Journal of Mobile Multimedia, Vol. 2, No. 2 (2006) 124–145
© Rinton Press

BAZAAR: A MIDDLEWARE FOR PHYSICAL WORLD ABSTRACTION

KAORI FUJINAMI and TATSUO NAKAJIMA
Department of Computer Science, Waseda University
{fujinami, tatsuo}@dcl.info.waseda.ac.jp

Received March 1, 2006
Revised May 1, 2006

In this paper, we propose a middleware, *Bazaar*, for building location and context-aware services without the need to consider the detail of information capturing, but to allow a developer to concentrate on his/her main task, i.e. the application logic development. *Bazaar* abstracts the physical world by separating the structures of information and the usage from utilization. It also serves as a shared "physical information repository" to maintain consistency among various applications in the environment, which is often called the *world model.* The world model in *Bazaar* contains a *location model* and an *artefact model.* The former represents the containment relationship between a unit space like a house and unit regions like the washroom. The static specification and dynamically changing status of an artefact is handled within the artefact model. Here, a sensor augmented artefact acts as a building block for a smart space. Usually, the artefact has a primary role and we have a prior understanding of the meaning of the state-of-use. We call such an artefact a *sentient artefact.* This paper describes the design and implementation of *Bazaar* with a programming model that reflects the relationship between locations and objects in the physical environment. Furthermore, we describe experiences from various application developments with Bazaar.

Keywords: Ubiquitous Computing, Context-Awareness, Sentient Artefact, World Model, Middleware
Communicated by: I. Ibrahim

1 Introduction

The advancement of technologies, such as wireless communication and downsized computation affected by high performance, allows intelligences to be spontaneously added into an everyday living space and connected us networks[9]. Such computing environments are often referred to as a ubiquitous computing environment[28], and they have been investigated since early 1990's. In ubiquitous computing era, the notion of context-awareness plays an important role, and it is one of the exciting research topics[7][21][20]. A system that is aware of its operating situation, i.e. context, can adapt its behavior to the user. We consider that this makes two major contributions to our everyday lives. One is assisting us by extracting relevant information from the information overloads. For example, let us consider a case in which networked room lights are installed into an office and a user staying in one room wants to move to another. The user needs not to be aware of the names or network addresses of the lights to turn off and on. Instead, he/she should only move to the room, and a context-aware room light controller should do the rest. The other is enriching our daily living with value added services. For example, if someone forgets to take an important thing that is required for him/her based on the sensed context, a context-aware memory aids is able to alert that it is left behind[13].

Contextual information is required to be extracted as implicitly as possible through interaction between users and surrounding environments. This leads to the need for a capability to handle a wide variety of contextual information from the physical world, e.g. the existence on particular location[16], co-locating[4], the state of use of an everyday object[14], and so on. Here, an everyday object (artefact) usually has a primary role, and we have a prior understanding of the meanings of the state-of-use. Therefore, we consider that an artefact becomes a building block to build a smart space. We call such a sensor augmented artefact a *sentient artefact*[11].

Moreover, the accuracy of the extracted information is important. If the information corresponds to the real situation and the user's experience, the user will feel less burdensome when the system acts autonomously. Furthermore, various different applications in a physical space can often consume the same kind of contextual information, so the consistency of the information is required. Such information should neither be extracted by a single application nor used only by itself. Therefore, it should be managed apart from the application itself and shared with others.

In this paper, we propose a middleware, *Bazaar*, so that a developer can concentrate on building location and context-aware application logics. *Bazaar* encapsulates the structure of the information about the physical world. Such information is often called the *world model.* The information is shared with different applications so that they can refer a single view of the world to work consistently. Furthermore, the developer can share the information with an application to build a smart space easily as if he/she were in the space. The world model contains a *location model* and an *artefact model.* The former represents the containment relationship between a unit space like a house and unit regions like the washroom. The static specification and dynamically changing status of a sentient artefact is handled within the artefact model. The design and implementation of *Bazaar* is described with a programming model. Furthermore, we report the experiences from various application development experiments.

2 Smart Space with Sentient Artefacts

In this section, we describe the motivation for building a sentient artefact-based smart space with a concrete scenario.

2.1 A Scenario

The following scenario illustrates how the appropriate information and services are provided at a proper timing in a daily life.

A Tokyo resident, Hanako, made an appointment to meet her friend in Yokohama. She registered the appointment with the schedule management application running on her cellular phone, and she put it on her favorite phone cradle. She went into bed after setting the alarm clock on the bedside. She received a call during the night, but as her cradle knew she was sleeping, the call was not notified with a sound. On the next morning, she woke up before the *alarm and went to the washroom to brush her teeth. The alarm was cancelled automatically. During brushing her teeth, information was presented on the surface of the mirror and she noticed that the weather in Yokohama would get worse and that there had been a railway accident on the train line she was thinking to use. Therefore, she decided to change her clothes as well as the path to go there. After she returned to her bedroom, she noticed that the cradle was dazzling to tell her about the call received. In the night, she met her younger sister and she was asked to exchange their cradles because her cradle was so cute.*

2.2 Sentient Artefact-based Smart Space Building

The above scenario can be realized by existing technologies. Location sensing systems that provide highly accurate information help the applications to extract more abstract contextual information than presence at a specific location. However, they require complex infrastructures embedded into the environment from the very beginning of the construction, and this increases the deployment and management cost[15]. Also, new type of devices that require the user to learn their usage might provide him/her a cognitive burden.

To address these issues, we are working on augmenting daily objects with computing capabilities, such as sensors and actuators. We call the daily object a "*sentient artefact*"[11]. We use a sentient artefact as a daily object that has inherent and ordinary functionalities. In addition, it is utilized as a "context sensor" that detects its state-of-use as a basis to extract more abstract context, which is natural because every artefact has its specific role (or characteristic) and the state-of-use should reflect it. In the above scenario, an alarm clock is utilized as an ordinary alarm clock. However, a system can additionally perceive whether the user is sleeping or not, and it can change its behavior according to the contextual information. We also consider that various services that utilize a user's context will be installed into our surrounding environments in the near future. This development moves us away from only using traditional computers. In this case, the artefacts can be utilized as outputs by leveraging their natural understandings, e.g. a mirror[10]. Table 1 shows that the specific application functionalities in the scenario are realized using the artefacts that have the specific capabilities. The important things here are that every artefact is responsible for a specific functionality of the application and that an artefact is replaceable with another that has the same role or characteristic. Thus, we believe that the sentient artefact approach allows a developer to build context-aware applications easily. Furthermore, from the user's point of view, he/she can utilize a context-aware service implicitly and naturally through the interaction with various sentient artefacts.

Table 1 Artefacts as Building-blocks

Application Functionality	*Role or Characteristic*	*Artefact*
Call notification while awake	Detection of "sleeping"	Alarm clock
	Detection of the exclusive tasks	Toothbrush
Information provision during tooth brushing in front of a mirror	Identification of the user	Toothbrush
	Provision on the periphery	Mirror

A sentient artefact is expected to play a key role in realizing a ubiquitous computing environment in a practical way. The advantages are, as follows:

- A smart space can be incrementally built by the growing numbers of sentient artefacts based on the requirements of an application.

- The user does not need to learn the artefact's usage because it keeps its metaphor and does not change its original functionalities.
- The artefact can extract the user's context implicitly and naturally through the original usage.

2.3 World Model

To be benefitted from the characteristics of the sentient artefact, we should introduce the notion of the *world model* [16][19]. The term *world model* is utilized to represent structured information related to the physical world and the computational entities existing there. Now, we describe the role of a world model consisting of sentient artefacts.

2.3.1 Shared Knowledge between Applications

Different applications might need to access the same artefact in the environment. Let us think about the characteristics of a toothbrush; it is not utilized during sleeping and it is not shared with others. One application might consume the state-of-use as a cue of the user's *waking up*, while another wants to know the *user* of the toothbrush. To keep the information utilized by both applications consistent, it should be shared with applications. Moreover, combining information from multiple artefacts can make an application more reliable in terms of context inference, as well as extracting higher level knowledge. The user's state *sleeping* will become more reliable when information from the alarm clock and from the toothbrush are combined. These two examples indicate that a shared model of the world is required. Otherwise applications have to extract contextual information by themselves even if there are two applications that utilize the same information. This is burdensome for the developer, and there is a chance that the user is provided incorrect information/service because of the low reliability and inconsistency.

2.3.2 Shared Knowledge between a Developer and the Environment

In addition to the advantage of sharing information among applications, the developers can benefit from the similarity to the world in terms of the structure they live in. They can easily find and use an appropriate component for building an application, e.g. physical object, sensor, and context, by physical attributes, e.g. location, color, or state of object of interest. For example, if a query like "select *states* from the *mirror* where *the location is the washroom*" is put to the world model, a software component which represents the states of the *mirror* located in the *washroom* is expected to return. This requires neither the knowledge of the IP address nor the ID of the mirror. Instead, this utilizes highly abstract information which is familiar to a person living in the physical world. The developer can build an application as if he/she were in the application space.

3 Middleware Design

The developer needs to access to the world model without taking account of the detail of sensing and extracting context, executing an actuator (optional), getting information from location systems, etc. In this section, we describe the design of a middleware called *Bazaar* that provides abstractions suitable for dealing with sentient artefacts distributed in indoor environment.

3.1 Requirements

A list of requirements for the world model is as follows:

1. Wide variety of shared information
2. Extensibility of the information
3. Minimum level of sharing
4. Extensibility of the smart space
5. Transparent use of non-sentient artefact's information

The first requirement means that various information can be used as contextual information and some of them need to be shared. Such information includes the dynamic features like location and state-of-use as well as the more static information, e.g. the type of an artefact, the owner, etc. For example, in the previous scenario, the owner and the user of a toothbrush can be linked because of the fact that a toothbrush is hardly ever shared with others. During the long life span of actually using a smart environment, the amount and types of information is expected to increase. This requires the extensibility of the shared information (the second requirement). The third requirement indicates that a minimum level of information should be shared and a mechanism should be provided so that the developer can easily compose the application specific information. For example, the context "sleeping" should not be shared because the meaning differ from application to application. This requirement can avoid making the structure of the shared information complex. The fourth requirement calls for the extensibility of the smart space, which is crucial for our sentient artefact-based approach. In this approach, we do not need to install any dense location sensing infrastructure from the beginning for each room or building. Instead, appropriate sentient artefacts and their identification system are added in an on-demand basis. Finally, the fifth requirement is important because non-sentient artefact that has no computational capability can be utilized as a landmark like "in the same area as Hanako's bag".

3.2 Architectural Design

In this section, we describe the overall architecture of *Bazaar* and the representation of the world model with an example.

3.2.1 Prerequisites

We assume that every artefact is assigned with an ID observable by an external entity. The detailed information about the artefact that is linked to the ID is provided by the manufacturer. There could be various schema to represent the information. However, in our model, it is unified or transformed into a single representation before the integration by a translation service or a standardization process. A central processing entity, such as a home gateway, a set top box, etc., is responsible for managing the world model and providing higher level access to the applications.

3.2.2 Components

Figure 1 shows the overall architecture of *Bazaar*. *Bazaar* consists of six major parts. This includes, 1) the identifiable object (artefact) as a source of low level contextual information, 2) ID detector that serves as a location identifier, 3) the Bazaar World Model Manager(BWMM) that constructs and maintains a certain unit space, 4) the context extraction framework that interprets and makes the low level contextual information available to applications as highly abstract information, 5) the application logic that a developer has to develop, and 6) the IDResolver that resolves the location of the file describing the detail of the artefact on the Internet. It is similar to Auto-ID's Object Name Service (ONS)[2].

Bazaar's core concept is the representation of a physical world with self-descriptive objects. The detailed information, i.e. self-descriptive information, is obtained from the manufacturer's site once the ID assigned to a specific artefact is detected by an IDDetector. The name of the location that the detector is installed to is then interpreted as the location of the artefact and the descriptive information is integrated into the Bazaar's world model. This allows the developer to scale-up the size of the space very easily, which is pointed as the fourth requirement. Moreover, this method can integrate non-sentient artefact seamlessly (the fifth requirement). *Bazaar* also provides a programming model that is specific to the sentient artefact-based application development, which we will describe in section 4. In the next section, we show the world model consisting of a location model and an artefact model.

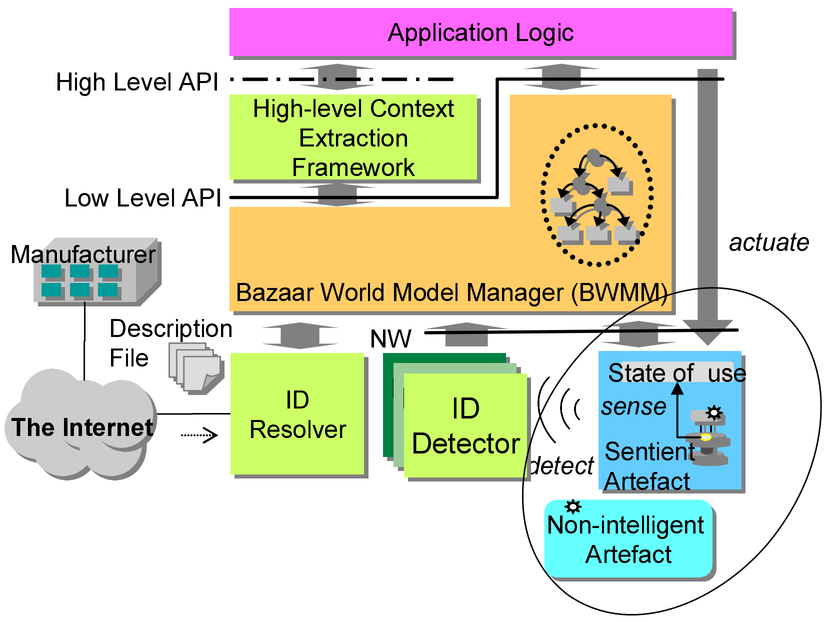

Fig. 1. Overall architecture of *Bazaar*

3.3 World Model Design

3.3.1 Representation by RDF: A Brief Introduction of RDF

Resource Description Framework (RDF)[18] that is getting much attention in the semantic web technology[25] provides interoperability between computational entities and utilized for the representation of the world model. In RDF, every attribute can be simply represented as a form of triples: a *subject* (or *resource*), a *predicate*, and an *object*. For example, a statement like “the mirror located in the washroom is being used by someone” can be represented as follows.

```
(urn:abc001, v1:type, "mirror")
(urn:abc001, v2:detectedBy, urn:detector:b)
(urn:detector:b, v1:name, "washroom")
(urn:abc001, v3:state, urn:state:m)
(urn:state:m, v3:current, "appeared")
```

The first column is a *subject*. Here, “`abc001`” in the *subjects* is the ID of the mirror. The second and the third are a *predicate* and an *object* which stands for the name and the value of the attribute, respectively. In an RDF representation with triples, an *object* needs not only to be literal value, but it can also be a *resource* pointing to another triple. This allows the description to be extensible in a recursive manner. Furthermore, the manipulation is easy as well as the expression is simple and powerful. In case of adding a new attribute like “the owner is Hanako Yamada”, only one row needs to be written:

```
(urn:abc001, v1:owner, "Hanako Yamada")
```

In addition, RDF allows powerful searching methods that utilize pattern matching. For example, a query like, “What is located in the washroom?” can be queried with three steps: 1) search the *resource* of the location `washroom` and get the answer `urn:detector:b` (the third row), 2) find an object whose location is `urn:detector:b` and get `urn:abc001` (the second row), and 3) get the final answer `mirror` by obtaining the *object* of the *predicate* `v1:type` (the first row).

Thus, representation by RDF provides strong expressiveness while the syntax is quite simple. Our first and second requirements are met due to the flexibility and powerful characteristics of RDF. However, the semantic transcoding is quite a challengeable issue. In the above statement, three vocabularies are used: `v1`, `v2`, and `v3`. They exist as a prefix in the *predicate* of each statement and indicate namespaces. If two same kind of objects have different vocabularies, for example `v2:location` and `v4:loc` as a location of an object, then a software entity should internally exchange one to another like a translator to encapsulate the difference. Furthermore, there is a more complex case with 1-to-N relationship. As assumed in section 3.2.1, our model does not consider this[a]. In the following two sections, both the location model and artefact model are presented.

[a]The predicate should also be represented with an appropriate namespace, however, we omit this in the latter part of this paper for simplicity.

3.3.2 Location Model

As can be seen in Figure 2, the location model in *Bazaar* represents the containment relationship between a unit space, e.g. room, building, and unit regions identified by symbolic names. Any place that has a significant meaning to an application can be a unit region: the washroom, Hanako's room, the entrance, around the kitchen table, etc. The unit region is defined for each IDDetector, and the detected physical object (artefact) is linked to the IDDetector that detects the artefact. Thus, the location of the artefact is fixed and represented in the world model.

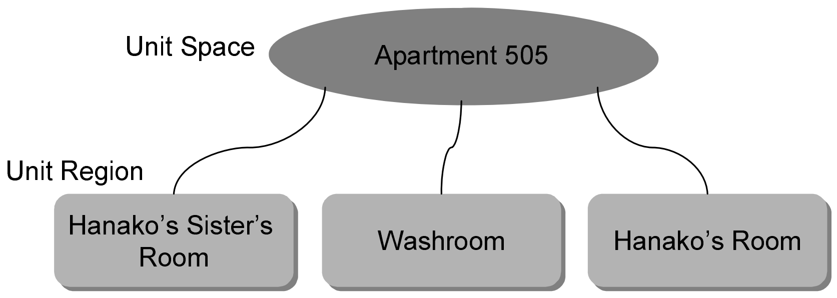

Fig. 2. The relationship between a unit space and unit regions

Figure 3-a) illustrates an example of the location model by the RDF graph representation.[b] The graph indicates that three IDDetectors exist in a unit space (apartment) named 505, and they correspond to the three unit regions representing "Hanako's sister's room", the washroom, and "Hanako's room". Additionally, an artefact identified by `urn:abc001` is detected by the IDDetector linked to the washroom, while `urn:abc002` and `urn:abc003` are in Hanako's room. Our location model excludes the topological relationship like neighborhood between unit regions and the containment between unit spaces because of the simplicity. The latter means that one BWMM can manage more than two unit spaces, however the relationship is defined outside the BWMM. It should be dealt as an application specific matter.

3.3.3 Artefact Model

The artefact model represents the static specification and dynamically changing status of an artefact. Six types of static information are specified as the low level shareable information: the type, a state-of-use, a location (detector), the specification of the actuation functionality, the owner, and the network information (IP address). An example of the RDF graph representation is shown in Figure 3-b), which indicates that "The owner of the mirror with ID (`urn:abc001`) located in the washroom is Ms. Hanako Yamada. It has an actuation functionality with commands: `turn_on` and `turn_off` to show personalized information. The `turn_on` command accepts the user ID as an argument by *String* and returns the result by *boolean*. Now, someone is in front of it."

In the model, one artefact might have more than two locations at a time since this could

[b]In Figure 3, the oval, arrow, and ractangle indicate *resource*, *predicate*, and *literal*, respectively. The source of the arrow represents *subject*, while the destination *object*.

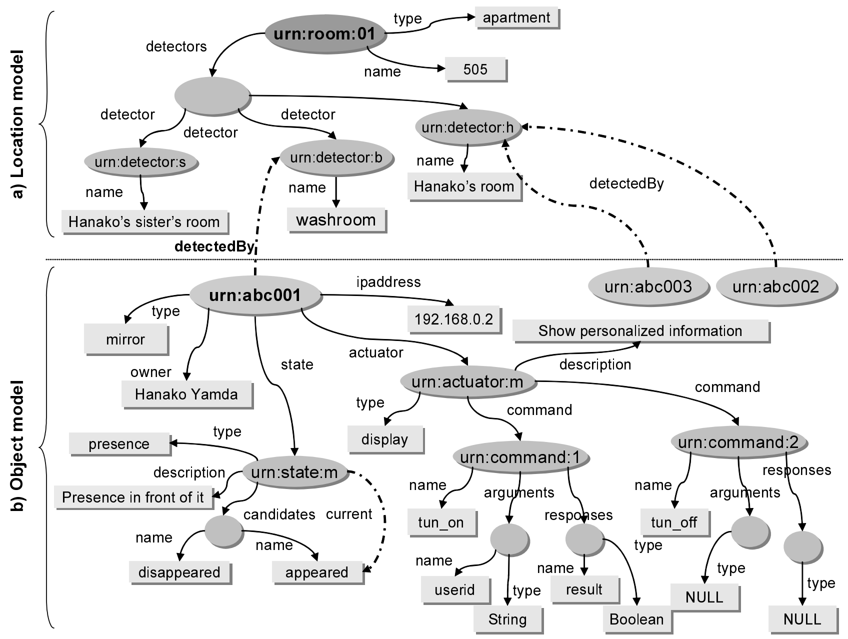

Fig. 3. A World model by the RDF graph representation: a) Location model and b) Artefact model

happen near the border of the detection ranges of IDDetectors. We consider it is a fact of the physical world and it should be handled on the application side in an appropriate way. Also, more than two state-of-use can belong to one physical object. For example, a chair might have both "seat state" and "direction of the seat back" as the state-of-uses. Furthermore, information to access a sentient artefact might be set to an artefact that has an actuation functionality. This includes the name of the actuation command, the name and the type of the argument, and those of the return value. In addition to requesting the callee artefact a specific actuation, these types of information are utilized to validate the parameter that a developer sets and the value that the callee returns.

These are illustrated as the nodes under `urn:actuator:m` in Figure 3. In terms of the *owner* attribute, it is replaceable with the "user" as previously described. Finally, the network address is utilized to access the sentient artefact from the application and BWMM sides. The links of the state-of-use and the location are dynamically changed on the detection of the change (the dotted lines in Figure 3), while other information is updated (rewritten).

4 API Design

In this section, the application programming interface (API) is designed.

4.1 Programming Model

As described in the previous section, the sentient artefact-based approach aims at extracting more information than the location specific context. Also, information/service provision is done through sentient artefacts. Therefore, the API needs to be designed to support the developer for this purpose. The utilization of the API is separated into two types: 1) "*Bazaar*-driven" and 2) "application-driven".

4.1.1 Bazaar-Driven Access

This is often referred to as an event-driven model, where the application is notified of an event if it registers the interests in a specific change of states, i.e. state-of-use and location. Figure 4 shows three types of events generated by *Bazaar*: ES, EL1, and EL2. The change of the state-of-use is generated by an artefact (ES). Regarding the location specific event, the following two types are supported: EL1) the change of the location of a specific artefact, and EL2) the change of contents at a specific location. For example, in the case EL1, the event is published when Hanako's toothbrush is taken from the washroom to Hanako's room. For EL2, the detection of whether a toothbrush is in the washroom generates the event. However, the reference to the software object corresponding to the event source needs to be obtained in a way that is described in the next section.

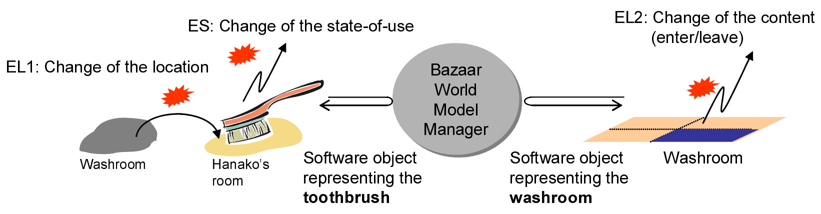
ES: Change of the state-of-use
EL1: Change of the location
EL2: Change of the content (enter/leave)
Bazaar World Model Manager
Washroom
Hanako's room
Software object representing the **toothbrush**
Software object representing the **washroom**
Washroom

Fig. 4. The event model in *Bazaar*: ES and EL1 are generated from an artefact. They are indicating the change of the state-of-use (ES) and that of the location (EL1), respectively, while EL2 is raised on the detection of an artefact's entering/leaving.

4.1.2 Application-Driven Access

To provide an application the way to obtain the information or the reference to a specific software object, the API contains so-called "*getter*" methods. Especially, the objects representing an artefact and location are the basis of the application programming and should be obtained in the early stage. They are shown as *resources* represented with `urn:abc`*`XXX`* and `urn:detector:`*`X`*, respectively. After obtaining them, the other types of information are retrieved through the appropriate *getter* methods, where the information represented as *resources* in Figure 3 is provided with inherent classes, and *literals* (rectangular form) are represented with *String* or a primitive type in Java. The latter includes, for example, the owner and the type of the artefact.

4.2 API

A generic class is provided for each type of the *resource* depicted in Figure 3 since the artefact model is represented in a unified schema. So, the class includes `World`, `Artefact`, `Location`, `State`, `Actuator`, and `Command`, where they correspond to a unit space, a sentient/non-sentient artefact, a unit region, a type of state-of-use, an actuation functionality, and a control command to the actuator. The concrete information that is depicted in Figure 3 is stored as a *literal* in the instantiated object, and accessed by a generic *getter* method. Therefore, the API can handle the increasing number of the attributes for a long period of time, which allows the extensibility of the information listed as the second requirement in section 3.1. In the following sections, typical methods are introduced.

4.2.1 `World`: The Class of a Unit Space

This is the entry point to obtain the basic classes, i.e. `Artefact` and `Location`, by a wide varieties of attributes from BWMM. We consider that this sort of *interaction* between the developer and BWMM is easy to understand because it is similar to the action of searching objects in the physical space. The methods with the postfix `ByAttrs` indicate that they return the required information by handling the passed arguments in a hash table, where the multiple elements are combined with the logical-AND operation in a query condition. This type of methods also exist in the other classes, however, we omit it.

```
Location[] getLocationsByAttrs(Hashtable a)
Location[] getLocationsByID(String id)
Artefact getArtefactByID(String id)
Artefact[] getArtefactsByLocation(Location l)
Artefact[] getArtefactsByState(String s, String v)
Artefact[] getArtefactsByAttrs(Hashtable a)
```

4.2.2 `Location`: The Class of a Unit Region

This class represents the notion of the unit region. It provides methods for adding/removing the callback `DetectionListener` for the event notification as well as obtaining the symbolic name. The callback method is invoked when something enters to or leaves the location, and the `DetectionEvent` is thrown with the name of the location and the time of the event.

```
void addDetectionListener(DetectionListener dl)
void removeDetectionListener(DetectionListener dl)
```

4.2.3 `Artefact`: The Class of an Artefact

This is the class representing the artefact model, where the methods for acquiring the dedicated references to the software objects in the model , i.e. `Location`, `State`, and `Actuator`, are provided. Also, the methods for special attributes like the name, the type, and the owner, are included. Additionally, it contains methods for adding/removing the callback `LocationListener`. The listener is called when the location of the artefact is changed. The method for removing the callback is useful when the application is only interested in the existence at a specific location. The application removes it after it is notified about the change of the location so that it can avoid receiving unnecessary events.

```
State getStateByType(String t)
Location getLocationByName(String n)
Actuator getActuatorByType(String t)
void addLocationListener(LocationListener ll)
void removeLocationListener(LocationListener ll)
```

```
String getType()
String getOwner()
```

4.2.4 **`State`***: The Class of a State*

This class indicates a type of state-of-use of a sentient artefact, where every instance of the class contains the type of the state and the value at a certain point. For example, the type of the state-of-use of the mirror in Figure 3 is "presence", and the value is either "appeared" or "disappeared". An event `StateChangeEvent` is generated on detection of the change of the value. This class also provides the method to get the most recent value on demand as well as the methods for adding/removing an event listener `StateListener`.

```
void addStateListener(StateListener sl)
String getCurrentState()
```

4.2.5 **`Actuator`***: The Class of an Actuator*

This class is used to control an actuator. So, it has one or more references to `Command` objects internally, and it provides a method `execute` to invoke the commands. The method below takes two arguments: the name of the command as `String` and the parameter as `Hashtable`, and returns the response including name-value pairs of the results.

```
Response execute(String n, Hashtable p)
```

4.2.6 **`Command`***: The Class of an Actuation Command*

This class encapsulates the processing of a remote command execution from the developer by passing only contents as arguments. The network address registered on the artefact initialization stage is internally utilized to access it. Here, `Response` indicates the return value of the command, and is forwarded to the caller, i.e. `Actuator`.

```
Response execute(Hashtable params)
```

The API presented here is primitive and referred to as low level API in Figure 1. By leveraging this, high level APIs are to be developed: inferencing context that is based on the first order predicate, modeling the spatio-temporal relationship among events, etc.

4.3 Code Example

In this section, we show an example program code utilizing the API. The example is a part of the scenario introduced in section 2.1, which describes the following logic: "When the condition that someone is in front of the mirror in the washroom and a toothbrush is used there is satisfied, the information related to the user of the toothbrush is provided through the mirror." The possible situations are illustrated in Figure 5, where only the person 2 is provided the personalized information. The code is shown in Figure 6. The double-quoted character strings correspond to the *predicates* (arrow) or *objects* (rectangle) in Figure 3. Also, `w` indicates the reference to the unit space `World`. In this figure, the logics to handle a zero-result in querying and an exception are omitted.

Let us walk through the code. The method `init` is called on the first stage, where

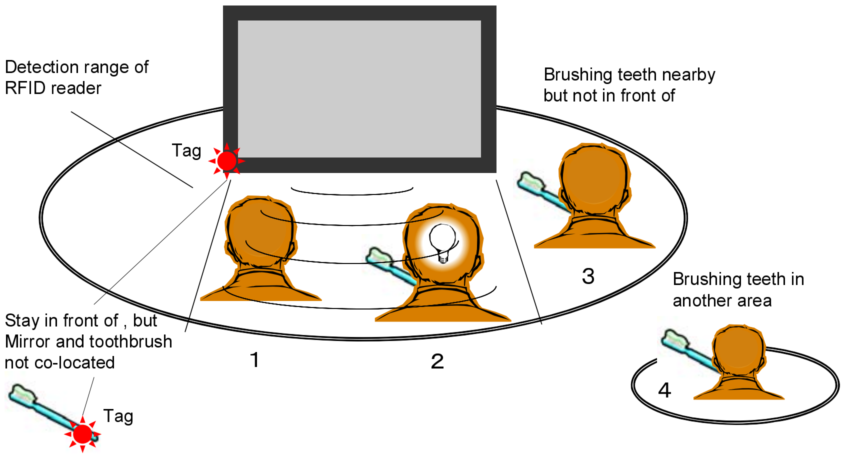
Detection range of RFID reader
Tag
Brushing teeth nearby but not in front of
3
Brushing teeth in another area
Stay in front of , but Mirror and toothbrush not co-located
1
2
4
Tag

Fig. 5. The possible situations among a toothbrush, mirror, and person. The personalized information is provided only when the person using a toothbrush is detected in front of the mirror. (case 2)

```
boolean appeared, brushing, lsnrAdded;
Artefact mrr, tooth;

void init(){
  Location wr = new Location(“washroom”);
  wr.addDetectionListener(this);

  Artefact[] m = w.getArtefactByLocation(wr);
  for( int i=0;i<m.length;i++){
    String artefactName=m[i].getType();
    if(artefactName.equals(“mirror”)){
      mrr=m[i].clone();
      mrr.addStateListener(this);
      break;
    }
  }
}

void onEntered(LocationChangeEvent e){
  Artefact a=w.getArtefactByID(e.getID());
  String type=a.getType();
  if(type.equals(“toothbrush”)&&!lsnrAdded){
    tooth=a.clone();
    tooth.addStateListener(this);
    lsnrAdded=true;
  }
}

void onRemoved(LocationChangeEnvet e){
}

void onStateChanged(StateChangeEvent e){
  String s=e.getType();
  String v=e.getChangedState();
  if(s.equals(“presence”)){
    if(v.equals(“appeared”) appeared=true;
    else appeared=false;
  }
  if(s.equals(“brushState”){
    if(v.equals(“started”) brushing=true;
  }
  if( appeared && brushing ){
    String owner=tooth.getOwner();
    showInformation(owner, true);
  }else{
    showInformation(null, false);
  }
}

void showInformation(String owner, boolean on){
  Actuator act=mrr.getActuatorByType(“display”);
  if(on){
    Hashtable arg=new Hashtable();
    arg.add(“userid”, owner);
    act.execute(“turn_on”, arg);
  }else{
    act.execute(“turn_off”,null);
  }
}
```

Fig. 6. Sample code using *Bazaar* Low-level API. This represents the logic “When the condition that someone is in front of the mirror in the washroom and a toothbrush is used there is satisfied, the information related to the user of the toothbrush is provided through the mirror.”

`Location` representing the washroom is obtained (line 4), and `DetectionListener` is registered with it for the notification of the change (line 5). Then, an array of `Artefacts` is acquired by the `Location` (line 6), filtered out objects for a mirror since there might be various types of artefacts in the location, and `StateChangeListener` is registered to be notified of the change of the mirror's state (line 7-11). These are the basic procedures in the initialization phase.

The methods in `DetectionListener` are `onEntered` (line16) and `onRemoved` (line 25), and they handle the event in the washroom. On the detection of *entering*, `StateListener` is registered only when the detected artefact is a toothbrush and it has not been registered yet. On the other hand, in `onRemoved` method, the removal of the registered listener and the control of the display functionality, `turn_off` need to be written, however, they are omitted here.

The method `onStateChanged` (line 27) is defined in `StateListener`, and it is utilized to handle the event indicating the change of the state-of-use. The type of the event source artefact and the changed value are obtained from `StateChangeEvent`, which is utilized to make a decision on the timing of displaying information on the mirror (line 37). If the condition in line 37 is satisfied, the owner is obtained (line 38) and `showInformation` is called, where the actuator for the display is acquired (line 45), set the required parameter (line 48), and invoked the displaying command (line 49). Thus, the programming on *Bazaar* is simple and easy to understand with the support of the API.

5 Implementation

In this section, we describe the implementation of *Bazaar*. Figure 7 illustrates the relationship between a sentient artefact, an IDDetector, BWMM, and an application.

5.1 Language and Operating Environments

BWMM is written in Java (Sun J2SE1.4 or higher), and as assumed in section 3.2.1, it runs on a central server like a home gateway, a set top box, etc. A sentient artefact discovers BWMM using multicasting, and they communicate each other via HTTP, which means the sentient artfefacts can be implemented by any language. An IDDetector also discovers BWMM and registers its information, i.e. the corresponding name of the location and the ID. We have utilized RFCode Inc.'s Spider active RF-tag reader as a location detection system. It has approximately 3 meter accuracy. However, as described before, any kind of location system that detects/recognizes the artefact's ID is applicable. The IDResolver component in Figure 1 is currently realized by looking up the table containing the artefact's ID and the URL of the descriptive file. However, if the ID is based on the Electronic Product Code (EPC) standards, the Object Naming System (ONS) proposed in the community is applicable. In the current implementation, the context-extraction framework depicted in Figure 1 does not exit. So, the applications described below were implemented using the low level API shown in section 4.1.

As described in section 3.3.1, both the location and artefact models are represented by RDF. We have utilized Jena2[23] as a library to parse, query, and manipulate the RDF-based models, where Jena2 internally processes the RDF Data Query Language (RDQL) [22]. There are several ways to encode RDF-based information like XML, Notation3[5], N-Triples[3], etc. However, we have utilized XML since it represents the structure of the model that is readable by human and highly expressive compared with others.

5.2 World Model inside *Bazaar*

The description file for each artefact is downloaded from the manufacturer's site once the artefact is installed into the environment. The RDF document is parsed into an internal tree model and continuously updated on receiving the events representing the change of the location and state-of-use. At the same time, the attributes are stored in the dedicated objects.

The duality comes from the flexibility of RDF-based querying and the usability of the inherent classes in *Bazaar*. The tree model is utilized to search the keys to get the requested objects from the *rmiregistry*, and also utilized to identify the node in the tree to update changes. The RDQL query mechanism is so powerful that the issuer can obtain the results by partial matching. However, the issuer needs to take into account of the structure of the tree, which makes the development complex. Therefore, we have decided to provide the inherent classes described in section 4.2. Certain *getter* methods, like `getArtefactsByAttrs` in `World`, internally issue the RDQL queries.

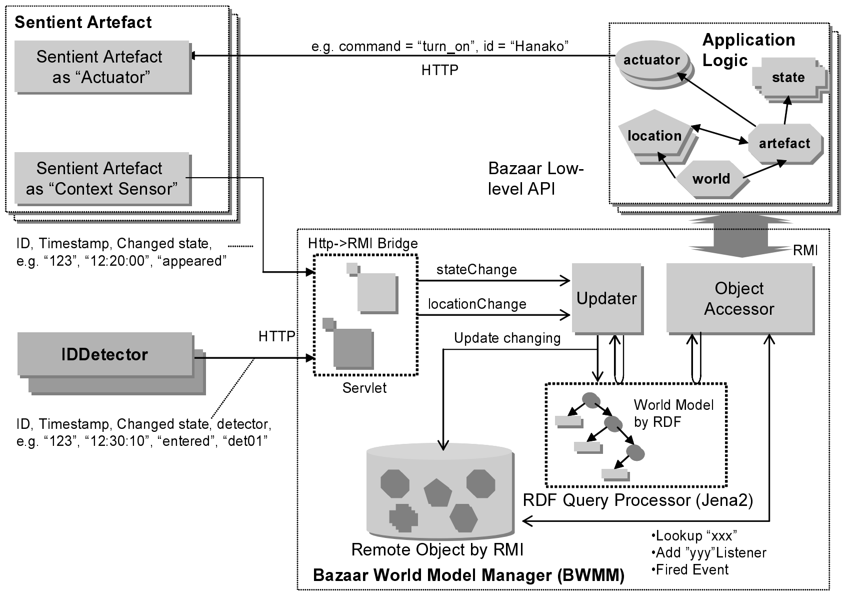
Sentient Artefact
Sentient Artefact as "Actuator"
Sentient Artefact as "Context Sensor"
e.g. command = "turn_on", id = "Hanako"
HTTP
Application Logic
actuator
state
location
artefact
world
Bazaar Low-level API
RMI
ID, Timestamp, Changed state, e.g. "123", "12:20:00", "appeared"
Http->RMI Bridge
stateChange
locationChange
Updater
Object Accessor
Update changing
IDDetector
HTTP
Servlet
World Model by RDF
ID, Timestamp, Changed state, detector, e.g. "123", "12:30:10", "entered", "det01"
RDF Query Processor (Jena2)
Remote Object by RMI
•Lookup "xxx"
•Add "yyy"Listener
•Fired Event
Bazaar World Model Manager (BWMM)

Fig. 7. Relationship between sentient artefact, IDDetector, BWMM, and applications

The updating of both the objects and RDF tree is handled by a component "Updater" in Figure 7. For example, in Figure 3 the value pointed by `current` is switched from `appeared` to `disappeared` when BWMM receives the event with the ID `abc001` and `presence` for the type of the state-of-use. The ID is utilized as the key to identify the corresponding `State` object in the *rmiregistry*, and then the change is notified to the application that registers `StateListener`. At the same time, the RDF tree is updated by identifying the corresponding node using an RDQL query.

The four classes in the API, `World`, `Location`, `Artefact`, and `State`, are implemented as remote objects in Java RMI. So, an application logic running on the different host can communicate with BWMM for the listener registration and the event notification.

6 Evaluation through the Application Development

In this section, we evaluate *Bazaar* as a tool to support the application developer with the abstraction of the physical world access. We have developed several applications on top of *Bazaar*. Here, we introduce two applications: "Unobtrusive Cradle" and "AwareMirror". The usage scenarios were introduced in section 2.1. Figure 8 illustrates the relationship between the two applications and *Bazaar*, where the information from a phone cradle, an alarm clock, a toothbrush, and a mirror, and their locations are shared in *Bazaar*. The two *integrated applications* control two actuators: the vibration of the cradle and the display of the mirror. Also, Figure 9 shows the deployment of the artefacts and a scene of using them.

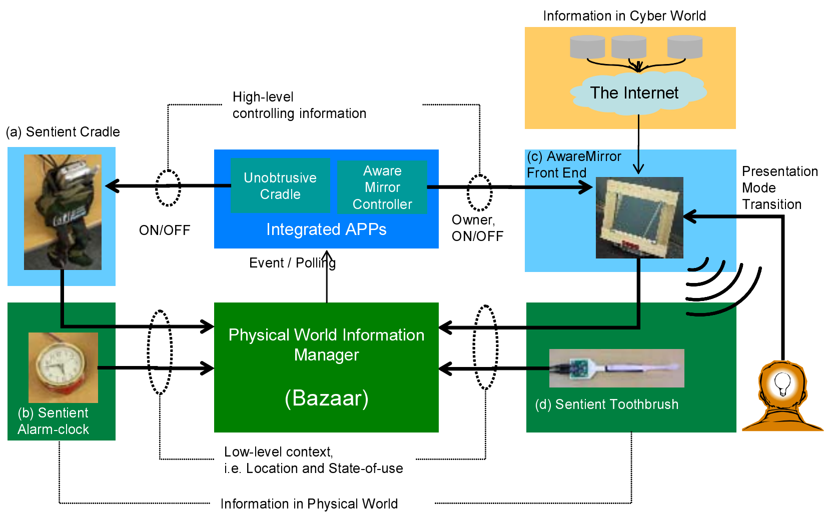
Information in Cyber World
The Internet
High-level controlling information
(a) Sentient Cradle
Unobtrusive Cradle
Aware Mirror Controller
Integrated APPs
ON/OFF
Owner, ON/OFF
(c) AwareMirror Front End
Presentation Mode Transition
Event / Polling
Physical World Information Manager
(Bazaar)
(b) Sentient Alarm-clock
(d) Sentient Toothbrush
Low-level context, i.e. Location and State-of-use
Information in Physical World

Fig. 8. The relationship between the two applications and *Bazaar*

6.1 Unobtrusice Cradle

An application "Unobtrusive cradle"[12] controls the actuation functionality of a cradle based on the user's interruptibility, i.e. *sleeping* or *not sleeping*. The cradle is augmented with two-axis accelerometer and starts vibrating on detecting a specific pattern of a call on a cellular phone. A servo motor controlling the cradle's leg movement makes noise. So, it should move only when its owner is not in sleep, which also contributes to reduce the power consumption. The context *sleeping* is defined by two complementary types of the state-of-use: 1) the state-of-use of an artefact that is not utilized during sleeping, and 2) the one that is used for the sleeping. We have utilized a toothbrush and an alarm clock for these purposes, respectively. Brushing teeth is "exclusive" activity during sleeping. On the other hand, an alarm clock is basically used for waking up at a certain time. Therefore, this common sense and prior understanding about the artefacts support extracting higher level information easily. The toothbrush we have utilized was augmented with two axis accelerometer to detect the shaking pattern, while the switch for the alarm is utilized to detect whether the alarm clock is used or not.

The issue in the development is to consider the "exchanging" with other persons. In the scenario, her younger sister uses Hanako's cradle. This means that the application cannot utilize the information of the cradle's owner to control the vibration because it might be utilized by the different user of the toothbrush. It might be the same case as the alarm clock. So, we have decided to pay attention to the unit region named "Hanako's room". Firstly, `Location` for the region is obtained, and then `Artefacts` representing the cradle and the alarm clock are retrieved using the object reference. This means that the *co-location* of the two objects is utilzied as can be seen in Figure 9-(a). In terms of the toothbrush, we have leveraged the pre-knowledge of the developer that the owner of the room is "Hanako", and thus we obtain the object representing the toothbrush and the owner is also "Hanako". However, the pre-knowledge is not machine-understandable although the name of the location is "Hanako's room" (see Figure 3). This means the "linkage" needs to be established by the developer. In Figure 3, appending a new *predicate* representing the owner, e.g. *owner*, to the *resource* of an IDDetector `urn:detector:`*N* can address this issue. This allows the application to obtain the toothbrush from `Location` without the knowledge of the developer. Thus, we can say that the two requirements: the heterogeneity of the shared information and the extensibility of the world model are satisfied.

An artefact like a cradle can be moved by someone during its long lifetime. So, the artefact becomes useless if the application is interested in the artefact at a specific location. Moreover, an artefact with the same type but with the different ID might be installed later, i.e. replacement. Therefore, the registered event listener for the state-of-use (`StateListener`) needs to be removed on the detection of the removal. And then, the developer should implement the procedures like: 1) finding alternatives, 2) waiting for the missing artefact, 3) notifying the user or administrator of the space, etc. We consider `LocationListener` supports this with the notification of the arrival/removal of the artefact.

In this application, *Bazaar* encapsulates the detailed ways to capture information of 1) the locations of the cradle and the alarm clock, 2) the owner of the toothbrush and the cradle, and 3) the state-of-use of the three artefacts. It also allows the developer to control an artefact, i.e. the cradle, without being aware of the distributed nature. The developer just has to retrieve the corresponding software object and invoke a method with the controlling parameters. The important thing here is that the higher level of activity context, i.e. "sleeping or not", can be easily extracted using the artefacts' state-of-use since their original roles help the system to narrow the candidates. Moreover, the redundancy of the low level contextual information can improve the confidence of the inferred context, where an erroneous extraction by a sentient artefact should be handle in an appropriate way. In the current implementation, information from the two sentient artefacts is simply processed by the logical-OR operator in the application logic. However, there could be a case that the command is "turn on" if one of them is misinterpreted that the user is "not sleeping". Therefore, a more sophisticated algorithm that resolves the conflict needs to be investigated. That might work either in the application logic or high-level context extraction framework layer that can be seen in Figure 1. The latter is, for example, an algorithm that utilizes the confidence parameter of the provided information with a threshold to select only the reliable one. In any case, *Bazaar* provides information for this purpose in a unified and consistent way.

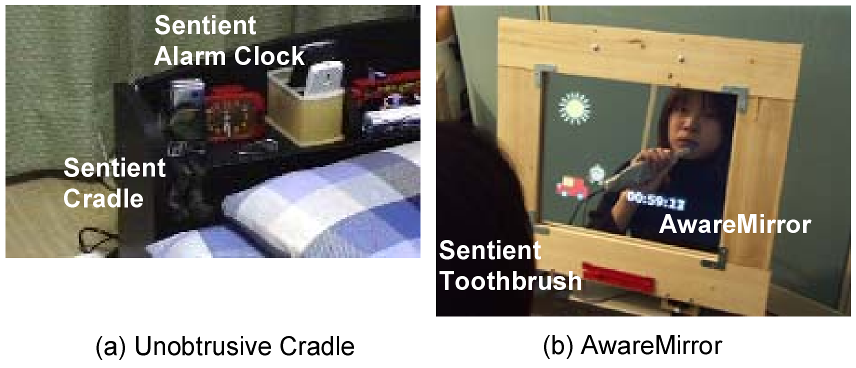

Fig. 9. Sentient Artefacts Deployment in the Prototype Applications: a) Unobtrusive Cradle, and b) AwareMirror

6.2 AwareMirror

AwareMirror (on the right side in Figure 9-(b)) is an augmented mirror that displays information relevant to the person in front of it on the periphery of his/her sights[10]. As can be seen in the scenario, the user can change his/her behavior through the information provided by AwareMirror in a very natural way. A fragment of the application logic and the figure illustrating the possible situations has already been provided in section 4.3. The state-of-use of the mirror is defined as the detection of something in front of it, which has been realized by two infra-red range finders in consideration of a feeling of privacy violation. Because of the low-level sensor data, it is not sufficient to identify the user by the mirror itself. The detection of utilization of a co-located sentient toothbrush (left in Figure 9-(b)) is utilized for this purpose, where the information is shared with "Unobtrusive Cradle" and kept consistent among them.

We consider that new attributes that indicates the *characteristic* and the *role* of an artefact make artefacts replaceable. Currently, the toothbrush is utilized for identifying the user, however a comb and a shaver are also applicable since they have the same characteristic, i.e. "non-shareable" personal items. So, in the method `onEntered` (line 16, in Figure 6), the application developer needs to list-up all the types of the possible artefacts that might appear in the future, which loses the extensibility of the application. On the other hand, the role of an artefact is considered to be the capability that the artefact can perform like "the identification of the user" and "the presence of an object". They can be easily applied to the current *Bazaar* implementation without any change in the program code. It is done by adding a *predicate* like `characteristic` and `role` into the artefact model. The information is obtained by using the generic method, `getArtefactsByAttr`, that accepts key-value pairs like. This realizes the heterogeneous and extensive information sharing.

For this application, another toothbrush was not installed into the space since "Unobtrusive Cradle" had already utilized one. So, the developer just needed to install an IDDetector in the washroom, and the location model was automatically extended by adding a new *object* for the *predicate* `detector`. The simple and high extensibility listed as the fourth requirement allows the smart space to "grow up" incrementally.

7 Discussions

7.1 Gap between the World Model and the User

Bazaar provides a world model that reflects the information in the physical space and computational entities existing there. The more accurate the information becomes, the better the application can act on behalf of the user. However, it is quite difficult to provide completely matched one since the model itself is designed by a person, which means it depends on his/her view of the world. Furthermore, the difficulty comes from the inaccuracy in the sensor data and the context, i.e. state-of-use, and its extraction algorithm. The issue is still an open research issue, where the minimum involvement of the user is crucial requirement.

7.2 Installation Policies of RF-tag Reader

As described previously, we have utilized an ID recognition-based system, i.e. RFID, to identify approximate position of a tagged object. Regarding the installation of the location system, there can be two options: 1) installation at a fixed position with no intention, and 2) installation into a place where a significant activity can occur. The former means the detectors are located in a certain order, e.g. lattice. We have utilized the latter one, namely the installation into a closed area "`Hanako's room`" and the "`washroom`". We believe that the approach is right since the installation is on-demand basis that allows the user to be provided the service at a minimal setting and cost.

7.3 Extension of the Unit Space

We consider that the flexible extensibility of the unit space is important for the large scale world model management, e.g. floor scale, building scale, campus scale, etc. Currently, the unit space is the top level notion in the world model, which means the number of the unit region and that of the artefacts in the regions are huge if the unit space is defined as a university campus. For load balancing and fault tolerance, the interconnection with the unit spaces that are distributed in different hosts is required. One way to realize this is what we call "micro World Model", where the unit space is defined as a limited set of information around the user or artefact, and it is stored on each device, e.g. the user's cellular phone, mirror, etc. In the "micro World Model"-based architecture, the operation is basically done against the local world model, and the query goes outside the *world* to obtain missing information.

7.4 Toward Reliable Context Extraction

A semantically rich model of the physical world allows an application to utilize reliable contextual information. For example, suppose that a single alarm clock is detected in two locations which are 20 meters away almost at the same time, e.g. within one second. In this case, one or both of them is considered to be incorrect detection. This can be detected if the world model has topological relationship and geometric value between detectors and a knowledge representing the impossibility to move so fast is utilized. However, we consider that such knowledge should not be handled within the world model since it is application specific. In addition, in case that a sentient artefact extracts contextual information with low confidence, it can improve this by asking appropriate one(s). Here, the world model is utilized to search suitable one(s) with many kinds of attributes. Thus, the world model supported by *Bazaar* is expected to provide an application with reliable context-awareness.

8 Related Work

Here, we examine related work regarding: 1) smart space construction, 2) representation of a physical world model in an infrastructure, and 3) API.

8.1 Smart Space Construction

We have utilized an augmented artefact, *sentient artefact*, as a building block for a smart space. To extract a user's context, location-based[1, 27], and image analysis-based[6, 17] approaches are well known. However, for the precise location detection, the detection system needs to be carefully installed, where the sensors are densely attached on the ceiling considering the effect of obstacles. Besides, it requires continuous maintenance, which costs a lot[15]. In case of image analysis-based approach, the advantage is that the person does not need to take anything with him/her for the ID detection. However, the sensitivity against the changes of the shape and background requires the system high computational power. Furthermore, the user might feel privacy obtrusiveness because of the "observing" nature.

On the other hand, our sentient artefact-based approach does not require precise location information since sentient artefact provides various information beyond mere state-of-use, which means the location information is not the primary context information. Therefore, as described in section 7.3, the location detector can be installed on demand basis, and thus it allows to reduce the total cost of ownership (ToC).

The MediaCups project [4] and its succeeding project of SmartITs[24] provide insights into the augmentation of artefacts with sensing and proecssing. The notion of artefacts computing composed of sensor augmented artefact provides a mean to obtaining human context implicitly, which has been greatly influenced the notion of sentient artefact. We are working on representing an artefact formally and integrating them systematically, which must be applicable to sentient artefacts based on the SmartITs platform.

8.2 Physical World Representation

An ontology-base approach is utilized to define an ontology that represents a meaning of a concept like a person, an object, a place, etc., and a relationship between these concepts. The approach tries to realize highly intelligent processing like reasoning and automatic adaptation. Among these, CoBrA[8] and Semantic Spaces[26] are infrastructures for building a smart space, where their own ontologies are developed so that the meaning of information can be annotated as a kind of tag. This differs from *Bazaar*'s approach in that the existence of the shared information repository. However, the integration of the ontological aspect into *Bazaar* allows a system to flexibly define the description files described in section 3.3. Moreover, the functionality/characteristic and replaceability introduced in section 6 can be defined, which allows a smart space to be built flexibly as well. The inference or reasoning engine is provided as a "high-level context extraction framework" that can be seen in Fig. 1.

8.3 API

Bazaar represents general concepts like an artefact, location as classes, rather than a particular type of things like a toothbrush, an entrance. On the other hand, The Sentient Computing project[1] targets an indoor application with the Active Bat location system. It has limited numbers of physical object model like people, computers, keyboards and telephones. It was implemented using Common Object Request Broker Architecture(CORBA), and thus the object modelling requires careful analysis. In contrast, *Bazaar* defines the artefact specific information only in the description file with generic methods to access the specific attributes. Therefore, it provides a system with robustness against the increasing types of information after installation.

9 Conclusions

In this paper, we proposed a middleware *Bazaar* to support the developer with the abstraction of the physical world access. We also introduced the notion of a sentient artefact as a building block of smart spaces. *Bazaar* manages a world model consisting of the two models: *location model* and *artefact model*. In the artefact model, various information like the type, the owner, the state-of-use, etc. are defined. The programming model provides a high level access to the world model encapsulating the detail of the device access and querying mechanism so that the developer can concentrate on the application logic development. We have showen the effectiveness and expressiveness of the proposed API. The architecture maintaining the world model can flexibly extend the smart space by defining a new attributes and adding a new unit region of interest, which is required to realize the true notion of ubiquitous computing.

References

1. M. Addlesee, R. Curwen, S. Hodges, J. Newman, A. Ward, and A. Hopper. Implementing a Sentient Computing System. *IEEE Computer Society*, pages 50–56, Aug. 2001.
2. Auto-ID Center. Web site:. `URL: <http://www.autoidcenter.org/>`.
3. D. Beckett and A. Barstow. N-Triples, W3C RDF Core WG Internal Working Draft. `URL: <http://www.w3.org/2001/sw/RDFCore/ntriples>`.
4. M. Beigl, H.-W. Gellersen, and A. Schmidt. MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects. *Computer Networks, Special Issue on Pervasive Computing*, 35(4):401–409, March 2001.
5. T. Berners-Lee. Notation3: Ideas about Web Architecture - yet another notation. `URL: <http://www.w3.org/DesignIssues/Notation3>`.
6. B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving: Technologies for Intelligent Environments. In *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, pages 12–29, September 2000.
7. G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, 2000.
8. H. Chen, T. Finin, and A. Joshi. Using OWL in a Pervasive Computing Broker. In *Proceedings of the Workshop on Ontologies in Agent Systems(OAS2003)*, pages 9–16, 2003.
9. Computer Science and Telecommunications Board. *Embedded, Everywhere, A Research Agenda for Networked Systems of Embedded Computers.* National Research Council, 2001.
10. K. Fujinami, F. Kawsar, and T. Nakajima. AwareMirror: A Personalized Display using a Mirror. In *Proceedings of International Conference on Pervasive Computing, Pervasive2005, LNCS 3468*, pages 315–332, May 2005.
11. K. Fujinami and T. Nakajima. Sentient Artefact: Acquiring User's Context Through Daily Objects. In *Proceedings of the 2nd International Symposium on Ubiquitous Intelligence and Smart Worlds*

(UISW2005), LNCS 3823, pages 335–344, December 2005.

12. K. Fujinami and T. Nakajima. Towards System Software for Physical Space Applications. In *Proceedings of ACM Symposium on Applied Computing(SAC) 2005*, pages 1613–1620, March 2005.
13. K. Fujinami, T. Yamabe, and T. Nakajima. "Take me with you!": A Case Study of Context-aware Application integrating Cyber and Physical Spaces. In *Proceedings of ACM Symposium on Applied Computing(SAC) 2004*, pages 1607–1614, Mar. 2004.
14. H. Gellersen, A. Schmidt, and M. Beigl. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. *Journal on Mobile Networks and Applications, Special Issue on Mobility of Systems, Users, Data and Computing (MONET)*, 7(5):341–351, Oct. 2002.
15. R. K. Harle and A. Hopper. Deploying and evaluating a location-aware system. In *MobiSys*, pages 219–232, 2005.
16. A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. In *Mobile Computing and Networking*, pages 59–68, 1999.
17. J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera Multi-person Tracking for EasyLiving. In *Proceedings of the 3rd IEEE Workshop on Visual Surveillance*, July 2000.
18. O. Lassila and R. Swick. Resource Description Framework(RDF) Model and Syntax Specification. `URL: <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>`.
19. D. Nicklas, M. Großmann, T. Schwarz, S. Volz, and B. Mitschang. A Model-Based, Open Architecture for Mobile, Spatially Aware Applications. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases: SSTD 2001*, pages 117–135, Jul. 2001.
20. B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, 1994.
21. A. Schmidt. Implicit Human Computer Interaction Through Context. *Personal Technologies*, 4(2-3):191–199, June 2000.
22. A. Seaborne. RDQL - A Query Language for RDF. `URL: <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>`.
23. Source forge. Jena-a semantic web framework for java. `URL: <http://jena.sourceforge.net>`.
24. The Smart-ITs project. The smart-its. `URL: <http://www.smart-its.org/>`.
25. W3C. Semantic web. `URL: <http://www.w3.org/2001/sw/>`.
26. X. Wang, J. S. Dong, C. Y. Chin, and S. R. Hettiarachchi. Semantic Space: An Infrastructure for Smart Spaces. *IEEE Pervasive Computing*, 3(3):32–39, July-September 2004.
27. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transaction on Information Systems*, 10(1):91–102, January 1992.
28. M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, pages 94–104, Sep. 1991.