A Context Gathering Framework for Context-Aware Mobile Solutions

Anusuriya Devaraju
University Technical Malaysia Melaka
Locked Bag 1200, Hang Tuah Jaya, Ayer Keroh, Melaka, Malaysia
+606-2332318
anusuriya@utem.edu.my

Simon Hoh
British Telecommunications Plc
1B-17, Plaza Sentral, Jalan Stesen Sentral 5, Kuala Lumpur, Malaysia
+603-20919328
simon.hoh@bt.com

Dr. Michael Hartley
University of Nottingham
Jalan Broga, Semenyih
Selangor Darul Ehsan
Malaysia
+603-89248137
michael.hartley@nottingham.edu.my

ABSTRACT

One of the fundamental design issues in context-aware mobile services development is the necessary support for adequately powerful yet efficient querying of the sensory data. This issue argues for research into the creation of a technology-independent, high-level software application programming interface (API) that provides mechanisms for dealing with the heterogeneity of sensors providing raw context data. In this paper, we review approaches in existing context-aware platforms especially those that consider with sensory data acquisition. The review formed the basis for the design and development of the context gathering framework which consists of sensor data model, messaging and communication protocol and software application programming interface. These components form as one of the enabler to support the development of context aware mobile applications.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems – *Client/Server, Distributed Applications*

General Terms

Design, Standardization, Languages.

Keywords

Context Awareness, Sensor, Framework, Protocol

1. INTRODUCTION

There is an abundance of information around us that is often taken for granted. At times, the subconscious mind acts upon these environment stimuli without the individual even noticing it. Research in context awareness seeks to enable a new breed of applications and services which would extend the functionality of the human's subconscious, being able to provide or act at the right time, with the right level of information.

As computing becomes increasingly mobile and pervasive today, it is necessary that mobile applications and services must 'aware' and adapt to highly dynamic environments to enhance end user's experience. In order to develop enablers for context aware mobile applications, one of the initial requirements is a mechanism for gathering raw context information itself. Here, we propose a generic context gathering framework that will simplify the process of acquiring sensor data and delivering them to our context-aware service platform.

A context-aware computing scenario from [13]:

"Cheryl has invited her boyfriend and his parents for dinner tonight. The dinner will take place at her house at 8pm and she is currently sitting in an important meeting with her manager. Fortunately, Cheryl has her Reminder Buddy, RB, running on her mobile phone, to take care of notifying her, if she forgets something. Knowing calendar of Cheryl, RB assumes she has forgotten the dinner, therefore RB decides to notify Cheryl about the dinner menu preparation. However, RB realizes, after verifying with the manager's personal assistant, that Cheryl is currently having a meeting, and the notification had better be given after the meeting is over. The meeting is over, and Cheryl walks to her office. RB informs Cheryl about the dinner tonight. When she enters car at the parking lot, she asks RB for menu preparation. RB communicates with Cheryl's electronic Household Buddy, HB, which registers any discovered item via RFIDs that come with each product. HB indicates that there are not enough ingredients in her refrigerator, thus he suggests a menu and a related shopping list to RB. The shopping list is not only based on the contents of the refrigerator, but it also takes into account the Cheryl's and the parent's favorite recipes. Based on responses from HB, RB checks the availability and price of products in the supermarkets along the route to Cheryl's home. RB selects one or two markets, which offer all the products in order to avoid several stops, and displays the info to Cheryl. Related recipes are automatically transferred to HB and HB will display them on LCD in the kitchen once Cheryl enters the house."

The example above refers to a context-aware computing scenario. This scenario reveals how the 'contexts' or characteristics of the surrounding environment (for example, Cheryl's 'home environment', 'office' and 'way to work') that determine and adapt the behaviors of applications (RB and HB). The applications are able to specify the request according to the interest in a certain part of the available context information. To be able to use context in above application, of course, there must be a mechanism to sense the current context data from sensors such as RFIDs at the products and respective readers, to deliver them to the interested applications.

1.1 Background

Common factors that lead to the difficulties in context-aware system design and implementation are the plethora of sensing technologies with differing data formats. As such, it is difficult to abstract the data for the application in a standard manner. For example, mobile devices may acquire location information in the form of geographical coordinates from outdoor GPS receivers or indoor positioning systems. But tour guide applications would make better use of higher-level information such as street or building names [20].

Addressing above difficulties, we have followed the 'middleware' approach to develop a context-aware service platform, named CASP that can assist the context-aware service providers to build and deploy services. CASP provides the fundamental required components as an abstraction layer for the developed context-aware services. The components provided by our platform are sixfold: Context Sensing, Context Modelling, Context Association, Context Storage and Retrieval.

The *Context Sensing* in CASP relies on sensors to observe aspects of the context. To make the sensory outputs made available for the platform, a framework is required to interface with sensors and to deliver the sensed data to the *Context Modeling* component in CASP.

The paper is organized as follows. Section 2 clarifies terminology used throughout this paper. This is followed by a brief introduction to context-data acquisition methods when designing context-aware systems. The relative strengths and weaknesses of context sensing components in chosen context-aware platforms appear in Section 4. In Section 5, we present requirements and design of the context gathering framework. Next, we open a discussion highlights some likely future directions of the framework development. Finally, conclusions are presented.

2. TERMINOLOGY

Context awareness is based on a group of interrelated areas of research: mobile computing, ubiquitous and pervasive computing, serviceware networking, programmable networks, autonomic communications and ambient computing. In these research areas, context has been used to enhance human-computer and computer-computer interaction, thereby providing seamless computing and networking anytime, anywhere [18].

2.1 What is Context?

With the many implementations of applications and services that make use of context information, context has been interpreted very differently. Here, we investigate how different researchers define context, starting with a generalized definition from dictionary references.

Dictionary references define 'context' as

a. WordNet : Lexical Database for the English Language [24]
    i. *'discourse that surrounds a language unit and helps to determine its interpretation'*
    ii. *'the set of facts or circumstances that surround a situation or event'*
b. The Free On-line Dictionary of Computing: '*That which surrounds, and gives meaning to, something else'*.

Context Synonyms are [23] *Circumstance, situation, phase, position, posture, attitude, place, point; terms; regime; footing, standing, status, occasion, surroundings, environment, location, dependence*.

Dey [5] defined context as *'Any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and application themselves'*.

Panayiotou [16] defined context as *'A set of premises expressed in some language, gathered intentionally or unintentionally in a relevant, coherent manner and which can itself constitute and adequate set of inferences (meaningful) or lead to some meaningful results (inferences)'.*

Rakotonirainy et al [17] stated that *'Context (from an entity's viewpoint) is information that can be used to characterize the situation of an entity and can be obtained by the entity, where an entity can be a person, place, physical or computational object'.*

Schmidt et al [22] defined context as *'Knowledge about the user's and IT device's state, including surroundings, situation and to a less extent, location'.*

Chen et al [4] suggested context as *'Set of environmental states and settings that either determines and application's behavior or in which an application event occurs and is interesting to the user'.*

Moran et al [15] referred context as *'Physical and social situation in which computational devices are embedded'.*

From the various refinements in the definition of the term context awareness, Dey's definition have become widely accepted and adopted to provide a consistent understanding of the subject studied by researchers [10].

2.2 What is Sensor?

According to [21], one can broadly define sensor as device, hardware, or software or their combination, that can be used to acquire context information. This definition of sensor is broad; devices not normally thought of as sensors might be also used to return context information and, therefore, are sensors under this definition, for example, the computer clock accessed using an operating system call or a video camera. In our work, the level of abstraction of context information is a key idea related to our definition of sensor. For instance, a GPS sensor on mobile device can provide user's location readings, but an application querying a 'positioning service' to return current location of that user can also be regarded as sensor from the perspective of an application. Therefore, in this paper we refer 'sensor' as any resource that outputs raw context data.

In our opinion, there is distinction between 'sensor data' and 'context data'. Generally, sensor data has several properties which affect how it is interpreted as higher-level context data. In our work, we use the term '*raw context data*' or '*sensor data*' to represents raw data sensed from the sensors. When the subsequent pre-processing and reasoning applied to the sensor data by our platform, the resulting information regarded as '*context data*' or '*context information.*'

3. CONTEXT ACQUISITION METHODS

The method of context-data acquisition is very important when designing context-aware systems because it predefines the architectural style of the system at least to some extent [2]. Chen [3] defines the following are three categories of context acquisition methods:

i. *Direct access to hardware sensors***.** The sensors are integrated into the devices. The client software gathers the context information directly from these sensors rather than by a specialized infrastructure.
In this approach, typically drivers for the sensors are hardwired into the applications. The high-level applications can have great controls over the operations of the low-level sensors, thus can have better knowledge about how different data is collected and computed. However, this method is not suited for distributed systems due to its direct access nature which lacks a component capable of managing multiple concurrent sensor accesses [2]. Further, typically, only small amount of context can be determined and supplied in a resource constrained device.

ii. *Facilitated by a middle-ware infrastructure*
The idea is that instead of letting the applications to manage the low-level sensing details, middleware infrastructures are provided to facilitate sensing. Context acquisition middleware typically built into the hosting devices or platform on which the context-aware applications operate.

The advantage of this method is context-aware applications' implementations can focus on how to use context while middleware can focus on how to acquire the context. This separation means that both components can be developed and replaced independently of each other. Compared to direct sensor access this technique eases extensibility since the client code has not be modified anymore and it simplifies the reusability of hardware dependent sensing code due to the strict encapsulation [2]. The drawback of middleware approach is it imposes additional computation burden on the hosting devices because it consumes certain amount of computation resources to maintain a generic programming interface between the high-level applications and the low-level sensors. This might lead to resource contention problem in resource-less devices, for instance, mobile phones and embedded devices.

iii. *Acquire context from a context server*
This approach shifts the context acquisition procedures into the implementation of a server entity that runs on a resource-rich device. The server entity provides contextual information to different context-aware applications in a distributed environment.

This approach overcomes the drawbacks noted with hosting device of a context-aware application that has limited computing resource. Besides the reuse of sensors, the usage of a context server has the advantage of relieving clients of resource intensive operations [2]. As the server runs on a resource-rich device, it can preserve the history of more contextual data sensed. It can detect and resolve inconsistent information that may have been acquired from unreliable sensors. However, one has to consider about appropriate protocols, network performance, and quality of service parameters and so on when designing a context-aware system based on client-server architecture [2].

4. RELATED WORKS

A comprehensive context aware system has the ability to illicit information from the environment supports information interpretation that is relevant to any application/service and the situation at any point in time, and be able to formulate a reaction to the said situation. Figure 1 [21] categorized the components of a Context Aware Pervasive system as sensing, thinking and acting.

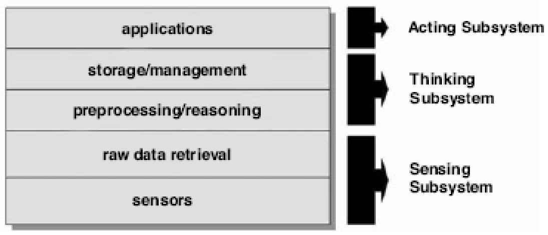


Figure 1. Abstract layered architecture for context aware systems.

Sensing forms the initial part of the equation where it looks at how raw environmental information could be acquired, represented and interpreted in a cohesive manner. A significant body of related research has already been carried out in the area of developing appropriate high-level programming abstractions and toolkits to simplify the process of sensory data acquisition. The following section discusses the strengths and weaknesses of existing implementation.

i. Context Toolkit

Dey and Abowd [6] have defined Context Toolkits in 1999 to create a framework for the Context Aware application development. In the architecture, the context toolkits have three primary components which are widgets, aggregators and interpreters. These three components provide the abstraction to the context aware application on contextual information. The widgets are the source of contextual information. It extracts contextual information and translates raw data from sensors that are monitoring the environment. The interpreters will then further derive the information to more meaningful higher level contextual information. Lastly the aggregators help to aggregate the contextual information to minimize the complexity of context aware applications.

The major drawback of the Context Toolkit is, therefore, its context model, a set of attribute-value tuples. Such attributes do not have a meaning, thus no contextual data modeling supported to enable context knowledge sharing and context reasoning [2]. Besides, using non-ontology based models requires a lot of programming effort and tightly couples the context model to the rest of the system. The tight coupled widget approach is not robust to component failures. In addition, there is no quantifiable aspect for quality of context within the Context Toolkit.

ii. Technology for Enabling Awareness (TEA)

The TEA project introduces a three layer approach to abstracting and representing context information. The lowest layer is the sensors itself, captures raw data from a set of heterogeneous sensors. The proceeding layer represents the functionalities of each sensor in terms of functional cues. The third layer, context layer, aggregates the cues from various sensors to suggest the context.

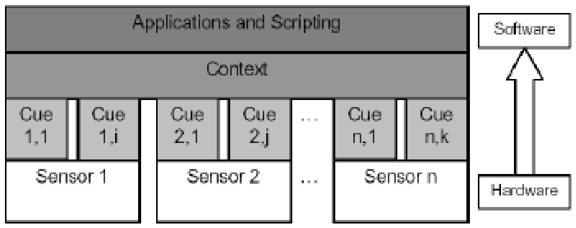


Figure 2. TEA architecture

Cx = ∑ Cn,k [where n = sensor number and k = cue of sensor n]

Here, context is said to be made up of one or more cues from the various sensors that are capturing the environment. This enables a context action to take place when the correct cues are met, demonstrating the context aware behavior in applications. Nevertheless, the TEA project lacks formality in modeling of the context information [10]. The proposed context model couples the representation of context with its direct use, thus fail to clearly separate concerns making it less extensible. The architecture does not prescribe the specific methods for calculating context from cues; rule-based algorithms, statistical methods and neural networks may for instance be used. Conceptually, context is calculated from all available cues [8]**.**

iii. A Service-Oriented Context-Aware Middleware (SOCAM)

Service-Oriented Context-Aware Middleware (SOCAM) is a middleware architecture that targets to enable rapid prototyping of context aware services [9]. SOCAM models the contextual information based on the ontology using OWL to resolve the issues of semantic representation, context reasoning, context classification and dependency. SOCAM define ontology in OWL to enable it to describe context semantically which is independent from any programming language and enabling computer system to understand the semantic value. This combination of technology enables the formal analysis on domain knowledge that could be done automatically by the computer system. A set of independent services is provided within SOCAM to facilitate the context aware applications and enabling contextual information exchange with other context providers. These services provide the fundamental functionalities such as context acquisition, context discovery, context interpretation and context dissemination.

iv. Context-awareness sub-structure (CASS)

The CASS [7] is a server-based middleware to facilitate context-aware applications on mobile computers connecting over wireless networks. In CASS, the sensory component is implemented via sensor nodes, essentially computers with sensors attached to collect sensory data. One or more sensors may be attached locally at a sensor node. Because the middleware is server-based, it does not suffer from the processor and memory constraints that would apply on a mobile computer. This allows use of a database and artificial intelligence components as required, as well as the facility to store large amounts of data. However there is a reliance on communications between the mobile platform and the server hosting the middleware. Because of this reliance, CASS supports applications in the use of local caching of information to reduce the effect of intermittent connections.

4.1 Summary of Related Works

Currently, there is no standard description language or ontology for sensing contextual information from various sources to enable reuse across various middleware systems and frameworks. Therefore, proprietary solutions as used by the different frameworks have emerged as summarized in Table 2 [2].

Table 1. Context Sensing Summary

Project	Architecture	Sensing	Context Model
Context Toolkit	Widget based	Context widgets	Attribute-value tuples
CASS	Centralized middleware	Sensor nodes	Relational data model
Context Broker Architecture (CoBra)	Agent based	Context acquisition module	OWL Ontology, SOUPA
Cooperating Real-time sentient objects (CORTEX)	Sentient object model	Context component framework	Relational data model
Gaia	Model-View-Controller (MVC) (extended)	Context providers	4-ary predicates (DAML + OIL)
Hydrogen	Three layered architecture	Adapters for various context types	Object-oriented
SOCAM	Distributed architecture	Context providers	OWL Ontology
Me-Centric Domain Server	Domain Server Architecture	relational database (for the internal knowledge base)	Ontology (RDF)
Context Mediated Framework (CMF)	Centralized blackboard architecture	Resource servers	Ontology based
STU21 (associates with the CoBrA context	Distributed agent-based	Sensor Agent	OWL & RDF

broker model, expanding that model using concepts from JCAF)			
Java Context-Awareness Framework (JCAF)	Service-oriented, distributed, event-based	Context Monitor	Object-oriented models in Java

5. CONTEXT GATHERING FRAMEWORK

After examining previous work on context gathering frameworks, we propose the following as essential requirements for our framework.

- Programming Interface: Uniform interface to provide easy access to sensor data
- Active/Passive Sensor mode: Provide mechanism for sensors to both report data automatically or when queried
- Simplicity: Simplification of the implementation of sensors via API
- Multi-transport Support: XML-based messaging to allow use of different transport technologies
- Separation of Concerns: Abstraction of sensor, sensor data acquisition and sensor data dissemination requires to be 'loosely-coupled' to promote extensibility of the framework
- Sensor Data Model: Extensible XML-based syntax for Sensor Messaging protocol

5.1 Architecture

The context gathering framework has been designed in a way to facilitate the operational requirements of the other components in the platform. For instance, the data modelling requirements from the ontology component would influence how the data structure is designed to encapsulate the sensory data.

The following Figure 3 shows the overall design principles of the developed context gathering framework. The sensor abstraction written by the developers supports the collection of low-level sensing from the physical sensors. They typically run on the machines where the sensors are connected to. The raw data will be reported to the CASP platform via the context gathering framework. The framework and sensor abstraction part is ‘loosely coupled’ to generalize the framework to other implementations.

The framework consists of two components- *Client-side Sensory API* and the *Server-side Sensory API*, which are connected over TCP. To communicate with the platform, the client needs to specify the IP address or hostname of the platform and the port number of the platform that is used for sensory interface communications. This Client-side Sensory API translates the raw data supplied by the sensor abstraction into platform-understandable format and then send the request to the platform, while the Server Sensor API on the platform receives the formatted request and unmarshall them. The unmarshalling output will be used by the Ontology Manager to create ontologies expressed in the form of OWL descriptions. To process many simultaneous incoming connection efficiently, the Server Sensory API supports a non-blocking I/O. The requests and replies between the two components of the framework are encoded in Extensible Markup Language (XML) and each component runs an XML parser. Both components are written in Java.

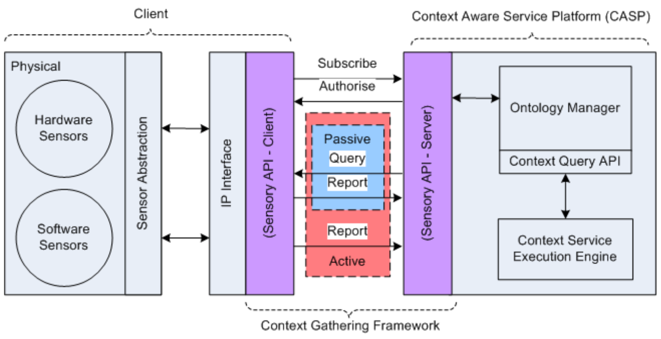


Figure 3. Overview of Context Gathering Framework

5.2 Sensor Data Representation

The following diagram describes the sensor data model. The model comprises sensor profile, entities and properties. According to the model given in Figure 4, each sensor data object has one basic profile which is used to identify the sensor and the mode of operation.

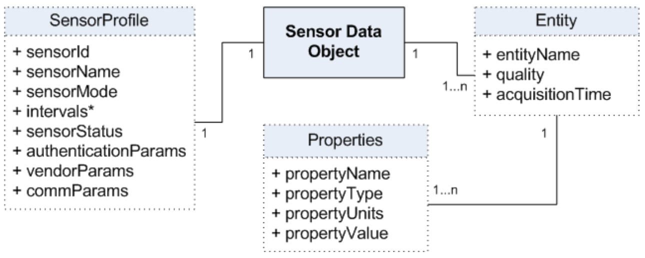


Figure 4. Sensor Data Model

The sensed data itself is represented using entities and properties. Entities are simply context categories for example Subject, Location, Activity, Time, and so on. An entity includes timestamp attribute to define the date and time values the raw data captured. This piece of information can be used to handle context history and sensing conflicts. Further, the quality attribute can be used to rate the reliability of raw context data. Each entity contains a collection of properties, and property describes some characteristics of an entity. For example, entity ‘Location’ can be described by using properties such as LatitudeDirection, LatitudeDegree, LongitudeDegree, Address, IP address and so on. Besides, a sensor data object can be modeled after a group of existing entities, for instance, ‘location of a person’ can be represented using the ‘Location’ and ‘Subject’ entities.

5.3 Sensor Messaging Protocol

The framework supports a uniform messaging protocol based on XML to provide an easy access to sensor data. The expressive power and flexibility of XML make it a good for representation language for heterogeneous context data. Further, an information model that describes interrelationships between different types of context data could provide an easy mechanism for accessing and reasoning about context [14]. Apart from that, the sensory data representation can be easily extended to accommodate new and unanticipated elements without any reprogramming in the

Sensory API module. The following are types of messages supported in the framework developed.

i.	`REGISTER`	: Sensor Subscription
ii.	`OK`	: Positive/successful reply
iii.	`NOT_OK`	: Negative/unsuccessful reply
iv.	`ACQUIRE`	: Raw context data acquisition
v.	`REPORT`	: Raw context data reporting
vi.	`ONLINE`	: Ready/Active state
vii.	`OFFILINE`	: Idle state
viii.	`UPDATE_LOC`	: Network location change notification
ix.	`GET_STATUS`	: Support for querying the sensor state
x.	`DEREGISTER`	: Remove sensor subscription
xi.	`RESET`	: Restore the sensor configurations
xii.	`ABORT`	: Abort the given operation
xiii.	`ACK`	: Generic Acknowledge

5.4 Sensing Context

In our work, we categorised the sensor into two types, one is passive and another one is active. ‘Passive’ sensor means sensor data that need to be queried while ‘Active’ sensor means sensor data that can be queried or configured to report periodically.

A normal sequence of operation of an ‘active’ sensor begins with the sensor subscription with the platform via a REGISTER message. The registration request consists of sensor profile and the list of sensor data properties the sensor can provide. Upon receiving the request, the platform determines the network location and the port number the client component receives communications. Then it stores acquired communication parameters with registration parameters in the sensory database. The sensor properties information will be used as parameters to handle context query in future, whereas the ‘sensorid’ and communication parameters will used to ensure platform delivers subsequent message to the correct client component. An ‘ONLINE’ message is sent to platform when the sensor is ‘active’ or ready to report sensed data. This is followed by the ‘ACQUIRE’ command from the platform to request that sensor queries for the last acquired value or newly sensed data. The data will be reported periodically according to the time interval specified. This operation is interrupted by ‘OFFLINE’ message which indicates the sensor’s idle state. The following sequence diagram (Figure 5) summarizes the normal sequence of operation of an ‘active’ sensor.

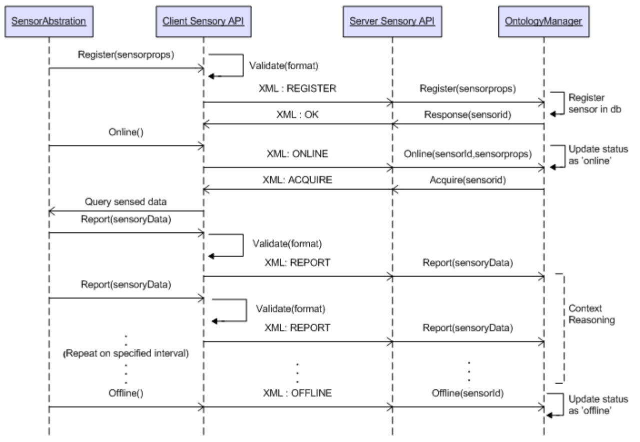
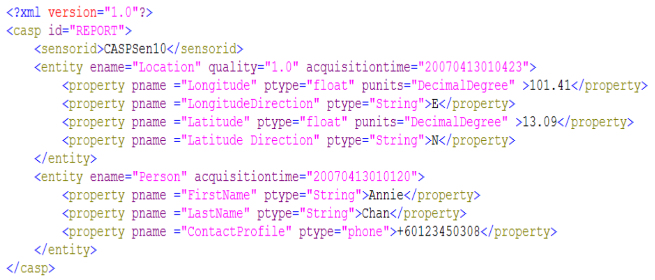


Figure 5. Typical sequence of operation of an ‘active’ sensor

This following Figure 6 is a typical example of raw context data describing the user’s identity and his location, retrieved from a built-in GPS sensor on a mobile phone. The Client Sensor API includes ‘sensorid’ with the sensed parameters to form a ‘REPORT’ xml and then delivered it to the platform. The acquisition time of each context type is expressed as ‘`yyyyMMddhhmmss`’. The quality attribute has a value ranging from ‘0.0’ to ‘1.0’, can be used to rate the reliability of raw data. We suggested that data captured from sensors can be rated as reliable while, the data retrieved from interpreters should be rated as less reliable.

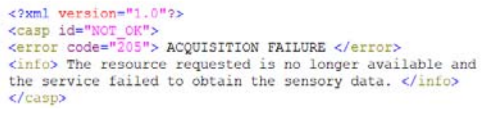


```xml
<?xml version="1.0"?>
<casp id="REPORT">
    <sensorid>CASPSen10</sensorid>
    <entity ename="Location" quality="1.0" acquisitiontime="20070413010423">
        <property pname ="Longitude" ptype="float" punits="DecimalDegree" >101.41</property>
        <property pname ="LongitudeDirection" ptype="String">E</property>
        <property pname ="Latitude" ptype="float" punits="DecimalDegree" >13.09</property>
        <property pname ="Latitude Direction" ptype="String">N</property>
    </entity>
    <entity ename="Person" acquisitiontime="20070413010120">
        <property pname ="FirstName" ptype="String">Annie</property>
        <property pname ="LastName" ptype="String">Chan</property>
        <property pname ="ContactProfile" ptype="phone">+60123450308</property>
    </entity>
</casp>
```

Figure 6. REPORT xml

5.5 Error Handling

The framework also supports messaging protocol to report errors. Further error details can be indicated in <info> element as in Figure 7.



```xml
<?xml version="1.0"?>
<casp id="NOT_OK">
<error code="205"> ACQUISITION FAILURE </error>
<info> The resource requested is no longer available and
the service failed to obtain the sensory data. </info>
</casp>
```

Figure 7. Error message

The error codes supported by the framework are divided in ranges as below. Few examples of possible errors for each range are also listed.

i. 1xx (Message Validation Error)
FORMAT ERROR, DATA MISSING, PROTOCOL ELEMENT NOT SUPPORTED

ii. 2xx Client Error (Bad or illegal request and the request cannot be fulfilled)
UNKNOWN SUBSCRIBER, ILLEGAL OPERATION, ACQUISITION FAILURE

iii. 3xx Server Error (The server failed to fulfill an apparently valid request)
SYSTEM FAILURE, NOT IMPLEMENTED, SERVICE UNAVAILABLE

6. FUTURE WORKS

Currently, the work on context gathering framework delivered in this paper is in its final development stage. Although it has proved useful in its current state, the framework still has room for improvements. The communication mechanism will be extended to support other kind of transport protocol, for example XML over HTTP. An accurate processing of context data requires their quality to be taken into explicit consideration. Since the relevant quality characteristics may be application-specific, context data quality has to be represented in a generic and extensible manner

[10]. The 'quality' element for each context entity will be refined to accommodate other quality attributes such as the accuracy, the precision, the correctness, and the level of trust. We plan to enhance the current sensory data specification by introducing a message extension mechanism allowing the addition of new elements. This mechanism works by specifying an element called '`<extension>`', encapsulating the actual parameters being added into sensory data structure. The server component should ignore any extension that is not recognized and process the message as if the extension is not available.

7. CONCLUSION

In this study, we have deliver a context gathering framework that introduces the sensor data formats, set of interfaces and messaging protocols that allows context gathering from sensor resources. The framework collects raw context data from the physical sensors via sensor abstraction programs written by the sensor providers, translates it to CASP platform-understandable format, and then delivers it to the platform for context modeling and reasoning.

The framework has looked at how to provide a software application programming interface that assist with reporting data from the sensor implementation programs without requiring the application developers to deal with much programming tasks and communication mechanism with the platform. By freeing application writers from the specifics of sensing technologies, the framework encourages the creation new context-aware applications by helping to amortize development efforts.

8. REFERENCES

[1] Abhishek, S. and Conway, M., Survey of Context aware Frameworks – Analysis and Criticism. Online Article, Information Technology Services, University of North Carolina of Chapel Hill, 2006. http://its.unc.edu/teap/tap/core/cafreview.html

[2] Baldauf M., Dustdar S., Rosenberg F., A Survey on Context-Aware Systems. International Journal of Ad Hoc and Ubiquitous Computing, Inderscience Publishers, January 2006.

[3] Chen, H., An Intelligent Broker Architecture for Pervasive Context-Aware Systems, PhD thesis, University of Maryland, Baltimore County, 2004.

[4] Chen, G. and Kotz, D., A Survey of Context-Aware Mobile Computing Research, Dartmouth College, Department of Computer Science TR2000-381, 2000.

[5] Dey, A. K., Providing Architectural Support for Building Context-Aware Applications, PhD. Thesis, Georgia Institute of Technology, 2000.

[6] Dey, A.K. and Abowd, G.D., The Context Toolkit - Aiding the Development of Context-Aware Applications. Proceedings of Human Factors in Computing Systems (CHI 99), 434-441.

[7] Fahy, P. and Clarke, S., CASS – a middleware for mobile context-aware applications. In Workshop on Context Awareness, MobiSys, 2004.

[8] Gellersen, H.W., Schmidt, A., Beigl, M., Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. Mobile Networks and Applications (MONET), 2002, 341 – 351.

[9] Gu, T., Pung, H. K., Zhang, D. Q., A Service-Oriented Middleware for Building Context-Aware Services. Elsevier Journal of Network and Computer Applications (JNCA), Vol. 28, Issue 1, 2005, 1-18.

[10] Haseloff, S., Context Awareness in Information Logistics, Ph.D. Thesis, TU Berlin, Germany, 2005.

[11] Howe, D., The Free On-line Dictionary of Computing, Department of Computing, Imperial College London, http://foldoc.doc.ic.ac.uk/foldoc/

[12] Jason I. Hong, The Context Fabric: An Infrastructure for Context-Aware Computing. Conference on Human Factors in Computing Systems, 2002, 554 – 555.

[13] Kellerer, W., Tarlano, A., Context Aware Wireless Ubiquitous Computing. In Proceeding of 10th Wireless World Research Forum (WWRF), New York, USA, Oct. 27-28, 2003.

[14] Maria, E., Guerney, H., and Hui, L., Joint Special Issues on Context-Aware Computing, IEEE Pervasive Computing and IEEE Wireless Communications, 2002.

[15] Moran, T. P. and Dourish, P., Special Issue on Context Aware Computing, Human Computer Interaction, vol. 16, 2001.

[16] Panayiotou, C., Context Awareness, Proceedings of the Conference on Human Factors in Computing Systems, 2000.

[17] Rakotonirainy, A., Loke, S. W. and Fitzpatrick, G., Context-Awareness for the Mobile Environment, Proceedings of the Conference on Human Factors in Computing Systems, 2000.

[18] Raz, d., Juhola, A.T., Serrat-Fernandez, J., Fast and Efficient Context-Aware Services, John Wiley & Sons Canada, Ltd.; 1 edition (May 17 2006), ISBN-10: 047001668X.

[19] Salber, D., Abowd, G. The Design and Use of a Generic Context Server. Technical Report GIT-GVU-98-32, Georgia Institute of Technology, 1998.

[20] Salber, D., Dey, A.K., and Abowd, G.D. The Context Toolkit: Aiding the development of context-enabled applications. Proceedings of CHI'99, 1999, 434-441.

[21] Seng Loke, Context-Aware Pervasive Systems: Architectures for a New Breed of Applications. AUERBACH, December 7, 2006, ISBN-10: 0849372550.

[22] Schmidt, A., Aidoo, K. A., Takaluomo, A., Tuomela, U., Laerhoven, K. V., and Velde, W. V. d. , Advanced Interaction in Context, Proceedings of 1st International Symposium on Handhelds and Ubiquitous Computing, 89-101, 1999.

[23] Thesaurus.com, Lexico Publishing Group, LLC. http://www.thesaurus.com

[24] WordNet : a lexical database for the English language, Cognitive Science Laboratory, Princeton University http://wordnet.princeton.edu/perl/webwn