Pers Ubiquit Comput (2011) 15:221–233
DOI 10.1007/s00779-010-0329-1

ORIGINAL ARTICLE

Toward a cooperative programming framework for context-aware applications

Bin Guo · Daqing Zhang · Michita Imai

Received: 17 November 2009 / Accepted: 22 April 2010 / Published online: 6 August 2010

Abstract OPEN is an ontology-based programming framework for rapid prototyping, sharing, and personalization of context-aware applications. Unlike previous systems that provide programming support for single group of users, OPEN provides different programming support for users with diverse technical skills. According to the programming requirements of different users, several cooperation patterns are identified, and the mechanisms to facilitate resource sharing and reuse are built into the framework. Three corresponding programming modes are elaborated by showing how a context-aware game has been developed with the support of the OPEN framework, and the usability of our system is validated through an initial user study.

Keywords Context-aware computing · Cooperation · End user programming · Semantic web

1 Introduction

With the advent of sensor networks and the prevalence of networked embedded devices, we are moving much closer to realizing the vision of context-aware spaces. As we create context-aware systems, we must think ahead to consider how they can be programmed and tailored by users. Here, the term "users" refers to both professional *developers* and non-expert *end users*. There have been recently various context-aware applications and prototypes developed by researchers to demonstrate the benefits of this new paradigm. However, the dynamic nature of ubicomp environments and the diverse, changing requirements from end users often make these "hard-programmed" applications inadequate in terms of flexibility.

Enhancing end user participation in the initial design of context-aware systems can somehow relieve this problem. However, given the fact that end users differ greatly in terms of their environments and expectations about how their applications should behave under different situations, and it is hard to precisely identify all these possibilities beforehand, the traditional "design-before-use" method is not sufficient to program evolving context-aware spaces. Instead, it is more reasonable to empower end users to build and adapt systems that fit their individual needs. This insight has fostered a new research topic called end user programming (EUP), whose main goal is to provide a set of methods and toolkits that allow non-expert users to create or modify a software system [1].

A number of toolkits have been developed to allow programming of context-aware applications by either developers or end users (see related work in Sect. 2). These systems always provide certain kind of programming abstraction, such as context processing APIs for developers, and graphical or tangible interfaces for end users, to ease the development process.

However, there are still several challenges that need to be addressed. First, existing systems do not take the diversity of user skills and interests into account. From a user-centered design perspective, they each provide single

B. Guo (✉) · D. Zhang
Telecommunication Network and Services Department,
Institut TELECOM SudParis, 9, Rue Charles Fourier,
91011 Evry Cedex, France
e-mail: bin.guo@it-sudparis.eu; bingo@ayu.ics.keio.ac.jp

D. Zhang
e-mail: daqing.zhang@it-sudparis.eu

B. Guo · M. Imai
Keio University, Hiyoshi 3-14-1, Kohoku-Ku,
Yokohama 223-0061, Japan
e-mail: michita@ayu.ics.keio.ac.jp

programming support that fits only a certain group of users (e.g., professional or non-expert level), failing to support other category of users. Second, existing systems rarely consider the cooperation among users. For example, the communication gap between developers and end users can lead to a situation where a professional application created by a developer cannot be easily tailored by an end user according to his particular needs, and on the other hand, the feedback from end users cannot be easily communicated to application developers.

To address these issues, we have developed the OPEN framework, with an aim at allowing a broad category of users' participation and cooperation in the development of context-aware applications. Unlike previous studies, OPEN follows the "*gentle slope*" principle [1] by designing a set of programming modes with diverse complexity, which enables different users to develop and tailor more context-aware applications. Taking advantage of the Semantic Web technologies, people utilizing different programming modes can cooperate with each other by publishing, reusing, and tailoring shared resources (e.g., application templates, rules, etc.), which substantially enhances the cooperation among users and speeds up the development process. The effectiveness of our system is evaluated through a user study, with users testing different programming modes.

2 Related work

Over the last decade, a lot of researchers have been working on programming support for context-aware systems. Much attention has been paid to providing support to professional developers. The pioneering work of Context Toolkit presents an object-oriented architecture for rapid prototyping of context-aware applications [2], including a set of programming abstractions that separate context acquisition from context consumption. Similar to Context Toolkit, the Solar platform also uses attribute-value pairs to represent contexts [3]. It facilitates developers' work by providing a series of reusable 'planets' for context collection and a subscription mechanism for context consumption. The two systems all used informal method to represent contexts, which cannot provide good support on knowledge sharing and reuse. Therefore, most developers have to start from scratch when building new context-aware applications. Recent studies have explored the Semantic Web technologies to address this issue [4–8]. In the CoBrA project [4], Chen et al. leverages the Web Ontology Language OWL to define ontologies that can be shared and reused by developers in the smart meeting room domain. Wang et al.'s Semantic Space infrastructure exploits the Semantic Web-related technologies to support explicit representation, expressive querying, and flexible reasoning of contexts in smart spaces [5]. Gu et al.'s SOCAM [6] is an ontology-based, service-oriented middleware that supports rapid prototyping of context-aware services. Yu et al.'s CMM context-aware architecture [7] aims at supporting context-based content filtering and recommendation in the development of multimedia applications. Our earlier work [8] provides a rule-based infrastructure that allows developers to deal with possible sensor failures in context-aware systems by writing heuristic rules. All these systems intend to provide programming-level abstractions for developers, rather than targeting non-expert users developing context-aware applications.

In recent years, a few researchers began to work on enhancing end user programming to ease the development of context-aware applications, and they explore different ways to achieve this. Some systems attempt to provide alternative input mechanisms (in most cases, a visual programming environment) to avoid coding, as reported by iCAP [9]. The UbiPlay project utilizes a "finite-state-machine" metaphor to enable non-expert users to program a smart playground [10]. Physical or tangible programming methods are also exploited. The two projects, StoryRoom [11] and AutoHan [12], empower end users to construct programs by manipulating physical objects. A detailed survey on existing high-level abstraction programming toolkits for context-aware applications can be found in Tang et al. [13]. While these are very imaginative methods, they are only designed for certain kind of end users. For example, some systems, like [11, 12], are easy to use but their functionality is limited; other systems, like [9, 10], support relatively complex application designs but they may place a significant cognitive load upon non-expert users (most of them are based on the structured programming metaphor which is mainly familiar to programmers).

As previous systems are designed to suit the needs of a certain group of users (e.g., professional or non-expert level), the sharing and reuse in resources, applications and user experiences, are almost neglected, though it has been investigated in [9] that there is quite big overlapping among users' needs over context-aware applications.

Our system differs from and perhaps outperforms previous work in the following respects. To meet both the "functionality" requirement from developers and the "simplicity" requirement from end users, multiple programming supports are provided, ranging from *program tailoring* to *program creation*. Furthermore, we identify several cooperation patterns among users and embed them into our programming framework, which greatly facilitates the programming process as well as the creation of more flexible context-aware applications.

3 Requirements and solution guidelines

A typical context-aware application consists of a set of IF–THEN templates (e.g., if a person enters a room, then turning on the light of this room), 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 enters a room) and the THEN part is specified by desired *actions* (e.g., turning on a light).

Traditional context-aware programming frameworks usually provide a programming toolkit for users to define inference rules (we call it the *inference-rule part*) and specify actions in response to context changes (we call it the *action setting part*) at a certain abstraction level. The term 'action setting' used here has a broad meaning in terms of adapting system behavior to context changes, which includes specifying device behaviors (e.g., turning on a light) and configuring system parameters (e.g., QoS settings). Our work mainly supports device behavior specification in its current implementation. To support the programming toolkit's work, two modules are often associated: a context manager for representing and processing contexts, and a context aggregator that collects contexts from heterogeneous sensors. The design goal of our framework is to support a broad category of users' participation and cooperation in the development of context-aware applications. To build such a framework, many new challenges are specified. We focus here on three key issues.

(1) *A unified, evolving context model to be shared among context-aware applications*. Context representation provides the foundation for sharing, reuse, interoperation among context-aware applications. As a collaborative environment, users must agree on a shared conceptualization of a particular domain (e.g., smart homes, smart offices), which provides a unified way for context representation and an unambiguous way for user cooperation. Emerging ontology standards, such as RDF and OWL, have been previously explored to explicitly represent contexts [4–8]. As those systems are designed for a small group of users, the ontology used can simply be extended to meet the evolving context-aware environment (e.g., a new kind of sensor is equipped). However, this may result in context inconsistency under a collaborative environment, given that users from different groups may extend the domain ontology in an arbitrary way. Therefore, a unified, consistent management mechanism should be used to control evolvable context ontology.

(2) *Providing multiple programming modes with different complexity*. Users differ from each other on their technical abilities. Professional developers are able to exert control over all components of a context-aware environment (e.g., adding new sensors, extending domain ontology). Some end users have certain technical skills and they are willing to create applications using a EUP toolkit. Most other end users have little or no technical skills and they may lose interest in maintaining a context-aware system if the operations are complex. Therefore, there are broadly three different technical levels—high, middle and low—which makes it hard to meet their different requirements by using a single programming method. Instead, we should provide appropriate support to users with diverse skills and interests.

(3) *Supporting cooperation among users*. Despite users are diverse in their technical abilities, there is big overlapping among their needs on context-aware services, which leads to the requirement on user cooperation and sharing. Users can cooperate with each other in various styles (e.g., individual-based or group-based, synchronously or asynchronously, etc.). Here, we concentrate our discussion on potential cooperation patterns raised by diverse technical background of individual users. Indeed, we exploit the following three cooperation patterns.

- *Cooperation between developers and end users*: The diverse technical ability of these two user groups leads to an interesting bi-directional cooperation pattern. On one hand, a developer can develop a "template" application and publish it as a shared one. An end user can simply configure some parameters to the inference-rule part of this application and specify preferred actions in its action setting part. For actions, we mean physical actions (e.g., turning on a light) or virtual/multimedia actions (e.g., playing a music clip, displaying an image). Users can share multimedia resources to enrich their action setting operation. On the other hand, when end users find some problems or new requirements in using the shared application, they should be able to send their feedback to developers, which results in the requirement for a communication channel. In summary, from this cooperation pattern, three types of sharable resources can be derived, they are, *application templates*, *multimedia action resources*, and *a communication channel*.
- *Cooperation among developers*: When creating applications, developers define various rules to detect different situations. Since an individual situation can involve different applications (e.g., a no-person-in-house situation should both be recognized by an energy-saving application and a

safety-monitoring application), the associated inference rules can be reused by different developers. This leads to another type of sharable resource—*inference rules*.

- *Cooperation among end users*: If an end user tailors an interesting application, he may expect to share his finding or experience with his friends or colleagues. Therefore, there should be a way to facilitate this process.

4 The OPEN cooperative programming framework

Based on the identified requirements, we developed the OPEN cooperative programming framework. As shown in Fig. 1, it consists of several collaborating modules: *the context providers*, *the context manager*, *programming toolkits*, and *the resource-sharing server*.

- *Context providers* provide context abstraction to separate 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, abstracting raw contexts or adjusting the behaviors of actuators according to decisions made by applications. Decoupling wrappers from actual implementation of sensors/actuators ensures the openness and makes it possible to include new sensors available (e.g., OSGi [7, 14]).
- *The context manager* is a middleware layer, and it can separate context processing from context usage. It integrates several components. The context aggregator gathers context makeups from wrappers and asserts them into the individualized ontology server. The individualized ontology server maintains the individualized context ontology extended from the domain ontology shared through the resource-sharing module (depicted in Sect. 4.1). 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 programming toolkits* provide support for users to program context-aware applications. Three different user technical levels, *high*, *middle* and *low*, are specified in last section, and we provide three programming modes accordingly: *incremental mode, composition mode, and parameterization mode*. While the incremental mode and the composition mode allow developers (high and middle level) to program new applications, those applications can be transformed into templates that can be used in end user (low-level) programming. With the aid of the *configuration interface design widget*, end

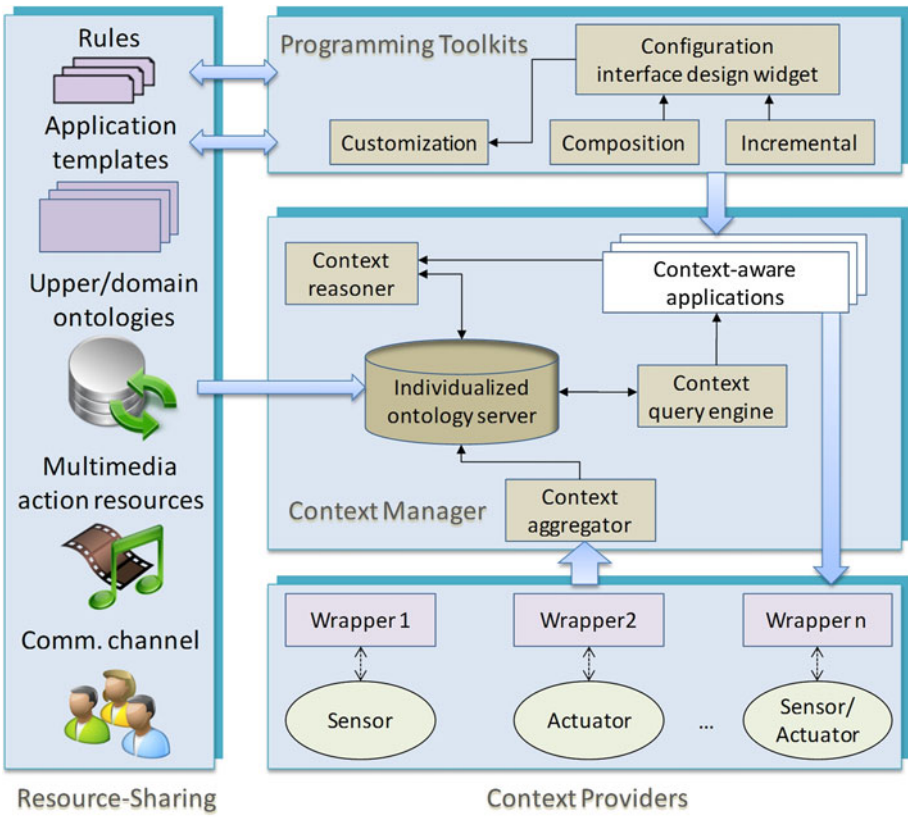

Fig. 1 The OPEN framework

users (low-level) are empowered to customize the templates created by the two higher-level modes through the parameterization mode. More details about each programming mode will be discussed in Sect. 4.2.

- *The resource-sharing module* enables users to share and reuse resources such as application templates, rules and multimedia files. On one hand, the components in this module provide common building blocks on templates for the three programming modes. On the other hand, the programming toolkits generate new components (e.g., rules, application templates) for this module. The cooperative interaction process between these two modules will be explained in Sect. 4.3.

4.1 Evolvable context model

As a collaborative programming environment, users must agree on a shared conceptualization of a domain. For the OPEN framework, we adopted the ontology-based method for context modeling. The ontology, called *SS-ONT*, is built using the Semantic Web language—OWL.

4.1.1 Hierarchical definition

As illustrated in Fig. 2, the *SS-ONT* ontology is defined using a hierarchical approach. The higher two levels of ontology, *upper ontology* and *domain ontology*, follow the general ontology-design principle proposed in [5, 14]. While the upper ontology defines the high-level concepts that are common among different context-aware spaces (e.g., homes, offices), the domain ontology is an extension of upper ontology, defining the details of general concepts and their properties for a particular domain (More detailed information about the defined *SS-ONT* ontology concepts can be found in our prior work [8]) The third level of ontology, named the *individualized ontology*, contains both the information defined in a domain ontology (e.g., a smart home) and the context instances. The context instances contain:

- User-defined context instances: some contexts, such as user profile, relationship among users, ownership of objects, seldom change in an individual space, and the information is often supplied by end users.
- Sensed context instances: referring to the dynamic context instances acquired from various context providers (e.g., smart artifact status, user location, etc.), which are collected at runtime.

4.1.2 Ontology evolution

In OPEN, domain ontologies are designed and maintained by a community of domain experts (e.g., service providers and sensor makers) and shared through the resource-sharing module, while each individualized ontology is maintained by the ontology server of an individual smart space. Since the evolution nature of the context-aware field, the context ontology often needs to be extended to meet new requirements. To ensure concept-level's consistency among different-level's ontologies, the following ontology-updating principles are deployed in our framework:

- The domain ontology can only be extended by domain experts;
- Users are strictly restricted to specify the context instances related;
- Once the domain ontology is extended, an "update" command will be automatically sent to individualized ontology servers to update their ontology.

4.2 Three programming modes

In order to support users with different technical skills, the OPEN framework provides three different programming modes. In this section, we explain the difference and relationship among them, as well as the user interface implementation in OPEN.

4.2.1 Programming modes

The three programming modes supported by OPEN differ from each other in various aspects, as summarized in Table 1.

- *Incremental mode* is a programming mode for high-level users, which supports the creation of new context-aware applications. The developers can construct the inference-rule part of an application by preselecting existing rules, modifying existing rules and writing new rules. All rules are represented using a formal rule language—SWRL (http://www.w3.org/Submission/SWRL/). For the action setting part, developers are allowed to specify default action settings (physical or multimedia actions) for the application. A sample context-aware application (called Smart Agent) is shown in the upper part of Fig. 3.
- *Composition mode* is a programming mode for middle-level users. Similar to the first mode, it also empowers developers to create new applications. However, to reduce complexity, this mode only allows them to select and modify the rules that other developers have already created. Developers of this mode can also specify default action settings just as the incremental mode allows.
- *Parameterization mode* is a programming mode for low-level users. In this mode, users can customize an existing application by setting the parameters

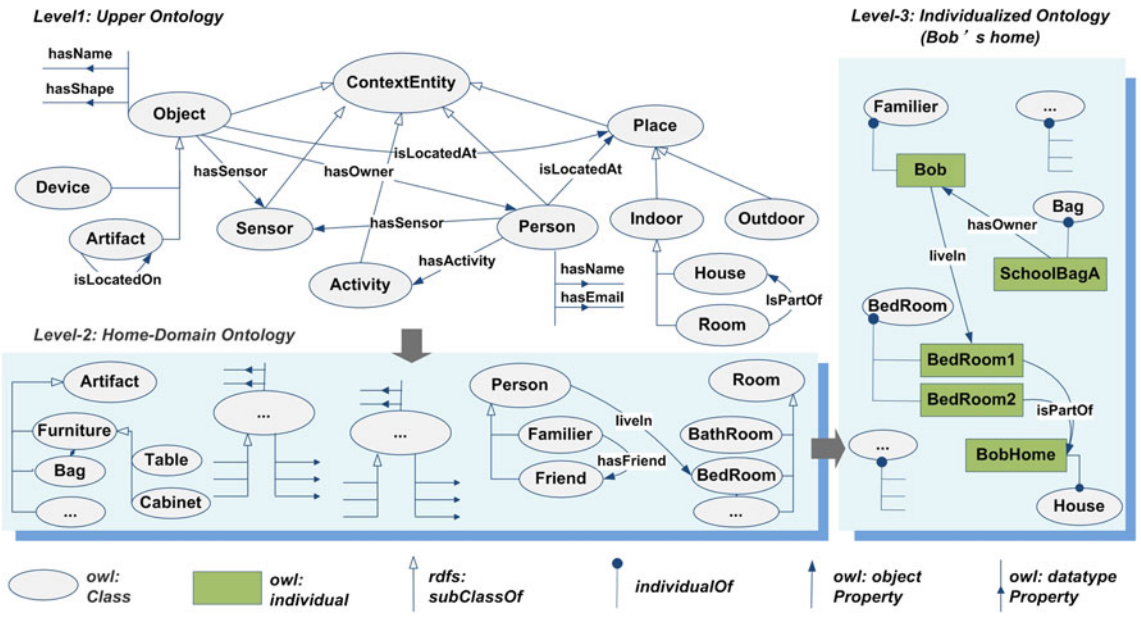

Fig. 2 SS-ONT ontology design: three levels

Table 1 Support for different programming modes

Support	Parameterization (customization)	Composition	Incremental programming
Creating new rules	–	–	√
Constructing new applications	–	√	√
Configuring applications	√	√	√
Power of expression	Low (simple words)	Middle (reading rules)	High (writing/modifying rules)

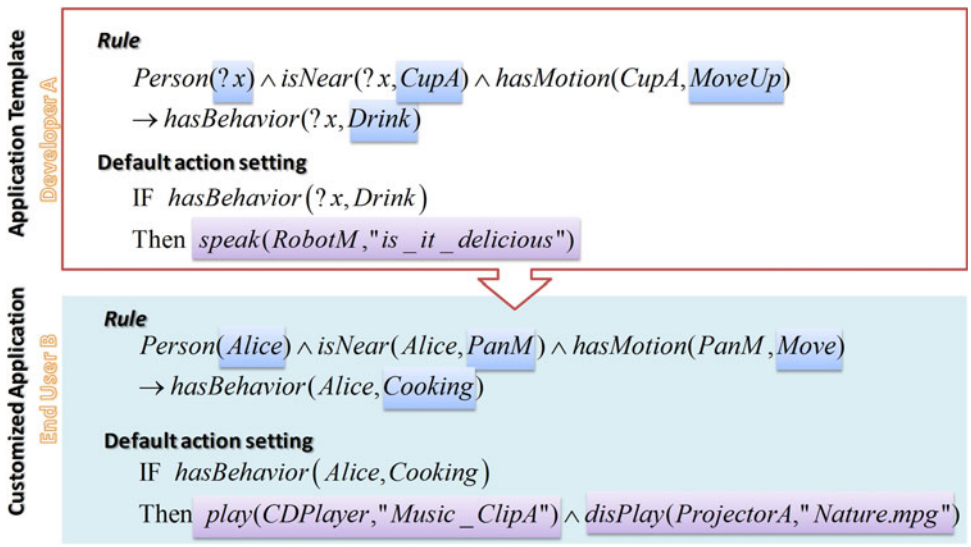

Fig. 3 The application customization mechanism

predefined by its developer, and thus they can tailor the behavior of the application according to their particular settings. The parameterization mechanism is described in the next subsection.

4.2.2 Relationship among the three modes

Both incremental mode and composition mode can create new applications. The parameters of applications are then

specified by the developers and can be configured by end users through the parameterization mode. The parameters usually come from different parts of an application and thereby lead to distinct parameterization methods.

- *Individualizing a variable*: The developer can specify a variable of a rule 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 by a preferred instance. For example, in the example shown in Fig. 3, '*?x*' is selected as a parameter and user-B replaces it by the '*Alice*' person instance.
- *Specifying constant values*: Developers can also specify a constant used in a rule as a parameter. End users are allowed to configure a constant-based parameter by resetting its value within its scope. For example, in Fig. 3, '*CupA*' is an instance of class '*Object*' and an end user can replace it by another instance ('*PanM*') of this class.
- *Specifying actions*: All default action settings designed by developers can be reconfigured by end users. As shown in Fig. 3, after configuring the constant value from '*Drink*' to '*Cooking*' in the rule, the user action setting can be changed from speaking a sentence to playing a combination of audio and video clip.

4.2.3 User interface

Having described the three programming modes, in this subsection, we present how OPEN's user interface supports these programming modes.

The interfaces for the incremental and composition programming modes are shown in Fig. 4. To create an application by using the incremental mode, the developer should first create a new application via the main interface page (shown in Fig. 4a). He can then start constructing the inference-rule part of the new application. This can be done by preselecting existing rules and writing new rules. Taking the sample application illustrated in Fig. 3 as an example, to develop the inference-rule part of it, the developer firstly selects rules with the keyword 'Behavior', as demonstrated in Fig. 4b. After checking existing rules, he finds out that existing ones cannot meet the application's requirement but give him good references to create the required rule. The new rule can be created through the rule-creation page, as shown in Fig. 4c. When creating a new rule, the developer can meanwhile specify the parameters that can be configured by end users, as well as the default action settings. Finally, to allow end users to simply configure the parameters, the developer should make a "simple-word"-based configuration front-end setting for end users, as illustrated in the configuration-interface-design page (see Fig. 4d). At this step, each configurable rule will be given a configuration introduction that guides other users' configuration. For example, the following introduction for the inference rule in Fig. 3 is given: "*The application can detect your behavior according to your interaction with a smart object in your house. For example, if a cup is in your hand and the cup is moving up, it informs that you are drinking. Please specify the following parameters according to your daily behaviors*". Besides, each parameter (e.g., '?x' in Fig. 3) in the rule will be assigned a so-called 'nickname' (e.g., "Your name" for '?x') that can be easily understood by end users.

Programming through the composition mode is similar to the incremental mode. However, in the composition mode, the developer does not need to create new rules through the rule-creation page (see Fig. 4c).

The *parameterization mode* implements all the three parameterization methods mentioned in Sect. 4.2.2. In order not to place high cognitive load to end users, it follows a simple preference-setting approach applied by many Web sites (e.g., city setting in Yahoo Weather), where the parameterization tasks are mapped to simple-word-based form-filling questions (see Fig. 5a). This kind of configuration interface is designed by application developers through the configuration-interface-design page (see Fig. 4d). In Fig. 5a, the sample example shown in Fig. 3 is used to illustrate how a "code-based" application is configured by end users through the form-based configuration interface. The configuration tasks such as parameter settings and action settings (e.g., multimedia contents) can all be performed through this configuration interface.

Providing three programming modes with different complexity ensures that a user can find the right programming tools according to his skill set and can easily move up from simple function/programming support to more complicated ones with a moderate increase in complexity. In other words, it builds a "gentle slope" for users to scale up from low-level programming activities to higher-level ones, and vice versa.

4.3 Collaborative programming patterns

Having described the three programming modes, we discuss here different cooperation patterns across the three programming modes.

4.3.1 Pattern-1: cooperation between developers and end users

According to our analysis previously, this bi-directional cooperation pattern is achieved through sharing and reusing three resource types via the resource-sharing module.

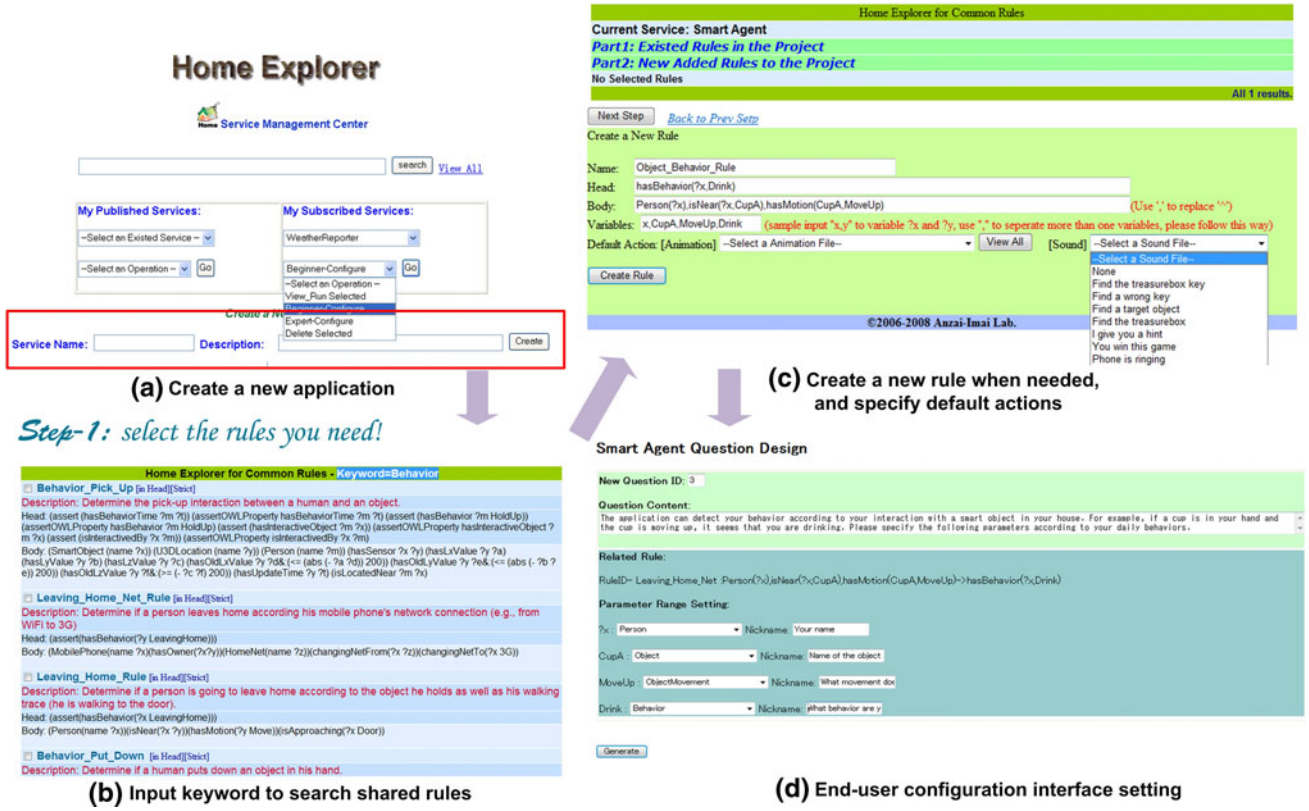

Fig. 4 Screenshots of the main-interface (resource-search) page (**a**) and the pages for incremental/composition programming mode (**b–d**)

- *Applications*: Through the parameterization mode, application templates created by developers can be used by end users (as depicted in Sect. 4.2.3).
- *Multimedia resources*: Sharing multimedia contents can greatly enrich the resources for action setting. Users can on one hand publish the multimedia resources they generated (flash movies, images, video/audio clips) in the resource-sharing module, and on the other hand, they can import the resources created by other users (as shown in Fig. 5b, c).
- *Communication channel*: Feedback information from end users can help developers to improve the development of context-aware applications. In OPEN, users can choose to communicate with each other by sending instant messages, sending emails or presenting problems in a Web-discussion forum. For example, through the Smart Agent configuration interface (shown in Fig. 5a), users can find the developer's contact information (an email address) from the "application description" part. If any questions are found, they can directly contact the developer for help. This bridges the communication gap between developers and users.

4.3.2 *Pattern-2: cooperation among developers*

The cooperation among developers is achieved in the form of reusing the rules they created. This pattern is reflected both in the composition mode and incremental mode. New created rules via the incremental mode will also be published to the resource-sharing module. As illustrated in Fig. 4, developers can search among the shared rules through the rule-browsing page (Fig. 4b). The new rule created in Fig. 4c will be added into the rule repository for sharing.

4.3.3 *Pattern-3: cooperation among end users*

When an end user finds an interesting context-aware application, he can share his experience with other end users (e.g., his friends) by the following two ways. First, the end user can simply recommend this application to others through the communication channel. For example, he can send an email to his friend to recommend an application. Second, he can authorize others to share and modify the application he has customized. As shown in Fig. 5a, the configuration interface allows a customizer to

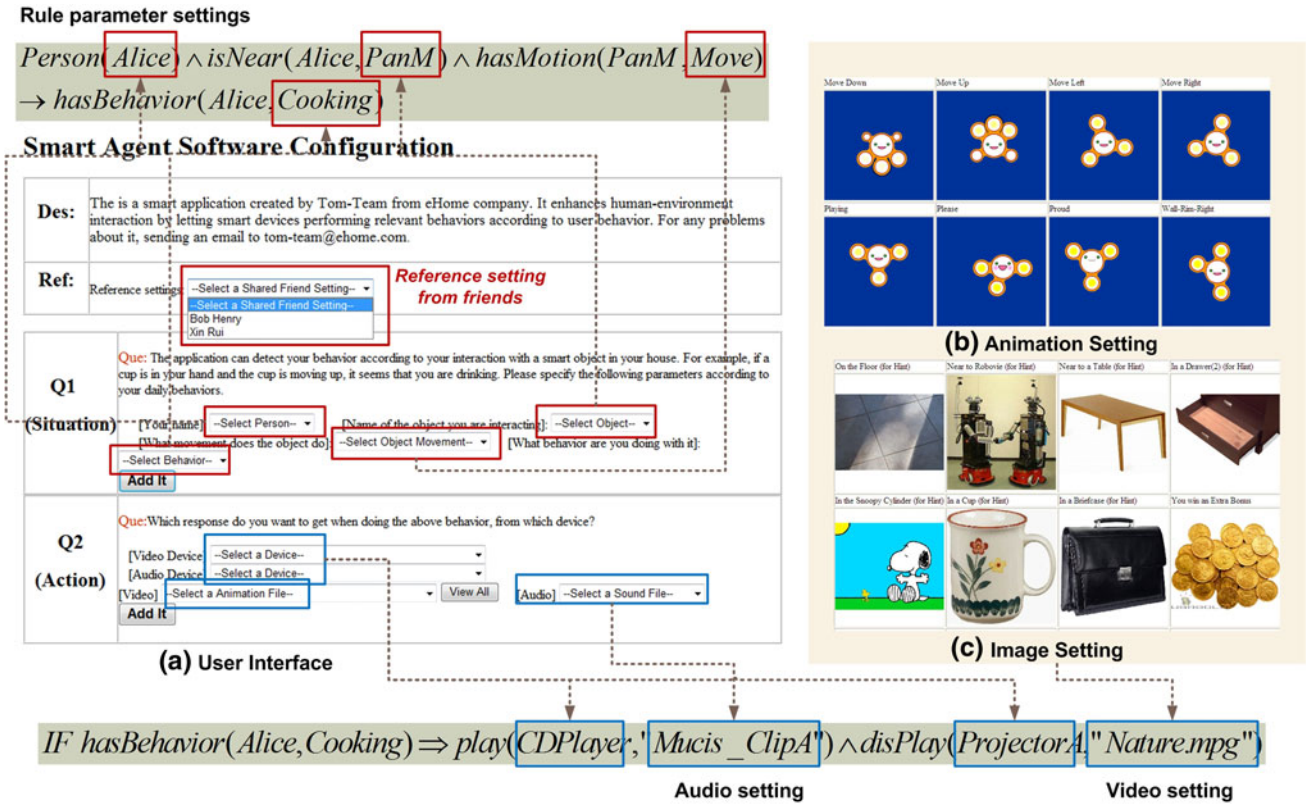

Fig. 5 Screenshots of the configuration front-end for the parameterization mode (**a**), and the animation and image setting pages (**b**, **c**)

view reference settings shared by his friends. One reference setting can be displayed in the form of an html that summarizes the relevant parameters and contents specified by the sharer.

5 The treasure-game scenario

We demonstrate how to provide different programming support with OPEN by developing a context-aware game. Similar to the previous entertainment systems [10, 11] that developed for smart homes, the game exploits a few smart artifacts in a smart home as interactive game props (e.g., a box can be used to act as a treasure-box), and enables people to play it by physically interacting with these objects. Several smart devices are used to present multimedia actions in response to the interactions between players and objects. This game, called Treasure, is described as follows:

> There are various smart artifacts in a smart home. A game designer can choose some of them to be hided. The selected objects are specified by the designer to act as different roles in the game. He should firstly select a box and a key to act as the roles of "treasure box" and "treasure-box-key". If the player finds both the two objects, he will win this game. Besides the two fixed roles, the designer can select a few other objects to act as other imagined roles. For example, a cup can be used to act as a "guide" that hints the player about the hidden place of the "treasure-box"; a drawer can be used to act as the "shelter of a monster" and when it is opened, a monster residing in it appears on the wall and shouts "Don't disturb me, I am now sleeping".

5.1 Programming the application

Programming of the context-aware game "Treasure" contains three distinct phases: *ontology individualization*, *application development*, and *application customization*.

Before developing Treasure, the context ontology needs to be first agreed by all developers and end users. To include new concepts used in game applications, such as status of smart artifacts (e.g., hidden/found status) and games (e.g., win/lose status), we extend the *SS-ONT* home-domain ontology by adding some new terms (the whole definition is available at "http://www.ayu.ics.keio.ac.jp/members/bingo/SS-ONT-v1.5.owl"). As there are different smart artifacts deployed in different homes, in the *ontology individualization* phase, users should insert these instances into the extended ontology. The individualized ontology will be maintained in the individualized ontology server.

In the *application development* phase, the developers identify and construct the context-aware game functionalities by using the composition or incremental programming modes. First, the situations to be detected are examined. In Treasure, two particular situations are identified: (1) the situation that a hidden object is found should be detected, and (2) the condition that a player wins the game should be defined. The developers then check whether the rules to detect these situations have been defined by others (via the main interface page shown Fig. 4a). Though there are no rules that can directly report the hidden status of smart artifacts, two rules that can determine the update status of the location sensors equipped on the smart artifacts are found. From the descriptions on the rule-browsing page (see Fig. 4b), the developers learn that this kind of sensor does not update data when it is not exposed to its readers, which can be indirectly used to detect hidden objects. These rules are imported into the new application (see upper part of Fig. 6) and a new rule is created to describe an object-found situation by using the conclusions from the two imported rules:

$$\begin{aligned}&\text{Object}(?x) \wedge \text{U3D}(?y) \wedge \text{hasSensor}(?x, ?y)\\&\quad \wedge \text{status}(?y, \text{Updated}) \rightarrow \text{status}(?x, \text{Found})\end{aligned}$$

As the second identified situation is an application-specific situation, the developers cannot find a related rule to detect it and he creates the following rule by himself.

$$\begin{aligned}&\text{Game}(\text{Treasure}) \wedge \text{Key}(?x) \wedge \text{Box}(?y)\\&\quad \wedge \text{status}(?x, \text{Found}) \wedge \text{status}(?y, \text{Found})\\&\quad \rightarrow \text{GameStatus}(\text{Treasure,Win})\end{aligned}$$

To enable others to customize the application, the developers also specify rule-interfaces and associated configuration settings for this game, as illustrated in Fig. 6. For each included rule, a default action setting for it can be given by choosing from shared multimedia resources (see Fig. 5). For example, according to the settings in Fig. 6, once an object is found, a "*smiling*" agent will be displayed on the wall and inform the player that "*You have found a target object*".

In the *application customization* phase, end users can customize the shared Treasure game via the parameterization mode. This is achieved by answering form-filling questions through Treasure's configuration interface. For example, one of the form-filling questions asks the customizer to select several objects from the ones he owns to play the roles (e.g., a treasure box) of this game. Four objects are selected to be hidden in the given example shown in Fig. 6, where a room key and a keyboard box are used to play the roles of the treasure-box-key and the treasure box. Two other objects, a bicycle key and a wallet, are respectively used to confuse players and to transmit cues to players.

End users can either follow the default actions given by developers or change the settings (*partly* or *totally*) according to their imagination. For example, if an end user imagines that a wallet can hint the player of treasure-box-key's location when the wallet is found, he can replace the video to be displayed by an image of a cardboard box (the treasure-box-key's hidden place), as shown in Fig. 6.

After playing the customized game, the end user who customizes this game can communicate with the developers for any technical problems (supposing that a hidden object cannot be "found" though exposing to the "air") or potential improvements. If he finds the Treasure game very interesting, he can share his experiences with his friends by sending recommendations to them (e.g., the website of this game) or, further, authorizing some friends to access the game application he customized. In the latter case, his friends can play Treasure by only performing simple modifications.

5.2 System implementation

Corresponding to the Treasure scenario, we built numerous smart artifacts by attaching sensors to everyday objects. As shown in the left part of Fig. 7, two types of sensors are used: ultrasonic location tags to localize objects [8, 15, 16] and MICA2 Mote sensors (http://www.xbow.com) to detect object status.

To present multimedia actions to players, two smart devices are used. The smart projector displays information in the form of animation or image on the wall that the player is currently facing, and the smart ultrasonic speaker generates voices related (it makes players feel that the voices are coming from walls, please refer to [17, 18] for technical details). A snapshot of this use scenario implementation is shown in the right part of Fig. 7.

Our prototype system is built upon the Java 2 platform. We adopted SPARQL (http://www.w3.org/TR/rdf-sparql-query/) as the context querying language. An open-source Java library—Protégé-OWL API (http://protege.stanford.edu/plugins/owl/api/)—is used for parsing OWL at the programming level. Jess (http://www.jessrules.com/), a forward-chaining inference engine is used to execute user-defined rules. The interaction between SWRL rules and the Jess rule engine is implemented through the SWRL-Jess Bridge API [19].

6 Evaluation

We conduct a user study to evaluate OPEN's usability. The main purpose is to validate whether users can correctly program context-aware applications using the three different programming modes.

		Rules	Rule-Interfaces	Video Setting	Audio Setting
Game Developer	Shared Rules	U3D_Not_Updated	No	No	No
		U3D_Updated Rule	No	No	No
	New Rules	Object_Find Rule	?x: Name of the object		"You have found a target object."
		Game_Win Rule	?x: Name of the Target Key ?y: Name of the Treasure-Box		"Oh, my god. You win this game. Congratulations!"
Game Customizer	Customized Rules	Object_Find	A room-key (act as target-key)		"You have found the treasure-box-key."
		Object_Find	A used keyboard-box (act as treasure box)		"You have found the treasure-box"
		Object_Find	A bicycle-key (to confuse the player)		"Do not touch me! I am sleeping now"
		Object_Find	A black-wallet (to transmit hints)		"Thanks for rescuing me, I will give you a hint about the key"
		Game_Win	The room-key The keyboard-box		"Oh, my god. You win this game. Congratulations!"

Partial
New
Default

Fig. 6 Programming the treasure game

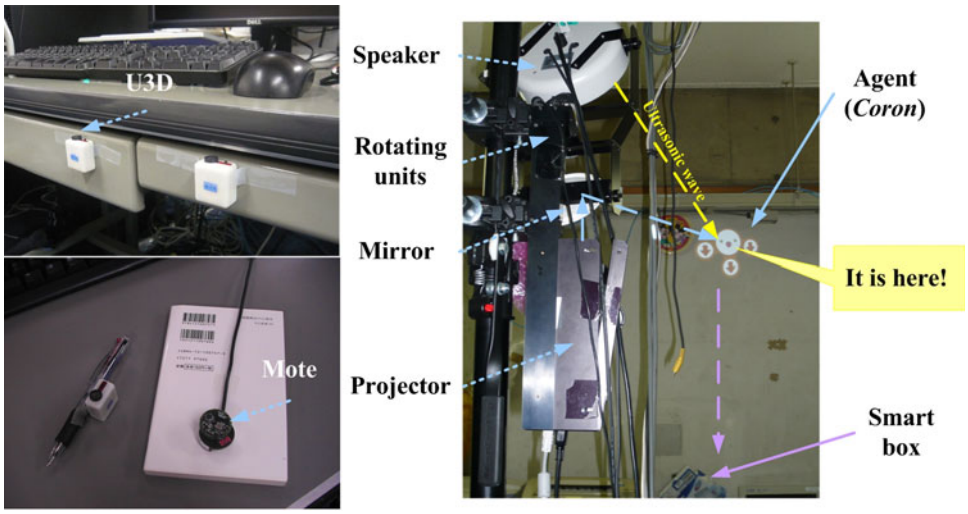

Fig. 7 Prototypical smart artifacts and the action devices

To conduct the user study, fifteen students from Keio University were recruited by email (ages ranging from 21 to 34). The subjects varied widely in gender (three females), discipline, and programming ability (20% of them have good knowledge of programming, while 67% of them have none). Each subject was invited to program the designed context-aware application with OPEN, answer a questionnaire, and give feedback about our system at the end of each session. It lasted for approximately 60–80 min for each subject's test session.

We first evaluated the composition mode. Subjects were asked to construct an application by selecting three rules from the rule-browsing page. All subjects except three successfully chose the right rules from the fifteen rule candidates in a mean time of 5 min. This illustrates 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% of subjects considered that it was not difficult to program using this mode.

For the incremental mode, we evaluated it by examining whether subjects could create a new rule. This evaluation was designed in a form-filling style, and subjects had to build a rule (choosing four right atoms from thirteen candidates) with the premise that it can remind the household who is going to leave home to take an umbrella if it rains outside. Less than half of the subjects (40%) gave the right answer with little or no assistance. Three subjects corrected their answer after a discussion with us. The rest (40%) had one or more errors in their answer. The result shows that even designed in a relatively simple testing form, it is still difficult for many users to create new rules. The questionnaire result shown in Fig. 8 illustrates that about two-thirds of the subjects found this programming mode difficult, whereas there were still five subjects (33%) thinking that they were able to learn to create rules after several days' training during the feedback session.

We finally evaluated the parameterization mode. To test this, we asked the subjects to customize the shared Treasure game through the configuration interface (see Fig. 5a). 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 8 to 17 min). Many interesting scenes were defined by them. For example, one subject added a "bonus" scene, which could give the player an added bonus when he found a "hidden-magic-book". The data shown in Fig. 8 clearly reveal that most subjects (80%) found it much easier to program using this mode. But there were still 4 subjects who did not perform the task according to instructions. For example, the requirement given by the first configuration item, namely "*You must specify a box and a key to act as treasure-box and treasure-box-key*", was ignored by some of them. The feedback from them indicated that the text-based interface was boring to them and made them easily miss some important points. They suggested us to improve it by using more graphical icons.

Allowing users to program context-aware applications creates a number of issues concerning correctness, consistency, as well as security. Although OPEN has built up a communication channel among users for discussion of technical problems, some criticisms were still raised on this. Some subjects suggested that in addition to diagnosing the problems arising at runtime, the system should provide better error-checking mechanisms that could automatically detect errors and prompt users to assist them during the programming process.

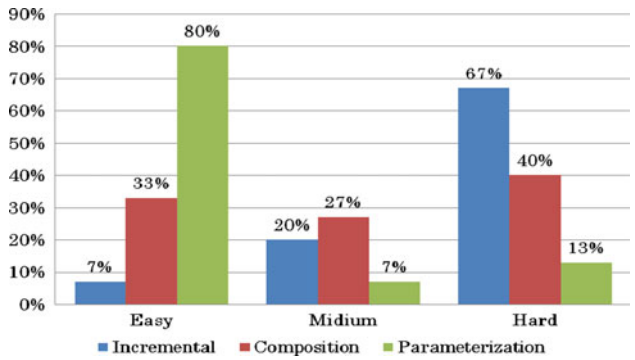

Fig. 8 Evaluation results

7 Conclusion

OPEN represents our early efforts to build a cooperative programming environment for context-aware applications. To meet diverse user requirements in the development and customization of context-aware applications, three programming modes with diverse complexity were proposed and implemented. OPEN further identifies and implements three user-cooperation patterns to facilitate the sharing and reuse of resources and user experiences. The evaluation results from our preliminary user study reveal that OPEN can provide appropriate programming support to a broad category of users, ranging from novices to programmers. Based on the feedback from the subjects, we intend to make OPEN more user-friendly by replacing text-based configuration interface with graphical ones. The privacy and security issues among cooperation patterns are also planned to be explored in our future work.

References

1. Lieberman H, Paternò F, Wulf V (2008) End user development. Springer, Dordrecht
2. Salber D, Dey AK, Abowd GD (1999) The context toolkit: aiding the development of context-enabled applications. In: Proceedings of CHI'99, pp 434–441
3. Chen G, Kotz D (2002) Solar: an open platform for context-aware mobile applications. In: Proceedings of the 1st international conference on pervasive computing, pp 41–47
4. Chen H, Finin T, Joshi A, Perich F, Chakraborty D, Kagal L (2004) Intelligent agents meet the semantic web in smart spaces. IEEE Internet Comput 19(5):69–79
5. Wang XH, Zhang DQ, Dong JS, Chin CY, Hettiarachchi S (2004) Semantic space: an infrastructure for smart spaces. IEEE Pervasive Comput 3(3):32–39
6. Gu T, Pung HK, Zhang DQ (2005) A service-oriented middleware for building context-aware services. Elsevier J Network Comput Appl 28(1):1–18
7. Yu ZW, Zhang DQ, Zhou XS, Chin C, Yu ZY (2006) An OSGi-based infrastructure for context-aware multimedia services. IEEE Commun Mag 44(10):136–142
8. Guo B, Satake S, Imai M (2008) Home-explorer: ontology-based physical artifact search and hidden object detection system. Mobile Inf Syst 4(2):81–103
9. Dey AK, Sohn T, Streng S, Kodama J (2006) iCAP: interactive prototyping of context-aware applications. In: Proceedings of pervasive 2006, pp 254–271

10. Mattila J, Väätänen A (2006) UbiPlay: an interactive playground and visual programming tools for children. In: Proceedings of the conference on interaction design and children, pp 129–136
11. Montemayor J, Druin A, Chipman G, Farber A, Guha ML (2004) Tools for children to create physical interactive storyrooms. Comput Entertain 2(1):12
12. Blackwell AF, Hague R (2001) AutoHAN: an architecture for programming the home. In: Proceedings of the IEEE symposium on human-centric computing languages and environments, pp 150–157
13. Tang L, Yu ZW, Zhou XS, Wang HB, Becker C (2010) Supporting rapid design and evaluation of pervasive applications: challenges and solutions. Personal and ubiquitous computing
14. Gu T, Pung HK, Zhang DQ (2004) Toward an OSGi-based infrastructure for context-aware applications. IEEE Pervasive Comput 3(4):66–74
15. Guo B, Satake S, Imai M (2006) Sixth-sense: context reasoning for potential objects detection in smart sensor rich environment. In: Proceedings of the IEEE/WIC/ACM international conference on intelligent agent technology (IAT'06), Hong Kong
16. Nishida Y et al (2003) 3D ultrasonic tagging system for observing human activity. In: Proceedings of IEEE international conference on intelligent robots and systems, pp 785–701
17. Nakadai K, Tsujino H (2005) Towards new human-humanoid communication: listening during speaking by using ultrasonic directional speaker. In: Proceedings of robotics and automation, pp 1495–1500
18. Ishii K, Yamamoto Y, Imai M, Nakadai K (2007) A navigation system using ultrasonic directional speaker with rotating base. In: Proceedings of HCI07, pp 526–535
19. O'Connor MJ, Knublauch H, Tu SW (2005) Supporting rule system interoperability on the semantic web with SWRL. In: Proceedings of the 4th international semantic web conference