A Middleware Framework for Context Acquisition in Ubiquitous Computing Systems.

N.Q. Hung, S.Y.Lee, L.X. Hung
Computer Engineering Dept.
Kyung Hee University, Republic of Korea
{nqhung, sylee, lxhung}@oslab.khu.ac.kr

Abstract

The aim of ubiquitous computing is to combine the worlds of computing and communication in a way to provide seem-less services to the end user by augmenting everyday objects with physical and logical information. Context aware computing, as one important ingredient of ubiquitous computing, helps to realize this dream. Due to plethora of sensors with variable granularity of context, we aim at providing middleware for context aware services named CAMUS[1] (Context-Aware Middleware for Ubiquitous Computing Systems) whose functionalities range from gathering raw sensor information, adapting it to application understandable format and then dispatching this information to interested applications. This paper presents the design and architecture of the CAMUS middleware framework, with focus on context fusion/acquisition from extracted features of diverse sensors. The ultimate goal is to come up with a reusable middleware framework ranging from low-level sensor-extracted features, context fusion from extracted features, and context dissemination to diverse application, resulting in a toolkit-like collection of algorithms for extracting features from sensors, and reasoning mechanisms for deducing context data.

1. Introduction

The general trend in computing is progressing towards the vision of Ubiquitous Computing (or Pervasive Computing), in which devices are seamlessly integrated into the life of everyday users, and services are readily available to users anywhere all the time [1, 2]. Ubiquitous computing is closely associated with visions of smarter devices and environments capable of providing proactive services to their users. Sensing is a key enabling technology to make this possible. As devices and services become more complex and sophisticated, everyday users often find themselves spending more time and efforts in configuring and instructing these devices and services. Context-aware computing is an emerging paradigm to overcome these issues by providing intelligent services that can anticipate the needs of users and act on their behalf, becoming a part of the user environment, disappearing from his awareness. By enabling computer systems to understand their situational context, context-aware computing frees users from being the slaves of their computer systems. Thus, it helps the users to achieve more by doing less [3, 4].

With advancement in computing technologies and in keeping with Moore's Law, sensing and computing devices are getting cheaper, smaller, and more powerful and at the same time the device power consumption is getting efficient. In near future, sensors will be deployed pervasively in the environment, even on our bodies (wearable computers), to carry out sensing and monitoring tasks. Real world objects will be enriched with information processing capabilities and wirelessly networked in an ad hoc, spontaneous manner. In such a ubiquitous computing environment, a middleware infrastructure is needed to act as a mediator between environment and users, to maintain sensing information and context data, as well as to provide reasoning/delivery services for context-aware applications. From the application developer's point of view, the main hurdles in development and deployment of a context-aware system are i) handling diverse and potentially unreliable sensor data, ii) dealing with synthesizers to deduce context from sensor data, and iii) maintaining semantic and communication interoperability between different systems. Delegating these issues to middleware will enable the application developer to focus on application logic and subsequently system deployment will become flexible. In this paper we address these fundamental issues by proposing a context-aware

[1] This work is supported in part by the Ministry of Commerce, Industry & Energy, and the Korea Science and Engineering Foundation

middleware framework with a composite solution covering different issues ranging from useful context representation, dynamic context composition, using perception techniques and sensor network management at sensor layer, and providing useful services for context subscription/publication at the application layer. Focusing on context fusion/acquisition from extracted features of diverse sensors, the paper addresses a reusable middleware framework, resulting in a toolkit-like collection of algorithms for extracting features from sensors, and reasoning mechanisms for deducing context data

We discuss the motivation in Section 2. In Section 3 and 4 we describe the *feature* abstraction, and CAMUS architecture. Section 5 overviews several perception techniques used for extracting features from diverse sensors and deducing context information from those extracted features. Section 6 discusses several advantages of the proposed framework. Related work and future directions are discussed in Section 7 and 8 respectively. Section 9 will conclude the article.

2. Motivation

A desired feature of acquired context is its relation to situation of the entity in focus. Situational context reflects the multifaceted characterization of a situation that typically require substantial analysis and fusion of data from individual sensors of diverse types. For example, to be aware of a user's current situation like sleeping, watching TV, reading etc requires the system to sense and decide location type (outdoors (GPS) or indoors (i-Badge)), light level (dark or bright), time of day, audio level (low, high, silent), specific motion patterns (or absence of them, e.g. lack of arm movement), etc. The described multifaceted characterization of situational context requires diverse sensor types (sometimes deployed redundantly to reduce the inherent ambiguity in sensed data) to be used at a single time for formation of useful context. Moreover, a context management system is also needed to deduce and deliver context information to diverse applications running on heterogeneous devices. Where existing context aware systems have accommodated for context management functionality, they have not addressed sensor management issues adequately [5, 9, 11]. Our proposed framework provides a sensor middleware layer to manage diverse sensor types, extract the useful *features* from sensor data stream to deduce context data. In this way, the higher layers are provided with a uniform abstraction to the heterogeneous sensor environment.

The most demanding feature that motivates further research and development in context-aware computing comes from the fact that existing context-aware computing projects take application-oriented approach, supporting some target application scenarios. As a result, the sensor types used are fixed and dictated by the scenario which makes it hard to deploy new sensor types or adapt the middleware system to new applications needs. Thus existing systems lack reusability and scalability in the middleware framework. CAMUS masks the application developers from heterogeneity of sensors and their access patterns through middleware support for encapsulation.

The proposed framework provides separation of concerns in which context can be modeled separately from sensor technologies and properties of sensors. Moreover, CAMUS middleware architecture will be supported by an application development toolkit which will allow robust development and deployment of context aware applications in a ubiquitous environment.

3. The *feature* abstraction

Every sensor in the system is expected to produce a large amount of values over time. Some of them will produce such a large stream of data, giving such a low-level description, that it is almost impossible to use this directly as input for a recognition system. The values of a light sensor can, for instance, be replaced by the mean and the variance over a sliding history window. Another – and perhaps better - example is the microphone since it produces an even larger amount of values. A range of transformations and filters are traditionally applied by default for this purpose (e.g. time domain analysis, power spectrum, Fast Fourier Transformation (FFT) for base frequency, etc.).

***Features*.** The most basic way to pre-process a data stream from a sensor is to use common elements from statistics, such as minimum, maximum, average or standard deviation. These values are usually referred to as features, descriptors, or cues, as they describe a stream of data by extracting just one value. The features can be interpreted as values that are the results of often simple and lightweight calculations on data that is sent in a dense stream. Many features are derived from the light sensor data in a standard period, such as the average brightness, standard deviation, base frequency (of artificial light from lamps), and so on. From the temperature sensor data, we get

the features: maximal and minimal temperature, average temperature, changing speed, etc.

Features are extracted not only in time domain but also in frequency domain, for example the base frequency. The data from light sensor is transformed into frequency domain through FFT, and then used a linear window to find out the base frequency of in the date. This base frequency should be a stable value when there is artificial light near the light sensor (to discriminate between outdoor/indoor, reading book/watching TV, etc). Correlation and wavelet transform can be used to extract features from sensors as well.

Features have several benefits. By using these features instead of the raw stream of sensor data, bandwidth and amount of data can be reduced. This enables any slower adaptive learning algorithms that work on the features instead of the raw sensor data to be as near real-time as possible.

The data from some sensors, especially from light sensor, involves some random noises that usually occur with no more than two sequential values in one sampling cycle. Before analyzing the data from this kind of sensors, a mid value filter (or median function) with K-value-size window could be used to do the preprocessing.

Utilizing features also optimizes the system's generalization performance since a slightly more abstract interpretation of the data is processed. The higher-level interpretation makes it furthermore easier to inspect any rules that adaptive algorithms may form afterwards.

3. CAMUS middleware framework

The proposed middleware framework CAMUS provides a two-layered abstraction as depicted in figure 1. The first abstraction, called ***Feature Extracting (FE) Layer***, separates the heterogeneous sensor field from sensor data consumers (upper layers). And the second abstraction, ***Context-awareness (CA) Layer***, separates the context acquisition and synthesis from the application layer.

3.1 Feature Extracting (FE) Layer

This layer provides an abstraction from raw sensor data and is the key element for masking the diversity of sensors and providing mechanisms to acquire diverse sensor data. The name Feature Extraction refers to the fact that at this layer sensory data is formulated into features. From the data stream of one sensor, a computing node can extract diverse features by applying low level functions for the consumption of upper layers. For example, an audio sensor might be generating a continuous audio stream. At the FE layer, features such as base frequency, ration between zero crossing and direction change points (peak) and noise level can be extracted to discriminate between voice, music, silence etc.

The FE layer hides the details of sensor interfaces from the context-consuming layers it serves by providing a smaller, uniform interface defined as set of features describing the sensed system environment. In building different applications, the concept of features will be very useful to make changes in hardware (sensors, devices such as PDAs) transparent to the context recognition layer. When including new sensors with different characteristics, only changes in the corresponding feature functions need to be adapted. This way, the FE layer strictly separates the sensor layer and context consuming layers which means context can be modeled in abstraction from sensor technologies and properties of specific sensors. Here, the architecture provides the advantages from separation of concerns.

CAMUS provides a feature tuple space (***FT***) to support interoperability between heterogeneous sensor nodes in the environment (e.g., RFID tags or iBadges for users and objects identifying, audio, video and light sensors for activities discrimination and detection, etc.) The feature tuple space is a multi-dimensional space that includes, among other things, sensor and feature types (Sensor_ID and Feature_ID), and the data value of the extracted feature (which is not sensed data of the environment in some cases, e.g., contents in the memory of an RFID tag). Additional dimension for timestamp may be included for consistency in the distributed tuple space:

$$FT = \{Sensor\_type\_ID, feature\_ID, feature\_value, timestamp\}$$

Consumers can access the feature tuple space in a sensor network via uniform FE APIs. Back-end system accesses the sensor network for features needed in deducing different levels of context (location context, activities, etc). Handheld devices may access sensor network for features to deduce some primitive context directly (as identity context, temperature, etc) using query/notification mechanism or for transfer of information to back-end system (as a transient access point). During system development or maintenance periods,

handheld devices can also be used to access the feature tuple space for debugging.

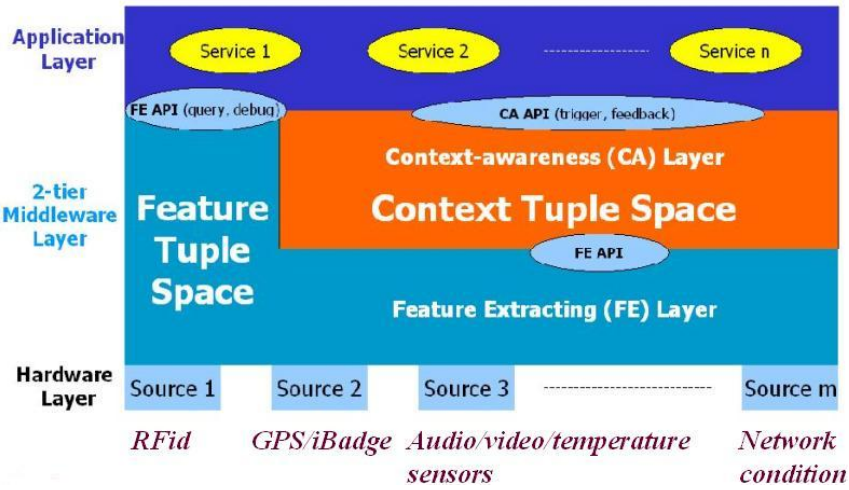

Figure 1: CAMUS Architecture

3.2 Context-Awareness (CA) Layer

This layer fuses the diverse environmental features received from different sensors to provide context synthesis. Context data is derived as a function of diverse available features. The methods for generating context from primitive features can be rule-based algorithms, statistical methods, neural networks, or reasoning machine.

While features are assumed to be generic, context is considered to be more closely related to the application scenarios. The feature extracting layer needs to produce only those features that are required by the context consumers, ignoring the remaining – unnecessary – features. This reflects the philosophy "application-knowledge-in-node" of CAMUS, which helps to reduce the computational and communication cost.

At this layer, CAMUS supports the application developers in the process of context acquisition and synthesis by providing ready-to-use contexts in the Context Tuple Space (***CT***).

CT = {Context_ID, Attribute_ID, context-data, time, probability}

The context information (CT) is identified by an ID and has several associated context attributes. Each attribute also has an ID and data value stored in context-data, a data type defined by the system developer. The structure given above allows the provision of including probabilities for the context value if the perception system offers this information. Place and time information is not context in its own; instead it adds meaning to other contextual information. Hence it is treated as meta-information about a context. We discuss how to represent/modeling context data using Web Ontology Language (OWL) [6, 7, 8] in another paper [12].

3.3 Hierarchical Tuple Space

CAMUS follows a bottom-up approach, starting with the actual data from the hardware sensors, and generalizing towards the user's description of the observed data. This method is the different with the traditional design of a sensor-based system where a specific application dictates which concepts are useful, and what sensors are required. The bottom-up approach leads to an abstract hierarchical tuple space in CAMUS framework as depicted in figure 2, and its advantages are discussed in section 6. Here the feature extracting functions and algorithms, along with context reasoning mechanisms (stated in section 5 – Perception methods) are represented in two libraries, FELib and CALib respectively.

4. CAMUS Core middleware services

The heterogeneity and dynamism inherent in smart/active spaces leads to high levels of heterogeneity at the link and transport layers. Services can be developed using existing technologies and deployed at the infrastructure level to mask such heterogeneity and provide support for handheld devices and applications executing in resource constraint environments. These services provide communication, amongst the framework components at different levels, and assistance to applications. In the CAMUS architecture, infrastructure services exist at three

levels: Core Feature Services, Context-Feature Mediator Services and Core Context Services as depicted in figure 3. We discuss details of communication mechanisms and services in another paper [12].

Core Feature Services deal with extraction of environment and context features from sensor signals. These services also register event subscriptions, trigger event notifications, handle queries and generate responses related to the basic feature information at the sensors.

query/response and event registration/ notification tasks for the applications.

Though the aim of the system is to provide context sensitive information to interested devices and applications, it is nevertheless desirable in some cases that raw sensory data be made available to applications/devices to infer on their own. This provision is also addressed in CAMUS architecture where sensory data from Feature Tuple Space will be made available to interested entities.

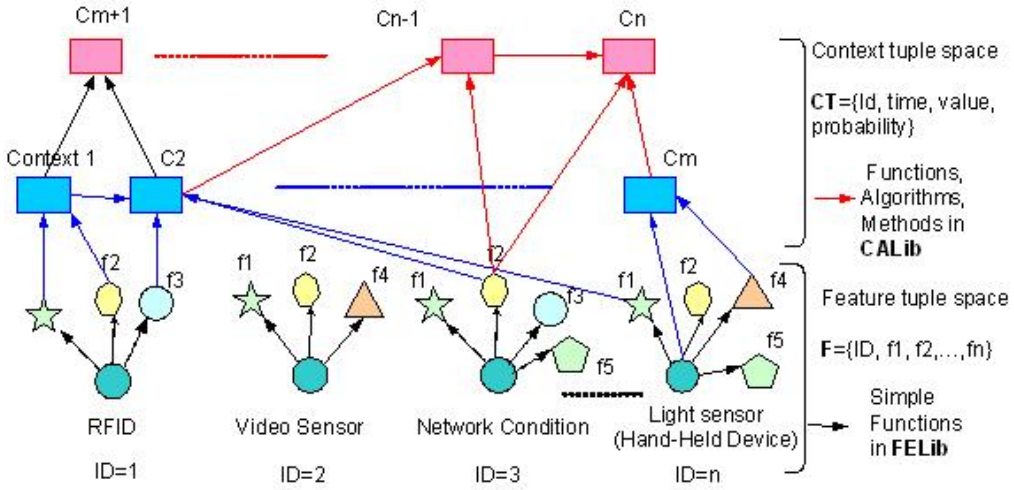

Figure 2. Hierarchical Tuple Space in CAMUS

Context-Feature Mediator Services aids higher-level services by mapping their requirements into queries to be forwarded to the feature tuple space. Moreover, the mediator also performs the reverse role of transforming features into context data to be passed from feature tuple space to the higher context services. In this way, this mediator performs the critical function of summing up features into context that is analogous to converting raw data into (useful) information. These services have input data (features) from the lower level Context Feature services. Foremost task of these services is to assemble feature data into utilizable context data. It is achieved through Data Aggregation and Fusion sub services. A reasoning engine is available to these services to aid them in their tasks. Secondly, these services present the acquired context to upper level services and applications. The higher level services can query these services and also register for contextual events of their interests with them.

The Core Context Services lie at the top of the architecture and consist of *asynchronous* Context Event service and *synchronous* Context Query service. These services handle the overall

5. Perception Methods

Context refers to information that describes some aspect of the conditions in which an application executes. There is no clear distinction about what is and is not a context, but the most interesting kinds of context are those that humans do not explicitly provide. With the advancement in sensing and automated means of perceiving physical environment, we can automatically collect much more implicit context then ever.

Lightweight algorithms are being investigated and embedded in sensor nodes to extract diverse useful features for context fusion [13], [14]. They can be basic statistical functions such as average, median, standard deviation, minimum and maximum, or first and higher order derivatives, which can be calculated consuming very low cost and at times on the fly without needing to save all samples. Time domain analysis may also be adopted, which is particularly useful for data from light and audio sensors. For audio the average itself has no meaning but it is useful to calculate further features. Knowing the average means that calculations on how often the average is crossed in

a certain time and also the average distance between crossing the average can be performed. It is also possible to calculate the distribution of the distances between crossing the average. This is an indicator for the base frequency and the stability of the base frequency in the signal. Counting the direction changes in the signal is also possible on the fly. The ratio between the average crossings and the direction changes gives an indication on the type of signal and allows discrimination between contexts. For example in the audio signal it is possible to discriminate music, speech, and noise, and in the acceleration signal it is possible to find characteristic values for certain patterns of movement. For fast changing signals like audio signals, the peaks or energy (root mean square) of the signal in small time windows (e.g. getting an indication every 100ms) provides information about the sampled data. Certain audio events (speaking of a word, ringing of the phone, applause, music) result in a characteristic series of values.

the neural network is noise-tolerant and can process the input features with plenty of noise. The other advantage is that neural network allows the system to be reconfigured according to the specified application instance. Many neural networks are computationally demanding. But there are still some methods that can be implemented on very restricted hardware platforms (handheld devices, and sensors) like back-***propagation neural networks, logical neural networks*** [15], etc. ***Nearest neighbor matching*** is a very simple technique for pattern matching. A representative vector is calculated and stored during the learning phase. When the system is in operational mode, an incoming vector is compared to the stored sample vectors and the distance is calculated.

System can use pre-defined rules written in some form of logic to infer, deduce different contexts. For example, based on the number of people in the room and the applications running in the room the system can deduce what kind of

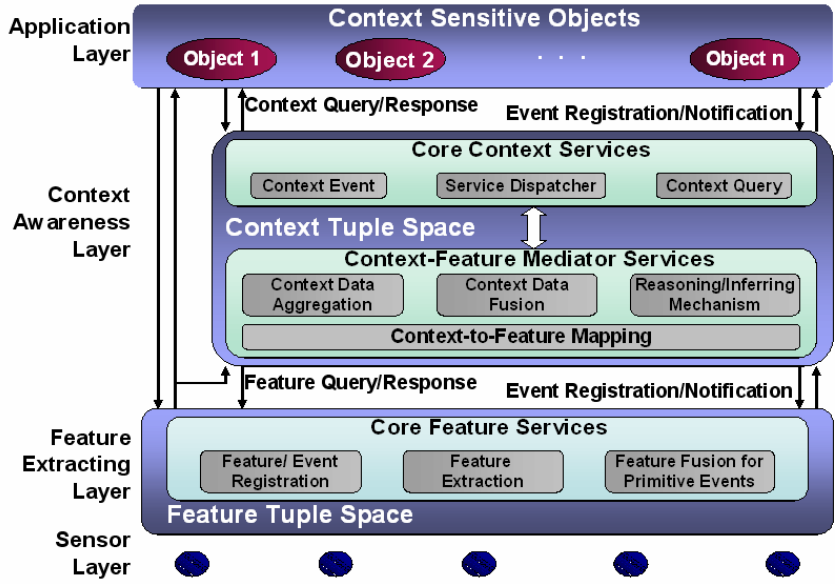

Figure 3: CAMUS Core services

To reduce inherent ambiguity in sensor data and to infer high-level context from the extracted features, high-level perception methods like pattern-matching engines and neural network will also be investigated and implemented in back-end systems. Most of the awareness of the contexts is based on more than one feature and even other contexts. The features and contexts are regarded as different dimensions of input vector of the fusion algorithm. There are two advantages of applying neural networks to fuse the decision. One is that

activity is going on in the room. It uses rules written in first order logic to perform the deduction. Some of the rules may be:

1. #People(Room 302, ">=" , 3) AND Application(PowerPoint, Running)
   => RoomActivity(302, Presentation)
2. #People(Room 302, ">=" , 1) AND Application(MPEG Player, Running)
   =>RoomActivity(302, Movie Screening)

6. Discussion

This proposed framework for context acquisition is mostly software-based design instead of a customized, hybrid, hardware-software design. The assumption that there are plenty of suitable sensors moves the focus to the algorithm that processes the sensor data. The system lets all sensors act together to generate descriptions for all applications, instead of having to design a tailored sensor system for each application.

There are several advantages of the proposed framework. First, the multi-step of abstraction provides separation of concerns in collecting raw data, extracting features from diverse and heterogeneous sensors, fusing/deducing those features into context information, and delivering the information/events to different applications running on diverse devices. This helps to model context data separately from sensor technologies and also filter/reduce traffic of data from sensor level to application level dramatically. Second, due to modularity and service-oriented design we benefit from system software reusability and evolution, sensor and context data abstraction, and maintainability. Finally also due to the modular design the middleware framework can be deployed in a distributed manner and achieve the benefits of parallelism and scalability.

7. Related work

A lot of work has already been done in the field of context-aware systems for ubiquitous computing and a lot more is still to appear. The history dates back to 1988 when M. Weiser at el. started their work on next-generation systems at Xerox Parc. They utilized the concept of agents to represent user static and device dynamic properties allowing the system to behave accordingly. Context toolkit [14] used the concept of GUI widgets to represent abstract sensors which hid sensor details from the applications but required sensors' manipulation at design time. Solar [15] took it one step further by allowing customized functionalities to be inserted into the system using operator-graph abstraction. While Context Fabric's [16] automatic path creation allows path to be created from source to sink by selecting components internal to the system at runtime and employs infrastructure approach for context aware systems.

The context systems built until now have mostly been prototypes, investigating different approaches that can be adopted to enable context awareness in a ubiquitous environment. Though most systems presented viable new approaches and designs, they lacked in interoperability, flexibility and extensibility. CAMUS aims at providing these missing features and a middleware infrastructure for application development and deployment.

8. Future Directions

In Ubiquitous Computing environment, there exist the concerns for securing context information from unauthorized uses and respecting individuals' privacy. We are assessing these security and privacy concerns in CAMUS raised by the collection of the data, the information produced, and the dissemination of that information to other people in other times and places.

A successful Ubicomp project is proved by some useful applications. For this, novel and application scenarios will be explored in the vision of Ubiquitous computing. Building these applications will help evaluate and validate our CAMUS Middleware framework. A test bed is being setup for deployment and step-by-step evaluation of our proposed architecture. This will enable us to verify the architecture and provide effective solutions for real world applications.

The culmination of architecture will lead to development of a toolkit. This toolkit will aid the application developers in fast development and deployment of applications.

9. Summary

In this paper we describe our middleware framework for context-aware ubiquitous computing systems with the focus on context acquisition from sensory data. CAMUS provides two levels of abstractions to the applications in the form of features and context. Features are extracted from the sensor layer and stored in a feature tuple space. These features are converted into context data at the context-awareness layer and stored in a context tuple space. This proposed two-tier middleware framework provides separation of concerns in which context can be modeled separately from sensor technologies and properties of sensors. The main objective is to come up with a reusable middleware framework ranging from low-level sensor-extracted features, context fusion from extracted features, and context dissemination to diverse application, resulting in a

toolkit-like collection of algorithms for extracting features from sensors, and reasoning mechanisms for deducing context data. Thus application developers can benefit with a flexible and scalable context-aware middleware framework. A large selection of analysis algorithms can be applied to the sensor data, and different solutions may be provided in various situations rather than one specific algorithm that will be considered as the optimal solution in any circumstance.

10. References

[1] M. Weiser, "The Computer for the 21st Century," Scientific Am., Sept., 1991, pp. 94-104; reprinted in IEEE Pervasive Computing, Jan.-Mar. 2002, pp. 19-25.

[2] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," IEEE Personal Communication, Aug. 2001, pp. 10-17.

[3] MIT Project Oxygen, http://oxygen.lcs.mit.edu/Overview.html

[4] Sousa, J.P., Garlan, D., "Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments", Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, Montreal, Aug. 25-31, 2002.

[5] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S.Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," IEEE Pervasive Computing, joint special issue with IEEE Personal Communications, 1(3), Jul.-Sep. 2002, pp.33-40.

[6] Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein, "Web ontology language (owl) reference version 1.0.", W3C Working Draft, Nov. 2002.

[7] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider and Lynn Andrea Stein, "DAML+OIL (March 2001) Reference Description", 18th Dec. 2001. http://www.w3.org/TR/daml+oil-reference

[8] Jeff Heflin (Lehigh University), "OWL Web Ontology Language Use Cases and Requirements", http://www.w3.org/TR/webont-req/

[9] Dey, A.K., et al, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", anchor article of a special issue on Context-Aware Computing, *Human-Computer Interaction (HCI) Journal*, Vol. 16, 2001.

[10] Guanling Chen and David Kotz, "Context aggregation and dissemination in ubiquitous computing systems", In Proceedings of the Fourth 1EEE Workshop on Mobile Computing Systems and Applications, IEEE Computer Society Press, Jun. 2002.

[11] Context Fabric project, http://guir.berkeley.edu/projects/cfabric.

[12] N.Q. Hung, S.Y. Lee *et al*. "CAMUS – A Context-Aware Middleware Framework for Ubiquitous Computing Systems", submitted to 23rd IEEE International Performance Computing and Communications Conference (IPCCC2004)- April 14-17, 2004 - Phoenix, Arizona

[13] B. V. Dasarathy, "Information/Decision fusion – principles and paradigms", in Proceeding of the workshop on Foundations of Information/Decision Fusion, pp. 46-60, 1996.

[14] B. V. Dasarathy, "Sensor fusion potential exploitation – innovative architectures and illustrative applications", in Proceedings of the IEEE, pp. 24-38, January 1997.

[15] Aleksander. I. And H. Morton. "An introduction to neural computing" (2 ed.). London, U.K.:Chapman and Hall. 1995.