Experiences with Developing Context-Aware Applications with Augmented Artefacts

Fahim Kawsar, Kaori Fujinami, Tatsuo Nakajima

***Abstract*—Context-Awareness is a key concept of future ubiquitous computing. Although various academic and industry researches are going on around the world to realize context-awareness, we are yet to achieve any substantial success. In this paper we have reported our experiences of dealing with context awareness. We have presented our approach for building context-aware applications by demonstrating a real life application development process. Furthermore we have presented two context-aware application frameworks that we have deployed in the roadmap of our research towards context-awareness. We have discussed our findings and learned lessons on various issues of context-awareness.**

***Index Terms*— Context-Awareness, Middlewares, Sentient Artefacts, Smart Environment.**

I. INTRODUCTION

UBIQUITOUS computing envisions a future environment that will be aware of its operating context and that will be adaptive to ease our interaction [1][3][9][24]. We approach such an environment by the environment itself. That means taking the building blocks of the environment and making them smart and context aware by capturing people's implicit interaction. We have been developing such building blocks, specifically everyday life objects by augmenting various kinds of sensors. We call them sentient artefacts. Our vision is to utilize these objects for value added services in addition to their primary roles. For example, consider a frying pan, its primary use is in the kitchen. However we can utilize the frying pan by augmenting it with some sensors/tags to infer that its owner is in the kitchen or he/she is cooking while the frying pan is being used. Usually these artefacts differ from the explicit sensors in three ways:

1) Sentient artefacts require a small operating software/device driver that captures values from multiple sensors embedded in the artefacts and process these values in a logical way to provide information about its state of use, properties or anything the software/driver author wants to provide.

Manuscript submitted June 17, 2005.

Fahim Kawsar is a MS student in Computer Science Department at Waseda University, Japan (e-mail: fahim@ dcl.info.waseda.ac.jp).

Kaori Fujinami is an Assistant Professor of Computer Science Department at Waseda University, Japan (e-mail: fujinami@dcl.info.waseda.ac.jp).

Tatsuo Nakajima is a Professor of Computer Science Department at Waseda University. (e-mail: tatsuo@dcl.info.waseda.ac.jp).

2) Rather than providing an analog/digital sensor value, sentient artefacts provide a "statement" to the interested applications, for example a sentient chair may provide a statement like "The chair is being used by Rob".

3) Finally, sentient artefacts can also be actuators in some cases.

Some services like scheduler or weather forecast monitor etc. are also considered as virtual sentient artefacts. By augmenting sensors, we make these belongings (micro component of the environment) smart. Eventually this process recursively makes our environment smart and context aware in a bottom up approach. For context aware services we need to handle this environment by modeling them in multiple applications. These applications typically use various components of the environment. In our approach, these components are the sentient artefacts.

In this paper we have reported our experience on developing the context aware applications using these artefacts. We have specified the approach centered on the sentient artefacts that we have taken and the problems that we have faced during development process. For the ease of development we have written two context aware application frameworks from different points of view. These two middlewares' goal is to provide a seamless platform for application development automating many recurring tasks. However we could not achieve a substantial degree of success. In the paper we have reported the probable causes that limit our performance. For clearly demonstrating our approach, we have selected a context aware application and have shown the development process of that application. We have developed two versions of this application utilizing the two frameworks to identify the pros and cons of the frameworks. Rather than specifying the ideal requirements, we will focus mainly on sharing our experience of application development in the paper.

The remaining paper is organized as follows: In Section II we have specified the steps that we have followed for application development. Section III demonstrates our approach by presenting a hypothetical scenario followed by the capability requirements for the scenario functionalities and the implementation of the scenario. In Section IV we have presented two middlewares and their integration with the application. Section V discusses the problems and our

experiences. In Section VI we have cited the related works, finally Section VII concludes the paper.

II. Our Approach

In our lab we are constantly drawing real life practical scenarios. We use these scenarios as the design base for deploying the environment components with augmented sensors and for developing integrated applications to implement those scenarios. We believe capturing users' context implicitly by their natural interaction with the environment is a key issue for context awareness. Here natural interaction means interacting with natural interfaces like everyday objects. A natural interface activates the cognitive and cybernetic dynamics that people commonly experience in real life, thus persuading them that they are not dealing with abstract, digital media but with physical real objects. This results in a reduction of the cognitive load, thus increasing the amount of attention on content [28]. So our approach is to make the artefact aware but not to make the user aware of this fact by keeping the artefact's primary role and interaction technique intact. Users only use daily life objects in the way they are used to. However our infrastructure captures these natural interactions in order to generate user's context.

From our experience of development, we have identified the essential steps that we followed for building context aware applications. These are:

1) Fixing the goal and the target location of the application, that is *where and what services to perform*, *where and what information to provide* etc. It means drawing the application scenario.
2) Identifying the context information that is required for achieving the goal or implementing the scenario.
3) Identifying the source of the context information that is how to extract the context information from the target environment. This step can be further sub-divided to answer the following questions:
   a) What context sources are available in the environment that can provide the required context?
   b) How these sources are acquiring context? If sensors are used then how to model the context from the raw sensor data?
   c) Who is responsible for modeling the context, the application or the context source?
4) Identifying the suitable interaction technique for the end users and providing a suitable way for reflecting end users' preference.

These steps are not mutually exclusive and may overlap each other. The success of the application depends on satisfying them with the highest degree of precision. However from our experience we have found that it is very difficult to satisfy these steps. There are many issues that contribute to this complexity thus limiting the success of context-aware applications. In fact we cannot pick a single problem for solving the complexity. We will discuss in detail about this complexity in the discussion section.

III. Sample Application

The first step that we follow is to identify an application scenario. In this section we have presented a scenario, then we have demonstrated how we have implemented the scenario.

A. Another Morning for Binti

Binti is a broker at the New York Stock Exchange. During her daily morning routine in the bathroom, while she is brushing her teeth and putting on her make-up, her mirror provides all the information she needs to start her day. During these activities she can watch her daily schedule. Besides that she also sees what the weather will be like, so she can dress fittingly. Furthermore she finds out if the subway, which she usually takes from her house to the Stock Exchange, is running properly. The subway is often delayed or closed for maintenance, in which case the mirror shows her an alternative route making sure that she does not have to rush to be on time for the morning breakfast meeting with her team.

B. Scenario Implementation: What To Sense and How

Once we have drawn the application scenario, then we have to identify the context information and the capability requirements of the artefacts that we can use to extract the required contexts. This scenario requires a smart mirror to be installed in the washroom that can capture users context (specifically detecting user's identity and user's presence at mirror's location) and can present him/her with some useful information. We have deployed an application *AwareMirror* for the functionalities. In the following table we have summarized what is required to be sensed for the proper functionalities of the application and what sentient artifacts we have used to capture them:

Table I
Functionality mapping from requirements to sentient artefacts

	Scenario Functionality	Required Capability	Augmented Artefact Used
WASHROOM	Display weather, schedule, and transportation information related to the user on the mirror	Detecting user's presence and position.	Mirror augmented with proximity sensors
		Identifying user	Toothbrush as an authenticator of the user
		Extracting schedule, weather and transport information	Web service treated as virtual sentient artefact
		Displaying extracted information	Mirror augmented with acrylic magic mirror board

AwareMirror [16] is a smart mirror installed in the washroom as shown in Figure 1. In addition to its primary task of reflecting someone's image, it can also provide some useful information related to the person who is using the mirror. It uses two sentient artefacts: a mirror and a toothbrush. It also uses three web services to collect information about the users' schedule, transportation information and weather forecasting. When a toothbrush is used, its user is identified and the useful information related to the user is extracted from various web services and is shown on the mirror.

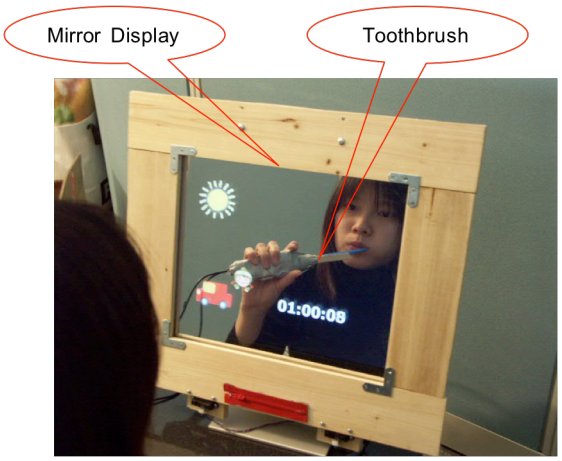

Fig. 1. AwareMirror in Operation, the mirror displaying weather, transportation and schedule information in abstract mode.

C. *Component View*

The following sentient artefacts and sensors have been used in the AwareMirror application:

1) AwareMirror: The mirror is constructed using an acrylic magic mirror board and an ordinary computer monitor. The acrylic board is attached in front of the monitor and only bright colors from the display can penetrate the board. Two proximity sensors have been embedded into the mirrors. These are used to infer users' distance/position from the mirror. Furthermore a slide sensor is fabricated on the mirror that is used to navigate between the abstract and detail mode of the displayed information.

2) Toothbrush: A Toothbrush is augmented with a two-axis accelerometer. It can detect start and end of brushing. This is achieved by monitoring zero crossing through the differentiation between two latest measurements, i.e. from plus to minus and vice versa. In addition an RFID tag is fabricated into it, which can be used to detect the toothbrush in a specific location. As a toothbrush is a highly personal belonging, which is rarely shared with others, we can infer that only its owner will use it. This fact we have utilized to infer the identification of a person (the owner) by the toothbrush's state of usage and location.

3) Web Services: For three categories of information we have used 3 distinct web services. Yahoo Japan has been used for the weather forecasting. iCalendar based scheduler service is used for tracking the user's schedule. We have used a dummy web service for the transportation information.

D. *Functional View*

The application's control flow can be stated as follows:

1) During the system initialization, all the components are enumerated accordingly.

2) The user initializes the system by providing necessary information.

3) If the user uses the toothbrush in the morning, while the system is running, the user is identified by the system and his/her preference information is loaded.

4) Accordingly the web services are contacted to collect the information.

5) The system then renders the information to the Mirror.

6) The display has two modes, initially the mirror displays very abstract information in appropriate positions within the mirror, making sure that information does not cover the main portion of the mirror.

7) The slider can be used to change the mode of the display. By using the slider fabricated in the mirror, the user can navigate to the detail mode that shows detailed information. In this case the mirror actually turns into a mere display.

In the next section, we have discussed how these functionalities are achieved by utilizing the two different frameworks.

IV. UNDERLYING INFRASTRUCTURE

It is obvious that we indeed need a stable platform that makes the application development simple and rapid by automating many recurring tasks. We have developed two infrastructures for providing the platform support to context-aware application developers. These two frameworks attack the problem in different manners. The first one *"Bazaar"* is a centralized architecture that attempts to model the environment in a bottom up approach exploiting self-descriptive objects, central repository and an inherent location model. Bazaar is more motivated to provide a suitable world model than providing support for contextual application. However it is flexible enough to provide support for handling contextual information. The second one *"Prottoy"* is a distributed one in approach that attempts to model the environment in a top down manner and primarily focuses on providing generic access to the environment for the developers. Prottoy is completely dedicated to provide a seamless platform for context-aware applications.

We have implemented two versions of AwareMirror applications using these two frameworks. However before giving our experience reports on the development, let us introduce the two architectures briefly and their integration with AwareMirror.

A. *Bazaar*

Bazaar [17][18] is an infrastructure for modeling the physical world populated with various sentient artefacts. It provides an application developer with a shared physical space information repository and with high level APIs to access information and notification of interesting events. The details of communication between the application space and the physical space are hidden in Bazaar.

Currently, Bazaar manages and provides the following types of information: 1) type, 2) location, 3) state-of-use, 4) owner, 5) timestamp of detection, and 6) IP address/port number and additional information to control a sentient artefact. Since Bazaar manages these kinds of information as a key-value pair, a new one can be easily added.

As can be seen in Figure 2 Bazaar consists of five major parts. This includes, 1) an identifiable artefact that serves as a source of low level contextual information and an actuator if any, 2) an ID detector that provides the approximate location of a detected artefact within its sensing range, 3) a shared information repository, 4) a contextual extraction framework that interprets and makes the low level contextual information available to the application as highly abstract information, and 5) an application logic that a developer has to implement. Every artefact is identified using a tag like RFID, where its related information and the location of the detector are linked together. Application can receive artefact specific information specified earlier by querying the shared repository.

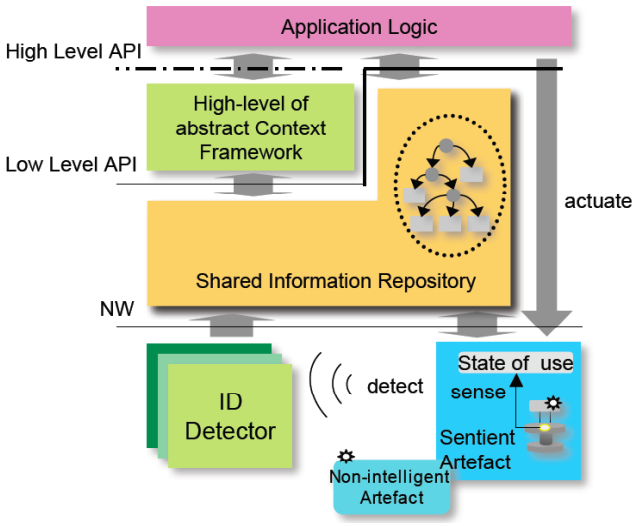

Fig. 2. The architecture of Bazaar, demonstrating major components

B. *Implementation of AwareMirror using Bazaar*

Figure 3 shows the overall architecture of Aware Mirror on top of Bazaar. AwareMirror, sentient toothbrush and an application "AwareMirror Controller" are the components that are glued with Bazaar. On top of Bazaar, an integrated application exists which aggregates information from the two artefacts and makes this information available to the AwareMirror Controller. Upon receiving appropriate context information, this controller communicates with the web services to extract proper information related to the user and present them on the mirror.

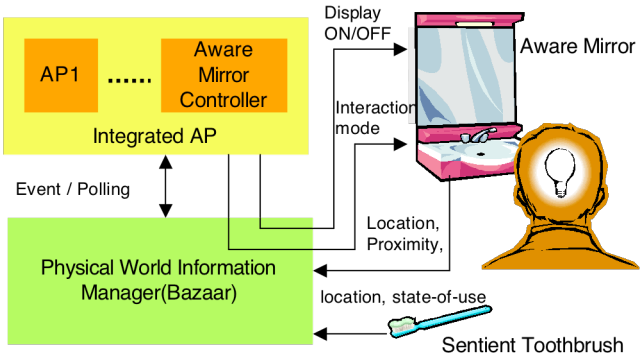

Fig. 3. Implementation of AwareMirror using Bazaar

So, here Bazaar acts as the physical space manager, it has the snapshot (location, state of use, properties etc) of the self-descriptive artefacts (AwareMirror, Toothbrush) available in the environment. The artefacts are identified by the ID detector component of Bazaar and accordingly their state of use and property information is aggregated in the central repository. The consumer application AwareMirror controller exploits this information accordingly to render the output on the mirror.

C. *Prottoy*

Prottoy [7][8] provides a generic interface to the applications for interacting with sentient artefacts in a unified way regardless of their type and properties Prottoy is composed of few core components and few pluggable components as shown in Figure 4.

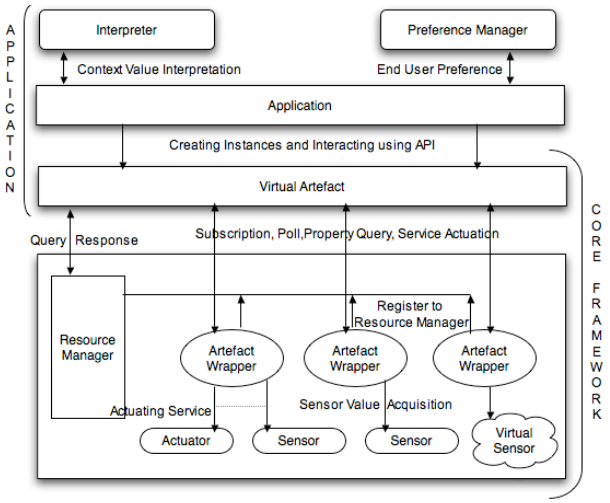

Fig. 4. The architecture of Prottoy, demonstrating major components

- **Core Framework Components:**

1) *Resource Manager*: As the name implies, it simply registers the properties, services and context information of the artefacts. When application query comes via virtual artefacts it responses accordingly

2) *Artefact Wrapper:* It encapsulates the sentient artefacts, sensors, actuators or virtual sensors like weather services, scheduler etc. We have provided a template for the developers to wrap their device drivers or software into this component. Developers provide the artefact property and location information into this template during deployment. Artefact wrapper has its own resource manager that can advertise its capabilities when the global resource manager is absent. In addition it has a simple security measure using IP filtering, that allows an artefact to control access to its service and protect information from malicious applications.

3) *Virtual Artefact:* It abstracts the smart environments and provides a unified view. Application constructs virtual artefact instances from specific context or service requirement. Virtual artefact communicates with the resource manager and if an artefact is found virtual artefact communicates with that artefact. If everything goes fine virtual artefact represents the artefact in the application. Application can subscribe to this artefact or can poll for contextual information. Application can also execute services of the physical artefact. If storage is enabled, virtual artefact creates storage in the application layer. If proxy is enabled then the proxy service of virtual artefact activates when the physical artefact is absent. The proxy provides the application with a calculated low accurate context value using the storage.

- **Components Pluggable to Application:**

1) *Interpreter:* It maps the context value to the interpreted value. We argue that context interpretation is highly application dependent as the same context can be interpreted in different ways based on the application requirements. So we put this component in the application layer.

2) *Preference Manager:* This component is designed for the end users of the applications, which are developed on top of Prottoy. It provides the facility to enable or disable the participation of any artefact of the environment in the application based on their preference.

D. Implementation of AwareMirror using Prottoy

Figure 5 shows the architecture of AwareMirror using Prottoy. The AwareMirror and the toothbrush are wrapped into the artefact wrappers. The three web services are also wrapped in three distinct wrappers. During the deployment, these artefacts communicate with the resource manager and register themselves by providing their identities. Application deals with both types of artefacts (physical like toothbrush and virtual like web service) using the virtual artefact interfaces. That means application creates instances of virtual artefact by providing the properties of the required artefact. Virtual artefact internally communicates with the resource manager to locate the AwareMirror, the toothbrush and the web services. Once these are found, then the application subscribes to the toothbrush and AwareMirror for context information. When the application receives appropriate context information (interpreted by the interpreter), it communicates with the web services through the virtual artefact and then actuates the rendering service of AwareMirror to display the information. Application uses the interpreter to interpret the context information received from the artefacts to map into application specific values. Also the end users of the AwareMirror application can control the participation of the artefacts by the preference manager component.

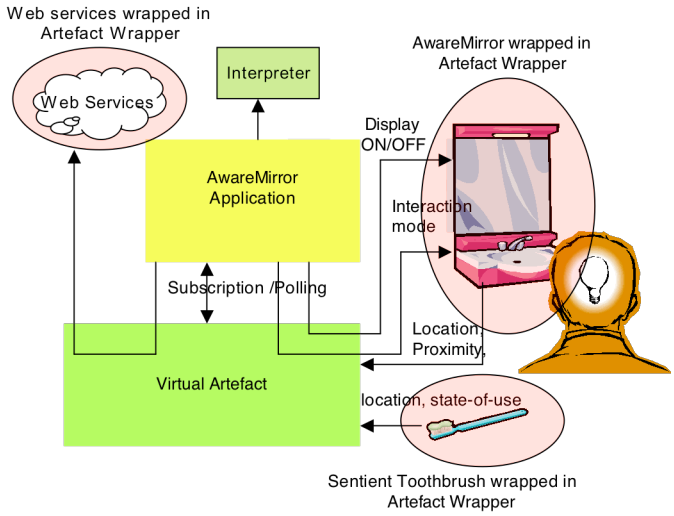

Fig. 5. Implementation of AwareMirror using AwareMirror

Virtual artefact solely hides all the communication details and isolates all access issues thus simplifying application development considerably.

E. Feature Comparison

Bazaar's primary focus is on providing a world model in a transparent way centering on the sentient artefact. For that purpose Bazaar's approach is different from Prottoy. As we have shown Bazaar provides various information of the real world beyond location to facilitate suitable modeling. However modeling the real world is slightly different than the daily life context aware applications. Thus we felt for another framework concretely focusing on the context aware applications and that's how Prottoy came into the scene. In the following (Table II) we are providing a comparison of the features between these two frameworks.

Table II
Comparison between Bazaar and Prottoy

Feature	Bazaar	Prottoy
Motivation	Providing a world model.	Providing a seamless platform for context-aware application development
Approach	Centralized and bottom-up. Artefact itself informs/notifies it's information.	Distributed and top-down, Application queries artefact from information/event notification
Location-Model	Inherent location model is present.	No location model is present. Location information is statically updated.
Attribute Independence	Complete. Application does need to know any attribute about the artefact.	Partial. Application needs to know the artefacts capability to acquire the information.
Security Support	No support.	Supported.
Plug-n-Play Support	No support.	Full Plug-n-Play via the Resource Manager
Event Notification	Partly event based. Application needs to query for events.	Completely event based. Application can be notified instantly about low-level events.
Context History Support	No historical information is provided.	Context historical support is provided.
Proxy Support	Not supported.	Supported.
Preference Support	No support.	Supported.

However, since Bazaar's initial goal is to provide a real world model, some features like proxy, security or preference support etc. should not be considered as evaluation criterions between the two frameworks. Of course both the architectures support all other standard requirements like context specification, separation of concern, transparent communication (spatial and temporal independence), constant availability etc.

V. Discussion

This section discusses about several issues. We have provided our experience on the basis of the performance, advantages, and drawbacks of sentient artefacts and the two middlewares. However one point to note here is that: though we have discussed only one application in this paper, our actual experience has been aggregated from various applications [7][8][17][18][26] that we have developed.

A. Sentient Artefact

For developing context-aware application, drawing practical real life scenario plays the key role. By augmenting appropriate artefacts with context sensing capability and by integrating these artefacts in one or more applications, we can eventually implement those scenarios. The major strength of considering everyday object as context source is the natural interaction. Users need not to learn any new technology; they are using the artefacts in the same manner they are used to. We are not offering them new devices, rather augmenting their daily life objects with sensing capability. As a result our approach is well accepted by the end users. This acceptance test was done for several applications that integrate multiple artefacts [7][16][17].

- **Major Observations:**

However, there are a few issues that we have identified and which require further discussions. First and most important is how can we make a generalized artefact that can be utilized in multiple scenarios. We must not develop application/scenario specific artefacts. Thus deploying artefacts with "*one to many"* attribute is a big challenge. From our experience we have identified that it is very difficult to generalize an artefacts capability for providing contextual information as same artefact can be used in different applications in different manner. Still now we could not come up with a guideline that can help us to build a generalized artefact.

The next issue is what types of sensors are suitable for an artefact and where to fabricate it. From our experience we have found that, to answer this question first we have to observe the target environment and the available artefacts. Then we have to identify what information we are looking for and what sensors are appropriate for providing the information. Once this analysis is performed, then we can pick appropriate artefact and can fabricate the sensors to them. In most of the cases there are several alternatives, so we cannot confine strict rules here. For example: In AwareMirror we need to identify the presence of a person. For that we used the toothbrush augmented with an accelerometer and an RFID tag. These two provides state of use and location. Thus combining these information we can infer users' presence. However we can also use a shaver or a comb for the same purpose. However, the reason for which we picked the toothbrush is that toothbrush is a personal belonging that is rarely shared than the other two. So such ad hoc reasoning may lead to the selection of appropriate artefacts.

Final issue related to the sentient artefact is what and how context information can be modeled. In our approach we have provided state-of-use, location and properties (owner, type, color etc.) of the artefact as the contextual information. The device driver running on the sentient artefacts aggregates this information. Also some of these are predefined or can be manually changed. However we are not claiming these are sufficient. A new application may come up with a new requirement. So, we need a guideline that will help us to model the low-level sensor data into higher-level context. One alternative can ontological based context modeling from raw sensor data at individual artefact. Further more same sensor data can be modeled in different manner for providing different context. We cannot yet confine any rule for prescribing such context calculation. Thus this issue poses a great challenge for the researchers.

B Bazaar

- **Performance and Observation:**

The primary motivation behind the development of these frameworks was to provide a seamless platform for integrating multiple sentient artefacts in context-aware applications. Bazaar was our first attempt where we tried to model the real world exploiting the features of sentient artefacts. Bazaar is successful in hiding the heterogeneity among the artefacts and providing a shared repository to the applications. Application developers can use the APIs provided by Bazaar to manipulate the artefact in an intuitive manner. They don't need to consider the low level detail of communication or context management. Very simple query semantics are provided by which application can query specific property of the artefacts. Also these properties are implemented as a key-value pair, thus adding new property to the artefacts is relatively simple.

The shared repository can support multiple applications at the same time with current context information. So integrating multiple artefacts has no effect on the applications' performance. Thus application development process is very simple, as we have seen in the AwareMirror application. Once the appropriate artefact is deployed, the control flow of the application is clear and smooth due the high level abstraction support from Bazaar. Application only queries the repository to know the context information and actuate service accordingly. So, application deals with the repository for contextual information. Further more Bazaar removes the typing conflict and attributes dependency. By providing a bottom up hierarchy Bazaar relieves the developer from knowing and analyzing the artefacts' classes and attributes in advance. Such simplicity makes Bazaar very flexible to handle.

- **Drawbacks of Bazaar:**

However, Bazaar has some drawbacks also. The centralized approach lacks from the fact of single point of failure. Also because of the single-entity, context management is difficult when artefacts increase thus paying a scalability penalty. Also Bazaar does not support plug-n-play features. Thus automatic join/leave of the artefact could not be supported. As we have seen, in the AwareMirror application, Bazaar could not handle the virtual sentient artefacts like web services. So the application developer needs to manipulate them manually. This requirement actually was not identified when we deploy Bazaar, when our focus was on real world modeling. Also complex querying is not well supported in Bazaar.

Though Bazaar isolates attribute based access to real world, from our experience we have identified that developers often made some mistakes that are revealed at run time. For example, if a developer wants to use a "*chair"*, but by mistake he/she types "*chari"*. Bazaar could not provide support for such errors. However, as we have mentioned in the future work section that we are working on providing artefacts' specification to the developer during the development time to reduce the probability of such errors.

C. Prottoy

- **Performance and Observation:**

To compensate some of these issues we have build the second framework. Prottoy's primary goal is to provide a seamless platform for the application development exploiting it's distributed nature. This is because during application development we felt the difficulty of accessing/interfacing with multiple sentient artefacts that have different access protocols. If the application developer has to handle these heterogonous protocols, application development becomes really cumbersome. One of the major goals of Prottoy is to hide this heterogeneity by providing a unified interface. Application development on top of Prottoy is fairly simple. To be specific, developers only provide the context to action mapping rules. Once the artefacts are wrapped in the artefact wrapper template, applications can communicate with them in a unified way via the virtual artefact interface. Even the application does not need to know the artefact specific information or even the resource manager. Such simplicity and isolation of the access issues make the application development very simple and rapid. Because of its distributed nature, scalability is not a problem. So any number of artefacts can be integrated in a seamless manner to several applications, as long as the artefacts are wrapped in artefact wrapper.

The virtual artefact and artefact Wrapper in conjunction provide the generic interface for everything from a sentient artefact to a single sensor to a web service and to an actuator. The artefact wrapper provides the generalization that allows the actual artefact to be replaced anytime with another one. The proxy service is a unique feature of Prottoy. Some of the existing systems provide storage functionality at the artefact layer, our argument is that if the artefact itself is absent in that case the storage is also absent. We think the best use of the context storage or history is the prediction of the context, so it should be somewhere that can be accessible when the artefact is absent. Virtual artefact perfectly solves the problem by hosting the storage and providing proxy service. While using Prottoy, application developers are free from network management issues. The three-layer architecture separates the application from the physical space completely. Prottoy's overall communication model is event based. All the events are handled at real time with proper functioning. Prottoy provides both subscribe-publish and request-response event models. Furthermore application code is completely independent of Prottoy.

Prottoy's distributed nature compensates the problems of centralized approach of Bazaar; also the resource manager (both global and local) can handle the issue of plug-n-play support. Further more in Prottoy any number of information/properties regarding artefact can be provided at deployment time or later. Also Prottoy hides the difference

between the physical artefacts and virtual artefacts like web service. Thus application developers can use them in a unified manner. So Prottoy could handle some of the issues that Bazaar suffers from.

- **Drawbacks of Prottoy:**

In spite of having these features, from our experience we have identified that Prottoy suffers from some problems that limit its performance. There is no inherent location model in Prottoy; so artefacts location information cannot be updated dynamically. Currently this information is manually updated. One solution to this problem is adopting Bazaar's location detector component. Prottoy cannot advertise automatic contextual information like Bazaar. (The local resource manager in Prottoy advertises artefacts capability) So applications need to explicitly subscribe/query the artefacts information. This is a vital issue when we consider the real world model. We need substantial level of information from the underlying environment beyond location. In this respect Bazaars performance is well justified. Also Prottoy's functionality is vastly dependent on artefact wrapper. All sentient artefacts must need to be wrapped by artefact wrapper for Prottoy's functionality.

Another issue is the query support. Currently Prottoy provides only *AND* operation, that is artefacts properties and capabilities can only be concatenated for artefact searching. Prottoy cannot handle other combinations (like *OR* or *XOR* etc.) However from the application development experience we have figured out that our mere *AND* support is not enough.

Prottoy attempts to provide a generic interface by hiding the communication detail, however we cannot prescribe this generalization for heterogeneous sensors. That means how can we generalize the sensor specific APIs. We could not find any suitable answer to this question. Also hiding communication from the applications does not always provide optimum performance. For example how can we ensure such communication in a generic way when conserving energy is a major requirement (like to communicate with the artefacts connected via wireless network)? Also the proxy and security component of Prottoy is immature in terms of functionalities.

B. *Future Work*

In the previous sub-section we have described the problems that we are facing currently. Our current research investigations are challenging those issues. We are focusing on finding a suitable location model that can be adopted for optimum functionality. The sentient artefact generalization and context modeling is another big hurdle that we are trying to cope with. Several other features that we have introduced in our two frameworks seek further clarification. For example, the access controls mechanism or proxy module's logic. We believe that the simple IP filtering technique used in the current Prottoy prototype is not suitable for all the applications. We are trying to figure out more appropriate mechanism for the security measure. Regarding the proxy service, currently we have used only the mean of the stored values and the most recent value, but more sophisticated technique may lead to better predictions. Also, what sort of applications are the appropriate clients for the proxy service and to what extent? The preference component in the current version only provides a selection-based approach via a GUI. However, we don't feel such GUI based preference is suitable for a context aware application. We are investigating to make this component more realistic and effective. One alternative is preference by demonstration. We feel this preference policy is very important for the practical deployment of the context aware applications.

Complex query support is another issue. Rule based technique has already in the literature but we believe such rule increases the cognitive burden of the developers. We are currently working on a prototype physical space programming IDE [20]. We feel physical space IDE will reduce the complexity of queries. With such IDE the developer will be automatically informed of available artefacts as he/she plugged into local environment thus isolating the necessity of complex queries. We are investigating all these issues with great interest and hope to come up with some interesting results soon.

VI. RELATED WORK

In this paper we have demonstrated our approach: sentient artefact as context source and middlewares for supporting application development. So, in this section we have cited the related work from two points of view; firstly current trend related to context source and secondly existing context-aware application frameworks.

A. *From Context Source Point of View*

Most of the context aware projects use artefacts that are either traditional general purpose computing platforms ranging from small handheld to large sized high end computers like ParcTabs, or dedicated artefacts designed for providing specific contextual information like Active Badge infrared sensor. However our work is different from these two approaches as we concretely focus on everyday objects for context capturing without compromising their primary role. Digital Décor [12] project augments traditional drawer and coffee pots to use as a smart storage and a media for informal communication respectively. However users are responsible for explicitly using these artefacts for their services. Also they only provide some services (searching, communicating with people etc.) rather than any contextual information. Tangible Bits [10] project attempts to bridge the physical world and virtual world by providing interactive surface, graspable objects and ambient media. However such explicit dedicated interfaces violates natural interaction paradigm and natural augmentation of conventional everyday objects.

Recently one Internet service [27] provides similar notion as ours by providing activity information of remote elderly by capturing the state of coffee pot. Although they have augmented everyday object the consumer of this information is not the person who uses the system. It is a kind of monitoring system, which does not provide any contextual information. Paradiso's work [15] in wearable computing arena matches our vision as he has exploited sensor-augmented footwear to obtain contextual information. TEA [11][22] project attempts to embed various sensors to augment handheld devices to provide contextual information. However they only focus on handhelds. MediaCup [21] projects and its succeeding SmartIts [29] provide insight into the augmentation of artefacts with sensing and processing. Our work is greatly influenced by them and exploits the Aware Artefact model introduced in [11]. However our sentient artefacts do not require any explicit interaction as MediaCup or SmartIts based artefact requires. Our approach is to make artifact aware but not their user aware of this fact. Sentient artefacts are mere everyday artefacts without any noticeable feature. Users manipulate them in the natural way they are used to with. They don't need to do something explicitly to make something happen. This natural feature distinguishes our work from other projects.

B. From Middleware Point of View

Currently there exist a number of context aware application frameworks in the literature. Usually, two approaches have been investigated for context-aware framework. One is the centralized server approach, like Schilits System [24] or Contextual Information Service [14] and the other is the distributed approach like Context Toolkit [2] or Speitzer's work [23].

Centralized frameworks provide fair performance from the point of view of context acquisition from the sensors and providing interpreted context via standard APIs. However they suffer from the fact of single point of failure and extensibility concerns. Also, collecting information from several sources in one place makes the framework complex and maintenance becomes difficult. Bazaar is centered on centralized repository. However, as the context sources are distributed among the sentient artefacts, the repository itself is not responsible for modeling or generating the context. Prottoy's approach is different from these as it completely distributes the context sources into multiple artefact wrappers. Application can communicate to specific context sources and can interact with them via the virtual artefact seamlessly. Thus scalability is well supported in Prottoy.

Schilit's System [25] deals with the context awareness by Device Agents that maintain the status and the capabilities of the devices, User Agents that maintain the user policies and Active Maps that maintain the location information of the devices and the users. Resource manager and preference component in Prottoy provide the same functionalities. In addition, by introducing features of the artefact wrapper and the virtual artefact, it provides the developer much more control and flexibilities over the physical space and the application development process. Bazaar's information repository actively performs the task of Device Agents and Active maps. However the other components of Bazaar further enhance the application development process.

Among the distributed ones, Context Toolkit [2] focuses on the component abstraction by providing the notion of Context Widget and Context Aggregator. Discoverer manages these components and additionally there is a Context Interpreter component that performs the task of context interpretation. Context Toolkit provides very low-level abstraction. Developer needs to provide the details about the context source like location, port etc. Also, the application is inherently dependent on the framework as the application is tightly coupled with the architecture. That means application needs to extend the architecture component and manipulate accordingly. Such low level complexity makes application development very cumbersome. Bazaar approach is different than Context Toolkit. Application can access the sentient artefacts by the shared repository solely without concerning any low level detail. Prottoy takes the Context Toolkit architecture and generalizes it in a single component namely virtual artefact. Using Prottoy, the applications become independent from the context infrastructure as the virtual artefact handles all the lower level tasks. Even the applications do not need to know about the resource manager. Applications only use the virtual artefact as a generic component that provides all the infrastructure supports. Prottoy also hides the context implementation from the context specification

Speitzer [23] proposed another distributed architecture based on multicast communication, where context information is multicast among the members of the multicast group. However the disadvantages of their approach are the increasing computation and communications thus paying a scalability penalty. For example an application within a multicast group will receive context information even if it does not request for it. The Stick-e Notes system [13] provides simple semantics for writing rules that specify what action to perform based on the acquired context, mainly focusing on non-programmers to author context aware services. Both Bazaar and Prottoy generalize this context acquisition from programmer's point of view. Stick-e Notes can be thought of as an application on top of Bazaar and Prottoy.

The Sentient Computing Project [1][20] utilizes Active Bat location system to provide an architectural base for indoor application exploiting a world model. The approach of Bazaar and Prottoy is different from the point of view that they offer the applications to create a context aware environment by constructing an array of artefacts. It means Bazaar and Prottoy specialize the world model creation by allowing developers to construct the model as they want. HP Cool Town [5] encapsulates the world by providing web presence of place,

people and thing and allows interaction with web presence of these entities primarily exploiting RF technology. Cool Town supports only web based context aware applications where Bazaar and Prottoy support any classes of context aware applications. Easy Living [4] focuses on an architecture that supports the coherent user experience as users interact with variety of devices in a smart environment. Easy Living also utilizes the notion of world model. Open Agent architecture [6] is an agent-based system, which exploits a centralized black board to support the contextual behavior. In contrast to these systems, Bazaar and Prottoy provides a more generic abstraction as developer has the flexibility to construct the model by manipulating the sentient artefacts.

VII. CONCLUSION

In this paper we have shared our experiences on building context-aware applications. We have demonstrated our approach by a real life application development process and have discussed our findings and difficulties. Two different frameworks have also been presented that we have exploited for developing the applications. Context-Awareness is a key concept of future computing, we believe our experience report will help the developers to understand the problems for realizing some of the issues that are limiting the success of this key concept. We hope the workshop will provide us with a good opportunity to share our experiences, to discuss the complex issues that mentioned in the paper with the researchers and to identify the probable path towards resolution.

REFERENCES

[1] A. Harter, A. Hopper and P. Steggles "The Anatomy of a Context-Aware Application," *In Mobile Computing and Networking*, pages 59-68, 1999

[2] A. K. Dey, G. Abowd and D. Salber "A Conceptual Framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction*, Vol-16 2001.

[3] A. Schmidt and V. Laerhoven. *"*How to build smart appliances*," IEEE Personal Communications, pages 66-71, 2001*

[4] B. L. Brumittet, B. Meyers, J. Krumm, A. Kern and S. Shafer "Easy Living: technologies for Intelligent Environments" *In the proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing* '2000

[5] C. Deborah and P. Debaty. "Creating Web representations for Places*" In the Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing '2000.*

[6] C. Philip R. A. Cheyer, M. Wang and S. C. Baeg "An Open Agent Architecture," *In the proceedings of the AAAI Spring Symposium Series on Software Agents,'94*

[7] F. Kawsar, K. Fujinami and T Nakajima "Design and Implementation of a Software Infrastructure for Integrating Sentient Artefacts," *The 2nd Annual International Conference on Mobile and Ubiquitous Systems: Network and Service*, July 2005 (To be published)

[8] F. Kawsar, K. Fujinami and T Nakajima "Prottoy: A Middleware for Sentient Environments," *The 2005 IFIP International Conference on Embedded And Ubiquitous Computing.* Dec 2005, (Submitted)

[9] G.D. Abowd *"Software Engineering Issues for Ubiquitous Computing",* In Proceedings of the 21st International conference on Software Engineering, pages 75-84. IEEE Computer Society Press 1999.

[10] H. Ishii, and B. Ullmer "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms*," CHI* 1997

[11] H.W. Gallerson, A. Schmidt and M. Beigl "Adding Some Smartness to Devices and Everyday Things," *WMCSA 2000*

[12] Itiro Siio, J. Rowan, N. Mima and E. Mynatt "Digital Decor: Augmented Everyday Things," *Graphics Interface 2003*

[13] J. P. Brown "The Stick-e Document: A framework for creating context aware applications", *In the proceedings of the Electronic Publishing '96*

[14] J. Pascoe *"Adding generic Contextual Capabilities to Wearable Computers ," In the Proceedings of the Second International Symposium on Wearable Computers, Pages 129-138. IEEE Computer Society Press. Oct,* 1998

[15] J. A. Paradiso, K. Hsiao and A. Benbasat "A. Interfacing the Foot: Apparatus and Applications," *CHI 2000 Extended Abstracts*

[16] K. Fujinami, F. Kawsar and T. Nakajima. "AwareMirror: A Personalized Display using a Mirror" *In the proceedings of International Conference on Pervasive Computing* (Pervasive 2005), May 2005

[17] K. Fujinami and T. Nakajima. "Towards System Software for Physical Space Applications" *In Proceedings of ACM Symposium on Applied Computing (SAC) 2005*, March 2005.

[18] K. Fujinami, T. Yamabe, and T. Nakajima. "Bazaar: A Conceptual Framework for Physical Space Applications." *In The 2nd International Symposium on Ubiquitous Computing System (UCS2004),* November 2004.

[19] K. Fujinami, T. Kambe and Tatsuo Nakajima, "BASE: An Interactive Location-aware Development Tool for Sentient Environments", In the Poster Proceedings of the Seventh International Conference on Ubiquitous Computing, UbiComp2005 (Submitted)

[20] M. Addlesee, R Curwen, S. Hodges, J. Newman, P. Steggels, A. Ward and A. Hooper. *"Implementing a Sentient Computing System" Cover Feature in IEEE Computer,* Vol. 34, No. 8, August 2001 pp 50-56

[21] M. Beigl, H.W. Gallerson and A. Schmidt *"*Media Cups: Experience with design and use of Computer Augmented Everyday Objects*," Computer Networks, special Issue on Pervasive Computing, Mar '2001*

[22] M. Beigl, T. Zimmer and C. Decker. "A Location Model for Communicating and Processing of Context," *Personal and Ubiquitous Computing* 6(5-6): 341-357, Dec, 2002

[23] M. Spreitzer "Providing Location Information in a Ubiquitous Computing Environment". *In Proceedings of the fourteenth ACM symposium on Operating System Principles, pages 270-283. ACM Press* 1993

[24] M. Weiser *"The Computer for the 21st Century"* Scientific American, Vol. 265, No.3 1991

[25] N. B. Schilit. "System Architecture for Context Aware mobile Computing*"* PhD Dissertation, Columbia University New York, 1995

[26] T. Yamabe, K. Fujinami and T. Nakajima "Experiences with Building Sentient Materials using various sensors," *In the Proceedings of the 4th International Workshop on Smart Appliances and Wearable Computing (IWSAWC)*, pages 445-450, Mar. 2004

[27] www.mimamori.net

[28] www.naturalinteraction.org

[29] www.smart-its.org

Fahim Kawsar is a MS student in Computer Science Department of Waseda University. His research interest includes Context-aware Computing, Human Computer Interaction, Operating Systems and Web Programming.

Kaori Fujinami is an assistant professor of Department of Computer Science in Waseda University. His research interest includes Context-aware Computing, Physical Space Oriented Programming, and Information Presentation.

Tatsuo Nakajima is a professor of Department of Computer Science in Waseda University. His interests are Operating Systems, Distributed Middleware, Real-time Systems and Ubiquitous Computing.