A Middleware for Building Context-Aware Mobile Services

Tao Gu [1,2], Hung Keng Pung [1], Da Qing Zhang [2]

[1] Department of Computer Science, National University of Singapore

[2] Connected Home Lab, Institute for Infocomm Research
Singapore

gutao@comp.nus.edu.sg, punghk@comp.nus.edu.sg, daqing@i2r.a-star.edu.sg

Abstract —**Computing becomes increasingly mobile and pervasive today; these changes imply that applications and services must be aware and adapt to highly dynamic environments. Today, building context-aware mobile services is a complex and time-consuming task. In this paper, we present a Service-oriented Context-Aware Middleware (SOCAM) architecture for the building and rapid prototyping of context-aware mobile services. We propose an ontology-based approach to model various contexts. Our context model supports semantic representation, context reasoning and context knowledge sharing. We take a service-oriented approach to build our middleware which supports tasks including acquiring, discovering, interpreting, accessing various contexts and interoperability between different context-aware systems.**

Keywords –Pervasive Computing; Context-aware Middleware; Context-aware Mobile Services; Mobile Commerce

1 INTRODUCTION

With the recent advances in mobile computing technology and the penetration of wireless networks, the nature of the services is moving towards mobile services that are becoming accessible from network-enabled mobile devices. Mobile services are expected to expand significantly in the next few years in term of popularity, variety and complexity. To avoid increasing complexity, and allow the users to concentrate on their tasks, mobile services must be aware of their contexts and automatically adapt to their changing contexts - known as context-aware mobile services. By context, we refer to any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object [1]. For example, contexts can be a person's current location, the user profile, the current time and date, or the temperature, etc. Contexts-aware mobile services can be used in many areas, including:

- presenting context information to mobile users
- triggering actions/behaviors on the occurrence of a set of contexts
- adapting of presentation of services to mobile users

A number of context-aware systems have been developed to demonstrate the usefulness of various contexts. However, context-aware mobile services have never been widely available to everyday users and developing such systems is still a complex and time-consuming task. Aiming at providing a middleware-level support for building and rapid prototyping of context-aware services, we propose a Service-Oriented Context-Aware Middleware (SOCAM) architecture.

In the SOCAM architecture, we present a formal context model based on ontology. Contexts are represented as predicates written in OWL [2] - an ontology markup language. The benefit of the ontology-based approach is that context knowledge can be shared among different entities and reasoning about context becomes possible.

Our middleware provides supports for most of the tasks involved in dealing with context - acquiring context from various sources; interpreting context; and carrying out dissemination of context. The main feature of the SOCAM 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. Each component in SOCAM is designed as an independent service component which can be advertised, located and accessed through a Service Locating Service. Context-aware mobile services can be easily built by using various types of contexts with different levels of complexity.

The rest of this paper is organized as follows. Section 2 discusses on related work. Section 3 describes our context model. We present the SOCAM architecture in Section 4 and its implementation and evaluation in Section 5. Finally, we conclude in Section 6.

2 RELATED WORK

A number of context-aware systems have been developed to demonstrate the usefulness of context-aware computing technology. In the earlier stage of the research, many researchers focused on building *application-specific* context-aware systems such as the Active Badge [3] project which provided the phone redirection service based on the location of a person in an office; and the Cyberguide [4] project which provided a context-aware tour guide to visitors. In MIT's AIRE [5] project, an intelligent room equipped with computer vision and speech recognition systems was created to experiment with different forms of natural, multimodal human-computer

interaction. The HP's Cooltown [6] project is a web-based system for context-awareness. These systems are typically proprietary and difficult to obtain and process context information due to the "ad-hoc" approach they deployed. They may depend on the underlying hardware and operating system.

Some researchers take a *Framework-based* approach to provide basic structures and reusable components for common functionalities to enable easy creation of context-aware applications. The ParcTab [7] system was the earliest attempt on general context-aware framework. By using an object-oriented approach, the ContextToolkit [8] provides a framework and 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 work has focused on providing infrastructure support for context-aware systems. The advantage of the infrastructure-based systems has been pointed out in [9]. In the Context Fabric [9] infrastructure, Hong et al. took a database-oriented approach to provide context abstraction by defining a Context Specification Language; and a set of core services. However, the design of a proprietary context specification language may lead to the lack of a common model. Ranganathan et al. [10] developed a middleware for context awareness and semantic interoperability, in which they represented context ontology written in DAML+OIL [11]. In the CoBrA [12] project, Chen et al. proposed an agent-oriented infrastructure for context representation, sharing knowledge and user's privacy control. However, they did not provide a performance evaluation for the feasibility of context reasoning in pervasive computing environments; and there is no explicit support for mobile services.

3 An Ontology-Based Context Model

The basic concept of our context model is based on ontology which provides a vocabulary for representing and sharing context knowledge in a pervasive computing domain, including machine-interpretable definitions of basic concepts in the domain and relations among them. An ontology-based model for context information allows us to describe contexts semantically in a way which is independent of programming language, underlying operating system or middleware; enables formal analysis of domain knowledge, i.e., context reasoning.

The design of context-aware mobile services often concerns with the limitation of mobile devices such as low CPU speed and small memory. To meet such requirement and reduce the burden of context processing on mobile thin clients, we design our context ontologies in a two-level hierarchy. We divide a pervasive computing domain into several sub-domains, e.g., home domain, office domain, vehicle domain, etc; and define individual low-level ontology in each domain. We also define a generalized ontology which describes the general concepts in upper level to link up all the low-level context ontologies as shown in Figure 1. Domain-specific ontologies can be dynamically "bounded" or "re-bounded" with the upper ontology when the domain is changed. For example, when a user leaves his home to drive a car, the home-domain ontology will be automatically replaced by the vehicle-domain ontology.

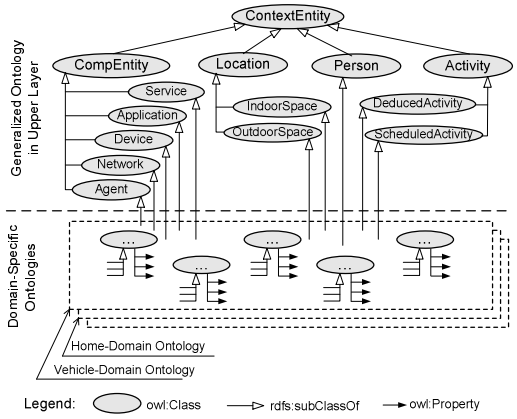

Figure 1. The ontology-based model with a two-level hierarchy

The structures and properties of different contexts are described in an ontology which may include descriptions of classes, properties and their instances. The ontology is written in OWL as a collection of RDF triples, each statement being in the form (*subName, predicate, objName*), where *subName* and *objName* are ontology's objects or individuals, and predicate is a property relation defined by the ontology. Some other issues related to context modeling, i.e., classification, can be found in our paper [13].

4 The SOCAM Architecture

Based on our context model, we design a Service-Oriented Context-Aware Middleware (SOCAM) architecture which aims to enable rapid prototyping of context-aware services in pervasive computing environments. The middleware converts various physical spaces where contexts are acquired from into a semantic space where contexts can be easily shared and accessed by context-aware services. It consists of Context Providers, Context Interpreter, Context Database, Service Location Service and Context-aware Mobile Services as shown in Figure 2.

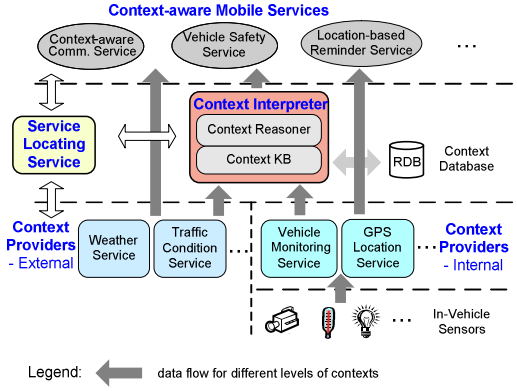

Figure 2. Overview of the SOCAM architecture

4.1 Context Providers

Context Providers provide context abstraction to separate the low-level context sensing from the high-level context manipulation. Each context provider needs to be registered into a service registry by using the Service Locating Service mechanism and can be discovered by others.

External context providers obtain contexts from external sources, e.g., a weather information server which is able to provide weather information on a particular place, or a road traffic provider which provides traffic information on a particular area. Internal context providers acquire contexts directly from ubiquitous sensors located in a sub-domain, i.e., a vehicle environment. For example, a location provider can provide the location of a vehicle acquired from an in-vehicle GPS receiver.

Different context providers acquire various context data from internal physical sensors or external virtual sensors and represent them as context events in the form of OWL descriptions. For example, a high-level description of context event "My car is approaching a supermarket" may be produced by a GPS location provider in the OWL description as shown below:

```
<socam:Car rdf:about="MyCar">
  <socam:hasNearbyPlace rdf:resource="#Supermarket"/>
</socam:Car>
```

4.2 Context Interpreter

The Context Interpreter also acts as a context provider as it provides high-level contexts by interpreting low-level contexts. It consists of a context reasoner and a context KB.

The context reasoner has the functionality of providing deduced contexts based on direct contexts, resolving context conflicts and maintaining the consistency of the context KB. Multiple logic reasoners can be incorporated into the Context Interpreter to support various kinds of reasoning tasks. Currently we have RDFS reasoner, OWL reasoner and a general rule-based reasoner built-in to our architecture. Different inference rules can be specified and preloaded into various logic reasoners. Developers can easily create their own rules based on predefined format.

The Context KB provides a set of API's for other service components to query, add, delete or modify context knowledge. The Context KB contains: context ontologies in a sub-domain and their instances. These instances may be specified by users in case of defined contexts or acquired from various context providers in case of sensed contexts. The context ontologies and their instances of defined contexts are pre-loaded into the Context KB during system initiation; the instances of sensed contexts are loaded during runtime. To ensure freshness of context information, we deploy an event triggering mechanism to allow updating of a particular context ontology or instance. Different information requires different update frequency. For example, the instance of a defined context may require updating every month or year, whereas the instance of a sensed context may need to be updated more frequently due to dynamic nature of the sensed data.

4.3 Service Locating Service

The Service Locating Service allows user, agents and applications to locate different context providers. We have developed the Service Locating Service mechanism in [14]. The main feathers include scalability, dynamics and multiple matching.

It supports wide-area discovery as a context provider, e.g, a weather service provider may be physically located in external networks. An internal context provider may change a context by adding and removing physical sensors or by reconfiguring a set of contexts supported. The service Locating Service is able to track and adapt to the dynamic changes of context providers. It also deploys a multiple matching mechanism to allow context providers to advertise their supporting contexts in different forms. Context providers can either use a service template or use OWL expressions to specify the kinds of contexts they provide. An application wishes to find out a context, for example the location context of "MyCar", will send the query "*socam:locatedIn(MyCar ?x)*" to the Service Locating Service. The Service Locating Service will load the context ontologies stored in the database and context instances advertised by different context providers, and apply semantic matching to find out which context provider provide this context. If a match is found, the reference to the context provider will returns to the application.

4.4 Context-aware Mobile Services

Context-aware mobile services are applications and services that make use of different level of contexts and adapt the way they behave according to the current contexts. By querying the service registry provided by the Service Locating Service, we are able to locate all the context providers which provide a set of interested contexts. To obtain contexts, a context-aware service can either query a context provider or listen for events sent by context providers.

TABLE I. FOL RULES FOR DESCRIBING CONTEXT-AWARE BEHAVIORS

Context-aware mobile services in a smart phone	IF socam:locatedIn(John,socam:MyCar) $\vee$ socam:status(John,DRIVING) THEN forward incoming calls to the in-vehicle phone or voice mail
Location-based reminder services	IF socam:locatedIn(John,socam:MyCar) $\wedge$ socam:hasNearbyPlace(MyCar, SupermarketA) $\wedge$ socam:hasActivity(MyCalendar, Supermarket) THEN notify me(John) to buy something
Vehicle safety services	IF socam:locatedIn(John,socam:MyCar) $\wedge$ socam:status(SeatBelt,LOOSE) THEN alert me(John)

To construct context-aware mobile services, a common way is to specify actions that are triggered by a set of rules whenever the current context changes. In the SOCAM, service developers can easily write pre-defined rules and specify what methods to be invoked when a condition becomes true. All the rules will be saved in a file and pre-loaded into the Context Reasoner. Developers also can make changes on the rule file and load it during runtime. Table 1 shows some context-aware behaviors which specified by a set of FOL rules.

4.5 Interaction Between Components

The SOCAM middleware components are designed as independent service components which may be distributed over heterogeneous networks and can interact with each other.

A context provider which provides a set of contexts may acquire context data from heterogeneous sensors. Different context providers resided in an internal network or an external network register and advertise their services through the Service Locating Service. Context consumers, i.e., the context interpreter or context-aware mobile services, are able to locate a context provider and obtain a piece of context. The context interpreter and context-aware mobile services can also register themselves to the Service Locating Service or other service discovery mechanisms so that they can be discovered and accessed by other context-aware systems.

In the SOCAM architecture, context dissemination is done in both push and pull modes. We provide a set of procedures and APIs to support both context query and context event subscription mechanisms. Users or services can either issue a query for a particular piece of context or subscribe a context event to a context provider. When the event is triggered, the particular context in the form of OWL descriptions will be return to the subscriber.

5 Implementation and Evaluation

Our prototype system consists of an OSGi-compliant mobile service gateway, various computing devices and physical sensors in an intelligent vehicle environment.

We incorporate the SOCAM middleware into the OSGi (Open Service Gateway Initiative) service platform [15] to provide secure and reliable service delivery, and remote management of context-aware mobile services. The OSGi-compliant mobile service gateway is a Java embedded server that connects local networks, i.e., in-vehicle network, to wide-area networks. The gateway enables and manages communications to and from local networks. Various types of networked devices such as in-vehicle computers, mobile devices, and in-vehicle sensors are attached to the gateway via Ethernet, WLAN, Bluetoth, etc. The SOCAM middleware is built on top of the OSGi service platform to provide a middleware level support for context-aware systems.

The gateway is designed based on Intel Celeron 600M CPU and 256M memory and runs Linux 2.4.17 kernel. The Context Interpreter runs on the gateway and is implemented based on HP's semantic web toolkit - Jena2 [16].

We measured the performance for the reasoning process and service locating process and the experimental results are presented in this section.

In our implementation, the vehicle-domain ontologies consists of 19 classes and 30 properties. Context instances are provided by internal context providers. The Context Interpreter validates and parses these OWL expressions into RDF triples [17] and performs the reasoning process when a context query is received. On average, it takes about 0.5 seconds to load and merge different OWL files containing context ontologies and different types of context instances. The average runtime for the reasoning process is about 1 seconds and the memory consumption is about 6MB.

The result shows that the runtime is acceptable for running context-aware mobile services. It also demonstrate that the Context Interpreter has a reasonable performance to perform reasoning over a small-scale (i.e., 5000 triples) context knowledge in pervasive computing environments. By tailoring the upper context ontology and domain-specific ontologies in our context model, the total number of context classes/instances used in a sub-domain can be controlled within the limits. By decoupling context consuming and reasoning process, we are able to execute context-aware mobile services on resource-constraint devices and leave the computational-intensive reasoning tasks to resource-rich devices such as an OSGi mobile service gateway.

We also measured the system searching performance for the Service Locating Service. We created and registered various context providers, the interpreter and mobile services, multiple mobile clients emulated using Pentium laptop with wireless LAN connectivity on separate machines made concurrent requests to a server. We measured the average elapse time taken for different number of concurrent requests. The average elapse time for each search request is around 250 ms and is nearly proportional to number of concurrent requests. More detailed performance results can be found in [14].

6 Conclusion and Future Work

In this paper, we present a service-oriented middleware architecture to enable the rapid prototyping of context-aware mobile services in an intelligent vehicle environment. The SOCAM architecture which is based on our formal context model provides efficient supports for context acquisition, context discovery and context dissemination. Our experimental results demonstrate reasonable performance for context reasoning and searching; and the SOCAM middleware is capable to meet the requirements of context-aware systems concerning limited memory and CPU resources. We will continue to build our prototype system to realize the SOCAM architecture in an intelligent vehicle environment and we will work on some of issues including service adaptation with regards to changing contexts.

References

[1] Dey, A. and Abowd, G., "Towards a Better Understanding of Context and Context-Awareness",

Workshop on the what, who, where, when and how of context-awareness at CHI 2000, April 2000.

[2] M.Smith, C. Welty, and D. McGuinness, "Web Ontology Lanugauge (OWL) Giude", August 2003.

[3] Roy Want, Andy Hopper, Veronica Falcao, Jonathon Gibbons, "The Active Badge Location System", ACM Transactions on Information Systems, Vol. 10, No. 1, pp 91-102, January 1992.

[4] Sue Long, Rob Kooper, Gregory D. Abowd, and Christopher G. Atkeson, "Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study", Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking (MobiCom'96), November 1996.

[5] http://www.ai.mit.edu/projects/aire/projects.shtml#835.

[6] T. Kindberg and J. Barton, "A Web-based Nomadic Computing System", Computer Networks (Amsterdam, Netherlands: 1999), 35(4):443–456, 2001.

[7] Shilit, B.N., "A Context-Aware System Architecture for Mobile Distributed Computing," Ph.D. thesis, Dept of Computer Science, Columbia University, 1995.

[8] Dey, A.K., Salber, D. Abowd, G.D., "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(2-4), pp. 97-166, 2001.

[9] Jason I. Hong and James A. Landay, "An Infrastructure Approach to Context-Aware Computing", In Human-Computer Interaction , Vol. 16, 2001.

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

[11] Ian Horrocks, "DAML+OIL: a Reason-able Web Ontology Language", In Proceedings of the 8th International Conference on Extending Database Technology (EDBT), Prague, March 2002.

[12] Harry Chen and Tim Finin, "An Ontology for a Context Aware Pervasive Computing Environment", IJCAI workshop on ontologies and distributed systems, Acapulco MX, August 2003.

[13] Tao Gu, Xiao Hang Wang, Hung Keng Pung, Da Qing Zhang, "An Ontology-based Context Model in Intelligent Environments", In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, California, USA, January 2004.

[14] Tao Gu, H. C. Qian, J. K. Yao, H. K. Pung, "An Architecture for Flexible Service Discovery in OCTOPUS", Proc. of the 12th International Conference on Computer Communications and Networks (ICCCN), Dallas, Texas, October 2003.

[15] The Open Services Gateway Initiative (OSGi), www.osgi.org.

[16] Jena 2 - A Semantic Web Framework, http://www.hpl.hp.com/semweb/jena2.htm

[17] Dan Brickley, R.V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema", World Wide Web Consortium, January 2003.