Encapsulation and Entity-Based Approach of Interconnection Between Sensor Platform and Middleware of Pervasive Computing

Shinyoung Lim and Abdelsalam (Sumi) Helal

Mobile and Pervasive Computing Laboratory
Computer & Information Science and Engineering Department
College of Engineering, University of Florida
P.O. Box 116120, Gainesville FL 32611, USA
{lim, helal}@cise.ufl.edu
www.ctia.ufl.edu

Abstract. In this paper, we present a unique mechanism that enables seamless interconnection and scalability of the interface between the sensor platform layer and the middleware layer in pervasive computing. The disadvantages of using one-to-one hard coding for various device drivers and firmware for interfacing sensor platform and middleware layers are primarily a lack of flexibility and scalability when the system changes environments. The encapsulation of access point of the sensor platform layer provides developers and designers with an effective way of interconnecting and scaling up with diverse and various kinds of sensors and actuators. In this paper, we define encapsulation of the access point of the sensor platform layer. The module for encapsulating the detected sensor data is called 'context representer'. The context representer converts detected sensor data to raw context. For interconnecting with the sensor platform layer and the middleware layer, 'entity manager' module in the middleware layer for each entity of context representer is defined. The entity manager is for transforming the raw context to a general context. Then, the middleware is able to interact with context-aware applications according to the reasoning with context in the middleware layer, upper layer requests and lower layer status, without being influenced by a change of sensor node and sensor platform environments. We present the encapsulation mechanism and entity manager for efficient interfacing and scalability of context-aware applications and compare them with other approaches.

1 Introduction

Pervasive computing is considered a comprehensive autonomous computing paradigm for the next generation computing environments [1]. One of major advantages in pervasive computing is the capability of building intelligent spaces, where people and environments can interact naturally and autonomously based on the situational contexts. A smart home is one of the most tangible scenarios for pervasive computing applications, and several projects have been proposed to construct a framework for the smart home [2, 3, 15]. In the smart home, various services are automatically and intelligently provided to the users without the users necessarily being aware of them.

Currently, some computer scientists considered that machines could understand contexts of situation relating to human beings. Thus, some definitions and operational methods for context-awareness have been proposed. Based on the methods, various context-aware applications have been designed, but their applications tend to become significantly more complicated as the number of applications increase. In utilizing various contexts, such applications require several modules to sense environmental data, collect and maintain the sensed data, and derive contexts from the sensed data. Thus, a well-defined context service framework is essential for the efficient design of applications using contexts. Recently, several research papers proposing toolkits and middleware for building context-aware applications have been introduced. From these approaches, users can design context-aware applications more easily. However, most of them are lacking in reusability, scalability and interoperability of applications is still high. Specifically, they do not have any method for efficiently maintaining the plethora of sensed data.

In this paper, we propose an encapsulation and entity manager (EEM)-based context service framework that is required to design applications for work on the smart home. The EEM is constructed as layered components that will satisfy certain degrees of requirements of applications, sensors and actuators. It also categorizes and maintains the collected data according to semantic entities. Such an entity manager-based algorithm and layered architecture can improve the reusability, scalability and interoperability of the contexts and reduce the time required to collect necessary contexts. It also allows developers to design lightweight context-aware applications quickly. Eventually, context-aware applications will be able to focus on adjusting behaviors based on contexts generated by the EEM. Experiment results show that the EEM can reduce the complexity of an application by more than 70%. Also, it improves feasible delivery time and has better response time than the other context service models. In this paper, section 2 introduces related work; section 3 provides a description of the raw context by encapsulating sensor data; section 4 provides a description of the generalization of context by entity manager-based aggregation; section 5 covers implementation; and section 6 provides a performance evaluation and comparison. The conclusion is presented in Section 7.

2 Related Work

There have already been several studies undertaken on the development and implementation of context-aware applications. Context Toolkit [4] is an example of the pioneering work done this area. The Context Toolkit suggests a well-defined conceptual framework that supports context-aware applications and provides several software components, which can aggregate and manage sensed data. Thus, developers can rapidly make context-aware applications by adopting the Context Toolkit.

After the advent of the Context Toolkit, a number of other research projects were completed. Some modify architectural features, others improve performance, and still others leverage emerging technologies such as semantic web and grid computing. CIS [5] (Contextual Information Service), a part of the Aura project [6], provides users with a convenience interface for obtaining contextual information. It adopts the concept of databases and provides an SQL-like query language because users are familiar with retrieving information from databases. Thus, users can access contexts as if they were using databases. Semantic Space [7] has a similar conceptual framework to that of Context Toolkit and enhances software components by using emerging technologies. It provides RDF query and ontology-based context representation. Ubi-UCAM (Unified Context-Aware Application Model) shares object-oriented unified contexts. It suggests a method to represent basic context based on 5W1H (who, when, where, what, why, and how), and can provide higher-level contexts by composition of basic contexts [8]. Other research on context-aware computing suggests additional context framework and software components [9, 10, 12, 13, 14].

The proposed EEM framework resembles these frameworks in that it provides contextual information to context-aware applications. While the prior research does not specify any method for efficiently maintaining the plethora of sensed data and does not reflect varying degrees of requirements for applications, we introduce a unique mechanism to resolve such problems by providing both raw context, which is an outcome of the encapsulation mechanism in the sensor platform layer, and generalized context, which is not affected by any applications, through an entity manager-based aggregation of raw context. In this paper, we assume the service environments to include various sensors, sensor platforms, actuators, a home area network and home server for the specific service scenario of watching a movie in the home.

3 Raw Context by Encapsulation of Sensor Information

To begin with, the concepts and characteristics of the EEM need to be discussed. We adopt the concept model of multi-layered context, where contexts are provided with various degrees to satisfy application requirements. Figure 1 shows multi-layered contexts of the proposed context service framework. It is hierarchically divided into 4 layers, i.e., raw contexts, general contexts, application-specific context, and inferred context. Higher-level context can be formed by composition of lower-level contexts.

In this paper, the raw context is a primitive form of contexts. Various sensors do the generation of raw context. For instance, a bundle of raw context that provides bundled sensor data, such as sensed temperature from a thermometer or an object's position from location sensors. General context is the smallest unit that can have a useful meaning. It is defined as a collection of raw contexts that are related to a specific entity. The application-specific contexts contain the composed contexts that are required for a specific application. Some parts of general contexts, or raw contexts, are composed as an application-specific context.

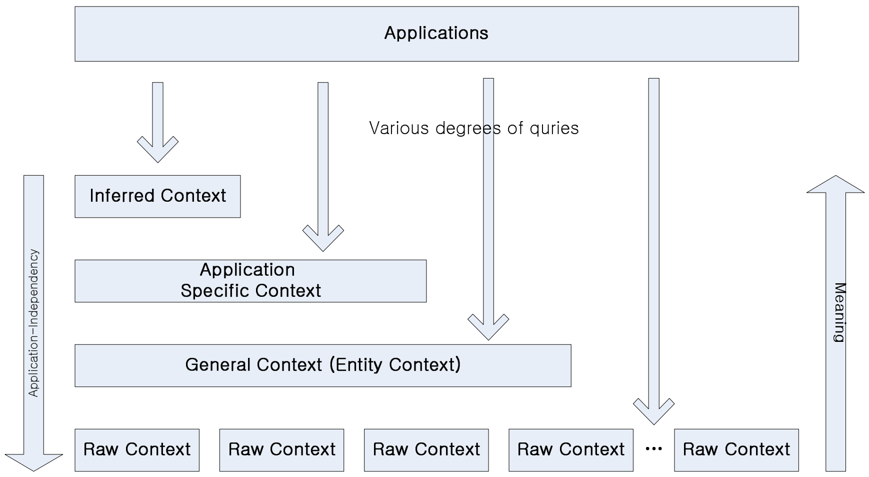


Fig. 1. The Concept of Multi-layered Contexts

Finally, inferred context is a context inferred by an inference algorithm for a specific situational context. However, it is different from application-specific contexts in terms of contexts, which are not simply a collection of lower-level contexts. This approach can provide two benefits; the variety and scalability of application programming requirements. In general, applications require various levels of contexts, from raw contexts to inferred contexts. In our framework, contexts are frequently transformed and modified. For example, additional information is added to a raw context, two contexts can be merged into one, and some information can be extracted from a single context. Because XML is a well-structured format and provides convenient APIs, we adopted the XML format to represent context. We defined the schema of context compatible with RDF (Resource Description Framework) [11] to raise interoperability, scalability and reusability of the context model. Figure 2 shows an example of context.

```
 <?xml version=“1.0”?>
 <edf:entity id=“livingRoom”
   xmlns:edf=“http://icta.cise.ufl.edu/entity-vocabulary”>
<edf:Description about=“temperature”>
     <edf:Defaults degree=“80” />
</edf:Description>
<edf:Description about=“humidity”>
     <edf:Defaults degree=“75” />
   </edf:Description>
.
.
 </edf>
```

Fig. 2. Example of the General Context

4 Generalization of Context by Entity Manager-Based Aggregation

Context can have a meaning only if the subject of a context is specified. For instance, suppose that a location sensor is used to position a location. We are not interested in the location of the location sensor itself. If someone or something being attached to the location sensor, the location information is considered to dedicate to a subject of meaningful value and then, we can use the sensed location. Here, entity is defined as an object, which can be a subject of a context. Thus, the EEM aggregates raw contexts and categorize them according to entities to generate and update general context. A sophisticated algorithm to find a particular entity related to the incoming raw context is required.

Figure 3 shows the overall architecture of the EEM, which comprises of five components. It is based on the multi-layered concept and each component is responsible for forming context in its associated layer. The *context representer* generates raw context by wrapping sensed data with XML format. The *entity manager* aggregates raw contexts and categorizes them according to entities, so that it generates general context. The a*pplication-specific context manager* generates application-specific context by composing general contexts, and the *inference module* can generate inferred context based on application-specific contexts and knowledge of application. In addition to the above four modules, *channel manager* is needed to control the flow of contexts between components. In this paper, we focus only on the context representer and entity manager for efficient interconnection between sensor platform layer and middleware layer of pervasive computing system.

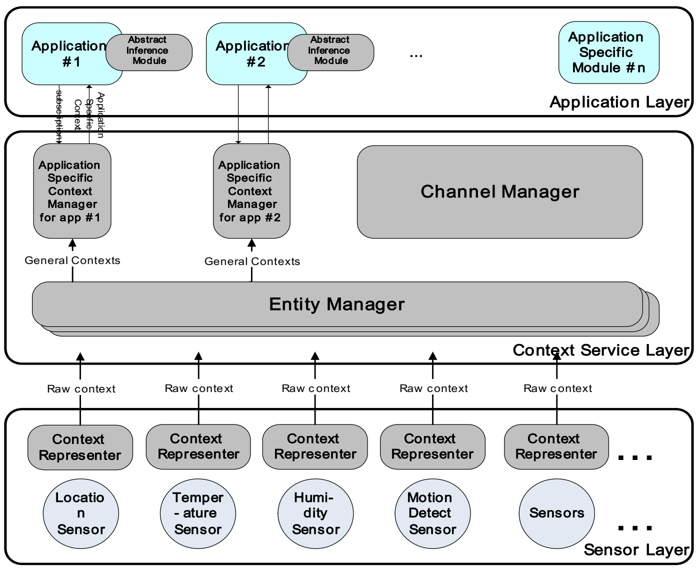


Fig. 3. Architectural overview of the proposed EEM-based Context Service Framework

The basic role representer is to wrap sensor data with a uniform interface. Sensors are varied from a physical thermometer to weather forecasting devices and they have different interfaces. Thus, it is too hard for upper-level components or applications to understand interfaces of various sensors. The context representer wraps sensor data with XML-based interface and formats of sensor data according to the XML schema. Therefore, upper-level components can obtain the raw context only by retrieving XML messages regardless of types of sensors. As we define the raw context has meta-data such as sensed time and valid period of the raw context by adding elements, the raw context can provide more accuracy and validity.

Figure 4 shows an example of raw context. All data, including both sensed data and meta-data in raw context, are expressed with ‘edf:description’ elements. If role attribute is marked as ‘meta’, it is meta-data.

```
 <?xml version=“1.0″?>
 <edf:sensor id=“lightSensor″
   xmlns:edf=“http://icta.cise.ufl.edu/entity-vocabulary″
   ref=“http://icta.cise.ufl.edu/sensor/lightSensor_type12″>
<edf:Description about=“brightness″ role=″main″>
     <edf:Defaults degree=“65″ />
</edf:Description>
<edf:Description about=“DetectionTime″ role=″meta″>
     <edf:Defaults location=“20060415123000″ />
</edf:Description>
<edf:Description about=“TimeToLive″ role=″meta″>
     <edf:Defaults location=“100″ />
   </edf:Description>
 </edf>
```

Fig. 4. An example of raw context

The entity manager generates and maintains general context by aggregating raw contexts and categorizing them by entities. For this, the entity manager includes a sophisticated categorization algorithm to find a particular entity, which is related to the incoming raw contexts and additional data structures. Figure 5 shows the internal diagram of the entity manager.

The entity manager comprises of an event processor, event buffers, a sensor registry, and an entity registry. All sensors are registered to the sensor registry. The sensor registry maintains ID, type, and location of sensors in the smart home. It classifies sensors into three types, i.e., predefined sensor, entity detector, and location-based sensor. The predefined sensors are sensors those are defined and registered individually for special purposes, i.e., for people with special needs. The entity detectors are various types of sensors attached to the objects excluding location sensors. For instance, RFID reader, temperature sensor, humidity sensor, or acoustic sensor can be an entity detector.

When a raw context comes to the entity manager, it is regarded as an event. The entity manager can extract information from the raw context, generate an event, and then store it in the event buffer. An event includes 4 parameters, i.e., sensor ID, sensed data, detection time, and valid period. Now, the event processor dispatches the event and processes it by following the algorithm shown in Figure 6.

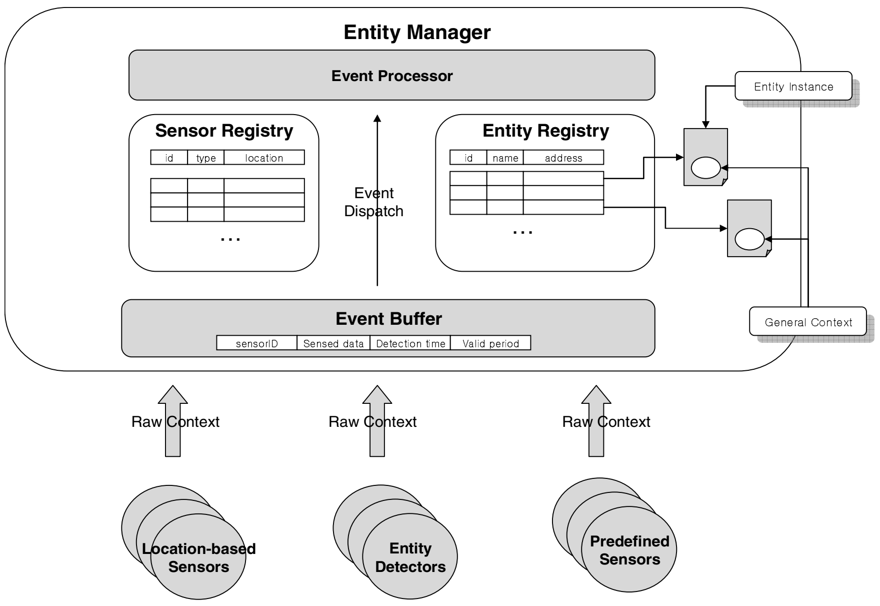


Fig. 5. Component diagram of the Entity Manager

First of all, it checks the type of the sensor by scanning the sensor registry. If the type of the sensor is predefined sensor, it can easily find the related entity because the subject of data from the sensor is already defined. It is the reason why the sensor is called predefined sensor. As a consequence, we can find the proper entity on entity registry and update its general context by using sensed data. Every entity has an XML document which represents the general context of the entity. Entity detector is the sensor that can identify entities. For example, RFID reader attached to the moving chair of the person with special needs can be an entity detector.

Basically, it provides not only the unique identification of the entity, but also can infer the location of the entity based on the location of the sensor. Thus, if an event occurs from the entity detector, the event processor can modify the location of the entity. If it cannot find the entity on the entity registry, event processor creates an instance of the entity and registers it into the entity registry. Finally, if the type of the sensor is location-based sensor, the subject of sensed data is determined by matching the location of the sensor with the location of entities. Assume a scale that can measure body weight. When the scale sensed a body weight, the sensed data is subjected to the entity, which has same location with the scale.

Thus, if an event is coming from the location-based sensor, the event processor finds the entity that has the same location as the sensor, and updates the general context of the entity by using sensed data. As shown in Figure 2, a general context includes information from multiple sensors related to an entity.

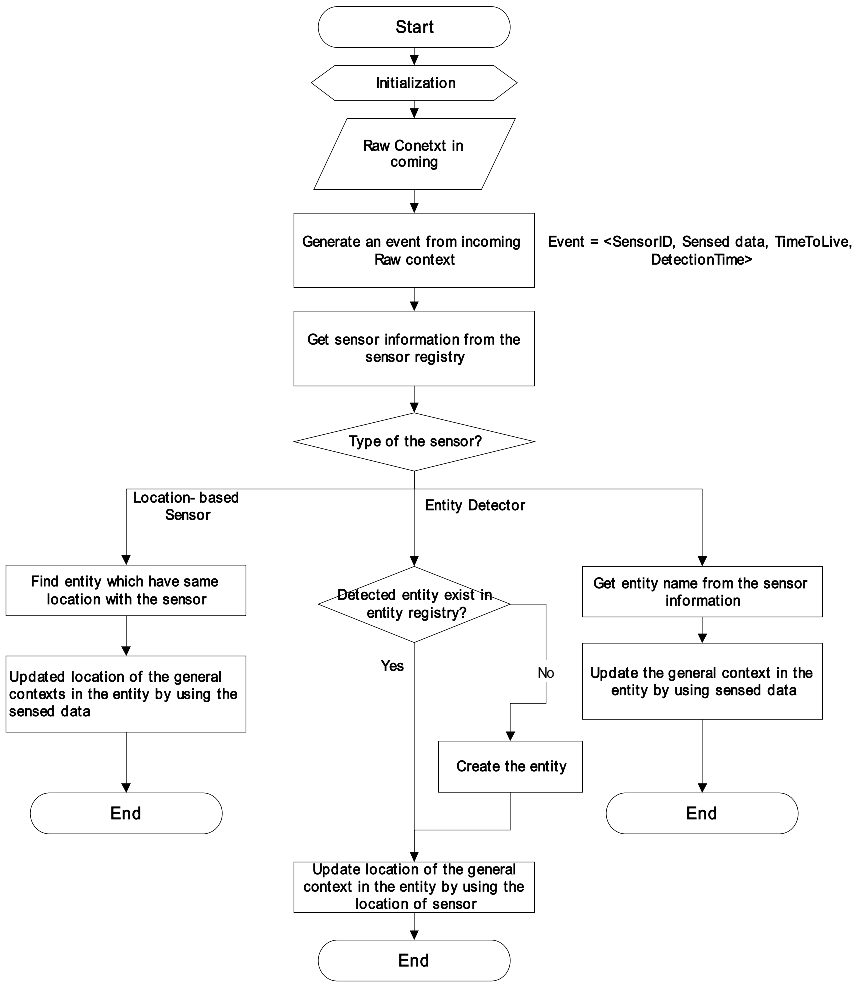

Start
Initialization
Raw Conetxt in coming
Generate an event from incoming Raw context
Event = <SensorID, Sensed data, TimeToLive, DetectionTime>
Get sensor information from the sensor registry
Type of the sensor?
Location- based Sensor
Entity Detector
Find entity which have same location with the sensor
Updated location of the general contexts in the entity by using the sensed data
End
Detected entity exist in entity registry?
Yes
No
Create the entity
Update location of the general context in the entity by using the location of sensor
End
Get entity name from the sensor information
Update the general context in the entity by using sensed data
End

Fig. 6. Operational flow of Generating and Updating a General Context

5 Implementation

Applications can retrieve general contexts by connecting the entity manager and raw contexts by connecting context representers. For example, assume an intelligent home theater application, which can control environments according to the user and user's situational status. Figure 7 shows a home theater service scenario utilizing the EEM-based context service framework, which depicts components functioning in the application and interactions between components. In this scenario, user 'Lim' watches the movie 'Matrix3' with his home theater system in his livingRoom. To detect the contexts for the application, the light sensor, the RFID, the video device, the motion detect sensor, and the location sensor continuously sense the contexts.

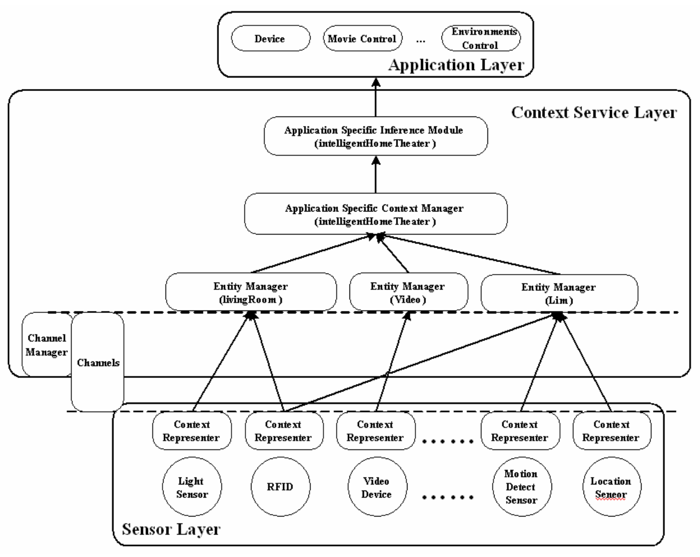


Fig. 7. Home Theater service scenario utilizing EEM-based Context Service Framework

The entity managers for the ‘livingRoom’, ‘Video’, and user ‘Lim’ aggregate the raw contexts from the sensors through the channels managed by the channel manager. The application specific context manager maintains the application specific contexts formed by the general contexts to handle the request of the application and the application specific inference module infers the situational contexts or the inferred contexts using the application specific contexts.

```
 <?xml version="1.0"?>
 <edf:entity id="lightSensor"
   xmlns:edf="http://icta.cise.ufl.edu/entity-vocabulary"
   ref="http://icta.cise.ufl.edu/sensor/lightSensor_type12">
<edf:Description about="location">
     <edf:Defaults location="livingRoom" />
   </edf:Description>
   <edf:Description about="brightness">
     <edf:Defaults degree="50" />
   </edf:Description>
 </edf>

 <?xml version="1.0"?>
 <edf:entity id="movementSensor_202"
   xmlns:edf="http://icta.cise.ufl.edu/entity-vocabulary"
   ref="http://icta.cise.ufl.edu/sensor/movementSensor_livingroom">
   <edf:Description about="movement">
     <edf:Defaults location="4_3" status="sit" />
   </edf:Description>
 </edf>
```

Fig. 8. Raw Contexts from a light sensor and a movement sensor

```
   <?xml version="1.0"?>
   <edf:entity id="Lim"
     xmlns:edf="http://icta.cise.ufl.edu/entity-vocabulary"
  ref="http://icta.cise.ufl.edu/temporary-testing/video/spv20060415">
  <edf:Description about="movement">
       <edf:Defaults status="sit" />
     </edf:Description>
     <edf:Description about="location">
       <edf:Defaults name="livingRoom">
     </edf:Description>
   </edf>

   <?xml version="1.0"?>
   <edf:entity id="video"
     xmlns:edf="http://icta.cise.ufl.edu/entity-vocabulary"
     id="spDemoDevice_110"
     ref="http://icta.cise.ufl.edu/temporary-testing/video/spv20060415">
     <edf:Description about="status">
       <edf:Defaults  status="play"  screen="16:9  wide"  audiodecode="dolby
digital"
         volume="12" />
     </edf:Description>
     <edf:Description about="content">
       <edf:Defaults movie="matrix3"
         ref="http://ictacise.ufl.edu/temporary-testing/matrix3-
movieguide.xml" />
     </edf:Description>
   </edf>

   <?xml version="1.0"?>
   <edf:entity id="livingRoom"
     xmlns:edf="http://icta.cise.ufl.edu/entity-vocabulary">
     <edf:Description about="People">
       <edf:Defaults number="1">
         <edf:Detail about="Person" who="Lim" />
       </edf:Defaults>
     </edf:Description>
     <edf:Description about="brightness">
       <edf:Defaults degree="50" />
     </edf:Description>
     <edf:Description about="device">
       <edf:Defaults id="spDemoDevice_110" name="video"
         ref="http://icta.cise.ufl.edu/temporary-testing/video/spv20060415"
/>
     </edf:Description>
   </edf>
```

Fig. 9. General Contexts about Lim, Video, and Livingroom

Figures 8 and 9 show several contexts generated by the context engine for the intelligent home theater application. Figure 8 shows the raw context format based on the XML technology, and 'edf' is the 'entity description framework' based on the RDF. The upper XML message on Figure 8 means that the degree of 'lightSensor' in the living room is 50. Moreover, the bottom message shows that the 'movementSensor' senses 'sit' status at location '4_3'.

General contexts are those categorized raw contexts collected for each entity. In this example, the specific user 'Lim', the 'video device', and the 'living room' are the entities. Figure 9 presents the case of general contexts. The beginning of message in Figure 9 shows contextual information related to user 'Lim', and the middle of message in Figure 9 shows the contextual information about device 'video' which can be referenced at 'http://icta.cise.ufl.edu/video/spv10311254'. Currently, the device plays 'matrix3' with '16:9 wide' screen and 'dolby digital' audio method. The bottom part in Figure 9 contains the general context of the 'livingRoom' entity. In this 'livingRoom', there are a person named 'Lim', and a 'video' device, which has id 'spDemoDevice_110'. Moreover the degree of its brightness is 50.

6 Performance Evaluation and Comparison

The implementing and experimental environments are AMD Athlon XP2500, 512 MB RAM, Windows XP, JESE 1.4.2_05 SDK and Eclipse 3.0.1. In this paper, our evaluation metrics are confined to the delivery time of sensed data and the average response time to retrieve a context between sensor platform and middleware of pervasive computing. To measure such metrics of the EEM, we deploy implemented components as shown in Figure 10. All components of the EEM are located in a machine named home server, and each sensor and application are placed in different machines. All machines are connected with local area network. Applications communicate with application specific context manager (ASCM) by using RMI calls and sensors send the data to the context representer (CR) by using TCP/IP socket API. Communication between components in home server also performed via RMI calls.

Also, we compare the EEM with two other context service models. One is single application model where an application includes all components to handle contexts

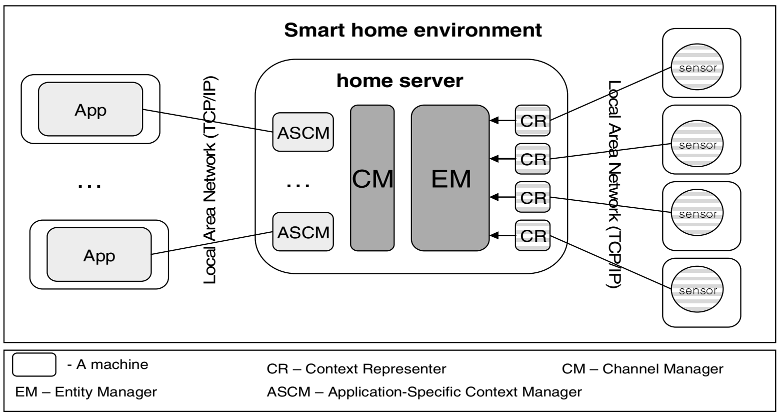


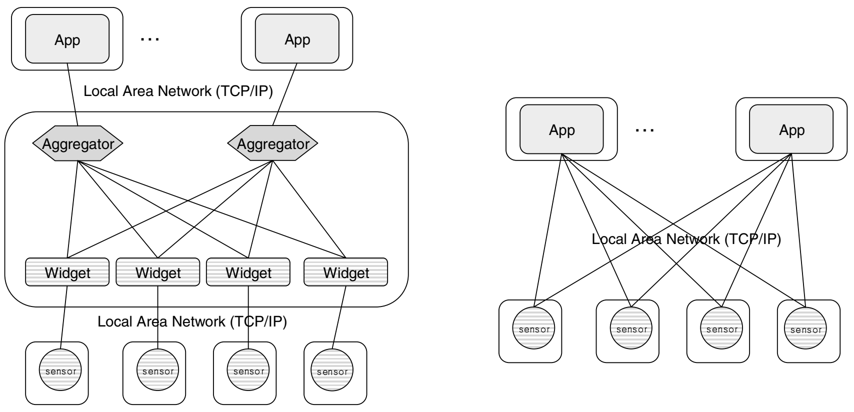
Fig. 10. Experimental Environment

like centralized application system. The other is Context Toolkit, which is a well-known context-aware application development kit. We deploy two models similar to our context service as shown in Figure 11.

6.1 Delivery Time

Delivery time refers to the time spent delivering data from specific sensor to an application. It is especially important in contextual information services because they have to guarantee that the information is provided to the applications in a pre-specified time period. Figure 12 show the delivery time from a sensor to an application in the EEM. It consists of three parts, including two communication parts and one processing part. The first part of the delivery time is the time of transmission of the sensed data from the sensor to the server where the context representer resides. Then, the datum is sequentially processed and formed as raw context, general context and application-specific context. The final part of the delivery time is the time taken to transmit the application-specific context to the application layer. The communication parts do not reflect any characteristics of the context service itself, but depends on the network status.

Processing time is determined by the operational characteristics of context services. Based on the sensed data, the EEM creates contexts as XML documents. Thus, most of processing time is to parse XML documents and to handle information in XML trees. Also, it includes local RMI invocation time between components. According to our experiments, such a processing time does not exceed 10ms and the total delivery time is about 15ms in our experimental environments as shown in Figure 10. We repeat the same experiment over the 1,000 times and obtain a mean value to guarantee high confidence interval. Specifically, 15ms of delivery time is quite feasible because valid



(a) Context Toolkit **(b) Single Application Model**

Fig. 11. Two-context service models for comparison

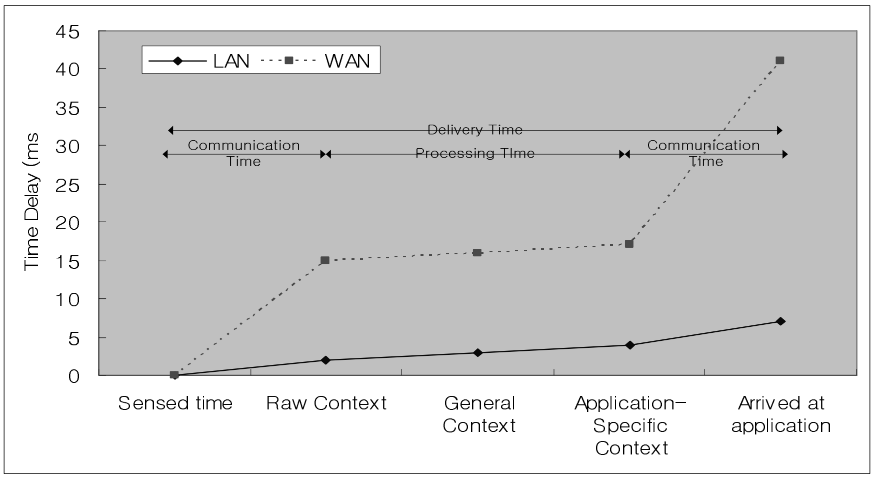


Fig. 12. Time delay of the EEM

time period of sensed data takes generally a couple of seconds. We also perform same test on the WAN environment, where sensors and applications are located in different spaces, which are far from the EEM. Dotted line in Figure 12 shows the delivery time of the test in WAN environment. In this case, communication environments dominate delivery time. But, the processing time is not different from the LAN case because processing is performed only in a computer system.

EEM vs. Single Application Model: Naturally, the delivery time for a single application model is fastest because the application gathers sensed data directly from the sensors. There is no additional overhead in delivering sensed data to an application. In this case, sensed data can be provided to an application in 5ms. But, the application has to process the sensed data after the data has arrived at the application. Thus, delivery time is not so different from the EEM.

EEM vs. Context Toolkit: Delivery time in Context Toolkit is similar to the EEM. It consists of two communication parts and one processing part. Communication parts are the same with the EEM and the processing part can be different. But, overall delivery time is not so different from the EEM in our experiments. Figure 13 shows the delivery time of three context service models in the same environment. Single application model shows 9.7ms, Context Toolkit shows 16ms, and the proposed EEM-based context service shows 15.7ms on the average. All models have similar delivery time and those are quite feasible. Finally, from the viewpoint of delivery time, although the proposed EEM-based context service includes several complicated processing, but it does not influence on the delivery time.

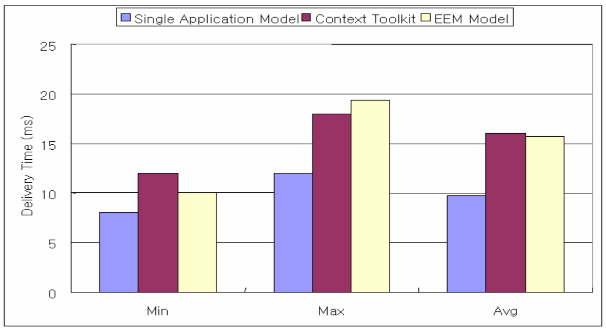


Fig. 13. Delivery time of three context models

6.2 Response Time

Response time is defined as the period the application waits for the data from the sensors. It seems to be similar to delivery time, but the meaning is quite different. Response time is regarded as the actual waiting time of applications in various situations, whereas delivery time can show only simple processing time of given components. For instance, response time can be the time it takes for sensors to send the required data, sensed data, to the application. This means response time can be influenced by the complexity of the system, i.e. various system parameters such as the number of sensors and applications. We compare the response time of three models while changing the complexity of the system.

EEM vs. Single Application Model: Basically, the EEM can provide better response time than the single application model because sensed data are already collected in the EEM and the requests of application are processed in the context service. Namely, the application do not concern about sensors. However, response time of single application model requires more than double of delivery time. And single application model is sensitive to the number of sensors and applications. When the number of applications increases, race condition occurs between applications using same sensors, so that response time can be degraded. Also, when the number of sensors used for an application increases, the overhead of application also increases because the application has to communicate with all the sensors directly and process sensed data by itself. As a consequence, the response time is degraded.

EEM vs. Context Toolkit: As Context Toolkit has a similar architecture with the EEM, it also does not concern about sensors, but just communicates with aggregators that already collect the necessary sensed data. However, Context Toolkit is also sensitive to the number of sensors and applications because it does not maintain generalized context. Typically, an aggregator in Context Toolkit is assigned to an application to collect relevant contexts, which are required to the application. Therefore, when the number of applications increases, there can be a race condition between aggregators, which use same widgets. Therefore, response time can be degraded. But the EEM maintains and provides generalized-level of context by the entity manager, so that the race condition occurs less even though the number of applications increases. Figure 14 shows the comparison in response times of each context frameworks. We measure the response time while changing the number of applications and the number of sensors and repeat same test over 1,000 times to verify the complexity by way of response time. As shown in the figure 14, response time of the EEM is lower than Context Toolkit by about 20% and lowers than single application model by 55% when we run simultaneously 5 application programs based on 15 sensors. The gap of response time is increased as the number of sensors and applications increases.

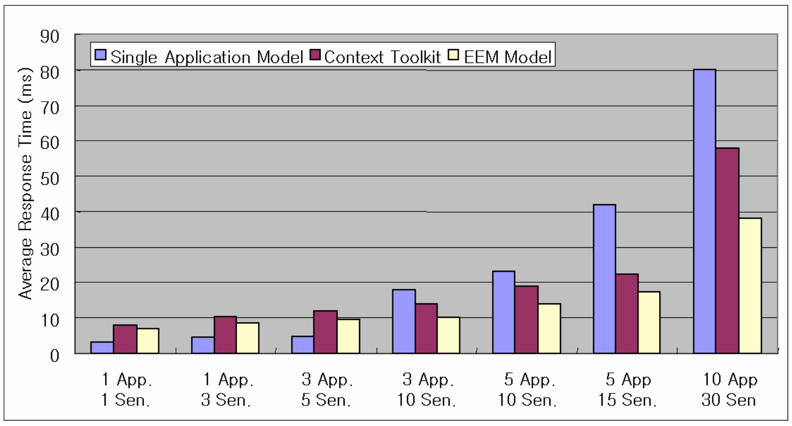


Fig. 14. Response time according to the number of sensors

7 Conclusions

In this paper, we proposed an encapsulation and entity manager-based context service to support context-aware applications in the smart home. It can provide various levels of context and categorize and maintain the collected data according to semantic entities. Thus, with the EEM model, applications can focus on adjusting their behaviors to the environment based on given contexts from the EEM. The result of experiment on delivery time and response time shows that the EEM can provide feasible delivery time and better response time than other context service models from the viewpoint of interconnection between sensor platform and middleware of pervasive computing.

References

[1] M. Weiser, "The computer for the 21st Century," *Scientific American*, 265(3), 66-75.

[2] B. N. Schilit, N. L. Adams, R. Want, "Context-aware computing applications," *Proc. of Workshop on Mobile computing systems and applications.*, Santa Cruz, CA. Dec. 1994.

[3] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. "EasyLiving: Technologies for intelligent environments," *Proc. of Second International Symposium on Handheld and Ubiquitous Computing, HUC 2000*, pp. 12-29, Bristol, UK, September 2000.

[4] A. K. Dey, D. Salber, and G. D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human Computing Interaction*, Vol. 16, pp. 97-166, 2001.
[5] G. Judd, P. Steenkiste, "Providing Contextual Information to Ubiquitous Computing Applications," *Proc. of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, 2003.
[6] J. P. Sousa and D. Garlan, "Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments," *Proc. of the 3rd Working IEEE/IFIP Conference on Software Architecture*, pp 29-43, Aug. 2002.
[7] X. Wang, J. Dong, C. Chin, and S. R. Hettiarachchi, "Semantic Space: An Infrastructure for Smart Spaces," IEEE Pervasive Computing, pp 32-39, Jul. 2004.
[8] S. Jang and W. Woo, "Ubi-UCAM: A Unified Context-Aware Application Model," *Lecture Notes in Artificial Intelligence*, Vol. 2680, Springer-Verlag, Berlin Heidelberg New York (2003), pp. 178-189.
[9] A. Ranganathan and R. Campbell, "An infrastructure for context-awareness basd on first order logic," Personal Ubiquitous Computing, 2003.
[10] T. Winograd, "Architecture for Context," *Human Computing Interaction*, Vol. 16, pp. 401-419, 2001.
[11] "RDF (Resource Description Framework) online at http://www.w3.org/TR/rdf-syntax-grammar/," *World Wide Web Consortium*, 10 February 2004.
[12] J. I. Hong and J. A. Landay, "An Infrastructure Approach to Context-Aware Computing, " *Human Computing Interaction*, Vol. 16, pp. 287-303, 2001.
[13] H. Lei, D. Sow, J. Davis II, G. Banavar, and M. Ebling, "The Design and Applications of a Context Service," Mobile Computing and Communications Review, Vol. 6, No. 4, pp. 45-55, 2002.
[14] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The Anatomy of a Context-aware Application," *Proc. of MOBICOM 1999*, Seattle, WA, August 1999.
[15] S. Lee and T. Chung, "System Architecture for Context-Aware Home Application," *Proc. of Second IEEE workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, Vienna, Austria, May 11-12, 2004.