Research Issues in the Development of Context-aware Middleware Architectures

Hung Quoc Ngo, Anjum Shehzad, Kim Anh Pham Ngoc, S. Y. Lee, Manwoo Jeon
Real Time & Multimedia Lab, Kyung Hee University, Korea
{nqhung, anjum, anhpnk, sylee, imanoos}@oslab.khu.ac.kr

Abstract

Context-aware middleware encompasses uniform abstractions and reliable services for common operations, supports for most of the tasks involved in dealing with context, and thus simplifying the development of context-aware applications. In this paper[1], we address some key issues of a middleware for context-aware ubiquitous computing, ranging from design considerations of a unified sensing framework, formal modeling and representation of the real world, pluggable reasoning engines for high-level contexts, and context delivery-runtime service composition mechanisms. Our implementation experience indicates that a comprehensive approach throughout the system layers results in a flexible and reusable middleware architecture.

1. Introduction

Context-aware ubiquitous computing emphasizes on using context of users, devices, etc. to provide services that are appropriate to particular person, space and time. Different approaches have been proposed for building context-aware applications and services. The Context Toolkit developed by Anind Dey et al [1] provides a number of reusable components to support rapid prototyping of certain types of context-aware applications. Besides the Toolkit approach, middleware infrastructures have been proposed [2], [3], [4] [5] encompassing uniform abstractions and reliable services for common operations, support for most of the tasks involved in dealing with context, and thus simplifying the development of context-aware applications. While different in approach, all infrastructures have the same goal of gathering environmental features and transforming them into deliverable context. The tasks involved in such systems can be generalized as sensor data aggregation and fusion, context reasoning, and context delivery mechanisms.

We have proposed a *Context-aware Middleware for Ubiquitous Computing Systems* (CAMUS [6], [7]) to address the discussed issues. This paper presents our development experience in CAMUS, focusing on the key characteristics of a context-aware middleware in order to be successfully deployed in real environment. Section 2 briefly describes the functional overview of CAMUS. The design considerations of a Unified Sensing Framework are provided in section 3. Issues in developing Context Repository and Context Reasoning are discussed in section 4 and section 5 respectively. Section 6 describes the Context Delivery and Runtime Service Composition Mechanisms. Section 7 concludes the paper with an analysis of our middleware architecture and future directions.

2. Functional Overview of CAMUS

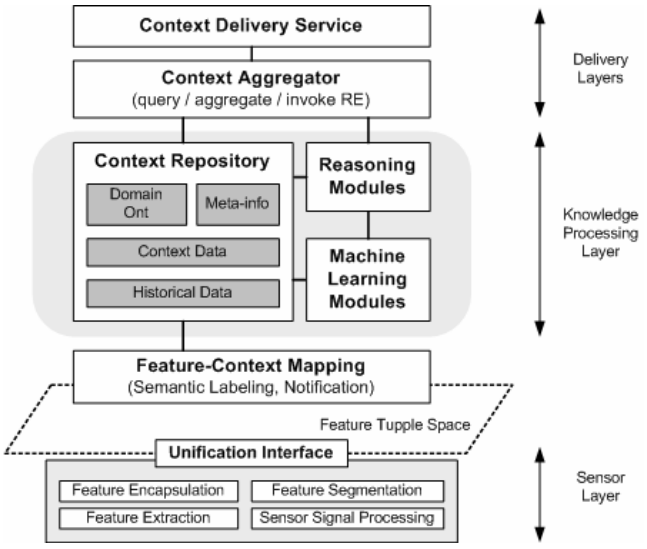

Figure 1. CAMUS Core Architecture

Our middleware architecture (depicted in figure 1) provides supports for gathering context information from sensors in a unified manner, incorporating different reasoning mechanisms for deducing high-level context,

[1] This research was supported by Ministry of Commerce, Industry and Energy, Korea.

and delivering appropriate contexts to applications as well as notifying the applications on context changes. More detailed definition and description of each functional module can be found in [6].

3. Design Considerations for Unified Sensing Framework

Ubiquitous environments contain diverse range of sensors each utilizing its native access mechanisms and output formats. This leads to potential problems and complexity in system design and implementation. The middleware needs to facilitate the device integration and information extraction from heterogeneous sensors, and present to the upper layers for deducing contexts in a standardized and unified manner.

CAMUS provides a software abstraction for sensing devices, namely Feature Extraction Agents (FXA), to hide the sensor technologies (i.e. low level communication details and specific processing algorithms of sensory data,) and expose to upper layers with a Unification Interface [8]. The feature extraction agents also hide the heterogeneity of outputs from disparate sensors, by encapsulating the features extracted from sensors in a common data structure of feature markup format [6].

With the abstraction of sensing agents, CAMUS lets developer deploy any type of sensors by implementing the native drivers for communicating with the sensor hardware; incorporating specific algorithms from domain experts for extracting and quantizing the most descriptive features from raw sensor data to get their semantic meanings; and providing the meta-data describing that Feature Extraction Agent. This solution enables a dynamic mechanism for mapping between the real world information and the virtual world of context representation. For example, when a new RFID tag is attached to an object, the developer just needs to append new information describing this tag (e.g. the name of the object carrying the tag) to the meta-data file of the RFID Readers, and no modification of the sensing modules is needed.

4. Context Repository & Query

Context Repository provides the basic storage services in a scalable and reliable fashion and contains the *Domain Ontology* and *Context Information* along with *Meta-information*.

4.1 Formal Context Modeling and Representation

In ubiquitous computing environments, applications need a shared understanding of context to communicate and transfer context effectively among them. This issue leads us to think of a formal context model for efficient utilization of context Based on different entities e.g. PDAs, mobile phones, ambient displays, sound intensity, light, temperature, traffic, software agents, persons, groups etc, we categorize them, in our framework, mainly into agents, devices, environment, location and time [6], [7]. OWL [9] is used for formal modeling of context since it allows us to define concepts and their inter-relationships e.g. describing person, devices, location etc., and to define instance data pertaining to some specific time and space, besides providing advantages of expressiveness, knowledge sharing, logic inference, knowledge reuse, and extensibility [9].

4.2 Domain Specific Data Model

A ubiquitous computing system may consist of many collaborative subsystems running on various domains such as home domain, office domain, university domain, etc. The use of ontology can help sharing the knowledge among different domains and systems. However, such a distributed and dynamic environment requires an efficient mechanism to store and retrieve context data over multi-domain repository. As CAMUS uses OWL format to store context data, it maintains a meta-graph to manage the meta-data about all the domain repositories. Using OWL format, the Context Repository can be backed by some kinds of DBMS such as MySQL, or just use text files if the system needs to run on some resource-constrained environment. When handling OWL data using Jena library [10], each database can be considered as a group of models, where each model is a collection of contexts. The ontologies defining context data schemas have hierarchical structure, so each context data model itself is a sub-graph of the large graph combining all the ontologies. Consequently, it is feasible to build a meta-graph of all graphs in a ubiquitous environment. That meta-graph stores the information about the models of each domain, names and namespaces of the models, and especially the contexts provided by each model in a hierarchical structure. Context data can be retrieved by RDQL [11] queries. The queries are parsed into list of condition triples. Then the contexts mentioned in condition triples are used to search all the models which can provide those contexts from the meta-graph. After that, for each concerned model, all the related statements are extracted using the template statement built from the condition triples. Jena library allows us to integrate many statement sets into one model before executing the query. Because each Context Repository Manager module runs as a service, it can advertise itself as well as discover other Repository Manager services. Whenever it discovers a new repository, it will integrate the meta-graph of the new repository in its own meta-graph.

5. Context Reasoning

A middleware infrastructure needs to provide support for incorporating different reasoning mechanisms into the system, as well as easily specifying the appropriate mechanism for reasoning each high-level context. This will facilitate not only the system internal modules to infer high-level context from low-level or predefined contexts, but also the applications to reason for their own application-specific context. A well defined set of Reasoning Engine APIs makes it possible to add and handle different reasoners as pluggable modules but, in return, it requires huge effort to come up with a uniform structure for different reasoning mechanisms.

The following piece of code illustrates how to add and invoke a rule-based reasoner in CAMUS, by declaring the *rule file name* and some *namespace abbreviations*:

```
/* declare the prefixes for namespaces */
      ContextReasonerManager.registerPrefix ("conagnt",
      rtmm.camus.vocabulary.contel.Agent.NS);
      ContextReasonerManager.registerPrefix ("env",
      rtmm.camus.vocabulary.contel.Environment.NS);
      ContextReasonerManager.registerPrefix ("conloc",
      rtmm.camus.vocabulary.contel.Location.NS);
/* add a new reasoner providing the rule file */
      ContextReasonerManager.addReasoner ("Location",
      ReasonerType.GENERIC_REASONER,
      "etc/contel.rules");
/* declare some statements */
      sms = new ContextStatement [] {PastLocationDescription,
      hasLocation};
/* invoke the reasoner to do reasoning, providing the reasoner
name, the context data name and the required statements */
      cdm.invokeReasoning ("Location", "Data", sms);
```

In CAMUS, the *ContextReasonerManager* manages all the reasoners in the system through a unified interface *ContextReasoner* (the class diagram of Reasoning Engine is omitted due to space limitation.) All the reasoners will implement this interface. The *ContextReasonerManager* service can be called to *addReasoner()*, get an existing Reasoner, and then do reasoning by *invokeReasoning()*. Developers just have to compose the rule sets, and decide the context data which should be used, and the middleware will take care of all other work from creating the reasoner to inserting the new inferred data into the repository.

6. Context Delivery & Aggregator Services

The main motivation behind context delivery services in CAMUS is two fold:

- Provide a discovery and registration mechanism which can utilize the underlying contextual information's syntax as well as semantics in order to make more intelligent and accurate context service selection
- Incorporate dynamic and autonomous access-control mechanisms in the context delivery process to ensure privacy and overall integrity of the system

Keeping in view the requirements of the context delivery service and the representation scheme of the underlying data model for context, semantic web concepts for matchmaking [12] along with support for dynamic composition of context-aware services is being developed.

The services (context aggregators) utilize the registration interface to make their information known to the applications. Lookup interface enables the applications to find appropriate matching context providers. Policies/rules database contains the system level policies as well as optional aggregator services level policies and rules defining the requirements or conditions to access some specific service provided by the context-aware middleware. This process is handled by the access control module. The matchmaking module matches the appropriate service with the client provided the access control policies are not violated. Further details of this module can be found in another paper [13].

Runtime Composition. Composition of services is basically used in the workflow management systems such as [14]. The idea is very powerful and applicable to context-aware ubiquitous computing services when the context requested by user is not provided directly by single service but can be composed by combining several services in a flow. Since semantic matchmaking is being employed at the delivery service, we register service along with quantitative and qualitative semantics of its interface. Quantitative semantics is related to context service specification i.e. service name, operations/methods provided by the service through its exposed interface along with the inputs, outputs and exceptions of those methods, while qualitative semantics are dealt with excellence of the service i.e. its execution time, context freshness, reliability and availability semantics. An optional hardware used attribute can be used to show which hardware was used to gather the elementary context, e.g. location can be got by using with RFID, iButton, or even simple WLAN. Once context service is fully described (both quantitative and qualitative semantics), it can be then registered with the broker service. E.g. if the client is interested in user location and it is willing to provide user URI and expecting Location in terms of GPS location, then it can be defined roughly as:

Domain (CT) = Location Provider Service
Ontological class of required operation = UserLocation
Ontological class of required input = URI
Ontological class of required output = GPSLocation

Context Freshness < 5 Sec
Hardware Used = RFID || iButton

In this way, the client specifies its requirements in more expressive way and there are more chances to find suitable service as compared to simple search based on keywords.

7. Analysis and Future Work

The benefits of using unified sensing framework approach are two fold. Firstly it provides the separation of concerns, that context modeling and reasoning mechanisms can be developed independently with sensing technologies. Secondly, the use of features allows better description of different environment parameters than raw sensor values and features can be organized, stored and delivered in an efficient manner. For permanent storage of context data, OWL data is converted into relational DBMS by using the Jena framework API. This has certain performance limitations which made us believe the database storage schemes especially for OWL should be investigated along with different efficient query mechanisms to retrieve stored data. Similarly, providing different reasoning mechanisms to infer higher level demands a uniform data structure to incorporate information required by different reasoning engines. Applying some data mining and AI techniques into middleware is being developed in CAMUS, e.g. from the historical information of user location, user activity, the environment features, combined with user profile, some data mining algorithms can be used to mine the association rules which describe user preferences or user routine. Another example is that we can build the decision trees to predict future actions of user. Also, incorporating access control requires a lot of information inflow on behalf of the applications i.e., the applications are required to provide some credentials to match the policies. This process might be slow in case some mobile user just wants to retrieve general information e.g., weather, light conditions, humidity, goods available in the market etc from the context-aware system which are not subject to privacy constraints. In such scenarios, policies can be written to grant unhindered access to services that provide such contextual information. However, there is also a need to carefully define the data structure to represent policies and semantics so that the representation scheme facilitates these mechanisms. Concerns like dynamic policies, granularity of access control, coping with runtime police changes and providing certain level of trust of users can be dealt if autonomic access control techniques (out of scope of this paper) are employed in context delivery.

To summarize, important elements that comprise middleware for context-aware ubiquitous computing have been discussed. Context sensing, modeling and representation, context repository and query, pluggable reasoning modules, aggregators and delivery services, and runtime composition, are all required components for a comprehensive context-aware middleware solution. The intermingling of all these components is necessitated to spotlight a comprehensive solution. Following a systematic approach makes CAMUS a flexible and reusable middleware framework.

References

[1] Context Toolkit project, http://www.cs.berkeley.edu/~dey/-context.html

[2] Hong, J. I., et al., "An Infrastructure Approach to Context - Aware Computing", HCI Journal, 2001, Vol. 16.

[3] Guanling Chen and David Kotz., "Solar: An Open Platform for Context –Aware Mobile Applications", Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002), Switzerland, June, 2002.

[4] Anand Ranganathan, Roy H. Campbell, "A Middleware for Context -Aware Agents in Ubiquitous Computing Environments", ACM/IFIP/USENIX International Middleware Conference, Brazil, June, 2003.

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

[6] Hung, N.Q., Shehzad, A., Kiani, S. L., Riaz, M., Lee, S., "A Unified Middleware Framework for Context Aware Ubiquitous Computing", EUC2004, Japan, Aug. 2004.

[7] Anjum Shehzad, N. Q. Hung, Kim Anh Pham, Sungyoung Lee, "Formal Modeling in Context Aware Systems", Workshop on Modeling and Retrieval of Context, CEUR, ISSN 613-0073, Vol-114, 2004.

[8] Saad Liaquat Kiani *et al.*, "A Distributed Middleware Solution for Context Awareness in Ubiquitous Systems", accepted for publication, RTCSA 2005, HongKong.

[9] W3C Web Ontology Working Group, "The Web Ontology language: OWL", http://www.w3.org/2001/sw/WebOnt/

[10] "Jena: A Semantic Web Framework for Java", http://jena.sourceforge.net/

[11] Andy Seaborne, "RDQL – A Query language for RDF", http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[12] Trastour, D., Bartolini, C., Gonzalez-Castillo, J., "A Semantic Web Approach to Service Description for Matchmaking of Services", HP Labs Bristol, HPL-001-183, 2001.

[13] Maria Riaz, *et al.*, "Service Delivery in Context Aware Environments: Lookup and Access Control Issues", accepted for publication, RTCSA 2005, HongKong.

[14] F. Casati, S. Ilnicki, L. J. Jin, V. Krishnamoorthy and M. C. Shan, "eFlow: a Platform for Developing and Managing Composite e-Services", HP Laboratories Palo Alto, HPL-2000-36, March 2000.