Mobile Information Systems 4 (2008) 81–103
IOS Press

Home-Explorer: Ontology-based physical artifact search and hidden object detection system

Bin Guo*, Satoru Satake and Michita Imai
Keio University, 3-14-1 Hiyoshi, Kohokuku, Yokohama 223-0061, Japan
E-mail: {bingo, satake, michita}@ayu.ics.keio.ac.jp

Abstract. A new system named Home-Explorer that searches and finds physical artifacts in a smart indoor environment is proposed. The view on which it is based is artifact-centered and uses sensors attached to the everyday artifacts (called smart objects) in the real world. This paper makes two main contributions: First, it addresses, the robustness of the embedded sensors, which is seldom discussed in previous smart artifact research. Because sensors may sometimes be broken or fail to work under certain conditions, smart objects become hidden ones. However, current systems provide no mechanism to detect and manage objects when this problem occurs. Second, there is no common context infrastructure for building smart artifact systems, which makes it difficult for separately developed applications to interact with each other and uneasy for them to share and reuse knowledge. Unlike previous systems, Home-Explorer builds on an ontology-based knowledge infrastructure named Sixth-Sense, which makes it easy for the system to interact with other applications or agents also based on this ontology. The hidden object problem is also reflected in our ontology, which enables Home-Explorer to deal with both smart objects and hidden objects. A set of rules for deducing an object's status or location information and for locating hidden objects are described and evaluated.

Keywords: Smart object, ubiquitous computing, semantic web, ontology, context aware

1. Introduction

The development of wireless sensor network makes it possible to embed sensors into everyday artifacts, creating smart artifacts, which can act as self-aware and environment-aware agents to deliver human-centric services. One such service supported by a ubiquitous smart artifacts environment, is the real world search. Unlike *Google* search in the virtual world, a real world search system can identify the real-time location, status and profile information of physical world objects. Such a search system can save people a lot of time and effort in organizing and managing their physical belongings.

There are several problems in building a physical world search system. One is designing smart objects. Related work includes the MediaCup project [19], in which various sensors are embedded in a cup to record and transmit its movement and temperature information, and the RFID-based Smart Toolbox, which can track and monitor the movable tools in a toolbox [20]. The second problem is how to locate

*Corresponding author: Bin Guo, Anzai & Imai Lab., Department of Information & Computer Science, Keio University, Hiyoshi 3-14-1, Kohoku-Ku, Yokohama 223-8522, Japan. Tel.: +81 45 560 1070 Ext. 43242; Fax: +81 45 560 1064; E-mail: bingo@ayu.ics.keio.ac.jp.

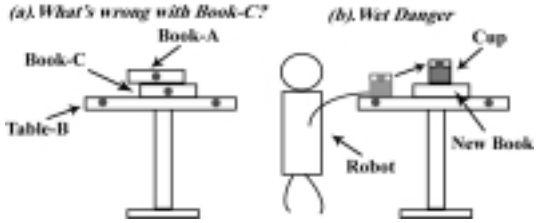


Fig. 1. Hidden object scenarios.

objects, which has also been studied a lot. The Active Bats system uses ultrasonic sensors and the triangulation location-sensing technique to locate objects [1]. The Smart Floor project embeds pressure sensors in the floor to locate humans and objects [23]. More detailed information about locating systems can be found in Hightower and Borriello [17]. Most other studies are centered on the modeling of contexts derived from smart objects. Context Toolkit represents context in the form of attribute-value tuples [9]. Henricksen et al. [18] focused on constructing a formality model to gather and manage the contexts by using the UML technology. Some other context-aware systems, like CoBrA [12] and Semantic Space [32], seek to use ontology to model contexts.

To implement a physical world search system, further researches are required. Although the sensor-enhanced smart artifact strategy has been the subject of many studies, little attention has been paid to the robustness of smart objects. We know that in ideal situations, the object equipped with sensors can be detected directly. However, in some circumstances, the hidden object problem may occur. There are several possible reasons for this problem: 1) Sensor faults (e.g. a sensor is broken or has a dead battery); 2) Sensors blockage (as shown in Fig. 1(a), the smart book *Book-C* is not detected by its ultrasonic sensor signal because it is blocked by *Book-A*); 3) Presence of non-sensor-equipped objects (e.g. a new book or belongings brought in by a visitor). Objects in these situations are all called hidden objects and locating them is sometimes very important (for instance in the scenario in Fig. 1(b), the robot did not notice the hidden object "new book" and put a full cup of water on it, exposing it to the "wet danger", i.e., the water may be spilled on the book). Therefore, detecting hidden objects like *Book-C* is a challenge that we should confront. Additionally, when a hidden object is detected, how to identify it so as to list it in the relevant search results is the other problem that an object search system should try to solve. The other common weakness of the previous systems is due to their context infrastructure. Current ad hoc context infrastructures that layer on programming objects (e.g., Java Classes), data structures or database schemes make it difficult to reason and to share and reuse knowledge. Although some pervasive computing research deals with these difficulties [2,11,12,32] using the normalized ontology technique, they take little account of the contexts derived from smart artifacts. Robustness of the embedded sensors (i.e., the potential hidden object problem) has been particularly neglected in their ontologies.

We propose an indoor physical artifact search and management system named *Home-Explorer*. This system is based on our previous work [4,5], but we have made great progress since those papers. Here, we report two main improvements over other studies. First, our system layers on a standardized and explicit knowledge infrastructure named *Sixth-Sense*, which explores Semantic Web technology to define a common ontology *SS-ONT* (*Sixth-Sense Ontology*) that aims to develop human-artifact interaction systems. Second, with the help of the underlying ontology infrastructure and the supporting tools for Semantic Web, *Home-Explorer* can both handle smart objects and detect hidden objects via rule-based context reasoning. In addition, to facilitate human search tasks, multiple search modes are incorporated into our system.

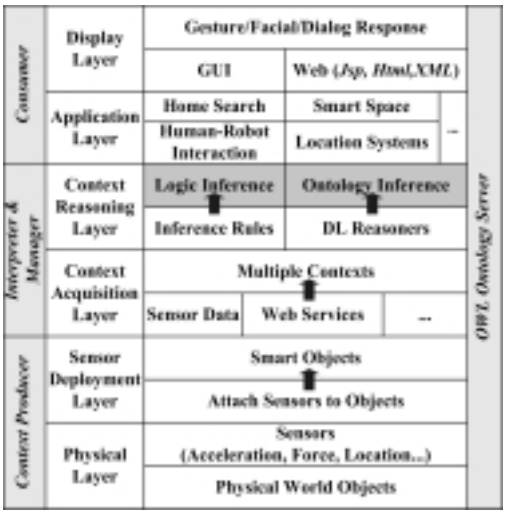


Fig. 2. Sixth-sense infrastructure architecture.

2. Overview of Home-Explorer

In this section, we present an overview of *Home-Explorer* and its knowledge infrastructure and system structure.

2.1. Sixth-sense infrastructure

Figure 2 is an illustration of the architecture of *Sixth-Sense*, our ontology-based context-aware infrastructure. As the figure shows, *Sixth-Sense* consists of one common ontology server and six collaborating layers.

Ontology server is an ontology repository that serves as a shared knowledge base where all the other layers can retrieve or store the defined (or deduced) information (we discuss it in detail in Section 3). *Physical Layer* tracks various artifacts and devices, most of which are frequented used in typical homes. *Sensor Deployment Layer* mainly concerns how to deploy sensors. *Context Acquisition Layer*'s work is to acquire low-level context information from heterogeneous sources (e.g., sensors, web services). Each derived context in this layer reflects one feature or attribute of a related entity (e.g., a cup) or an abstract concept (e.g., weather). *Context Reasoning Layer* is implemented at two different levels: (1) the ontology inference using the description logic reasoner Racer [30]; (2) rule-based logic inference using SWRL [25] and the Jess inference engine [15] to infer higher-level contexts from low-level ones. *Application Layer* includes a number of applications that use the defined or derived context knowledge to deliver expected human-centric services. *Home-Explorer* is one such application. *Display Layer* mainly considers about how to display and transmit the inferred information to users. In some computer-related applications, the information can be displayed as application interfaces like GUI or in the form of Web pages.

2.2. System structure

Home-Explorer's whole operating process is shown in Fig. 3, where it can be seen that its design is ontology-centric and it consists of three components, separately for smart object localization, hidden object detection, search engine and interface design. We will present them in the following sections.

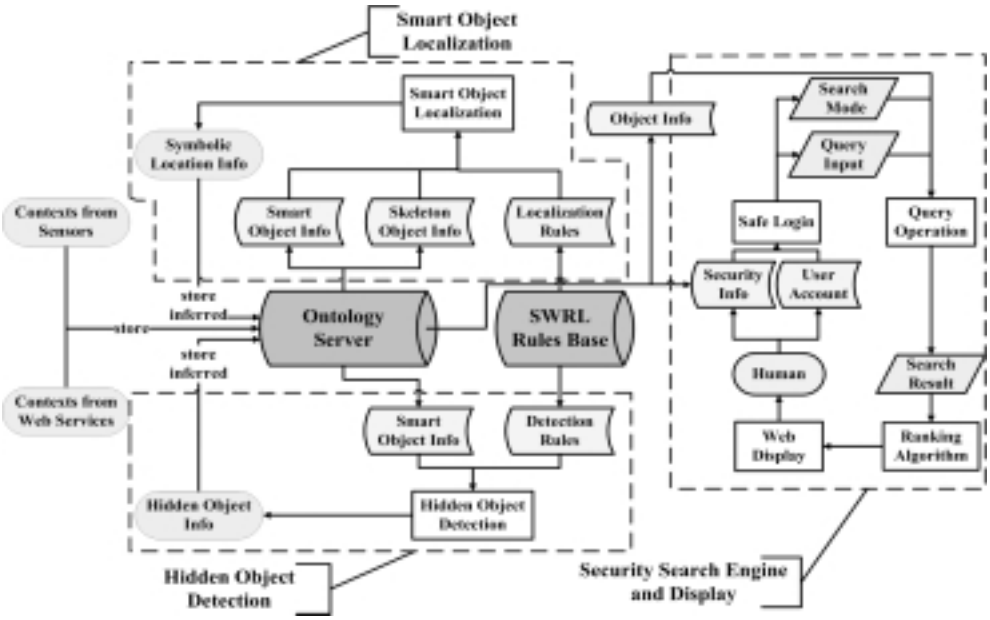


Fig. 3. Home-Explorer working process.

3. OWL ontology used in Home-Explorer

An explicit context infrastructure is a key element when building smart artifact systems. Our *Sixth-Sense* infrastructure uses ontology to model contexts because doing so enables context sharing, knowledge reuse, semantic query and context reasoning.

3.1. Why ontology?

As mentioned in the introduction, most current smart artifact systems rely on ad hoc representations of context information. In contrast to these ad hoc infrastructures, context modeling based on ontology is more normalized and promising and can at least offer the following three advantages. First, ontology's hierarchical structure enables developers to reuse domain ontologies (such as the modeling of smart artifacts) when building new domain-related applications. Second, an ontology provides a shared vocabulary base, which enables the independently developed applications and agents to interoperate based on common understanding of the context semantics. Finally, using ontology as a knowledge infrastructure allows the underlying system implementations to be better integrated with the existing ontology technologies (such as the ontology inference engines and querying mechanisms supported by the Semantic Web).

Our *Sixth-Sense* infrastructure explores the Semantic Web [28] technology to define a common ontology whose purpose is the development of intelligent human-artifact interaction systems. The *Sixth-Sense Ontology* (*SS-ONT*) is expressed by the emerging Semantic Web language OWL (Web Ontology Language) [8], which provides standardized approaches and support tools for knowledge sharing and logic reasoning. In the next subsection, we present detailed information about *SS-ONT*.

3.2. An extended ontology for Home-Explorer

SS-ONT is designed at two levels. The upper ontology is a high-level ontology that provides a set of

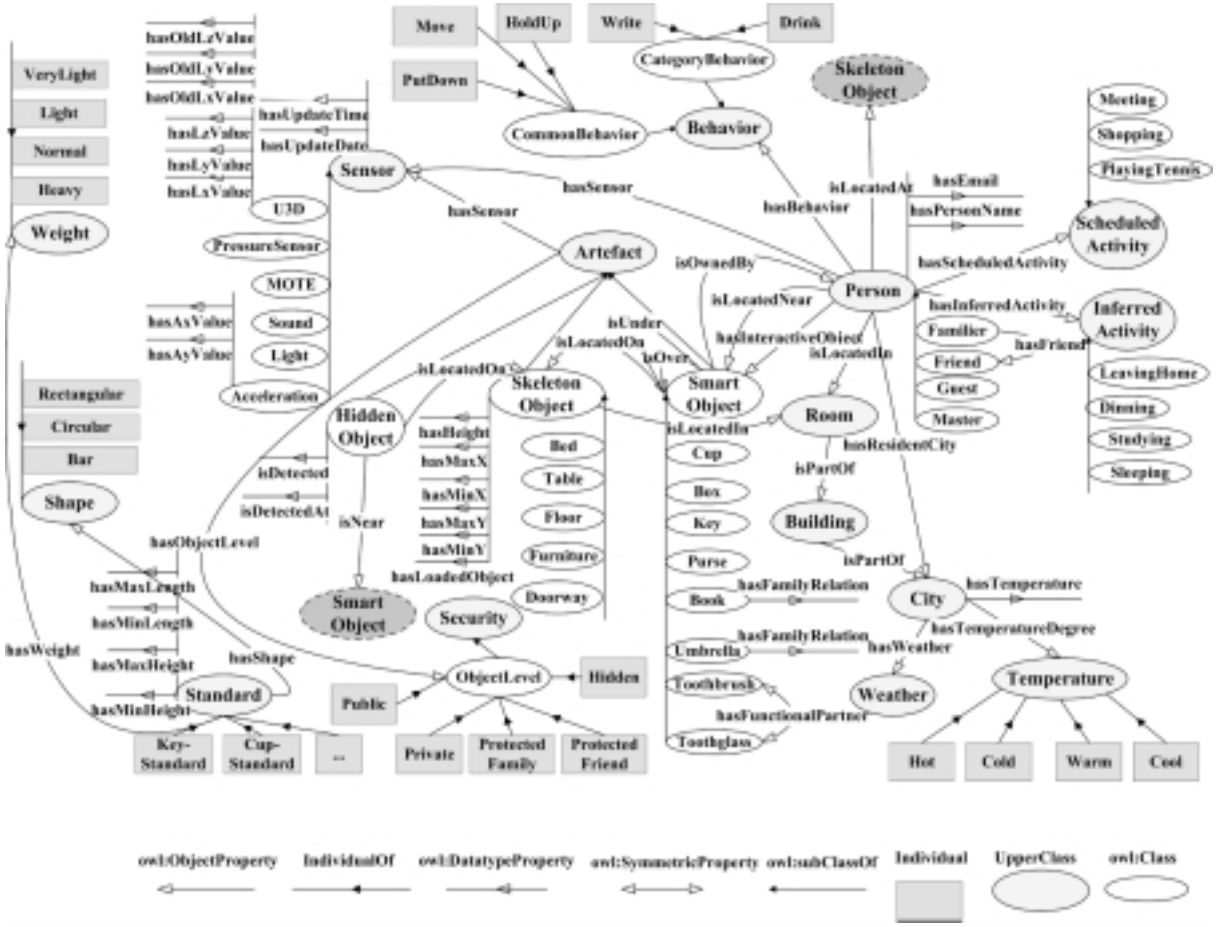


Fig. 4. An extended SS-ONT ontology.

basic concepts that are common among different smart spaces (e.g., home and office). The current SS-ONT upper ontology version (*ver-1.1*) is available at "http://www.ayu.ics.keio.ac.jp/members/bingo/SS-ONT-Upper.jpg". The domain-specific ontology is an extension to the upper ontology by adding detailed concepts and their features in different application domains. The two-level design makes SS-ONT more flexible and extensible, which enables developers to customize particular application ontologies by inheriting the classes defined in the upper ontology. Part of the extension of SS-ONT upper ontology used in *Home-Explorer* is shown in Fig. 4.

This extended ontology consists of 95 classes, 106 properties (including 55 OWL object properties and 51 datatype properties), and 72 individuals (The whole definition is available at "http://www.ayu.ics.keio.ac.jp/members/bingo/SS-ONT-v1.1.owl"). This ontology models a portion of the general relationships and attributes associated with artifacts and humans. The hidden object problem is particularly well dealt with in this ontology. Now we turn to an explanation of part of these defined concepts that are vital to our work.

3.2.1. Artifact

Explicit and precise definitions about the physical artifacts are important as smart artifact applications. In the current ontology, the defined artifact properties can all be classified as one of five types: object

profile and state information, location description, relationships with humans, relationships with other objects, and robustness.

(1) *Profile and state information:* these describe, respectively, the static (e.g., *hasName, hasShape, hasColor, hasLength, hasWidth, hasHeight or isLuminous*) and dynamic properties (e.g., *hasTiltAngle, isMoving*) of an artifact.

(2) *Location:* To approximate human cognition, we prefer to use symbolic (or relative) expression rather than absolute location (i.e., raw coordinate data). In this way, a key's location can be represented as: *"on* (*key, table*)*"*, which means the key is on the table. Obviously, it's much easier for a human to find an indoor landmark object (e.g. a table) than a small object (e.g. a key), so the symbolic expression delivers an efficient way for humans to find things. To achieve this goal, we borrowed two subclasses from the *Artifact* class: *SkeletonObject* and *SmartObject* (as Fig. 4 shows). Several large and mostly static artifacts are selected to serve as skeleton objects (i.e., indoor landmarks), while all other little and easily movable objects are categorized into the *SmartObject* class. Figure 4 lists several inherited subclasses of these two artifact types. In *SS-ONT*, the property "isLocatedOn" is defined to represent the location relationship between smart object A and skeleton object B (means A is located on B). Besides this property, we define two other terms to reflect the spatial relations between two *SmartObject* instances (e.g., a book is placed on another book). The two properties, *isOver* and *isUnder*, are defined as OWL inverse properties. That is to say, the assertion that A is placed on B is equivalent to the assertion that B is under A. Other than the purpose of locating an object (like *"isLocatedOn"*), these two properties are imported to enforce some reasoning tasks (described below). In addition, a *Boolean* datatype property *hasNoUnderObject* is defined to denote whether or not smart object B is placed under smart object A; if no such object B exists, A's *hasNoUnderObject* property will be set to *true*.
For the *SkeletonObject* instances, in view of the fact that they will act as reference objects to other smart objects, some particular properties to specify their coverage area are needed. For example, the coverage area of a skeleton object with a rectangular surface can be expressed by four OWL datatype properties: *hasMaxX, hasMinX, hasMaxY, hasMinY,* which involve the maximum and minimum horizontal coordinate values. There is also another datatype property *"hasHeight"* to denote the height of this skeleton object, and a *Boolean* property *"hasLoadedObject"* to express whether there is any object located on this skeleton object. An OWL object property *"isLocatedIn"* is defined to specify in which room the skeleton object is placed.

(3) *Relationships with humans:* This mainly reflects the relations between artifacts and humans. *isOwnedBy* and *isInteractedBy* are two such properties.

(4) *Relationships with objects:* Beyond the physical relations, there are also logical relations among artifacts (e.g., the functional relation between a toothbrush and a glass). Compared to the physical relations, the logical relations among artifacts can sometimes provide more precise and direct information (e.g. assuming a scenario in which we have to identify what is placed in a glass, as the known functional relationship, we may guess that it is a toothbrush but not a pen). To the best of our knowledge, the logical relations among physical artifacts have little been discussed in previous smart space research. However, by using OWL, we can easily reflect them in our ontology, and the current *SS-ONT* defines four such logical relation types.

Definition 1. (**Combinational Relation**)**:** Some objects work as components or accessories for other objects, e.g., the relation between mouse and computer (reflected by predicate *"hasComponent"* in *SS-ONT*).

Definition 2. (**Family Relation**)**:** Objects that belong to the same category sometimes have this relationship because, in daily life, some similar operational modes exist when humans deal with objects in the same category (e.g., people deposit all their books on the bookshelf, or place all their umbrellas in the umbrella stand near the door). We use an OWL *Boolean* property *"hasFamilyRelation"* to denote whether the object class has such a relation.

Definition 3. (**Partner Relation**)**:** Several objects can cooperate as partners to perform some human-centric tasks and always are arranged in the same way. Possible instances include tables and chairs, foreign language books and electronic dictionaries. The partner relation between objects is symmetric, and a symmetric property *"hasWorkPartner"* is used to represent this relation.

Definition 4. (**Functional Relation**)**:** Some objects are designed for a specific functional purpose, which is to serve or to support another kind of objects' work (e.g. a toothbrush and its supporter glass). We use *"hasFunctionalPartner"* to express this relation.

(5) *Robustness:* As mentioned in the introduction, the embedded sensor may sometimes not work well, which causes the related smart object to lose its smart status. Sometimes, too, there are no sensor-equipped objects. Either of these factors may cause a hidden object problem. A well-defined ontology should reflect such unstable factors. In *SS-ONT,* there is another subclass inherited from the *"Artifact"* class named *"HiddenObject"*, which represents these no-exact-information objects. To support detection and localization of these hidden objects via context reasoning, several properties are defined: *"isDetected"* is a *Boolean* property that denotes if a given hidden object is detected; *"isDetectedAt"* is a time property that denotes the instant the object is detected, and; *"isLocatedOn"* and *"isNear"* are two properties that represent the hidden object's location.

3.2.2. Security filter

If people prefer, for reasons of convenience, to live in a context-aware home, every reasonable measure should be taken to prevent information generated by the system about their private life from being accessed by others. Otherwise people will not find life in a smart house pleasant [24]. In an artifact management system, the resident manager should, on the one hand, have the authority to decide which artifacts can be publicly viewed and which cannot, and on the other hand to allow access to different users, e.g. friends. To this end, *SS-ONT* classified the *"Person"* class into four subclasses: *Master, Familier* (denotes a family member), *Friend,* and *Guest*. In addition, objects are classified into five types: *Public* (e.g., a story book), *ProtectedFamily* (objects shared by the whole family, such as furniture), *ProtectedFriend* (an object that is privately owned but set by the owner to friend level), *Private* (personal objects, such as diaries) and *Hidden* (hidden objects). The first four object levels are used for smart objects while the last one is for hidden objects. A *Master* user can track all the smart objects; A *Familier* user can view the private objects that belong to him (represented by the *"isOwnedBy"* property of the *Artifact* class) and the other three smart object types; If a *Familier* user A has a friend B, and A sets its private object *O* level to be *ProtectedFriend*, then *O* will be accessible to B; Finally, a *Guest* user can only access *Public* objects. In addition, for the sake of security, objects at the *Hidden* level can only be tracked by *Master* users. With this protection policy, if B wants to search an object that belongs to another *Familier* user C, our system will block his access.

Fig. 5. Mote sensor family (left) and smart artifacts (right).

3.2.3. Sensor

Our system uses various sensors, including U3D (ultrasonic 3D location) sensors, MICA2 mote sensors [7] (a combination of an acceleration sensor, a light sensor, and a sound sensor) and Kinotex pressure sensors [27]. Such sensors are either attached to smart artifacts or placed on skeleton objects. The MOTE sensor family and some sensor attached artifacts are illustrated in Fig. 5. To reflect the various sensor types, we borrowed several subclasses from the top class *"Sensor"*, like U3D and MOTE (see Fig. 4). A number of properties that represent sensor values are also defined. For instance, *"hasLxValue"*, *"hasLyValue"* and *"hasLzValue"* denote the real-time coordinate values derived from U3D, whereas *"hasOldLxValue"*, *"hasOldLyValue"*, and *"hasOldLzValue"* represent the sensor data for the last update time (i.e., a historical data record). *"hasAxValue"* and *"hasAyValue"* denote the two axis acceleration sensor values.

3.2.4. Category standard

It is known that people always use the concept of category to distinguish between objects. Each category should have one or more particular attributes that distinguish it from other object categories. A definition of this artifact category principle is given here.

Definition 5. (**Category Standard**)**:** Each object category has its product or design standards, such as shape, size, color or weight, which are all endowed with specific or unified definitions (e.g., the size of a sheet of paper can be A4, B5; while the shape of the bottom of a cup is usually circular). We call this category standard.

Use of category standard helps recognize objects and SS-ONT has a top class *Standard* to reflect this concept. As illustrated in Fig. 4, class *Standard* has a number of individuals (e.g., *Cup-Standard, Key-Standard*) that denote different category standards. A set of properties is defined to describe the standards. OWL datatype properties *hasMaxLength, hasMinLength, hasMaxWidth, hasMinWidth, hasMaxHeight,* and *hasMinHeight* are used to express the general dimension range of this object category. Properties such as *hasWeight, hasShape,* and *hasColor* are used to describe the common attributes of the same kind of object as well.

4. Finding indoor smart objects

Having presented our ontology infrastructure, we now describe the development of the *Home-Explorer* components that are based on it. First, we will explain how the system finds smart objects.

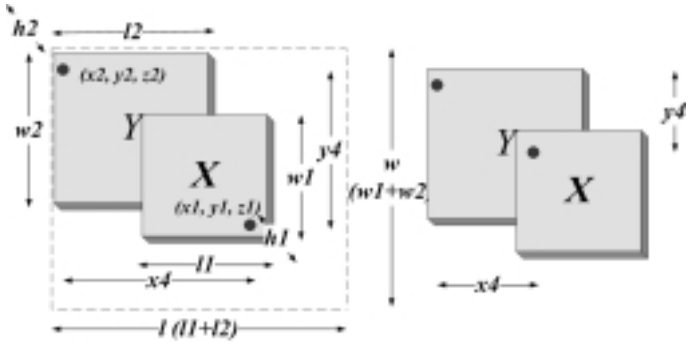


Fig. 6. Spatial relation between smart objects.

4.1. Localization of smart objects

For reasons explained above, we want our system to locate smart objects symbolically (i.e., in relation to skeleton objects), but the raw data received from the U3D sensors are absolute location data (i.e., 3D coordinates). Absolute location information is transformed to symbolic location information with a user-defined rule:

Rule-1: For smart object A and skeleton object B, if the downward projection of A is within the range of B, we conclude that A is placed on B.

SS-ONT supports the rules in the form of *Semantic Web Rule Language* (SWRL), which is intended to be the rule language of Semantic Web [25]. SWRL enables users to write Horn-like rules to reason about OWL individuals and to infer new knowledge about these individuals. *Rule-1* is formulated as SWRL in Eq. (1).

$$\begin{aligned}
&\mathit{SmartObject}(?A) \wedge \mathit{SkeletonObject}(?B) \wedge \mathit{hasSensor}(?A,?h) \wedge \mathit{U3D}(?h) \wedge \mathit{hasLxValue}\\
&(?h,?x) \wedge \mathit{hasMaxX}(?B,?x1) \wedge \mathit{hasMinX}(?B,?x2) \wedge \mathit{swrlb}:\mathit{lessThanOrEqual}(?x,?x1) \wedge\\
&\mathit{swrlb}:\mathit{greaterThanOrEqual}(?x,?x2) \wedge \mathit{hasLyValue}(?h,?y) \wedge\\
&\mathit{hasMaxY}(?B,?y1) \wedge \mathit{hasMinY}(?B,?y2) \wedge \mathit{swrlb}:\mathit{lessThanOrEqual}(?y,?y1) \wedge\\
&\mathit{swrlb}:\mathit{greaterThanOrEqual}(?y,?y2) \wedge \mathit{hasLzValue}(?h,?z) \wedge\\
&\mathit{hasHeight}(?B,?z1) \wedge \mathit{swrlb}:\mathit{greaterThanOrEqual}(?z,?z1)\\
&\rightarrow \mathit{isLocatedOn}(?A,?B) \wedge \mathit{hasLoadedObject}(?B,\mathit{true})
\end{aligned} \tag{1}$$

One of the most powerful features of SWRL is its ability to support a range of built-ins [6]. A built-in is a predicate that takes one or more arguments and is evaluated to be true if the arguments satisfy the predicate. For example, a *greaterThanOrEqual* built-in can be defined to accept two arguments and return true if the first argument is no less than the second one. Some SWRL built-ins, which are preceded by the namespace qualifier *swrlb*, are used in Eq. (1).

By using *Rule-1*, we can represent the smart object's location as its spatial relation with the skeleton objects. We also have other rules to represent the spatial relations between different smart objects, for instance, the *"isUnder"* property defined in *SS-ONT*. However, using the U3D sensors to determine if one object is under another one is not an easy thing. As Fig. 6 shows, owing to the relatively fixed spatial relations between the object and its sensors, and different locations of two objects X and Y, the horizontal distance, $(x4, y4)$, i.e., $(|x1 - x2|, |y1 - y2|)$, between them will be different (because

the data from their location sensors only represent two object locations in the real world but do not represent whole object spaces). For simplicity, we merely specify that all U3D sensors are attached to the top surface of smart objects. In Fig. 6, it can be seen that, although the value of the pair $(x4, y4)$ is changing, it cannot exceed the range of (l, w), where l and w separately denote the sums of object X and Y's lengths and widths. This condition can, to a certain extent (an exceptional case is discussed in Section 6.2), ensure that there is an overlap between X and Y. The second condition comes from the difference in height between these two U3D sensors, i.e., if the Z-coordinate value difference $(z1 - z2)$ is larger than the height of object X, we can conclude that Y is under X. This rule (*Rule-2*), represented in the form of SWRL, is given in Eq. (2).

$$
\begin{aligned}
&\mathit{SmartObject}(?X) \wedge \mathit{SmartObject}(?Y) \wedge \mathit{hasLength}(?X, ?l1) \wedge \\
&\mathit{hasWidth}(?X, ?w1) \wedge \mathit{hasHeight}(?X, ?h1) \wedge \mathit{hasLength}(?Y, ?l2) \wedge \\
&\mathit{hasWidth}(?Y, ?w2) \wedge \mathit{hasHeight}(?Y, ?h2) \wedge \mathit{hasSensor}(?X, ?c) \wedge \\
&\mathit{hasSensor}(?Y, ?d) \wedge \mathit{U3D}(?c) \wedge \mathit{U3D}(?d) \wedge \mathit{hasLxValue}(?c, ?x1) \wedge \\
&\mathit{hasLxValue}(?d, ?x2) \wedge \mathit{swrlb}:\mathit{subtract}(?x3, ?x1, ?x2) \wedge \mathit{swrlb}:\mathit{abs}(?x4, ?x3) \wedge \\
&\mathit{swrlb}:\mathit{add}(?l, ?l1, ?l2) \wedge \mathit{swrlb}:\mathit{greaterThanOrEqual}(?l, ?x4) \wedge \\
&\mathit{hasLyValue}(?c, ?y1) \wedge \mathit{hasLyValue}(?d, ?y2) \wedge \mathit{swrlb}:\mathit{subtract}(?y3, ?y1, ?y2) \wedge \\
&\mathit{swrlb}:\mathit{abs}(?y4, ?y3) \wedge \mathit{swrlb}:\mathit{add}(?w, ?w1, ?w2) \wedge \mathit{swrlb}: \\
&\mathit{greaterThanOrEqual}(?w, ?y4) \wedge \mathit{hasLzValue}(?c, ?z1) \wedge \mathit{hasLzValue}(?d, ?z2) \wedge \\
&\mathit{swrlb}:\mathit{subtract}(?z3, ?z1, ?z2) \wedge \mathit{swrlb}:\mathit{greaterThanOrEqual}(?z3, ?h1) \\
&\rightarrow \mathit{isUnder}(?Y, ?X) \wedge \mathit{hasNoUnderObject}(?X, \mathit{false})
\end{aligned}
\tag{2}
$$

4.2. Search modes for smart objects

To provide more friendly and intelligent human-centric services, we provide multiple search modes to facilitate the human search tasks. There are currently three ways of searching for smart objects.

Search by Name: The most common type of search in daily life is to find an individual object, such as *Bob's bicycle key*. By querying the ontology server with human input, this can be easily done. In the ontology's view, this mode can be interpreted as returning an individual (defined in the ontology) with a given name.

Search by Location: Sometimes people would like to know what is placed in a specific place, such as on a *study table*. Since our system locates smart objects based on their spatial relations with skeleton objects, if a skeleton object is named, the system can list all the objects that are on it, which will help people to make decisions or avoid dangerous situations (e.g. a cup of water and books and files placed on the same desk). In the ontology's view, this mode can be interpreted as returning all individuals whose *"isLocatedOn"* property value matches the human input.

Search by Category: In some cases people may not need to find an individual object and may want instead to find a series of objects that belong to the same category. For instance, Bob may want to choose an interesting book from among his books). In the ontology's view, this mode can be interpreted as returning all the individuals that belong to a targeted OWL class.

5. Mechanisms for searching for hidden objects

Home-Explorer provides special mechanisms to deal with the hidden object problem when the smart objects are lost due to the sensor faults or for other reasons. We should find a way to detect and locate hidden objects.

5.1. Rule-based hidden object detection

In *Home-Explorer,* hidden objects are also detected using a set of user-defined rules. The rules are mainly abstracted from common sense knowledge and various physical relations among objects. Look back to Fig. 1(a) to see how common sense works in daily life. By analyzing the physical relations between smart object *Book-A* and skeleton object *Table-B*, we can infer that *Book-A* is not placed directly on the desktop because there is a height difference between them, then common sense indicates that there must be something between *Book-A* and the desktop; however, by executing *Rule-2* in Section 4.1 for smart objects, we find that none of them is located under *Book-A*, so we may conclude that there is a hidden object. Based on this analysis we can conclude two things about the hidden object: first, that it is present and second, its approximate location. In other words, common sense reasoning can effectively and simultaneously detect and localize hidden objects.

All the inference rules derived from common sense knowledge are represented as SWRL rules and stored in the *SWRL Rules Base* (see Fig. 3). Based on the different types of knowledge the rules concern, we formulated four types, examples of which are presented below.

(*1*) *Spatial relation*

Rule-3: If smart object A is on skeleton object B and is laid flat (i.e., not tilted), there is no touch point between A and B, and there is no other smart object under A, then we infer that there is a hidden object C between them (as illustrated in Fig. 1(a)). This is formulated in Eq. (3).

$$\begin{aligned} &\mathit{isLocatedOn}(?A,\ ?B) \wedge \mathit{hasTiltAngle}(?A, \mathit{false}) \wedge \mathit{hasSensor}(?A,\ ?a) \wedge \\ &\mathit{hasHeight}(?A,\ ?b) \wedge \mathit{hasHeight}(?B,\ ?c) \wedge \mathit{hasLzValue}(?a,\ ?d) \wedge \\ &\mathit{swrlb}:\mathit{subtract}(?e,\ ?d,\ ?c) \wedge \mathit{swrlb}:\mathit{subtract}(?f,\ ?e,\ ?b) \wedge \\ &\mathit{swrlb}:\mathit{greaterThanOrEqual}(?f,\ 10) \wedge \mathit{hasNoUnderObject}(?A,\ \mathit{true}) \\ &\wedge \mathit{isDetected}(?C, \mathit{false}) \rightarrow \mathit{isLocatedOn}(?C,\ ?B) \wedge \mathit{isNear}(?C,\ ?A) \wedge \\ &\mathit{isDetected}(?C,\ \mathit{true}) \wedge \mathit{hasHeight}(?C,?f) \end{aligned} \tag{3}$$

Several comments should be made about Eq. (3). First, the height (size) of A and B is separately represented as b and c, and A's real-time z-coordinate value (data from its sensor) is represented as d, then the symbol $f(f = d - c - b)$ denotes the height of a potential object C. If this height value is above a threshold M (in Eq. (3), $M = 10$), then we conclude that hidden object C exists, otherwise not. Threshold M is used to reduce the influence of sensor noise, which causes fluctuations in sensor values even when the object does not move.

Second, the SWRL rules can not deduce that there is some new individual that has not been defined in the OWL ontology (though they can derive, assign, and update the property values of existing individuals), we must define an individual *HiddenObject* (e.g., *Hidden01*) in our ontology but set its *hasDetected* property to be *false* before a search using the SWRL rules is attempted. When, by some inference

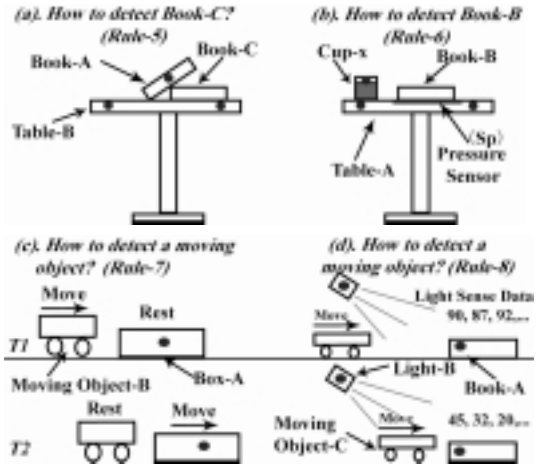


Fig. 7. Hidden object detection scenarios.

rule like *Rule-3*, a new hidden object is detected, the present individual *HiddenObject's* (*Hidden01's*) *hasDetected* property will be set to *true*. Meanwhile, a new individual *HiddenObject* (like *Hidden02*) will be created in our ontology. Of course, its *hasDetected* property will again be set to *false* when it is created.

Third, by using this rule, we not only detect the hidden object, but also get its relative location, which is reflected by *isLocatedOn* and *isNear* in Eq. (3).

Fourth, the *hasTiltAngle* property determines whether the relative object is laid flat or tilted, the value of which can be determined by the two axis acceleration values from the MOTE sensor the object equipped. This determining process is also implemented according to several rules, and one rule that asserts that the object is laid flat is given in *Rule-4* (Eq. 4).

$$\begin{aligned} &SmartObject(?x) \wedge hasSensor(?x,?y) \wedge MOTE(?y) \wedge hasAxValue(?y,?z) \\ &\wedge hasAyValue(?y,?a) \wedge swrlb:greaterThan(?z,\text{-}1.1) \wedge swrlb:lessThan(?z,\text{-}0.7) \\ &\wedge swrlb:greaterThan(?a,0.5) \wedge swrlb:lessThan(?a,0.75) \\ &\rightarrow hasTiltAngle(?x, false) \end{aligned} \tag{4}$$

The main principle is that different tilt angles can yield different acceleration values. First, we measured the acceleration value when the object was laid flat. Because there was noise, the measured results were bounded in two intervals (the x-axis in $[-1.1$ g, -0.7 g], and the y-axis in [0.5 g, 0.75 g] (the available range is $\pm$ 2 g in the MOTE sensor we used). This result is reflected in *Rule-4*.

Another rule for detecting a hidden object using spatial relations is given below.

Rule-5: If smart object A is located on skeleton object B, A is tilted, there is no other smart object under A, and nobody is using it, then we infer that there is an object C between A and B, as illustrated in Fig. 7(a). This is formulated in Eq. (5).

$$\begin{aligned} &isLocatedOn(?A,?B) \wedge hasTiltAngle(?A,true) \wedge isInteractedBy(?A,NoPerson) \wedge \\ &hasNoUnderObject(?A,true) \wedge isDetected(?C, false) \wedge \\ &\rightarrow isLocatedOn(?C,?B) \wedge isNear(?C,?A) \wedge isDetected(?C,true) \end{aligned} \tag{5}$$

It should be noted about this rule that, because some human action may have caused the object to tilt, we added the premise that nobody is using the object. The value of the property *isInteractedBy* will be assigned to a certain individual *Person* when human behavior is detected (*NoPerson* is a special individual *Person* defined in *SS-ONT* to indicate that nobody is present). Since human behavior is not the focus of this paper, we satisfy ourselves with giving a common principle for determining whether some artifact-centric human behavior has occurred: There are two major factors in this determination, first, the human must be near this artifact (reflected by the *Person* property *isLocatedNear* in *SS-ONT*, shown in Fig. 4), and; second, the state of the artifact changes when a human arrives.

(*2*) *Force relation*

Rule-6: If skeleton object A is subject to downward pressure in scope Sp (action zone of the force) and there is no smart object x located in Sp, then there will be a hidden object B on A, as shown in Fig. 7(b). This is formulated in Eq. (6).

$$\begin{aligned} &\mathit{SkeletonObject}(?A) \wedge \mathit{PressureSensor}(?p) \wedge \mathit{hasSensor}(?A,?p) \wedge \\ &\mathit{hasActionZone}(?p,?Sp) \wedge \mathit{SkeletonObject}(?Sp) \wedge \\ &\mathit{hasLoadedObject}(?Sp, \mathit{false}) \wedge \mathit{isDetected}(?B, \mathit{false}) \wedge \\ &\rightarrow \mathit{isLocatedOn}(?B,?A) \wedge \mathit{isDetected}(?B,\mathit{true}) \end{aligned} \tag{6}$$

In Eq. (6), the action zone of force *Sp* (derived from the pressure sensor values) is represented as an individual *SkeletonObject*. In this way, we can easily use the smart object localization rules, such as *Rule-1*, to determine whether a smart object is placed on it, i.e., to derive the value of *hasLoadedObject* (see Eq. (1)). Using this rule, the system can both detect hidden objects and determine some of the properties of the object, such as its size (by *action zone*) and weight (by *pressure intensity*).

(*3*) *Change of motion state*

Rule-7: If smart object A is located on skeleton object B, A's state changes during period P (from *rest* to *horizontal motion*), and there is no human interacting with A, then there will be a moving object C near A, as shown in Fig. 7(c). This is formulated by Eq. (7).

$$\begin{aligned} &\mathit{SmartObject}(?A) \wedge \mathit{SkeletonObject}(?B) \wedge \mathit{isLocatedOn}(?A,?B) \wedge \\ &\mathit{hasSensor}(?A,?s) \wedge U3D(?s) \wedge \mathit{isInteractedBy}(?A,\mathit{NoPerson}) \wedge \\ &\mathit{hasLxValue}(?s,?x1) \wedge \mathit{hasLzValue}(?s,?z1) \wedge \mathit{hasOldLxValue}(?s,?x2) \wedge \\ &\mathit{hasOldLzValue}(?s,?z2) \wedge \mathit{swrlb}:\mathit{subtract}(?a,?x1,?x2) \wedge \mathit{swrlb}:\mathit{abs}(?x,?a) \wedge \\ &\mathit{swrlb}:\mathit{greaterThanOrEqual}(?x,100) \wedge \mathit{swrlb}:\mathit{subtract}(?b,?z1,?z2) \wedge \\ &\mathit{swrlb}:\mathit{abs}(?z,\ ?b) \wedge \mathit{swrlb}:\mathit{lessThanOrEqual}(?z,10) \wedge \mathit{isDetected}(?C, \mathit{false}) \\ &\rightarrow \mathit{isLocatedOn}(?C,?B) \wedge \mathit{isNear}(?C,?A) \wedge \mathit{isDetected}(?C,\mathit{true}) \end{aligned} \tag{7}$$

In Eq. (7), if the x-coordinate value difference between two U3D sensor updates ($|x1 - x2|$) is above some threshold M (in Eq. (7) it is assigned to 100) and the z-coordinate value difference ($|z1 - z2|$) is below another threshold N (set to 10 in Eq. (7)), then we conclude that there is movement toward A. Thresholds M and N are also used to reduce the influence of sensor noise (as in Eq. (3)). However, Eq. (7) only makes an assertion based on the object's displacement relative to the x-coordinate, so a symmetric rule that relative to the y-coordinate is also required. As these two rules are very similar (they merely change the relative x variables in Eq. (7) to y variables), we don't list the second one in detail.

(*4*) *Sender-receiver pairs*

The term, sender-receiver pair, refers to several smart objects cooperating to monitor the environment. In such a pair, one smart object acts as the physical signal sender (such like a light source), and others act as signal receivers by using an attached sensor, such as a light sensor). *Rule-8,* given below, illustrates how to use smart object pairs to detect a hidden object.

Rule-8: Of smart objects A and B, B is a stable light source, and A can sense the light B transmits. If A's light sensing value changes during period P (from bright to dark) and A is not acted upon by any person, then there is an object C between A and B, as shown in Fig. 7(d). This is formulated in Eq. (8).

$$\begin{aligned}
&\mathit{SmartObject}(?A) \wedge \mathit{SmartObject}(?B) \wedge \mathit{isLuminous}(?B) \wedge \\
&\mathit{hasSensor}(?A,?s1) \wedge \mathit{MOTE}(?s1) \wedge \mathit{hasSensor}(?B,?s2) \wedge \mathit{MOTE}(?s2) \wedge \\
&\mathit{isInteractedBy}(?A,\mathit{NoPerson}) \wedge \mathit{hasLightValue}(?s1,?x1) \wedge \\
&\mathit{hasOldLightValue}(?s1,?x2) \wedge \mathit{swrlb}:\mathit{subtract}(?a,?x2,?x1) \wedge \\
&\mathit{swrlb}:\mathit{greaterThanOrEqual}(?a,30) \wedge \mathit{hasLightValue}(?s2,?y1) \wedge \\
&\mathit{hasOldLightValue}(?s2,?y2) \wedge \mathit{swrlb}:\mathit{subtract}(?b,?y1,?y2) \wedge \\
&\mathit{swrlb}:\mathit{abs}(?c,?b) \wedge \mathit{swrlb}:\mathit{lessThanOrEqual}(?c,20) \wedge \mathit{isDetected}(?C,\mathit{false}) \\
&\rightarrow \mathit{isNear}(?C,?A) \wedge \mathit{isNear}(?C,?B) \wedge \mathit{isDetected}(?C,\mathit{true})
\end{aligned} \tag{8}$$

In Eq. (8), $x2 - x1$ and $|y1 - y2|$ separately denote the change in A and B's light sensing value between two MOTE sensor updates. $x2 - x1$ is above some threshold M (set to 30 in Eq. (8)), butis less than another threshold N (set to 20 in Eq. (8)). Therefore, it can be concluded that B does not change but that A's light sense value does change.

5.2. Search modes for hidden objects

In some circumstances, the target of a search might be a hidden object. For instance, when a user wants to find the bedroom key and it happens to be a hidden object. In these cases, when *"bedroom key"* is input and the smart object search modes (presented in Section 4.2) are used, it is impossible to find it. Therefore, a new way of solving this problem should be found. As shown by the *"bedroom key"* problem, since it is not easy to search it in smart object modes, we may change our search focus to hidden objects. As in the smart object search, the system has multiple search modes for hidden objects.

Search All: This method lists all the detected hidden objects. In the ontology's view, this mode can be interpreted as returning all individuals that belong to the *HiddenObject* class. This search mode is useful for the *Master* user, who can get a global view of hidden objects and take measures to deal with them.

Search by Attribute Matching: One general way to recognize an unknown object is by its attributes. The more attributes the system recognizes, the more easily it can determine its identity. Therefore, to the system should acquire as much of the hidden object's attributes as possible. In reality the system acquires attribute information during the hidden object detection process, such as the height value in *Rule-4* and the weight and size attributes in *Rule-6*. A novel way of recognizing hidden objects can be implemented based on the derived attributes and the *category standard* (see Section 3.2.4) defined in *SS-ONT*. Figure 8 shows two entities: *Key Category* (the key category standard) and *Hidden06* (a hidden object). *Key Category* consists of five specific attributes, while *Hidden06* has four derived attributes. Note that *Hidden06*'s attributes all drop in the relative intervals or have the same value in comparison

Fig. 8. One example for attribute matching.

with those in *Key Category*. This means that we can assign a degree of probability or match rate, 0.8, to the proposition that *Hidden06* is a key.

The idea of this approach is similar to the topological or fingerprint mapping used for robot navigation and localization [22], which compares perceptional object-features with known database data (for place description) to estimate a robot's current position. It's also similar to the role-based classification approach mentioned by Beer et al. [31], which classifies an unknown object by comparing its attribute sets with the known attribute templates. Approaches like these all work with uncertainty, mainly informing users of the probability of each hidden object being the target, leaving the truth to be determined by the user.

Search by Associated Rules: This approach is based on the logical relations among objects that we discussed in Section 3.2.1. A definition of how the logical relations work is given below.

Definition 6. (**Associated Rules**)**:** there are various logical relations existing among objects. According to the natural qualities and functions of objects, or humans' daily habits or intentions, the objects with such relations are usually close to each other in the real world. We call these the associated rules.

According to our system, since the hidden objects are detected by nearby smart objects, according to Definition 6, associated rules may exist among them. Thus we can conjecture what the hidden object is using smart object information. For instance, when a hidden object *Hidden01* is detected by a smart object *Book-A*, since the Boolean *"hasFamilyRelation"* property is set to *true* for the *Book* category (see Fig. 4), we may conclude that *Hidden01* is also an individual *Book*. It should be noted that this kind of estimate is a low probability one (lower than attribute matching). However, because it is simple and efficient, we can still treat it as an effective method for our system.

6. Implementation and evaluation

6.1. Prototype implementation

In this subsection, we describe the current prototype implementation of our system from six aspects.

(*1*) *Ontology Design*

We adopted Protégé 3.3_bata (an open-source OWL editor) [29] to create and edit our *SS-ONT* ontology (upper and domain) and we stored it at the OWL files mode. To manipulate the ontology at the programming level (e.g., via Java), we used the Protégé-OWL API [13] to load, update and save these OWL files.

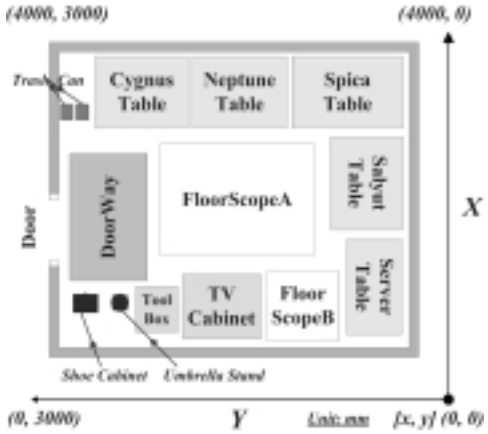


Fig. 9. The Home-Explorer test environment.

(*2*) *Context Query*

Massive query operations are unavoidable as systems for real-world searches. The Protégé-OWL API also provides simple methods to query the OWL ontologies. However, the efficiency and expressive power of this kind of query is not high. To support advanced queries, we adopted the SPARQL language, a standard RDF query language developed by W3C, as the context query language. Our SPARQL queries are backed by the Jena 2 Semantic Web API [14], which uses a module named ARQ to support SPARQL queries [3]. SPARQL queries consist of a series of triple patterns (i.e., <subject, predicate, object>) and modifiers. The statement for querying what objects are placed on the *DinnerTable* (*search by location*, the second search mode for smart objects) is given in Eq. (9).

$$\begin{aligned} &\textit{PREFIX ssont:<http://www.ayu.ics.keio.ac.jp/} \\ &\qquad \sim \textit{bingo / ontology / SS - ONT} \text{ - v1.1.owl\#>} \\ &\textit{SELECT } ?\textit{object} \\ &\textit{WHERE}\{?\textit{object ssont: isLocatedOn ssont: DinnerTable .}\} \end{aligned} \tag{9}$$

(*3*) *Context Reasoning and Inference Engine*

All the SWRL rules defined in our system are created and edited using Protégé SWRL Editor [26]. The current SS-ONT has 27 rules for different purposes, e.g., object state detection, human/object localization, human behavior/activity prediction, and rules for hidden object detection. A forward-chaining inference engine named Jess [15] is used to execute the rules based on the *SS-ONT* knowledge facts. The interaction between SWRL rules and the Jess inference engine is implemented through the SWRL-Jess Bridge API [21].

(*4*) *Test Experiment*

We used our workspace, a 4.0×3.3 m^2 portion of our laboratory, as the test-bed, (see Fig. 9). We deployed 16 ultrasonic receivers on the ceiling to work with the U3D sensors. Also, as Fig. 9 shows, there are several skeleton objects in our test space (e.g., *Neptune Table* and *Umbrella Stand*).

(*5*) *Search Interfaces*

Home-Explorer's main interfaces, which build on *JavaServer Pages* (*JSP*) technology, are shown in Fig. 10. To ensure security, the user should first login to our system with his authorized account, including

Fig. 10. Home-Explorer interfaces.

user type (e.g., *Master* or *Friend*), user name, and password (see Fig. 10(a)). In terms of the security policy defined in our ontology, different user types can only search the relevant level's objects (e.g., a *Friend* user can't search a *FamilyProtected* level object). This policy is reflected in the search results. The main search interface (Fig. 10(b)) is presented after user login, and the user can input the object search keyword on it. As described in Sections 4.2 and 5.2, we provide a total of six search modes: three for smart object search and three for hidden object search. Figure 10(c) illustrates the search results of books via the *"search by category"* mode, from which we can see that the relevant location and update time (to the embedded sensor) information is listed.

6.2. Evaluation

Because our search application is layered on a series of inference rules, the performance of the defined rules will directly influence the search results. In the following, we evaluate our system at two different aspects.

Table 1
Experiment results

Rule number	True positives	False positives	False negatives	Precision (%)	Recall (%)
1	48	0	2	100	96
2	47	5	3	90	94
3	45	7	5	87	90
4	44	3	6	94	88
5	42	10	8	81	84
6	46	0	4	100	92
7	40	12	10	77	80
8	42	11	8	79	84
Total	354	48	46	88	89

6.2.1. Evaluation of effectiveness

Eight typical inference rules are explained in this paper, and we tested them one by one. The experiments were performed as follows: For each inference rule R_i, we set the test time as 45 minutes, during which a situation similar to R_i's scenario recurred 50 times. However, to test R_i's performance in different situations, the 50 tests were performed alternately using object pairs in two different places (i.e., 25 times for each). For example, for *Rule-3*, there are three objects referred (expressed as (A, B, C)), so, in the test, the two test-groups can be set as (*book, table, book*) and (*box, floor, book*). Following we introduce three terms to distinguish different test results.

Definition 7. (**Diverse Test Types**)**:** In our system, an *accurate* test is one in which an object property (e.g., location) or a hidden object is detected at the right time by the right rule, it scores a true positive (TP). An incorrect claim scores a false positive (FP), and it's an *error* test. A *failed* test is one in which an inference rule should have been triggered but was not, which scores a false negative (FN).

We then used two standard metrics to summarize our system's effectiveness. *Precision* is the probability that a given inference about an object property (or a hidden object) is correct. *Recall* is the probability that *Home-Explorer* will correctly infer a given true hidden object (or smart object property). They can be expressed in formula Eq. (10):

$$\textit{Precision} = \frac{TP}{TP + FP}; \quad \textit{Recall} = \frac{TP}{TP + FN} \tag{10}$$

The results are shown in Table 1, where it can be seen that *Home-Explorer* correctly inferred that a hidden object occurs 88 percent of the time. For two inference rules (*Rule-1* and *Rule-6*), there were no false positives. Of the totally 400 tests (50 tests for each rule multiplied by 8) that actually happened, *Home-Explorer* detected 89 percent correctly. Rules for smart objects (*Rule-1*, *Rule-2* and *Rule-4*) performed better than the hidden object detection rules (the other five rules). This is mainly because most of the hidden object detection rules are founded on the smart object rules. For example, to use *Rule-5*, we must first derive information about the related smart object A that is referred to in this rule. This includes its location, its status, and whether there is any other smart object under it, which should be previously deduced by the smart object rules. The results show that our system can, to a certain extent, enhance the sensor network robustness when the hidden object problem occurs. By analyzing the real-time testing data, we generalized the following reasons that a failed or an error test may occur.

(1) *Sensor noise:* sometimes sensor noise exceeds the threshold value we set in the rules, which may trigger an inference rule, causing an error test.

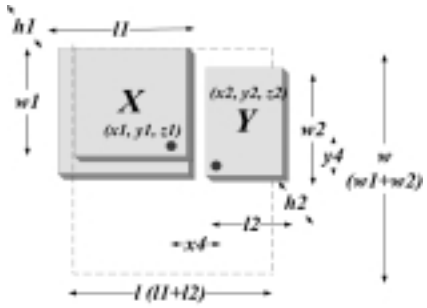


Fig. 11. An exception case for Rule-2.

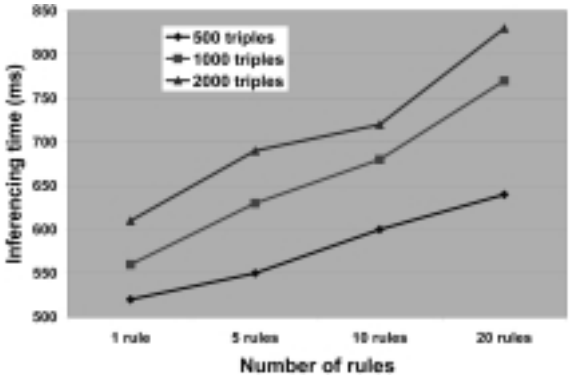
(2) *Object movement:* when a smart object is moving, it will take more time for the sensor receivers to relocate it, which may require a longer sensor update time and induce loss of important process data.
(3) *Sensor delay and data loss:* when more sensors are working, the sensor receivers have to cope with them one by one, resulting in more delays. Moreover, as the number of sensors increases, the processing capacity of the sensor system decreases. According to our experimental statistics, when 15 U3D sensors are working simultaneously, about 4% of the sensor data will be lost.
(4) *Inference rule defects:* some other reasons for failure may originate from the inference rules themselves. Because of the imprecise common sense knowledge background, it is sometimes difficult to create a faultless inference rule. Figure 11 gives an example of an exception to *Rule-2*. If the three objects X, Y, and Z (Z is under X and is a hidden object) are placed in this way, the horizontal distance between X and Y (i.e., ($x4$, $y4$)) still does not exceed the range of (l, w), and, since Z is under X, the height difference ($z1 - z2$) is also larger than the height of object X. Therefore, *Rule-2* will be triggered and will assert that Y is under X. With this incorrect conclusion, the detection of hidden object Z will also fail (as the *hasUnderObject* property value to X is true, *Rule-3* cannot be triggered).

6.2.2. Evaluation of runtime performance

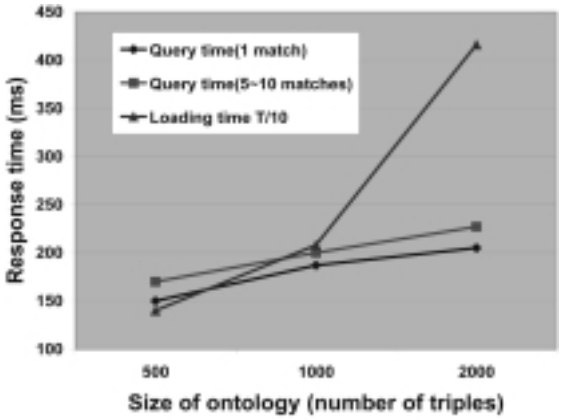
We also evaluated the performance of our system by measuring context reasoning and querying response time on a 1.06-GHz Pentium 4 workstation with 1.0 GB of RAM running Windows XP. We used three context data sets to evaluate our system's scalability. These three test data sets, including the real one we used in our test environment, i.e., *SS-ONT-v1.1*, which can be parsed into about 2000 RDF triples, and two other simplified versions of *SS-ONT* (with fewer classes and instances), separately amounts to about 1000 triples and 500 triples. Because the reasoning and querying operations are performed in the memory, the loading time of data sets from OWL files to the memory was measured as well.

We used four rule sets to test the performance of our context reasoner. The smallest rule set includes only one rule (*Rule-1* in Eq. (1)), and the biggest rule set consists of twenty rules (including the eight rules described earlier in this paper and some other rules created for other applications). The result of our experiment, shown in Fig. 12(a), definitely indicates that logic-based context reasoning is a computational intensive task, and the inferencing response time it costs greatly depends on the size of the ontology file and the number of rules applied.

We used SPARQL query statements like Eq. (9) to test the performance of context querying. The experiment result, shown in Fig. 12(b), clearly demonstrates that the increase of triples or matched facts



(a)



(b)

Fig. 12. Context reasoning performance (a) and querying performance (b).

results in a corresponding increase of the querying response time. Figure 12(b) also shows that the loading time of an ontology data set is, to some extent, proportional to the size of this data set.

By the above experiment results, we can conclude that the context reasoning time will be human perceivable if the smart space's scale increases. However, as a non-time-critical application, the real time requirement of searching smart objects or detecting hidden objects is not likely to be critical so that a perceivable delay (about one second) caused by context reasoning is acceptable. That's to say, as long as the real time requirement is not too strict, our context infrastructure can support a set of useful human-centric services, for example, an application that can remind a human what he forgets to carry (e.g., it finds that his wallet is still on the table) when he wants to leave home, and another application that can alert a human to the fact that he forgets to put the yogurt or butter back to a refrigerator after it is left on a table for more than 15 minutes, and so forth. The performance evaluation results also suggest that, under a suitable scale of ontology size and the complexity of rule set, our logic-based context reasoning mechanism can also work for some time-critical applications (e.g., security, emergency).

In summary of the above evaluations, to further improve our system's performance, better sensor technologies, transmission protocols, underlying common sense knowledge, and reasoning technology still need to be used.

Table 2
Comparison with other related systems

System name	Sensing technique	Smart artifact system	Object localization Absolute	Relative	Real world search	Context infrastructure	Inference engine	Hidden object detection
Media-Cup	Motion orientation, temperature	√	–	–	–	Ad hoc	–	–
Smart Toolbox	RFID	√	–	·	–	Ad hoc	–	–
Semantic Space	Environment sensors, WLAN and Bluetooth	–	–	·	–	ULCO-Ontology	Jena2	–
Active Bats	U3D	√	·	–	–	Ad hoc	–	–
Smart Floor	Pressure sensors	–	·	–	–	Ad hoc	–	–
CoBrA	Bluetooth and Ethernet	–	–	·	–	COBRA-ONT	Jess	–
MAX	RFID	√	–	·	√	Ad hoc	–	–
Home-Explorer	U3D and MOTE	√	–	·	√	SS-ONT	SWRL and Jess	√

6.3. Discussion

A survey of comparison of our work with other related systems mentioned in the introduction appears in Table 2. Different sensing techniques are used by them to acquire information from the physical world, including several different ways to locate physical entities (e.g., people, artifacts and devices), they are, U3D, RFID and pressure sensors for locating people and smart artifacts, and the Bluetooth and Ethernet technology for locating smart devices (e.g., mobile phone, Fax). Different locating technique has different merits and limitations, comparison of them lies outside the scope of this paper, detailed discussions can be found in Hightower and Borriello [17].

Besides *Home-Explorer*, smart artifacts are also the focus of several other listed systems, however, all these systems are based on ad hoc context infrastructures, which makes it difficult to share and reuse knowledge. Furthermore, the context reasoning tasks of them are implemented at the programming level, that's to say, they don't use any inference engines, which, inevitably, reduces their extendibility and increases their maintenance costs. Among the smart artifact systems listed in Table 2, the hidden object problem (or the robustness of sensor network) is only considered in our system.

CoBrA and Semantic Space are two only systems that use normalized ontology as their underlying context infrastructure, they also provide effective context reasoning mechanisms by using inference engines. However, compared to our system, these two systems focus more on the contexts acquired from smart devices or software applications, while the contexts from sensor equipped everyday artifacts are little concerned. As rule-based context reasoning systems, we three systems face the same challenging issues yet to address, especially the performance and time complexity of context reasoning in the presence of large-scale smart spaces and a vast number of rules. It has been reported in Wang et al. [33] that the runtime performance of rule-based context reasoning also depends on the hardware configurations (saving about 50% runtime with a double-speed CPU). Therefore, we expect that, as the rapid development of the computer hardware technology, the runtime performance of logic-based context reasoning will behave better in a few years. Even under current hardware configuration level, we can also improve our system's performance by exploring the distributed computing technology or introducing new control algorithms which can trigger only the related rules when a certain context changes (currently all the rules are re-evaluated when the context of the smart space changes).

7. Conclusion

We have presented our efforts to incorporate Semantic Web technologies into physical artifact search system development. Building upon an explicit, flexible OWL ontology infrastructure, it is easy to implement knowledge reuse, knowledge sharing, and common context semantic understanding among the independently developed applications. The extended version of our ontology also gives a good modeling of both smart objects and hidden objects. Coupled with Semantic Web supporting tools like SWRL, a set of inference rules that derive smart object information or that detect hidden objects are defined. Based on the derived information, the system provides multiple search modes for humans to search these objects. The results of our experiments illustrate that our system can reflect dynamic physical object information well and greatly enhance the robustness of the sensor network.

There are a number of possible ways to enhance our system: First, since the contexts among objects are complex and changing, *Home-Explorer* can deal only with a few hidden object search situations. In future works, we hope to study more about common sense and human cognition to create more general rules. Second, as our experiment result shows, when the number of inference rules increases, the evaluation of these rules will take more time ($O(n)$ time, n as the number of rules) and some conflicting relations may be concluded from different rules (because common sense reasoning is imprecise). Therefore, strategies are needed to improve the efficiency of the system's inferences, as discussed by Tan et al. [16]. Nonmonotonic reasoning technologies are also needed to ensure the consistency of the system's inferences, as discussed by Antoniou and Wagner [10].

References

[1] A. Harter, A. Hopper, P. Steggles, A. Ward and P. Webster, The Anatomy of a Context-aware Application, in *Proceedings of MOBICOM1999*, 1999.
[2] A. Ranganathan and R.H. Campbell, A Middleware for Context-Aware Agents in Ubiquitous Computing Environments, in *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, Brazil, 2003.
[3] ARQ – A SPARQL Processor for Jena, http://jena.sourceforge.net/ARQ/, 2004.
[4] B. Guo, S. Satake and M. Imai, 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*, 2006, 191–194.
[5] B. Guo and M. Imai, Home-Explorer: Search, Localize and Manage the Physical Artifacts Indoors, in *Proceedings of the IEEE 21st International Conference on Advanced Information Networking and Applications* (*AINA-07*), Niagara Falls, Canada, 2007, 378–385.
[6] Built-Ins for SWRL, http://www.daml.org/2004/04/swrl/builtins.html, 2004.
[7] Crossbow Technology, http://www.xbow.com/, 2007.
[8] D.L. McGuinness and F. Harmelen, Web Ontology Language Overview, http://www.w3.org/TR/owl-features/, 2004.
[9] D. Salber, A.K. Dey and G.D. Abowd, The Context Toolkit: Aiding the Development of Context-Enabled Applications, in *Proceedings of CHI'99*, 1999, 434–441.
[10] G. Antoniou and G. Wagner, Rules and defeasible reasoning on the Semantic Web, in *Proceedings of RuleML-2003*, 2003, 111–120.
[11] H. Chen, F. Perich, T. Finin and A. Joshi, SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, in *Proceedings of MobiQuitous*, 2004, 258–267.
[12] H. Chen, T. Finin and A. Joshi, An ontology for context-aware pervasive computing environments, *The Knowledge Engineering Review* **18**(3) (2003), 197–207.
[13] H. Knublauch, *Protégé-owl API Programmer's Guide*, http://protege.stanford.edu/plugins/owl/api/guide.html, 2006.
[14] Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net, 2003.
[15] Jess, the Rule Engine for the Java Platform, http://herzberg.ca.sandia.gov/jess, 2004.
[16] J.G. Tan, D. Zhang, X. Wang and H.S. Cheng, Enhancing Semantic Spaces with Event-Driven Context Interpretation, in *Proceedings of the 3rd International Conference on Pervasive Computing*, Munich, Germany, 2005, 80–97.
[17] J. Hightower and G. Borriello, Location Systems for Ubiquitous Computing, *Computer* **34**(8) (2001), 57–66.

[18] K. Henricksen, J. Indulska and A. Rakotonirainy, Modeling Context Information in Pervasive Computing Systems, in *Proceedings of first International Conference on Pervasive 2002*, 2002, 167–180.
[19] M. Beigl, H.W. Gellersen and A. Schmidt, Mediacups: Experience with design and use of computer-augmented everyday objects, *Computer Networks* **35**(4) (2001), 401–409.
[20] M. Lampe and M. Strassner, The Potential of RFID for Moveable Asset Management, in *Proceedings of Ubicomp 2003*, Seattle, 2003.
[21] M.J. O'Connor, H. Knublauch and S.W. Tu, Supporting Rule System Interoperability on the Semantic Web with SWRL, in *Proceedings of 4th International Semantic Web Conference*, 2005.
[22] P. Lamon, A. Tapus, E. Glauser, N. Tomatis and R. Siegwart, Environmental Modeling with Fingerprint sequences for topological global localization, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, 3781–3786.
[23] R.J. Orr and G.D. Abowd, The Smart Floor: A Mechanism for Natural User Identification and Tracking, in *Proceedings of CHI 2000*, New York, 2000, 275–276.
[24] S. Meyer and A. Rakotonirainy, A survey of research on context-aware homes, in *Proceedings of ACSW frontiers 2003*, Adelaide, Australia, 2003, 159–168.
[25] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.daml.org/2004/04/swrl, 2004.
[26] SWRLTab, http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab, 2006.
[27] Tactile Force Sensors, http://www.tactex.com/kinotex.php, 2006.
[28] T. Berners-Lee, J. Hendler and O. Lassila, The Semantic Web, *Scientific American* **284** (2001), 34¤-C43.
[29] The Protégé Ontology Editor, http://protege.stanford.edu, 2006.
[30] V. Haarslev and R. Moller, Racer system description, in *Proceedings of the International Joint Conference on Automated Reasoning*, Springer, 2001, 701–705.
[31] W. Beer, V. Christian, A. Ferscha and L. Mehrmann, Modeling Context-aware Behavior by Interpreted ECA Rules, in *Proceedings of Euro-Par 2003*, 2003, 1064–1073.
[32] X. Wang, D. Zhang, J.S. Dong, C.Y. Chin and S. Hettiarachchi, Semantic Space: An Infrastructure for Smart Spaces, *IEEE Pervasive Computing* **3**(3) (2004), 32–39.
[33] X. Wang, D. Zhang, T. Gu and H. Pung, Ontology Based Context Modeling and Reasoning using OWL, in *Proceedings of PERCOMW'04*, 2004, 18–22.

Bin Guo received the BS and MS degrees in computer science from Xi'an JiaoTong University, China, in 2003 and 2006, respectively. He is currently a Ph.D. candidate of the Department of Information & Computer Science at Keio University, Japan. His research interests include semantic sensor network, context-aware computing, semantic web and smart home control.

Satoru Satake received the B.E. and M.E. degree from Keio University, Japan, in 2003 and 2005, respectively. He is currently a Ph.D. candidate of School of Science for Open and Environmental Systems, Graduate School of Science and Thechnology, Keio University. His research interests include knowledge management for sensor networks and ubiquitous environments.

Michita Imai received his Ph.D. degree in computer science from Keio University, Tokyo, Japan, in 2002. In 1994, he joined NTT Human Interface Laboratories. He joined ATR Media Integration and Communications Research Laboratories in 1997. He is currently Assistant Professor, Faculty of Science and Technology, Keio University, and a Researcher at ATR Intelligence Robotics and Communications Laboratories, Kyoto. Dr Imai's research interests include autonomous robots, human-robot interaction, speech dialogue systems, humanoids, and spontaneous behaviors. He is a Member of the IEEE, the Information and Communication Engineers Japan (IEICE-J), the Information Processing Society of Japan, the Japanese Cognitive Science Society, the Japanese Society for Artificial Intelligence, the Human Interface Society, and the Association of Computing Machinery.