International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:3, No:4, 2009

Context Aware Lightweight Energy Efficient Framework

D. Sathan, A. Meetoo, and R. K. Subramaniam

***Abstract*—**Context awareness is a capability whereby mobile computing devices can sense their physical environment and adapt their behavior accordingly. The term context-awareness, in ubiquitous computing, was introduced by Schilit in 1994 and has become one of the most exciting concepts in early 21st-century computing, fueled by recent developments in pervasive computing (i.e. mobile and ubiquitous computing). These include computing devices worn by users, embedded devices, smart appliances, sensors surrounding users and a variety of wireless networking technologies. Context-aware applications use context information to adapt interfaces, tailor the set of application-relevant data, increase the precision of information retrieval, discover services, make the user interaction implicit, or build smart environments. For example: A context aware mobile phone will know that the user is currently in a meeting room, and reject any unimportant calls**.** One of the major challenges in providing users with context-aware services lies in continuously monitoring their contexts based on numerous sensors connected to the context aware system through wireless communication. A number of context aware frameworks based on sensors have been proposed, but many of them have neglected the fact that monitoring with sensors imposes heavy workloads on ubiquitous devices with limited computing power and battery. In this paper, we present CALEEF, a lightweight and energy efficient context aware framework for resource limited ubiquitous devices**.**

***Keywords*—**Context-Aware, Energy-Efficient, Lightweight, Ubiquitous Devices.

I. Introduction

THE last few years there has been major development in technologies with introduction of wireless communication medium, smaller and cheaper devices which has enabled people to get access to services anywhere, anytime. As a result ubiquitous computing is proliferating and populating public places (offices, shops, laboratories, museums, classrooms, hospitals), as well as private environment (homes, gardens). Ideally, a ubiquitous computing infrastructure should offer personalized support to a user's activities at any location. Such infrastructure would enable users to take full advantage of all the computing capabilities at each location. Thus the term context-awareness that enables ubiquitous system to make use of context information to provide enhanced and proactive services to users [6]. Especially in combination with ubiquitous devices, such mechanisms are particularly of high value and claim to increase usability tremendously. Context-aware systems can adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking the environmental context into account. Many researchers have made claims about the potential benefits of context-sensitive mobile computing since its first appearance more than a decade ago [22],[1],[19]. Such applications however remain difficult to develop and deploy due to a lack of architectural support and abstraction. In order to be able to provide context-aware applications, programmers are often required to write large amounts of code from scratch based on their own architecture and interact with sensor and actuator devices at a low level.

In the light of all the problems faced by context aware computing, a standard architecture is needed to support future applications that will enable seamless service provisioning in heterogeneous, dynamically varying computing and communication environments [5]. In this paper, a lightweight energy efficient context-aware architecture is proposed that will ease the development of context-aware systems.

The key to the proposed framework is threefold. First, there is a context history in CALEEF that continuously monitor and record the changes in context.

Second, a server-based approach is used to reduce the processing on the ubiquitous device which has limited resources.

Third, the sensor control module perform statistical inference on the data in the context history to determine the polling frequency of sensors thus reducing wireless communication.

II. Context-Awareness Techniques

It is observed that the sheer diversity of exploitable context and the plethora of sensing technologies are actually working against the deployment of context-aware systems [4]. One solution to the problem is to separate the application and the actual context sensing part. A middleware approach has been adopted by many researchers to achieve the separation of concerns between sensor data processing and applications, whose functionalities are to collect raw sensor information, translate it to an application-understandable format and disseminate it to interested applications [10].

D. Sathan is with the CSE dept, University of Mauritius, Mauritius (phone: 230-454-1041; fax: 230-465-7144; e-mail: d.sathan@ uom.ac.mu).

A. .Meetoo is with the CSE dept, University of Mauritius, Mauritius (phone: 230-454-1041; fax: 230-465-7144; e-mail: anuja.meetoo@ uom.ac.mu).

R. K. Subramaniamo is with the CSE dept, University of Mauritius, Mauritius (phone: 230-454-1041; fax:230-465-7144; e-mail: rks@ uom.ac.mu).

In the earlier stage of research in this field, many researchers focused on building application-specific context-aware systems such as the Active Badge [22] project which provided the phone redirection service based on the location of a person in an office. In these systems it was difficult to obtain and process context information due to the "ad-hoc" approach they employ. Furthermore, they were dependent on the underlying hardware and operating system. Some researchers take a framework-based approach [3] to provide basic structures and reusable components for common functionalities, and hence to enable easy creation of context-aware applications. The ParcTab [19] system was the earliest attempt made for a general context-aware framework. In the same line of thought is the Context Toolkit [6]; an object-oriented approach framework that provides a number of reusable components to support rapid prototyping of sensor-based context-aware applications. However, these systems do not provide a common context model to enable context knowledge sharing and context reasoning.

Recent research works focus on providing infrastructure support for context-aware systems. The advantage of the infrastructure-based systems has been pointed out by Hong and Landay[11]. In the Context Fabric infrastructure [11]; a database-oriented approach is used to provide context abstraction by defining a Context Specification Language and a set of core services. In the Context Broker Architecture (CoBrA) [4] project, Chen proposed an agent-oriented infrastructure for context representation, context sharing and user's privacy control.

The development of future context-aware applications will require tools that are based on clearly defined models of context [7] and system software architecture. So, an infrastructure that supports the gathering of context information from different sensors and the delivery of appropriate context information to applications is needed. In order to greatly simplify the tasks of creating and maintaining context-aware systems, as much of the weight of context-aware computing as possible must be shifted onto network-accessible middleware infrastructures [15]. By providing uniform abstractions and reliable services for common operations, service infrastructures can make it easier to develop robust applications even on a diverse and constantly changing set of devices and sensors. A service infrastructure would also make it easier to incrementally deploy new sensors, new devices and new services as they appear in the future, as well as scale these up to serve large numbers of people. Lastly, a service infrastructure would make it easier for sensors and devices to share sensor and context data, placing the burden of acquisition, processing and interoperability on the infrastructure rather than on individual devices and applications [11].

III. Literature

During the past few years, the research focus in context-aware computing has shifted to scalable and reconfigurable architectures. It can be used to automatically recognize user requirements from the application situation and accordingly adapt functionality and interaction patterns of the system [2]. One difficulty faced by context-aware application designers is the lack of generic infrastructure for developing context-aware applications. In addition, existing applications have focused mainly on location information. Several researchers have proposed and prototyped general architectures to support context-aware application; Schilit's infrastructure for ubiquitous computing [19] is probably the earliest attempt in this direction. It is agent-based and supports the gathering of context about devices and users. Applications build on the architecture can request for these context information from the different agents such as *device agents, user agents* and *active maps* to provide proactive services to users.

Dey's Context Toolkit [6] aims at facilitating the development of context-aware applications by offering a small set of generic 'base' classes from which an application developer can derive application specific classes. These classes are organized around high level context-aware application functions, namely collecting sensor data, combining data from multiple sensors and translating sensor data into alternate formats. Context Toolkit delivers a standardized way of implementing the syntactic part of context-aware systems. However, the reasoning about contextual information must be implemented for each domain and application. This makes it flexible only in the design and implementation phase, but and not during run-time.

MiddleWhere [17] is a middleware for location-awareness that separates applications from location detection technologies. It integrates multiple location technologies to present applications with a consolidated view of mobile object location and allows the addition of such technologies on the fly as they become available. A probabilistic reasoning technique is used to resolve conflicts and infer location given different sensor data. MiddleWhere also enables applications to determine spatial relationships between mobile objects and their environment.

The Service-oriented Context-Aware Middleware (SOCAM) project [10] is an architecture for the building and rapid prototyping of context-aware mobile services. It supports context acquisition, interpretation, discovery and dissemination and its main feature is its support for context reasoning through which high-level implicit contexts can be derived from low-level explicit contexts. Contexts are represented as predicates written in OWL (Web Ontology Language) which makes it flexible. Interpreter acquires context data from distributed context providers and offers it in mostly processed form to clients.

Gaia [16] aims at making physical spaces intelligent by providing services to manage the spaces and their associated states. *Context providers* are data sources of context information. Other agents can query them or listen on their event channel where they keep sending context events. They can also advertise the context they provide with the *context provider lookup service* that allows other agents to discover them. *Context history service* stores past contexts and may be queried.

CAPNET [18] is a context-aware middleware for mobile multimedia applications that fulfils the requirements of context-awareness, mobility of software components,

multimedia applications and adaptation. The CAPNET middleware is located below the application layer and above that of existing technologies. Each of them offers a particular service to the other components and applications. The *connectivity management component* controls the bandwidth and overall traffic while the *messaging component, service discovery* and *component management* provide transparency to mobile applications.

TABLE I
SUMMARY OF EXISTING ARCHITECTURES AND MIDDLEWARE

Existing Framework	*Context Specification*	*Separation of Concern*	*Context Interpretation*	*Constant Context Acquisition*	*Distributed Communications*	*Context Storage*
Context Toolkit	✓	P	✓	✓	✓	✓
Schilit's Architecture	P	P	X	✓	P	X
SOCAM	✓	N/A	✓	N/A	✓	✓
MiddleWhere	✓	✓	✓	N/A	N/A	✓
Gaia	✓	N/A	✓	N/A	N/A	✓
CAPNET	N/A	N/A	N/A	N/A	✓	✓

X = No Support P = Partial Support ✓ = Complete Support N/A = Not Available

IV. CHALLENGES

There are four main objectives of context-aware computing, namely to increase the number of input channels into computers, to encompass more implicit acquisition of data, to create better models that can take advantage of the increased input, and to use the increased input and improved models in new and useful ways [11]. This confirm the idea pointed out by Dey,2000 that to support context aware applications an architecture should have a mechanism to represent contextual information and abstraction models that are capable of handling it [5]. In summary, what is needed is a standard architecture supporting general, adaptive, and decentralized services that both scale to wide-level deployment and simplify context-aware application design.

Though recent advances in technology, namely in sensors, hardware, networking and software, have made it feasible to implement such architectures, it is extremely difficult to build effective and reliable context-aware applications. There are five primary reasons that account for this complexity, namely because the same context data can be acquired from several sources, context data is highly distributed, possibly coming from and being used anywhere, anytime, data models have normally been application-specific and inflexible, thus making sharing of context data complex, data models have not addressed security and privacy concerns and it is highly complex to program applications in an environment that is constantly changing in terms of sensors, services and context data [11]. So, considerable progress is still needed in these areas before a context infrastructure can be deployed across a wide area.

At the same time, future architectures for context-awareness should be able to acquire context from heterogeneous and distributed sources; so as to reduce the cost and difficulties of building context-aware applications. The architecture should also maintain consistent contextual knowledge, to prevent applications from making inaccurate decisions due to inconsistent knowledge. The architecture should detect and resolve any contextual knowledge that may be inconsistent or ambiguous, thus the term Quality of Context (QoC). Moreover, the architectures should enable knowledge sharing among applications entities. In a dynamic environment, it is more cost-effective for entities to share their knowledge. Context-awareness and mobile scenarios require an architecture that supports interoperability among components that have not been specifically designed to work together. There are many competing approaches to this problem, and a good deal of research and experimentation will be required before broadly usable architectures and standards appear

V. ARCHITECTURE

This section gives an overview of CALEEF and describes how it provides mobile context-aware application designers with support for high-level context derivation. CALEEF addresses the issue of separation of context-aware application code from high-level context reasoning and behaviors. More precisely, this allows an application's context reasoning and resulting behaviors to be changed without re-compilation. CALEEF applications need not communicate with each individual source of context directly but only with the context service layers and therefore they do not need to store low-level details of context sources. An efficient model for handling, sharing and storing context data is essential for a working context-aware system. CALEEF maintains a model of the current context that can be shared by all devices, services and agents in the same smart space. The shared model is a repository of knowledge that describes the context associated with an environment. As this repository is always accessible within an associated space, resource limited devices will be able to offload the burden of maintaining context knowledge. When this model is coupled with a reasoning facility, it can provide additional services, such as detecting and resolving inconsistent knowledge and reasoning with knowledge acquired from the space. Fig. 1 depicts the overall architecture of CALEEF and a brief description of its components follows.

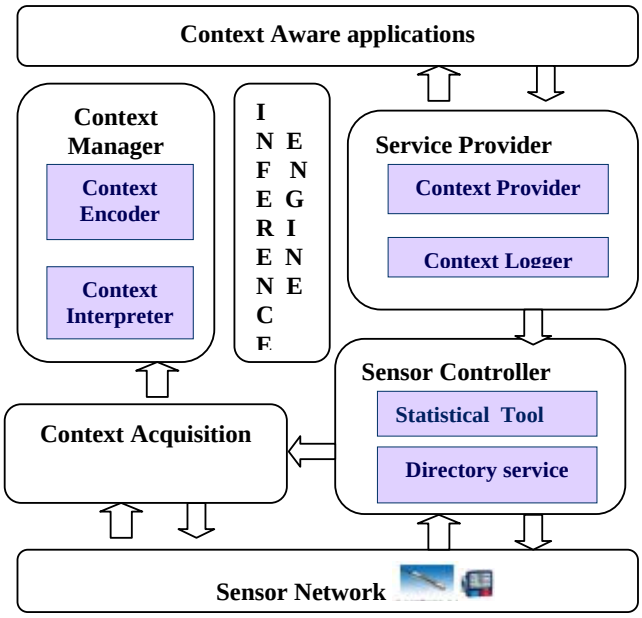

Fig. 1 CALEEF Architecture

A. Context Acquisition

Specific widgets are developed at the data acquisition layer to capture different kinds of contextual information. The context widget acts as a mediator between an application and its operating environment, separating context acquisition from context use. It relieves applications from context sensing issues by wrapping the sensor with a uniform interface. This makes applications design independent of the method of context sensing employed. The context widget encapsulates details of context acquisition thus leading to the flexibility of exchanging a context widget with another widget providing the same type of context information without affecting applications using it. Irrespective of the type of context a widget provides, it stores its latest sensed context value and allows applications to query. A context widget is independent of all executing applications and persistent. It continuously sends updated context information to the context encoder, which then send it to the service provider for storage and distribution to context consumers. It can also be shared among applications and is reusable.

B. Context Manager

The context manager is responsible of converting acquired data from sensors into context information that will provided to applications. It consists of two subcomponents:

Context interpreter: It performs context processing using logic reasoning. The tasks for context processing may include deriving high-level contexts from low-level contexts, querying context knowledge, maintaining the consistency of context knowledge and resolving context conflicts.

*Context Encoder***:** The context encoder encodes context information; acquired from sensors using OWL and pass the information to the context logger.

C. Inference Engine

The inference engine is used to perform context reasoning over stored facts and processes past and current context information to determine how an application should adapt its current behavior. It uses a technique called forward chaining, where known facts are used to infer other true facts. Forward chaining is a search technique useful for situations where the search space is wide with many potential goals, which is the case with context-aware systems. The inference engine works in conjunction with a knowledge base that stores facts and rules, and a context history that stores past and current context as facts.

D. Context Logger

Context-aware applications may be dependent on past context, in addition to current context information, to adapt their behavior. Thus, context information acquired from sensors is encoded and stored in a context history that may be queried by applications. The context logger is made up of a *context history* and *Knowledge Base*. The context knowledge base provides a set of API's for other service components to query, add, delete or modify context knowledge. Furthermore, it contains context ontology rules. The context ontology and its instances of defined contexts are pre-loaded into the context knowledge base during system initiation while the instances of sensed contexts are loaded at runtime. To ensure freshness of context information, an event triggering mechanism is deployed to allow updating of a particular context ontology or instance. Different information requires different update frequency. Having a separate knowledge base means that changes can be made to context inferences and goals relevant to an application without requiring changes in the application code.

E. Context Provider

The context provider keeps record of context consumers and issues a callback to them when updated context is available. Context users subscribe to the context provider for specific context types and receive notifications when they are available. Thus, context consumers listen for events that are sent by the context provider. Moreover, they can also query the context provider for context information. The inference engine will choose the best context information that will satisfy the consumer's query.

F. Directory Service

The directory service is used to register the information of the sensors in the surrounding environment of the user that will provide data to the context acquisition. The sensors will advertise their QoC attributes such as spatial information, refresh rate, correctness that will recorded by the directory service. With this mechanism, CALEEF can choose the most appropriate sensor that can be used to get context data thus improving the reliability and quality of context information. In case a sensor is down, an alternative sensor that provides the same context data can be used.

G. Context Consumers

The context consumers consume various types of contexts and adapt their behavior according to the current context. They obtain contexts either by querying the Context Provider or by listening for events that are sent by the Context Provider. Hence, applications can easily acquire the contexts they need to take decisions. The architecture also provides mechanisms for developers to specify context-sensitive adaptation behavior using rules. As such, CALEEF makes it very easy to develop and deploy context aware applications.

VI. Energy Efficient Sensor Control

Sensor devices are placed in ubiquitous computing environment, to capture user's context, and usually have limited communication capabilities due to energy and bandwidth constraints. Data can be obtained from the sensors by repeatedly contacting and querying the sensor. The polling frequency allows you to configure for each sensor how often this is done. When optimizing the polling frequency for sensor, it will reduce wireless communication between system and consumers of data; and will be more energy efficient.

For this reason, CALEEF employs a prediction algorithm that will statistically infer the polling frequency of sensors. The sensor control mechanism will use the context data in the context history to recognise context pattern change over time. Because even if the context data are changing it does not mean that the user has changed location as for example a user within a conference room is moving within the conference room itself. For this reason, the environment was grouped into zone; For example conference room will be allocated a range of context values using a supervised training algorithm. The algorithm will decide on the appropriate value that can be used to poll the sensors.

The sensor control will also use the context information from the context history to select the most appropriate sensors that have advertised their QoC attributes with the directory service. So that sensors within the spatial environment of the user will be taken into consideration. In this way any changes to the sensor status will be taken into account when polling sensors for data by the context widget and will guarantee reliability of data.

VII. Implementation

All the components of CALEEF are autonomous in execution. They are instantiated and execute independently of each other as shown in Fig. 2. These components as well as the context-aware applications are usually distributed on several computers for better performance and efficiency. CALEEF allows a peer-to-peer communication among them using HyperText Transfer Protocol (HTTP) and eXtensible Markup Language (XML). Messages are encoded in XML and wrapped with HTTP. This technique enables lightweight integration of distributed components; and to build interoperable context-aware services. Other alternatives considered include CORBA, RMI and SOAP. CORBA and RMI which are too heavyweight and require additional components. In addition, RMI would dictate the use of Java. SOAP, while being lightweight would require the use of a web server that supports the protocol. However, the default communication protocol can be modified to account for other protocols, e.g. Simple Mail Transfer Protocol (SMTP) simply by creating an object that speaks the SMTP protocol for outgoing communications and one for incoming communications. Thus CALEEF supports transparent distributed communications.

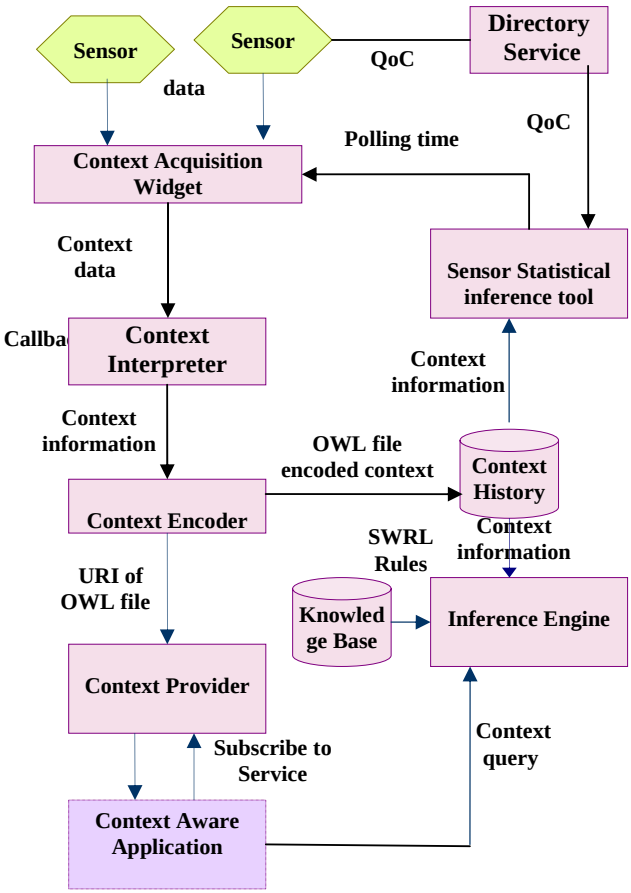

Fig. 2 Module interaction

Moreover, resource discovery mechanisms are rarely used in existing frameworks. However, such dynamic mechanisms are important, particularly in pervasive environment, where available sensors and the context sources change rapidly. CALEEF exploits Jini service discovery protocol (Marin-Perianu *et al.*, 2005) to allow service providers to advertise their capabilities and information one must know to access their services and consumers to locate available services. All the components of CALEEF are implemented in Java and are multithreaded. Thus, CALEEF is platform-independent and its components are capable of handling multiple incoming messages. CALEEF provides two classes, namely CALEEFclient and CALEEFserver to send and receive messages respectively. Application programmers need not write these classes, but merely create instances of them to use them. For instance, applications as well as CALEEF components create an instance of the CALEEFclient class to send requests and an instance of the CALEEFserver class to

receive replies. An instance of the CALEEFserver class acts as a super tiny web server that intercepts messages intended for the application or component that instantiated it. CALEEF provides a fundamental abstract class 'Widget' that caters for features common to all context widgets, namely (1) receiving and servicing requests from context consumers, (2) notifying the context manager when a context change is detected, and (3) sending a list of all context attributes a particular widget can provide. Classes of all types of widgets subclass from this class and thus inherit all its attributes and behaviours. The inference engine used by CALEEF is Bossam version 0.9b5 [14], developed by M. Jang, as it seems to best fit the architecture. It supports ontologies, can be embedded in Java programs and makes use of URI to refer to files storing facts and rules, thus making it apt for use with the distributed components of CALEEF.

VIII. Discussion

Building context-aware applications can be difficult and costly without the support of a proper infrastructure. CALEEF reduces the cost and difficulties of building context-aware applications by enabling distributed software components to access a shared model of contexts. CALEEF can also acquire context from a wide range of sources rather than only sensors that are embedded in the local environment. In order to prevent applications from making inaccurate decisions due to inconsistent knowledge, the architecture detects and resolves any contextual knowledge that may be inconsistent or ambiguous.

Moreover, the CALEEF architecture enables knowledge sharing among applications entities. The knowledge rules can be modified at any time to cater for a number of context aware applications. The architecture also takes into consideration the quality of context represented by selecting the most appropriate sensor to query for data from the Directory Service. The research goal is to find the right level of description, which abstracts away from implementation details, but is still specific enough to serve the purpose of inferring appropriate intent from context-assuming interactions.

Finally, an open issue in CALLEF is the definition of a context prediction method supporting the proactivity of context services in a dynamic environment which associates 'similar' context models. For the time being, CALEEF is only determining the polling time of sensors that will help in reducing the energy consumption by reducing wireless communication. But in the future, a prediction algorithm such as K-means might be used to predict the future context of a user based on current information available in the context history.

The goal of this research is to provide an architecture that makes it easier for application developers to use context. The architecture will enable developers to add context to applications that are not context-aware and to increase the use of context in applications that are already context-aware.

IX. Conclusion

Context-aware systems are a fast growing research area with a lot of diverse viewpoints and prototypes being proposed by researchers. Some of these existing context-aware architectures and supporting infrastructures which ease the development of context-aware applications have been discussed in this paper. And it has been observed that the techniques of accurately discovering context, efficiently disseminating contextual information and making use of the available context is still at the early stage. However there are major developments in the field and it is believed, that context awareness will be an important feature for new applications in ubiquitous computing. It has also been noticed that the main problem in the existing architectures is the variety of context encodings and ways to find or access context sources. It is accepted by researchers that standardized formats and protocols are important for the enhancements of context-aware systems to make the deployment of context services more widespread and it is the direction of most research in the field. Unfortunately, while good design alternatives for context architectures already exist, it will be a long time before standard, interoperable context information management becomes a reality.

In this paper, the issues that arise in supporting an emerging class of applications that operate independently of direct human control have been explored. CALEEF provides supports for most of the tasks involved in dealing with context, namely acquiring context from various sources, interpreting context and disseminating context. The main feature of the CALEEF architecture is that it supports context reasoning. Through context reasoning, high-level, implicit contexts can be derived from low-level, explicit contexts and applications can be given a notion of the confidence of different contexts before acting on it. Context-aware mobile services can be easily built by using various types of contexts with different levels of complexity.

A long-term goal of CALEEF is to make sensors and context platforms flexible and scalable enough to be widely adopted in various context-aware applications. Future work in this area will investigate the use of service-oriented and autonomic computing concepts for building context-aware service frameworks. Another issue that merit further investigation is to enable the architecture to detect failure of a component and automatically restart it as well as restore it to its last working state. Moreover, context prediction can also be used to support pro-activity of context services.

.

References

[1] Abowd, G..D., Atkeson, C.G., Hong, J., Long, S., Kooper, R. & Pinkerton, M. (1997). Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3, 421-433.

[2] Anagnostopoulos, C., Tsounis, A. & Hadjiefthymiades, S. (2004). Context Awareness in Mobile Computing: A Survey. *Proceedings of Mobile and Ubiquitous Information Access Workshop*, Mobile HCI '04, Glasgow, UK.

[3] Biegel, G. & Cahill, V. (2004). A Framework for Developing Mobile, Context-aware Applications. *Proceedings of 2nd IEEE Conference on Pervasive Computing and Communications*, (Percom) 2004, Orlando, FL.

[4] Chen, H. & Finin, T. (2003). An Ontology for a Context Aware Pervasive Computing Environment. *IJCAI Workshop on ontologies and distributed systems*, Acapulco MX.
[5] Dey, A. (2000). Providing Architectural Support for Building Context-Aware Applications. Ph.D. Thesis Dissertation, College of Computing, Georgia Tech.
[6] Dey, A. & Abowd, G.D. (1999). Towards a Better Understanding of Context and Context Awareness. Technical Report, GIT-GVU-99-22, Georgia Institute of Technology.
[7] Dey, A., Kokinov, B.N., Leake, D.B. & Turner, R.M. (2005). Modeling and Using Context. *5th International and Interdisciplinary Conference*, CONTEXT 2005, Paris, France.
[8] Dey, A., Salber, D., & Abowd, G.D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. *In the Proceedings of the 1999 Conference on Human Factors in Computing Systems*, CHI 1999, Pittsburgh, PA, 434-441.
[9] Fahy, P., Clarke, S. (2004). CASS: Middleware for Mobile, Context-Aware Applications. *Workshop on Context Awareness at MobiSys 2004*, Boston, USA.
[10] Gu, T., Pung, K. K. & Zhang, D. Q. (2004). A middleware for building context-aware mobile services. *In Proceedings of IEEE Vehicular Technology Conference (VTC)*, Milan, Italy.
[11] Hong J. & Landay, J. (2001). An infrastructure approach to context-aware computing', *Human Computer Interaction*, 16(2).
[12] Kuo, Y., Cheng, K., Hsu, J., Chu, H. & Huang, P. (2004). Architectural Support for Context-Sensitive Interaction in Ubiquitous Computing Environment. A proposal submitted to the IIS, Academia Sinica.
[13] Marin-Perianu, R., Hartel, P., & Scholten, H. (2005). A Classification of Service Discovery Protocols. Technical Report TR-CTIT-05-25 Centre for Telematics and Information Technology (University of Twente, Netherlands).
[14] Meditskos, G., & Bassilliades, N. (2005). Towards an Object-Oriented Reasoning System for OWL. *International Workshop on OWL Experiences and Directions*, B. Cuenca Grau, I. Horrocks, B. Parsia, P. Patel-Schneider (Ed.) (Ireland).
[15] Musolesi, M. (2004). Designing a Context-aware Middleware for Asynchronous Communication in Mobile Ad Hoc Environments. *In Middleware 2004 Companion Proceedings*, 304-308, ACM Press.
[16] Ranganathan.A and R. H. Campbell, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments", In ACM/IFIP/USENIX International Middleware Conference 2003, Germany, pp. 143 – 161, June 2003.
[17] Ranganathan.A, J. Al-Muhtadi, S. Chetan, R. Campbell and M. D. Mickunas, "MiddleWhere: A Middleware for Location Awareness in Ubiquitous Computing Applications", In Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, Canada, pp. 397 – 416, 2004
[18] Riekki.J, O. Davidyuk, V. Rautio and J. Sun, "Context-Aware Middleware for Mobile Multimedia Applications", In Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia (MUM'04), Maryland, Vol. 83, pp. 213 – 220, October 2004
[19] Shilit, B. N. (1995). A Context-Aware System Architecture for Mobile Distributed Computing. Ph.D. thesis, Dept of Computer Science, Columbia University.
[20] Smith, M. K., Welty, C., Volz, R., & Mcguiness, D. (2006). OWL Web Ontology Language Guide. W3C Recommendation February 2004. Online: http://www.w3.org/TR/owl-guide/.
[21] Veríssimo, P., Cahill, V., Casimiro, A., Cheverst, K., Friday A. & Kaiser, J. (2002). CORTEX: Towards Supporting Autonomous and Cooperating Sentient Entities. *Proceedings of European Wireless 2002*, Florence, Italy.
[22] Want, R., Hopper, A., Falcao, V. & Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems*, 10(1), 91-102.