Context Awareness in Large Scale Ubiquitous Environments with a Service Oriented Distributed Middleware Approach

Saad Liaquat Kiani, Maria Riaz, Sungyoung Lee, Young-Koo Lee
Real Time & Multimedia Lab, Kyung Hee University, Giheung, Yongin, Gyeonggi-Do, 449-701, South Korea
{Saad, Maria, Sylee}@oslab.khu.ac.kr, Yklee@.khu.ac.kr

Abstract

Various components of context aware middleware infrastructure work collectively to enable physical spaces to be transformed into computationally active and intelligent environments by providing context synthesis and provision services. In this paper, we discuss the requirement of physically separating the components of a context aware middleware in a distributed environment and present the design and implementation of Context Aware Middleware for Ubiquitous Systems (CAMUS)[1]. Issues related to distributed coordination within the middleware in terms of component discovery and management and multiple context domains are also discussed in order to elaborate service oriented approach for providing context awareness in large scale ubiquitous environments.

1. Introduction

Context aware computing provides services that are appropriate for a person's context, interpreted as any information that describes the setting of the user's activities, with emphasis on the physical attributes: time, place, people, physical artifacts, and computational objects. Augmented with ubiquitous computing [1], context awareness allows handheld devices, sensors, computer systems integrated with wired and wireless networks to play a functional role in our everyday life.

Different approaches have been proposed for building context-aware applications and services. Anind Dey et al [2] have built a Context Toolkit to support rapid prototyping of certain types of context-aware applications by providing a number of reusable components. Besides the Toolkit approach ([3], [4]), middleware infrastructures have been proposed [5], [6], [7] encompassing uniform abstractions and reliable services for common operations, support for most of the tasks involved in dealing with contexts, and thus simplify the development of context-aware applications. While different in approach, all infrastructures have the same goal of gathering environment features and transforming them into deliverable context. The tasks involved in such a system can be generally categorized as sensor data acquisition mechanisms, context formation techniques and context delivery methods.

The authors have proposed a *Context Aware Middleware for Ubiquitous Computing* (CAMUS) to address the discussed issues. Iin this paper we will focus our discussion of CAMUS towards the distributed nature of the infrastructure, coordination and management aspects. Sec. 2 presents the overview of the CAMUS and is diagrammatically depicted in Fig. 1.

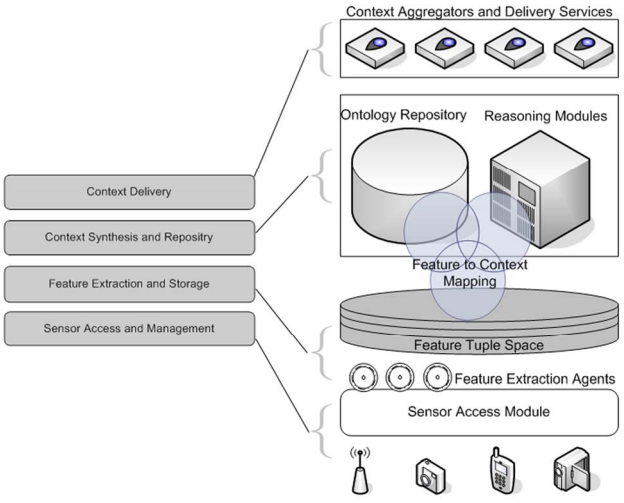

Figure 1. CAMUS middleware architecture modules mapped into 4 separate functions

Sec. 3 lists the issues which render a distributed middleware approach necessary for implementation and deployment of the middleware. The design considerations

[1] This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment)

for the coordination framework which lead to a service oriented design are explained in Sec. 4 and Sec. 5 respectively. Prototype implementations, runtime overview of CAMUS infrastructure and execution snapshots are given in Sec. 6. Sec. 7 concludes with an analysis of our framework and future directions in.

2. Functional Overview of CAMUS

The CAMUS architecture is modeled to provide context aware services in four steps that include sensor access, feature extraction, context synthesis and context delivery. The lowest layer of CAMUS consists of a Sensor Access Module (SAM) which provides unified access to hardware sensors. SAM, as depicted in Fig. 2, provides a Hardware Abstraction Layer (HAL) which masks the heterogeneity of the environment sensors from the upper layers of the system. It provides a sensor driver API which allows native sensor drivers to be used to access the sensors in a unified manner.

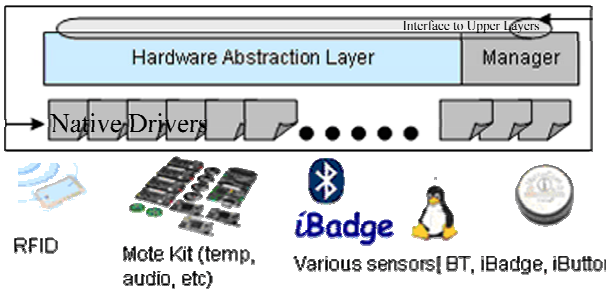

Figure 1. Sensor Access Module (SAM) includes a Manager, HAL layer, pluggable drivers for various sorts of sensors and provides a unified interface for upper layers to query the sensors and retrieve information

Feature Extraction Agents (FXA) are sensing agents that extract the most descriptive features from the sensors through the SAM. In order to have a more expressive representation of information related to the user's environments, features are further quantized or segmented, resulting in a set of symbolic values that describe concepts from the real world. The quantized features are encapsulated in the form of a 'Feature Tuple' and stored in *Feature Tuple Spaces* (FTS). The use of 'feature' abstraction allows context synthesis engine to work on more descriptive input rather than raw sensor values. This also allows for customization of feature extraction algorithms in a way that two entities requiring input from the same sensor can use different FXA that are tailored to their particular requirements. To facilitate the identification of individual features uniquely as well as collection of features by the same source a unique feature identifier is allocated in conjunction with the sensor and type identifiers to each Feature Tuple (FT) in the space.

FT= {Sesnor_ID, Type_ID, Feature_ID, Feature_Value, Probability, Timestamp}

Feature Tuple Space (FTS) is employed as underlying storage mechanism. Features are stored directly as objects independent in space and time and decoupled from the context generating processes; an important advantage in the ubiquitous environment in terms of interoperability and scalability. Various sub-modules for feature extraction and context formation dynamically interact in the middleware by mere flow of objects in and out of the FTS. Current FTS implementation in CAMUS is done on top of IBM TSpaces [8] with extended read, write methods and customized events.

Feature - Context Mapping layer performs the mapping required to convert a given feature into elementary context using reasoning mechanisms [9] and base on the meta-information saved in the ontology repository. *Ontology Repository* provides the basic storage services in a scalable and reliable fashion and contains the domain ontology (concepts and properties), contextual information (including both elementary and composite contexts), and meta-information. *Reasoning Engine* is a collection of various pluggable reasoning modules to handle the facts present in the repository as well as to produce composite contexts. Reasoning mechanisms based on various kinds of logic have been employed, e.g. description logic, first order logic, temporal and spatial logic, fuzzy logic, machine learning mechanisms like Bayesian and neural networks, depending upon the type of contextual information being produced. *Context Aggregator* is responsible for satisfying certain context queries and providing context to interested applications through *Context Delivery Services.*

3. The Case for Distributed Middleware

Most of the current context aware systems that have been prototyped are limited to providing context at a small physical scale e.g. a campus environment [9], a laboratory, a home [10] etc. At such a scale, the problems of a distributed environment do not present themselves adequately since the sensors are confined to a limited space (allowing easy sensor management), the context synthesis and delivery services run on a single system, system contact points are known [2] and dynamic discovery of system services is not required. In a practical scenario, an effective context aware system will provide context services over a vast stretch of environments ranging from homes, campuses, market places to city blocks and larger precincts. To manage such extended spaces it is necessary to separate the overall environment into smaller logical domains and incorporate a robust coordination and management framework as dictated by following reasons:

- The limitation in the communication range of most sensors makes it necessary for input gathering software components to exist in physical proximity to the sensors. Since sensors are diversely deployed in a ubiquitous environment, multiple input gathering components of a context aware system require coordination and management for data procurement.
- The synthesis of context is a complex process because environment sensors cannot pinpoint users activities in an exact manner and user context has to be interpreted using logical, rule based systems or systems based on Bayesian networks etc. The complexity involved in context synthesis may require special computing devices for efficient performance.
- Context is not limited to a confined physical space since a typical user of context provision services is mobile, moving from one domain to another e.g. a commuter traveling from home to office and then to some market place. Employing a single system to manage context synthesis and delivery for a large environment consisting of many sub-domains can be performance limiting. It is best that the whole environment is segregated into separate logical domains and clone sub-systems handle the steps involved in the context delivery process in each domain individually.

In order to carry out these varying specialized tasks and incorporate a considerably large number of hardware and software clients and contributors, a distributed setup becomes inevitable. Objects that interact in a distributed system need to be dealt in ways that are intrinsically different from objects that interact in a single address space [11]. Various techniques such as agents based interactions [12] and broker services [6], [7] have been employed for this purpose. In CAMUS, following requirements for coordination arise when functionality is distributed amongst components on specialized systems.

3.1. Logical and Physical Separation

The foreseeable problem that is related to heterogeneous sensors deployed in the environment is the limited communication range of sensors e.g. RFID, infrared, blue-tooth radio enabled sensors. It implies that the software components responsible for managing the sensors (start, stop, reset, discover, register etc) and retrieving measurements of environmental parameters have to be located in proximity to the sensors. However, deploying sensor access components close to sensors is a tricky task due to the distributed nature of sensors. With the distribution of sensors being wide and sparse, deployment of access components close to sensors becomes difficult and we have to adopt a divide and conquer strategy by expending more than one access module to cover the set of sensors exhaustively and combining their results for context generation later on.

In context aware systems, the raw values retrieved from sensors are initially encapsulated in a data format before being synthesized into context. Since an entity's context is an interpreted result from a collection of features, it cannot be derived from a single sensory source. This constraint necessitates that such intermediate data is placed in storage till adequate information sources contribute and reasonable context can be inferred. In CAMUS, this underlying storage mechanism for data acquired from sensors is the FTS as discussed in Sec. 2 and provides a domain-wide persistent space.

3.2. Context Domains

Ubiquitous computing environment is characterized by various domains e.g. home, office, university etc. To formally model context information to represent a particular domain, individual components need to be affiliated with a specific domain to relate coherent environments and entities, and to confine them within a logical boundary. A functional context aware system will be entrusted with context generation of a considerable number of entities and the combined effect of context inference, repository access and context history storage amounts to a computationally demanding task. Instead of employing a single context synthesis component to interpret context on behalf of all entities, the concept of separation of concerns based on geographical or logical boundaries has been implemented in CAMUS where individual domains are responsible for context management within domain boundaries e.g. home and university domain in Fig. 3.

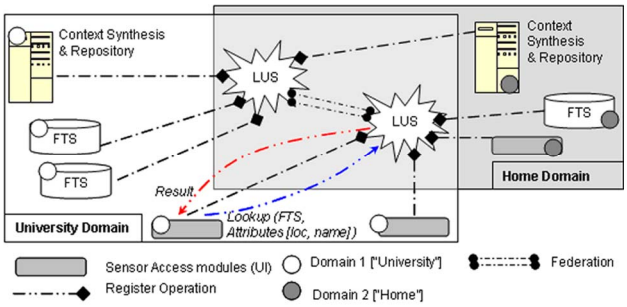

Figure 3. A scenario where individual components are associated with a single domain at a time

4. Design Considerations for Coordination

CAMUS has been designed to be deployed not as a single standalone infrastructure, but as multiple clone systems each responsible for managing context awareness in its allocated zone and interacting with other CAMUS

instances for inter-domain context sharing. In Fig. 3 the core components of CAMUS system, namely SAM, FTS and context synthesis components are logically bound to a single domain. Their functionalities are restricted to serving context in their respective area of allocation.

Since various components are deployed physically apart from each other within a given domain, the foremost issue that presents itself here is that of discovery of individual components. A mechanism is required by which all the components can make themselves, their capabilities and functionalities known to others. The requirements and design constraints which lead us to a service oriented solution are listed as follows:

1. Registration of components capabilities and services
2. Discovery of individual components by peers
3. Searching of components based on required functionality
4. System safeguard against temporary failures of individual components
5. Scalability to allow for system growth without reconfiguration

5. Service Oriented Approach to Middleware Coordination

CAMUS coordination sub-system is based on Service Oriented Architecture (SOA) [13] where core components are distributed services residing on the network to be published, discovered and invoked by each other. It also allows a software programmer to model programming problems in terms of distributed services offered by components to anyone, anywhere over the network. To implement CAMUS architecture based on SOA, several existing technologies were investigated including Web Services, Java RMI, Jini [14], UPnP and CORBA; which are capable of, to one extent or another, satisfying the stated requirements. For reasons stated in Sec. 7, Jini technology was found to be the closest match to our requirements. Figure 4 shows core CAMUS services interacting with each other and the flow of information is also depicted.

The discovery and registration process among modules is facilitated by Jini while the internal working of each module is independent of the underlying coordination mechanism. When a service becomes available, it sends a registration message to the lookup service and makes itself known to others. Modules can also discover each other by querying the lookup service. When a component becomes available, it joins a specific domain by registering the attributes and capabilities it affords along with a downloadable proxy with a *service registry, specifically a Jini Lookup Service* (LUS). Other components can discover the service by looking up specific attributes they are interested in. When the module of interest becomes available, its proxy is downloaded from the registry and is used for communicating back and forth between components.

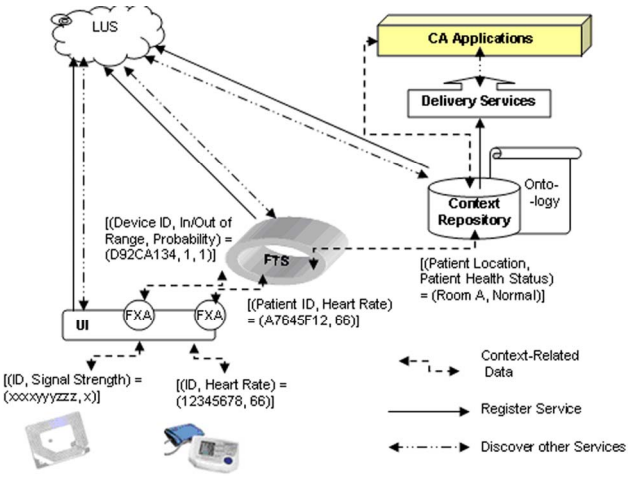

Figure 4. Interaction between modules via Jini with respect to data and communication messages

The attributes published by components vary from basic properties such as name, service type, location, and status to specialized capabilities of the module for more specific lookup provision. Components can specify the location during lookup operations so they can locate other components (FTS in this case) in their proximity. Similarly, queries can be further restricted to domains memberships, e.g. an FTS belonging to 'University' domain may search only for SAM instances located in that domain. For instance, an FTS publishes following set of attributes during its individual registration:

Entry FTSAttributes [] = new Entry [7];
FTSAttributes [0] = new Location ("room 312", "floor 3", "engineering building");
FTSAttributes [1] = new Name ("RTMM Lab FTS ");
FTSAttributes [2] = new Status (StatusType.NORM);
FTSAttributes [3] = new Domain ("University");
FTSAttributes [4] = new FTSCapability (READ);
FTSAttributes [5] = new FTSCapability (WRITE);

In a scenario where system users increase in number or the system has to manage a larger area equipped with a greater number of sensors, sudden changes and unpredictability in system configuration becomes inevitable leading to an increase in the responsibility of system components which may serve as a performance penalty. To meet this challenge at runtime, one of the solutions is to increase the number of system components handling the tasks that now offer an increased workload.

In CAMUS, an increase in the number of sensors in a domain can be accommodated by the deployment of additional sensor access modules and/or feature tuple spaces as shown in Fig. 6.

Efficient coordination amongst components requires that the increase in number of components does not hamper the discovery and registration process. For this purpose, multiple lookup services handling the task of discovery and registration are federated and the clients' (middleware services) queries are distributed across a number of lookup services to balance the load on the system and thus avoiding bottlenecks.

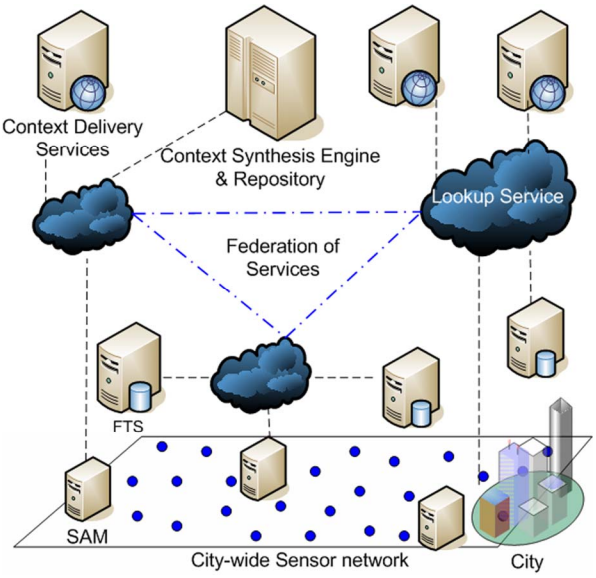

Figure 6. A hypothetical organization provides city wide context delivery service. A wide area sensors network is deployed to retrieve environment and user related data. Multiple SAMs are deployed to access sensors and feed features into FTS deployed conveniently across the city.

6. Implementation Overview

The authors have implemented the CAMUS middleware infrastructure using Java and the coordination infrastructure is based on Jini technology. Individual components of CAMUS are deployed as services and their responsibility is limited to logical domains. After individual startup of core components (SAM, FTS, Context Repository and Ontology Server, Context Aggregators and Delivery Services), the middleware self configures itself to form a domain. A collection of drivers for sensors and feature extraction agents for multiple purposes have been made available as Java jar files. Available sensors include audio, video, RFID tags and readers, Mote Kit sensors including light intensity, temperature and humidity sensors etc.

This collection of sensors and their respective feature extraction agents have been used to prototype a number of context aware applications, e.g. the well known meeting room scenario where presence of sufficient participants is detected by the system through sensor information, Presentation material is distributed to their devices if necessary, actuators are used to dim room lights, presentation is projected and room temperature is kept to an optimal level. Another application that prototypes the knowledge sharing between two domains, Home and University, has been implemented. The aim is to make a user's context information related to one domain accessible while he is physically present in another domain. The motivation behind this is the mobile nature of the user and a good example of such a user is a university student who commutes between his apartment and the university campus everyday.

7. Analysis and Future Work

In the existing solutions for context awareness, a variety of approaches have been tried to enable coordination in the framework. In [2], XML messages over HTTP (TCP/IP) are used between various components for this purpose. This approach is useful since both these protocols are practically ubiquitous. However, the toolkit does not provide a discovery mechanism and the communicating components need to know the exact location of each other in order to interact, hence the coordination is not dynamic and scalable. A number of middleware solutions use Corba based coordination and communication [16], [17] which provides dynamic resource discovery but due to its hard-state registration and lack of support for leasing mechanism, the system behaves reactively instead of proactively to the changing system dynamics.

Implementation of our system in Java allows us to utilize the Java activation framework [18] as a performance enhancing measure through which services can be transferred from main memory to persistent storage when idle, and can be made available as and when they are requested by a client. Activation also aids in fault tolerance when crashed service is restarted from the last known good configuration.

Jini was found out to be the closest match to the specific requirements of our system. Particularly, the possibility of querying components by attributes, downloadable proxies and independence from transport protocol were the main support features which were found lagging in other similar technologies such as UPnP. Moreover, the advantages of leasing, remote and distributed event notification model and event mailboxes provided by Jini can be utilized to full extent in distributed middleware architectures.

As a task for future enhancements, we aim to extend the service search capability available in Jini by identifying and incorporating service attributes related to services in a context aware domain. Another area is the representation scheme for data acquired from the sensors. Most systems utilize a single representation scheme for both elementary sensor data and higher level context by using a variety of schemes such as name-value pairs in XML format [2], black-board systems [15], object oriented representation [16], and ontologies [17]. However, separation of concerns dictates a two-level representation scheme for representing elementary data gathered from the sensors at a lower level and contextual data formed after synthesis at a higher level. We employ the black-board approach (tuple repository) for representing sensor data and ontologies for representing context. The combination of these two representation schemes results in greater flexibility and loose coupling at the sensor level and provision of common understanding and higher semantic representation at the contextual level.

To summarize, the authors have discussed the requirements related to distributed coordination within context aware middleware infrastructures in terms of component discovery and management, dynamic system state and multiple context domains. A service oriented coordination framework based on Jini Network Technology is presented to address the issues and example applications that use the distributed middleware for context awareness are also discussed.

References

[1] Weiser, M.: The Computer for the 21st Century. In: Scientific America. (Sept. 1991) 94-104; reprinted in IEEE Pervasive Computing. (2002) 19-25.

[2] Dey, A.K., et al.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. In: A Special Issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal, Vol. 16. (2001).

[3] S. Jang, Woo, W.: Ubi-UCAM: A Unified Context-Aware Application Model. In: Context 2003, Stanford, CA, USA. (Jun. 2003).

[4] Gellersen, H.W., Schmidt, A., Beigl, M.: Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts. In: Mobile Networks and Applications, Vol. 7. (Oct. 2002) 341-351.

[5] Hong, J.: The Context Fabric. http://guir.berkeley.edu/projects/confab/.

[6] Ranganathan, A. and Campbell, R.H.: A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In: ACM/IFIP/USENIX International Middleware Conference, Brazil. (Jun. 2003).

[7] Harry, C., Finin, T., and Joshi, A.: An Intelligent Broker for Context-Aware Systems. In: Ubicomp 2003, Seattle, Washington. (Oct. 2003).

[8] Wyckoff, P.: TSpaces, In: IBM Systems Journal, August 1998.

[9] Burrell, J. and Gay, G.K. (2002). E-Graffiti: evaluating real-world use of a context-aware system. In: Interacting with Computers 14 (2002), 301-312.

[10] Kidd, C.D., et. al.: The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In: Cooperative Buildings, http://citeseer.ist.psu.edu/kidd99aware.html (1999) 191-198

[11] Waldo, J., Wyant, G., Wollrath, A., Kendall, S.: A Note on Distributed Computing. In: Mobile Object Systems: Towards the Programmable Internet. Springer-Verlag, Heidelberg, Germany, April 1997, pp. 49-64.

[12] Yang, K., Galis, A.: Policy-Driven Mobile Agents for Context-Aware Service in Next Generation Networks. In: Mobile Agents for Telecommunication Applications (MATA), 5th International Workshop, Morocco, (Oct. 2003).

[13] Furmento, N., Hau, J., Lee, W., Newhouse, S.: Darlington, J. Implementations of a Service-Oriented Architecture on top of Jini, JXTA and OGSA. In: Proceedings of the UK e-Science Program All Hands Meeting 2003, Nottingham, UK. Sept., 2003

[14] Sun Microsystems, Inc.: Jini™ Architecture specification. http://www.sun.com/jini/specs/.

[15] Mamei, M., Zambonelli, F., Leonardi, L.: Tuples on the Air: A Middleware for Context-Aware Multi Agent Systems. In: 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), Providence, Rhode Island, USA, May 2003.

[16] Román, M., et al.: Gaia: A Middleware Infrastructure to Enable Active Spaces. In: IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

[17] Yau, S. S., et al.: Reconfigurable Context-Sensitive Middleware for Pervasive Computing," IEEE Pervasive Computing, joint special issue with IEEE Personal Communications, 1(3), July-September 2002, pp.33-40.

[18] William Grosso: Activation Framework. In: Java RMI, O'Reilly & Associates, Inc. CA 95472, USA, October 2001 Ch. 17.