asked to choose three rules, from a rule repository, to detect the three events. All subjects except three (80%) successfully chose the right rules from fifteen rule candidates in a mean time of five minutes. This shows that it is not a very difficult task for them to perform. This observation is further proved by the questionnaire result shown in Fig. 8, where about 60% (33% easy and 27% medium) of the subjects considered application creation to not be difficult using this activity.

The mashup plus activity allows users to use shared rules as well as create new rules. It was evaluated by examining (1) whether subjects could understand sample rules, and (2) whether they could create a new rule.

(1) Understanding sample rules. Two typical rules R1 and R2, formulated as (3) and (4), were used for this test. We listed them on a questionnaire and asked the subjects whether they could understand them. In addition, to determine if they really understood the two rules, we asked them several relevant questions, for example, “What does ‘25’ mean in R1?” “What does ‘hasMotion (Briefcase, Move)’ mean in R2 and how does it happen?” The only knowledge about the Jess rule language we provided to the subjects is that “In the atom ‘Person (?x), ?x’ is a variable, which denotes an instance of the class ‘Person’, such as Bob, Jim, and so on.” The testing results are shown in Fig. 9. For rule R1, 73% of the subjects could totally or roughly understand it; and for the relatively complex rule R2, the percentage still reached 53%. The results reveal that a mean of around two-thirds subjects could have a general understanding of the sample rules, though they were only given limited knowledge about it. This conclusion is further validated by the subjects’ answers to the aforementioned questions. All subjects succeeded in finding out the relation between “temperature degree” and the constant ‘25’ in R1. For the questions related to R2, 80% subjects stated that the given atom means that the briefcase is moving and around half of them (47%) more or less identified its implicit meaning, that is, the briefcase is carried by a person.

\[
\text{City(?x) \land \text{hasTemperature}(?x, ?y) : (\geq ?y25)} \rightarrow \text{hasTemperatureDegree}(?x, \text{Hot})
\]
(2) Creating rules. The subjects were asked to build a rule that can remind those in the household who are going to leave home to take an umbrella if it might rain outside. This evaluation was designed in a form-filling style, and subjects had to choose four right atoms from 13 candidates to build the rule. A total of nine subjects (60%), including three who corrected their answer after a discussion with us, gave the right answer with little or no assistance. The questionnaire result shows that even designed with a relatively simple testing form, it is still difficult for many users (67%) to create new rules. Comparing with the result of “sample rule understanding”, the result here indicates that it was more difficult for the subjects to write rules than understand sample rules. Further discussion about the rule creation activity is made in Section 7.4.

7.3. Error handling and failure detection

Failure and error handling is an important but difficult problem for ubiquitous computing. We have also designed a set of experiments to test the effectiveness of the error-handling mechanisms presented in Section 5.4.

7.3.1. Collaborative error handling

To test this, we set up a failure situation in our experimental ubicomp environment and asked the subjects to find the error in it with the help of several suggestions obtained from a discussion forum. The failure situation was related to the ultrasonic positioning technique, in that an ultrasonic-tag equipped wallet was placed under a book, and thus could not be detected by the location system. Therefore, the wallet could not be listed in the search result of an indoor object-search application. Several possible reasons were listed on a webpage, and users had to examine each of them in our experimental environment to identify the right reason. A comparison of pre- and post-questionnaire results of the possible failure reasons...
is shown in Fig. 10, which reveals that collaborative error handling can help users to identify some failures during system maintenance.

7.3.2. Automatic failure detection

We have also designed experiments to test the rule-based failure detection mechanism. As presented in our prior work [21], experimental results reveal that the rule-based failure detection mechanism is sound, and can detect, to a certain extent, the failures caused by ultrasonic sensors. In the future, we will explore other error-handling techniques that can be implemented within iPlumber.

7.4. Discussion

One lesson learned from the evaluation of the mashup plus activity is that, though powerful, writing rules using a traditional rule language like Jess is somewhat difficult for end users. During the feedback session, some subjects suggested to us that we should introduce more graphical elements into the rule creation activity: "It is complex to write rules, a graphical-connection method is better." "Can I create rules using a wizard-like method." These comments indicate a possible way to improve our rule creation method. In future work, we intend to add a new activity, named graphical rule maker, which can make rules easier to write by using graphical abstractions.

iPlumber is an ongoing project. As a collaborative, open source platform, the rule repository of our system will be enlarged when it is put into practice. On one hand, hosting and third-party service providers, sensor makers, will contribute much to the fundamental sensing rules. Experienced users, on the other hand, will create many interesting applications and rule examples. Since the experimental results show that many subjects were better at understanding and using existing rules (see Section 7.2), we believe that the creation of a large rule repository will, from another direction, lessen the burden of end users on application co-design.

Finally, we want to make a summary of the meta-design approach used in iPlumber. As shown in Fig. 11, the meta-design approach makes iPlumber a continuously evolving socio-technical system by (1) providing different degrees of ubicomp design and management activities in terms of diverse user abilities, and (2) facilitating collaboration and knowledge sharing among developers and end users. In this paper we mainly focus on application-level user collaboration, in the next step we intend to exploit the collaboration at fine-grained granularities, such as context exchange among different users/homes and community activity mining using shared contexts. Some preliminary results have been reported in [28].

8. Conclusion

iPlumber represents our early efforts to build a cooperative management system for ubicomp environments. By exploring the meta-design method, users can cooperate in making the ubicomp software meet diverse installation environments and ever-changing user preferences and needs. Different management activities are provided, ranging from basic operations such as software foraging, intelligent software installation, and software configuration, to advanced ones like software co-design and error handling. The evaluation results from the preliminary user study reveal that the management activities we provide are appropriate for users to control their ubicomp environment. As the evaluation results reveal that it was difficult for the subjects to create rules using a rule language, we plan to develop a graphical rule maker that can facilitate rule creation in the future work. In addition to research on application-level knowledge sharing and reuse, we are also exploiting context exchange and community context mining in multi-domain ubicomp environments.

References

K. Ishii, Y. Yamamoto, M. Imai, K. Nakadai, A navigation system using
T. Gu, H.K. Pung, D.Q. Zhang, Toward an OSGi-based infrastructure
Y. Ye, Y. Yamamoto, K. Nakadai, A socio-technical framework for
K. Healey, A. Schussman, The ecology of open source software
K.R. Lakhani, E. Hippel, How open source software works: “free”
B. Guo, S. Satake, M. Imai, Home-explorer: ontology-based physical
S. Benford et al., Coping with uncertainty in a location-based game,
A.F. Blackwell, R. Hague, AutoHAN: an architecture for programming
K. Henricksen, J. Indulska, Developing context-aware pervasive
A.K. Dey, T. Sohn, S. Streng, J. Kodama, ICAP: interactive prototyping
H. Chen, T. Finin, A. Joshi, F. Perich, D. Chakraborty, L. Kagal,
K.N. Truong, E.M. Huang, G.D. Abowd, CAMP: a magnetic poetry
J. Rangarajan, J. Al-Muhtadi, R.H. Campbell, Reasoning about
S. Benford et al., Coping with uncertainty in a location-based game,
I. Guo, S. Satake, M. Imai, Home-explorer: ontology-based physical
X.H. Wang, D.Q. Zhang, J.S. Dong, C.Y. Chin, S. Hettiarachchi,
K.R. Lakhani, E. Hippel, How open source software works: “free”
K. Healey, A. Schussman, The ecology of open source software
Y. Ye, Y. Yamamoto, K. Nakadai, A socio-technical framework for
T. Gu, H.K. Pung, D.Q. Zhang, Toward an OSGi-based infrastructure
K. Ishii, Y. Yamamoto, M. Imai, K. Nakadai, A navigation system using
B. Guo, L. Sun, D.Q. Zhang, The architecture design of a cross-domain

Bin Guo is currently a post-doctorate researcher at the Institute Telecom & Management SudParis, France. His research interests include wireless sensor networks (WSNs), context-aware middleware, pervasive gaming, Semantic Web and end-user programming in smart environments. He received his Ph.D. degree in computer science from Keio University, Tokyo, Japan, in 2009. He has worked in several projects in the field of ambient intelligence, including EU’s Feel@Home CELTIC Project in 2009 and Japan’s 21st Century COE Program “System Design: Paradigm Shift from Intelligence to Life” during 2006–2007. He is a member of IEEE.

Daqing Zhang is a professor on Ambient Intelligence and Pervasive System Design at Institute TELECOM SudParis, France. He initiated and led the research in Smart Home, Healthcare/Elderly care and context-aware computing from 2000 to 2007 at the Institute for Infocomm Research (I2R), Singapore. He was the Program Chair of the First International Conference of Smart Home and Health Telematics (ICOST2003) in Paris, France and the Sixth International Conf. on Ubiquitous Intelligence and Computing (UIC 2009) in Brisbane, Australia. He is the associate editor of ACM Transactions on Intelligent Systems and Technology, Journal of Ambient Intelligence and Humanized Computing (Springer), etc. He also served in the technical committee for conferences such as Ubicomp, Pervasive, PerCom, etc. Dr. Zhang’s research interests include pervasive elderly care, service-oriented computing, sensor based activity recognition, context-aware systems etc. He has published more than 100 papers in referred journals, conferences and books. He obtained his Ph.D. from University of Rome “La Sapienza” and University of L’Aquila, Italy in 1996.

Michita Imai is Associate Professor of Faculty of science and technology at Keio university, a researcher at ATR Intelligent Robot Laboratories, and a visiting researcher at the University of Chicago. He received his Ph.D. degree in Computer Science from Keio Univ. in 2002. In 1994, he joined NTT Human Interface Laboratories. He joined the ATR Media Integration & Communications Research Laboratories in 1997. His research interests include autonomous robots, human-robot interaction, speech dialogue systems, humanoid, and spontaneous behaviors. He is a member of Information and Communication Engineers Japan (IEICE-J), the Information Processing Society of Japan, the Japanese Cognitive Science Society, the Japanese Society for Artificial Intelligence, Human Interface Society, IEEE, and ACM.