# Prottoy: A Middleware for Sentient Environment

Fahim Kawsar, Kaori Fujinami and Tatsuo Nakajima

Department of Computer Science,
Waseda University, Tokyo, Japan.
{fahim,fujinami, tatsuo}@dcl.info.waseda.ac.jp

**Abstract.** Our approach towards context awareness is to retrieve contextual information by augmenting our daily life objects (like a chair, a mirror etc.) with sensing capabilities. We call such artefacts *sentient artefacts*. To avoid developing dedicated context-aware application integrating these artefacts, there is a need for a generic computing platform that can assist application programmers to develop and deploy applications easily and rapidly. We present a framework titled "Prottoy"for context-aware applications. The framework provides a generic interface for interacting with sentient artefacts in a unified way, regardless of their type and properties. As a result, application development is simple, rapid and independent from the underlying environments. This paper describes the design and implementation of Prottoy.

## 1 Introduction

Ubiquitous computing envisions a future environment that will be aware of its operating context and will be adaptive to ease our interaction. Our approach towards such an environment is the environment itself. That means focusing on the building blocks of the environment and making them smart and context aware by capturing peoples implicit interaction. We augment daily life artefacts like a chair, a table, a door, a mirror, a bed etc. with various kinds of sensors to capture contextual information. Our vision is to utilize these objects for value added services in addition to their primary roles. 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.

Based on our experiences of developing contextual applications integrating these artefacts, we felt the necessity of a software abstraction that hides the low level details while satisfying requirements like security support, preference reflection, reliability etc. To fulfill these requirements we are working on a software infrastructure called "Prottoy". Our primary goal is to provide a simple, lightweight and extensible framework that offers a unified view of the underlying physical spaces to the applications while supporting all other features.This paper discusses about the design and implementation of Prottoy.

The rest of the paper is organized as follows: In Section 2, the motivation and design principle of Prottoy are presented. Section 3 gives an insight in the architecture of Prottoy respectively. In section 4 we have presented three sample applications that are built on top of Prottoy. Section 5 discusses several issues and future directions. Section 6 discusses about the related work and finally Section 7 concludes the paper.

## 2 Motivation and Design Principle

A common goal for programming frameworks for context-aware computing is to provide a computing platform to easily develop and deploy context-aware applications. Programmers can focus on modeling and using the context information while the underlying platform takes care of the actual management and the distribution of this information[1]. There have been several investigations on context aware computing framework in the literature [8] [6] [7] [11] [10] [4] [13] [2] [12] . Many of these frameworks have several interesting features. Prottoys approach is to share all those interesting features while incorporating the features that are missing in other frameworks. In addition Prottoy is designed to satisfy some essential requirements.The major design principle of Prottoy is to provide a generic interface that isolates all environment access issues . Next it targets to separates the application and physical space completely. This separation ensures the decoupling in space and provides fair flexibility to the developers. Another important design issue of Prottoy is to make the application independent of the architecture. Prottoy also designed to be event driven and easily extensible. Final design goal of Prottoy is to provide fair support for context history, end users' preference reflection and artefacts' security.

## 3 Architecture of Prottoy

As illustrated in Figure 1 Prottoy is composed of few core components and few pluggable components. Core components are the architectural base that facilitate the underlying support for context acquisition and service actuation. Pluggable components run in the application space to assist the developers to exploit the context information.

- Core Framework Components:
  1. Resource Manager 2. Artefact Wrapper 3. Virtual Artefact
- Components Pluggable to Application:
  1. Interpreter 2. Preference Manager

These components internal structures and their corresponding programming models are discussed in the following.

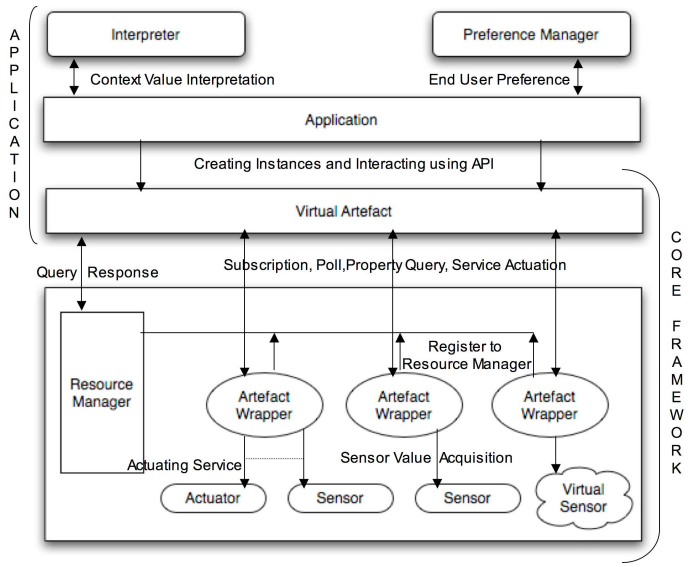


**Fig. 1.** Basic Architecture of Prottoy

### 3.1 Resource Manager

It acts as the information repository and service discoverer. It contains artefacts' information like what context or service they provide and their properties. Artefacts register and deregister themselves to the resource manager when they join or leave the environment respectively. Applications locate an artefact by querying the resource manager. If an application uses an artefact, it notifies the resource manager. Also when an application releases an artefact, the resource manager is notified. So the resource manager has the snap shot of the environment and the applications association with the environment all the time.

### 3.2 Artefact Wrapper

This component encapsulates the sentient artefacts, sensors, actuators and virtual sensors (like weather services, scheduler etc.). It can be used for acquiring context information from the environment and also can be used to actuate services to affect the context or environment. Artefact wrapper is essentially a template that contains various functionalities as already implemented. Artefact developers only need to provide their device drivers for context acquisition or for service actuation. Prottoy handles everything else like event handling, communication mechanisms, deployment techniques etc. Artefact wrapper is composed of the following modules:

1. Communication Module: Artefact wrapper supports two modes of communication. In *synchronous* mode, the context information is constantly dispatched to the interested application whenever the context information is requested, where as in *asynchronous* mode, the context information is dispatched to the interested applications only when context is changed. In case of the service actuation the artefact wrapper always sends an acknowledgement to the application stating the actuation status.
2. Local Resource Manager: Artefact wrapper has its own resource manager. If the global resource manager in the environment is absent, then the artefact can start its own resource manager to advertise its service or context information.
3. Security Module: Artefact wrapper has an internal security module that allows an artefact to protect its service and information from malicious applications. The artefact only responds to the requests coming from the applications running on its known domain.
4. Device Driver and Context Logic: This module provides a template for the artefact developers to plug in their device driver code and context calculation logic.

**Programming Model:** Artefact wrapper has built-in features for most of the architectural support. Developers need to provide their device driver and context calculation logic by overriding two functionalities, i.e context update and service execution method. A snippet of artefact wrapper code looks like the following:

```
1. public class Thermometer extends ArtefactWrapper{
2. public Thermometer()
3. public void update(){
4. /* Device driver code and context calculation logic */
5. setContextState(contextStatement);
6. notifyClient();
7. }
8. /* Provide the service execution code here */
9. public synchronized Hashtable executeService(String argument){
10. return null;
11.}
12.public static void main(String[] args){
13. Thermometer artefact = new Thermometer();
14. artefact.start();
15. }}
```

In the `update()` (line 3-7) function, after context calculation, the developer should specify the current context value by setting the context statement using `setContextStatement()` (line 5) and should fire `notifyClient()` (line 6), which eventually notifies all the subscribed applications. The `ArtefactWrapper.start()` (line 14) in the main function initiates the startup process of the artefact.

### 3.3 Virtual Artefact

Virtual artefact is the heart of our framework that encapsulates the smart environments and provides a unified view to the application developer. Using the APIs provided in the virtual artefact, the application developers can interact with the actual physical artefacts. Virtual artefact consists of the following components:

1. Communication Module: This module plays the key role for isolating all the access issues to the environment from application point of view. The developer only provides the required artefact functionality/properties using the APIs. This communication model handles the rest by hiding all the details like discovering and establishing the communication link with the actual artefact.
2. Storage Module: Virtual artefact internally hosts a transparent storage component. When an application uses an artefact, the interaction history of the application with the artefact is stored in this storage. Application can later query any historical context.
3. Proxy Module: This module utilizes the storage to facilitate the proxy support when the actual artefact is absent. It produces a calculated context value using some predefined logic. The value is tagged with a low accuracy value indicating that it is a calculated value.

**Table 1.** Virtual Artefact APIs Available to the Developers

| Virtual Artefact API | Functionality |
|---|---|
| `public VirtualArtefact(String context, String service, PropertyList props, boolean storage, boolean proxy)` | Constructor for creating the virtual artefact instances |
| `public void subscribe(Object source, String callback)` | For subscribing to artefact for the context information |
| `public void unsubscribe()` | For removing subscription from the artefact |
| `public synchronized Hashtable execute (String args)` | For executing artefacts service |
| `public Hashtable poll()` | For polling an artefacts context information |
| `public String getProperty(String propname)` | For querying artefacts specific property |
| `public PropertyList getPropertyList()` | For querying all the properties of an artefact |
| `Public Hashtable getTimeValue(Calendar time)` | For querying the historical context information |

**Programming Model:** Virtual artefact has methods for all the standard operations required by the applications for interacting with the artefacts. Table 1 summarizes the APIs and their functionalities.

To demonstrate these APIs usage, a very simple application is provided here. The application automatically adjusts a cooler to the comfort level based on the sensed air temperature.The first 4 lines of the snippet are used to create virtual artefact instances. Then using these instances the actual artefact can be polled (line 6) or can be subscribed to artefact events (line 7). In case of subscription the call back supplied to the framework, is generic and independent of architecture. Also artefact service can be executed (line 17) and properties can be queried (line 9,10) using the instance respectively.

It is visible in the code snippet that applications do not consider network or message management of the architecture and do not need to override or extend any component of the architecture; even applications do not have to look for the resource manager. The developers only provide the context to action mapping rules. Thus Prottoy makes application development fairly simple and rapid.

```
1. PropertyList props = new PropertyList();
2. props.add("location","lambdax");
3. VirtualArtefact thermometer =
            new VirtualArtefact("temperature",null,prop,false,false);
4. VirtualArtefact cooler =
            new VirtualArtefact(null, "cooler service",prop,false,false);
5. If(thermometer.status){
6. System.out.println(thermometer.poll());
7. thermometer.subscribe(this, "thermometerListener");}
8. If(cooler.status){
9. String owner=cooler.getProperty("owner");
10. props=thermometer.getPropertyList();}
11. thermometer.unsubscribe();
12. /*call back*/
13. public void thermometerListener(Hashtable data){
14. /* Hashtable contains context value, timestamp, accuracy and source*/
15. String context = data.get("context").toString();
16. If(cooler.status){
17. System.out.println(cooler.execute("turn-on");}}
```

### 3.4 Interpreter

This component maps the context value to the interpreted value. We argue that context interpretation is completely application dependent. For example, consider a chair that provides information about its state of use. We can use this information to infer its user is sitting/not sitting (activity) or its users location (at chairs location) based on the applications requirement. Similarly an RFID tag can be interpreted as a person name or a news category name based on the

application. Our argument is that, we cannot broadly confine the interpretation of the context information. So we separate it form the core and provide it as a pluggable component at the application layer. The interpreter provides very simple interfaces for the developers to map the context value received from the lower layers to application specific value. During initial deployment , the developer needs to provide the mapping information, which is later used when the real time context value is received.

### 3.5 Preference Manager

This component is dynamically generated when the application is deployed and runs at application space. This component is non-editable and is primarily created targeting the end users of the application developed using Prottoy. So we can consider it as a macro running on top of Prottoy. The preference manager provides endusers with the facility to enable or disable the participation of any artefact in the application.


**Fig. 2.** Sample Applications Developed using Prottoy

## 4 Sample Applications

While introducing a new framework perhaps the most important issue is the evaluation. We approach this issue by deploying several context aware applications on top of Prottoy. We will discuss here three applications as shown in figure 2 that capture three scenarios at three distinct places namely washroom, dining space and work place.

### 4.1 AwareMirror

AwareMirror [9] is a smart mirror installed in the washroom. In addition to its primary task of reflecting someones 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 users schedule, transportation information and weather forecasting.These services were wrapped in artefact wrapper. When a toothbrush is used, its user is identified and the useful information related to the user is shown on the mirror. Initial goal of this application is to identify artefact wrappers performance on handling heterogonous context sources like physical artefact and virtual artefact (web services). We have found that the artefact wrapper could handle these successfully. Another important observation was applications effect on artefacts replacement. For testing purpose, weather forecasting web service was replaced by a thermometer and was being simulated to provide the same information as the web service. Interestingly we did not modify single line of code for the proper behavior of AwareMirror. This finding proves Prottoys capability to handle such replacement seamlessly.

### 4.2 Byte N Dine

This application is designed for a public/private dining space scenario. The goal of the application is to provide the latest news to the user while dining. The application uses two types of sentient artefacts: a few chairs and a table. When the table and the chairs are used in conjunction, the table displays some news. We have assumed that the user will carry a RFID tag that represents his/her preferred topic. The artefact wrapper was used to wrap the artefacts, information source and the RFID reader. Artefact wrapper was successful to handle isolated context source like RFID reader.Another major observation of this application was the development time for such simple applications. Considering reusability of the sentient artefacts, we realized that virual artefact enables writing simple application in a very short span of time.

### 4.3 SmartAssistant

This application runs in user workspace and can track users activities by analyzing the states of the artefacts populated in the workspace. The application uses a chair, a desk lamp, a tray and a few mugs and jars. Each of these are wrapped in artefact wrapper. The application suggests the user to take a refreshment considering users activity and can provide the user with some predefined schedule notification. Further more, the application can control workspace lighting. This application is more complex in terms of the functionalities than the other two. However we have found that by using virtual artefact interfaces, we could put aside the concern of interacting with such heterogeneous artefacts.One important observation is: context history plays a key role in the analysis of user activity.So such applications are good consumer of historical context.

## 5 Discussions

In this section we discussed about several issues related to the evaluation of Prottoy and our future work.

### 5.1 Feedback from Programmers

We have found that application development using Prottoy is fairly simple. None of the applications that we have developed exceed more than 100 lines of code for interfacing with Prottoy. Several members of our lab have used Prottoy in their applications. Programmers impression on Prottoy is positive. They like its simplicity, especially the virtual artefacts features. They have reported that Prottoy APIs are very easy to use. However they asked for more interfaces at the artefact wrapper layer for the calculation of context.

### 5.2 Qualitative Evaluation considering Design Principles

The virtual artefact and the 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 support that allows the actual artefact to be replaced anytime with another one.While using Prottoy, application developers are free from network management issues. The three-layer architecture separates the application from the physical space completely. The unique callback creation mechanism features of virtual artefact and standard data structure usage provide the complete independence to the application from the architecture. Some of the existing systems provide the storage functionality at the artefact layer [8]. 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 the history is the prediction of the context. So it should be somewhere that can be accessible when the artefact is absent. The virtual artefact perfectly solves the problem by hosting the storage. Historical context information stored in the storage can be used for predicting future context or tracking users activity pattern. In Prottoy, the proxy component provides the context prediction support. The proxy service is a unique feature of Prottoy. No architecture yet in the literature supports this feature. Prottoy also provides the support for reflecting end users preference. Current version simply offers end users the flexibility to select an artefacts participation in the application. However our support is very minimal. Another issue is protecting artefact content. We argue that self-descriptive micro world component should be smart enough to protect their content. Some architecture [3] [4] tries to tag the context information as secured or non secured. However malicious applications can still exploit such information. Our approach is different in the sense that the artefacts or the context source can completely deny any request if the requester is not authenticated thus enabling artefact to protect their content from eavesdroppers.

### 5.3 Major Contribution

Prottoy's event driven design with interesting features facilitate seamless platform for context aware applications. In summary we can point out the following as the main contributions of Prottoy:

1. Generic Interface for all sorts of context source, (namely sentient artefacts, unit sensor, virtual sensor like scheduler, web services etc) and actuators.
2. Complete independence of the application from the underlying architecture.
3. Context storage and proxy service support.
4. Authentication and local resource manager support for the smart artefacts.
5. End users preference reflection support.

Essentially developers need to handle only two components, virtual artefact and artefact wrapper. Because of this, application is very simple and lightweight.

### 5.4 Future Work

One important drawback of Prottoy is that it does not have any inherent location model for maintaining the spatial relationship among the artefacts. Currently we have used a static location model. However any suitable location model can be adopted in Prottoy with minor modification. We are currently working on this issue. There are few other issues that we are further investigating. Like authentication and proxy module are currently too shallow in performance. We are working on these modules to come up with much better performance The preference component in the current version only provides a selection-based approach. However, we dont feel such GUI based preference is suitable for a context aware application. We are investigating to make this component more realistic and effective. We hope to come up with some interesting results soon on these issues.

## 6 Related Work

Currently there exist a number of context aware application frameworks in the literature. We have mentioned in Section 2, that Prottoy incorporates many of these frameworks features. However Prottoy also introduces several new notions as we have stated in the paper that differentiates it from the rest of lot. Usually, two approaches have been investigated for context-aware framework. One is the centralized server approach, like Schilits System [12] and the other is the distributed approach like Context Toolkit [8] or Speitzers work [13]. 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. Prottoys approach is different from these as it completely distributes the context sources into multiple artefact wrappers.Thus scalability is well supported in Prottoy.Schilits System [12] 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.

Among the distributed ones, Context Toolkit [8] 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. Prottoy is highly influenced by Context Toolkit, however it differs in several ways. 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. Even the applications do not need to communicate with 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. It means applications process contextual information at run time dynamically without concerning their acquisition mechanism. In Prottoy, the context storage is at virtual artefact layer that the proxy module utilizes when the actual artefact is absent. We have justified our argument regarding this storage location in the discussion section.

The Stick-e Notes system [5] 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. Prottoy generalizes this context acquisition from programmers point of view. The Sentient Computing Project [2] utilizes Active Bat location system to provide an architectural base for indoor application exploiting a world model. Prottoys approach is different from the point of view that it offers the applications to create a context aware environment by constructing an array of artefacts. It means Prottoy specializes the world model creation by allowing developers to construct the model as they want using virtual artefacts.HP Cool Town [7] 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 Prottoy supports any classes of context aware applications. Easy Living [6] 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 [11] exploits a centralized black board to support the contextual behavior. In contrast to these systems, Prottoy provides a more generic abstraction as developer has the flexibility to construct the model by manipulating a single component namely virtual artefact.

## 7 Conclusion

In this paper we have presented a framework "Prottoy" for context-aware application development.When looking at the related work, Prottoy shares many features of the research already done for creating and supporting context information. Our attempt is to take all those frameworks into consideration and come out with one that specializes all. However, Prottoy has several distinct features such as a unique generalized interface, storage and proxy support, security and preference policy. In this paper we have provided the ins and outs of Prottoy and its approach in a summarized way. We believe Prottoy promises to provide a seamless development platform for context aware application developer.

## References


1. G. Abowd. Software Engineering Issues for Ubiquitous Computing. In *21st International conference on Software Engineering*, 1999.
2. M. Addlesee, R. Curwen, S. Hodges, J. Newman, A. W. P. Steggels, and A. Hooper. Implementing a Sentient Computing System. *Cover Feature in IEEE Computer*, 34, 2001.
3. C. Bisdikian, J. Christensen, J. Davis, G. H. Maria R. Ebling, W. Jerome, H. Lei, S. Maes, and D. Sow. Enabling Location Based Applications. In *1st international Workshop on Mobile Commerce*, 2001.
4. J. E. Bradman. Applications of Context Aware Computing in Hospital Work Examples and Design Principles. In *2004 ACM Symposium on Applied Computing*, 2004.
5. J. P. Brown. The stick-e document: A framework for creating context aware applications. In *Electronic Publishing 96*, 1996.
6. B. L. Brumittet, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easy living: technologies for intelligent environments. In *2nd International Symposium on Handheld and Ubiquitous Computing*, 2000.
7. C. Deborah and P. Debaty. Creating web representations for places. In *2nd International Symposium on Handheld and Ubiquitous Computing*, 2000.
8. 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*, 16, 2001.
9. K. Fujinami, F. Kawsar, and T. Nakajima. AwareMirror: A Personalized Display using a Mirror. In *3rd International Conference on Pervasive Computing*, 2005.
10. H. Gallerson, A. Schmidt, and M. Beigl. Adding Some Smartness to Devices and Everyday Things. In *IEEE Workshop on Mobile Computing Systems and Applications*, 2000.
11. C. Philip, A. Cheyer, M. Wang, and S. C. Baeg. An Open Agent Architecture. In *AAAI Spring Symposium Series on Software Agents*, 1994.
12. N. B. Schilit. *System Architecture for Context Aware Mobile Computing*. PhD thesis, Columbia University New York, 1995.
13. M. Spreitzer. Providing Location Information in a Ubiquitous Computing Environment. In *14th ACM symposium on Operating System Principles, ACM Press*, 1993.