Available online at www.sciencedirect.com
SCIENCE DIRECT
Journal of Network and Computer Applications 28 (2005) 1–18
Journal of NETWORK and COMPUTER APPLICATIONS
www.elsevier.com/locate/jnca

A service-oriented middleware for building context-aware services

Tao Gu[a,b,*], Hung Keng Pung[a], Da Qing Zhang[b]

[a]*Network Systems and Services Laboratory, Department of Computer Science, National University of Singapore, 3 Science Drive, 2 Singapore 117543*
[b]*Context-Aware System Department, Institute for Infocomm Research, Singapore*

Received 12 March 2004; received in revised form 27 May 2004; accepted 4 June 2004

Abstract

The advancement of wireless networks and mobile computing necessitates more advanced applications and services to be built with context-awareness enabled and adaptability to their changing contexts. Today, building context-aware services is a complex task due to the lack of an adequate infrastructure support in pervasive computing environments. In this article, we propose a Service-Oriented Context-Aware Middleware (SOCAM) architecture for the building and rapid prototyping of context-aware services. It provides efficient support for acquiring, discovering, interpreting and accessing various contexts to build context-aware services. We also propose a formal context model based on ontology using Web Ontology Language to address issues including semantic representation, context reasoning, context classification and dependency. We describe our context model and the middleware architecture, and present a performance study for our prototype in a smart home environment.

Keywords: Context-aware middleware; Pervasive computing; Context-aware services; Network services; Context model; Context ontology

1. Introduction

Emerging pervasive computing technologies provide ‘anytime, anywhere’ computing by decoupling users from devices and viewing applications as entities that perform tasks on

* Corresponding author. Address: Department of Computer Science, National University of Singapore, 3 Science Drive, 2 Singapore 117543. Tel./fax: +65-68747381.
E-mail addresses: gutao@comp.nus.edu.sg (T. Gu), punghk@comp.nus.edu.sg (H.K. Pung), daqing@i2r.a-star-edu.sg (D.Q. Zhang).

1084-8045/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2004.06.002

behalf of users (Henricksen et al., 2001). To avoid increasing complexity and allow the users is to concentrate on his tasks, applications and services must be aware of their contexts and automatically adapt to their changing contexts-known as context-awareness. 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 (Dey and Abowd, 2000). For example, contexts may include person (name, role, etc.), location contexts (coordinate, temperature, etc.), computational entity contexts (device, network, application, etc.) and activity contexts (scheduled activities, etc.). A context-aware service is a network service which uses various contexts and adapts itself to the change of environment dynamically and automatically. For example, a context-aware service for smart mobile phone is able to forward incoming calls to voice mail when a mobile phone user is currently sleeping in the bedroom.

Context-aware computing has been drawing much attention from researchers since it was proposed about a decade ago. However, context-aware services have never been widely available to everyday users. Recent research results show that building context-aware systems is still a complex and time-consuming task due to the lack of an adequate infrastructure support (Chen and Kotz, 2000). We believe such infrastructure requires the following supports:

(i) A common context model that can be shared by all devices and services. Context sharing is important as other entities can 'understand' contexts in a same smart space domain or across different domains. In previous systems, context information represented as string or modeled as Java programming objects are unlikely to be shared efficiently. Ontology has been widely used in many areas such as Artificial Intelligence and the Semantic Web (Berners-Lee et al., 2001). An ontology provides a vocabulary for representing knowledge about a domain and for describing specific situations in a domain. By applying ontology and the Semantic Web technology into pervasive computing environments, we propose an ontology-based context model using OWL (Smith et al., 2003)—an ontology markup language to enable context sharing by explicitly defining contexts in a semantic way. Most importantly, context reasoning becomes possible. Through reasoning, many high-level, implicit contexts can be derived from low-level, explicit contexts. Our model also captures information of context classification and dependency which is useful in the context reasoning process.

(ii) A set of services that perform context acquisition, context discovery, context interpretation and context dissemination. A challenge for building such infrastructure lies with designing a set of services to support various tasks. Some of these services may be highly application specific. By taking a service-oriented approach, we propose a Service-Oriented Context-Aware Middleware (SOCAM) which includes a set of independent services. A service component performs a specific task and interacts with other components to support context-aware applications. The SOCAM architecture which is built on our context model supports acquiring various contexts from different context providers, interpreting contexts through context reasoning, and delivering contexts in both push and pull modes. With these service components, context-aware services can be easily built by using different services to access various types of contexts with different levels of complexity.

The rest of this article is organized as follows. Section 2 discusses related work. In Section 3 we describe our modeling concepts, followed by the SOCAM architecture in Section 4. Section 5 presents our performance evaluation. Finally, we present our conclusions 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 (Want et al., 1992) project which provided the phone redirection service based on the location of a person in an office; and the Cyberguide (Long et al., 1996) project which provided a context-aware tour guide to visitors. In MIT's AIRE[1] 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 (Kindberg and Barton, 1999) 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 *toolkit-based* approach to provide basic structures and reusable components for common functionalities to enable easy creation of context-aware applications. The ParcTab (Shilit, 1995) system was the earliest attempt on general context-aware framework. By using an object-oriented approach, the ContextToolkit (Dey et al., 2001) 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 inHong and Landay (2001). In the Context Fabric (Hong and Landay, 2001) 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 and Campbell (2003) developed a middleware for context awareness and semantic interoperability, in which they represented context ontology written in DAML+OIL (Horrocks, 2002). In the CoBrA (Chen and Finin, 2003) project, Chen et al. proposed an agent-oriented infrastructure for context representation, sharing knowledge and user's privacy control. However, both Ranganathan and Campbell (2003) and Chen and Finin (2003) did not provide a performance evaluation for the feasibility of applying ontology in pervasive computing environments and their approaches did not address various characteristics of context information such as classification and dependency.

3. Context modeling and reasoning

In this section, we discuss the use of ontology and OWL; and describe our ontology-based approach for modeling contexts. The initial concept can be found in Gu et al. (2004a).

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

3.1. Ontology and OWL

The term 'Ontology' has a long history in philosophy, in which it refers to the subject of existence. In the AI literature, an ontology is a formal explicit description of concepts in a domain of discourse. It provides a vocabulary for representing knowledge about a domain and for describing specific situations in a domain.

Modeling context using an ontology-based approach allows us to describe contexts semantically in a way which is independent of programming language, underlying operating system or middleware. The main benefit of this model is that it enables formal analysis of domain knowledge, i.e. context reasoning using first-order logic, temporal logic, etc.

In our model, contexts are described by ontologies written in OWL. OWL, a key to the Semantic Web, is web ontology language proposed by W3C's Web Ontology Working Group. OWL is much more expressive than other ontology languages such as RDFS (Dan Brickley, 2003). We chose OWL rather than DAML+OIL as the latter has been merged into OWL to become an open W3C standard.

3.2. A formal context model

In our model, contexts are represented as first-order predicate calculus. The basic model has the form of *Predicate(subject, value)*, in which

- *subject* $\in$S*: set of subject names, e.g. a person, a location or an object.
- *Predicate* $\in$V*: set of predicate names, e.g. is located in, has status, etc.
- *value* $\in$O*: set of all values of subjects in S*, e.g. the living room, open, close, empty, etc.

For example, *Location(John, bathroom)*—John is located in the bathroom, *Temperature(kitchen, 120)*—the temperature of the kitchen is 120 °F, *Status(door, open)*—the door's status is open, etc.

The basic context model can be extended to form a complex context or a set of contexts by combining the predicate and Boolean algebra (union, intersection and complement). For example, *FoodPreference(familyMembers, foodItems)*—all family members' food preferences are a list of food items, can be represented as *FoodPreference(John, FoodList_1)* $\vee$ *FoodPreference(Alice, FoodList_2)* $\vee$ *FoodPreference(Tom, FoodList_3)*.

The structures and properties of context predicates 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 (*subject, predicate, object*), where subject and object are ontology's objects or individuals, and predicate is a property relation defined by the ontology. An example is shown in Fig. 1.

3.3. The design of context ontology

Context information has a great variety. Contexts include any information that describes physical objects, applications and users in any domain such as home domain, office domain, vehicle domain, etc. As a large proportion of context knowledge is unlikely

```
<rdf:RDF xmlsn:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlsn:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlsn:owl="http://www.w3.org/2002/07/owl#"
         xmlsn:socam="http://lucan.ddns.comp.nus.edu.sg/octopus/socam#"
>
...
<owl:Class rdf:ID="Adult">
   <rdfs:subClassOf rdf:resource="#Person"/>
   <owl:disjointWith rdfs:resource="#Child"/>
</owl:Class>

<socam:Adult rdf:ID="John">
   <rdfs:locatedIn rdf:resource="#bedroom"/>
</socam:Adult>
...
</rdf:RDF>
```

Fig. 1. A partial context ontology written in OWL.

to be processed and maintained efficiently in pervasive computing environments where the resource limitation of pervasive devices such as CPU speed and memory are often of concerned, we adopt a two-layer hierarchical approach for designing our context ontologies. Our context ontologies are divided into the common upper ontology for the general concepts and the domain-specific ontologies which apply to different sub-domains. The generalized ontology captures general contexts for all pervasive computing domains. The generalized ontology is fixed once defined and will be shared among different domains. The domain-specific ontology is a collection of low-level ontologies which define the details of general concepts and their properties in each sub-domain such as a smart home domain or a vehicle domain.

The separation of domains significantly reduces the scale of context knowledge, and hence releases the burden of context processing for pervasive devices in each domain. The low-level ontology for each sub-domain may be adaptive and dynamically 're-bind' with the generalized ontology when the environment is changed. For example, if a user is at home, the home-domain ontology is bound with the generalized ontology and used to derive high-level contexts. The user can access the context-aware services in a smart home scenario which are specified by a set of rules (described in Section 4.2.2). When the user leaves his home to drive a car, the vehicle-domain ontology will re-bind with the generalized ontology and a new set of rules for describing context-aware behaviors may also be loaded. A new set of context-aware services in a vehicle scenario now can be accessed, for example all incoming calls can be forwarded to the voice mail or to an in-vehicle hands-free phone when the user is driving. The change of domain is either initiated by the system automatically (i.e., when the user's location is detected by the network) or by the user. We assume no high-level service handover issues need to be considered as most current context-aware services are domain-dependent.

The generalized ontology defines the basic concepts of person, location, computational entity and activity and consists of 14 classes and six properties as shown in Fig. 2. The class *ContextEntity* provides an entry point of reference for declaring the generalized ontology. One instance of *ContextEntity* exists for each distinct user or service. Each

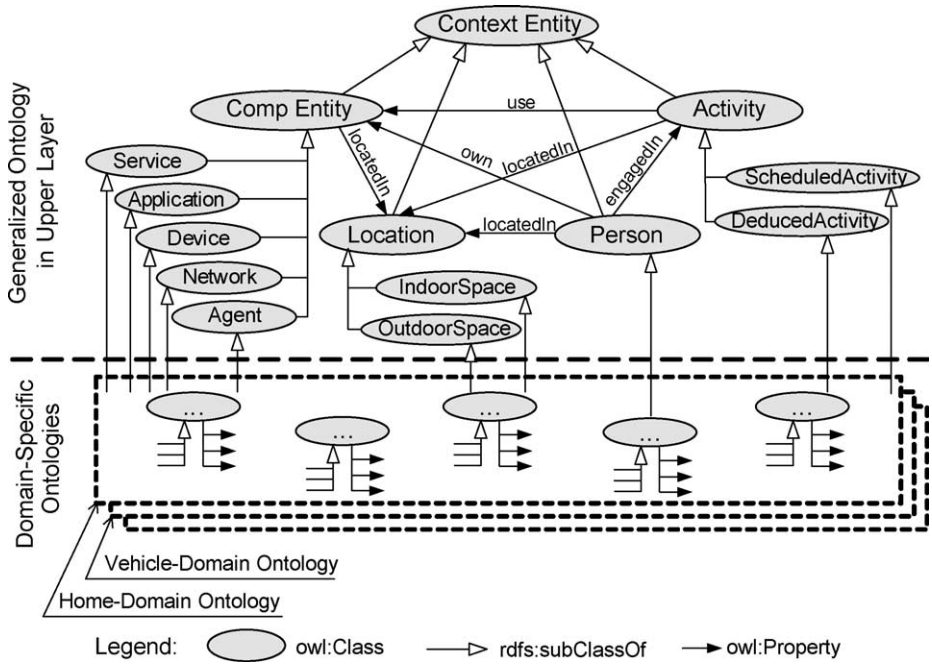

Fig. 2. Class hierarchy of the upper ontology.

instance of *ContextEntity* presents a set of descendant classes of *Person, Location, CompEntity* and *Activity*. The details of these basic concepts are defined in the domain-specific ontologies which may vary from one domain to another.

A domain-specific ontology defines the details of general concepts and their properties, for example, *IndoorSpace* in a smart home domain has subclasses such as *Entry*, *Room*, *Corridor*, etc. We have defined two domain-specified ontologies for smart home environments (as partially shown in Fig. 3) and vehicle environments.

3.4. Context classification and dependency

We classify contexts into two main categories—Direct Context and Indirect Context based on the means by which context is obtained.

Direct context is directly acquired or obtained from a context provider which can be an internal source such as an indoor location provider, or an external source such as a weather information server. Direct context can be further classified into sensed context and defined context. Sensed context is acquired from physical sensors such as a door's status context sensed by a door position sensor, or from a virtual sensor, e.g. a web service. Defined context is typically defined by a user such as user's foodPreference is specified by a particular person. Indirect context is derived by interpreting direct context through context reasoning. By using a context reasoner, an indirect context can be inferred from direct contexts. For example, the current status context of a person (*Showering*) can be inferred

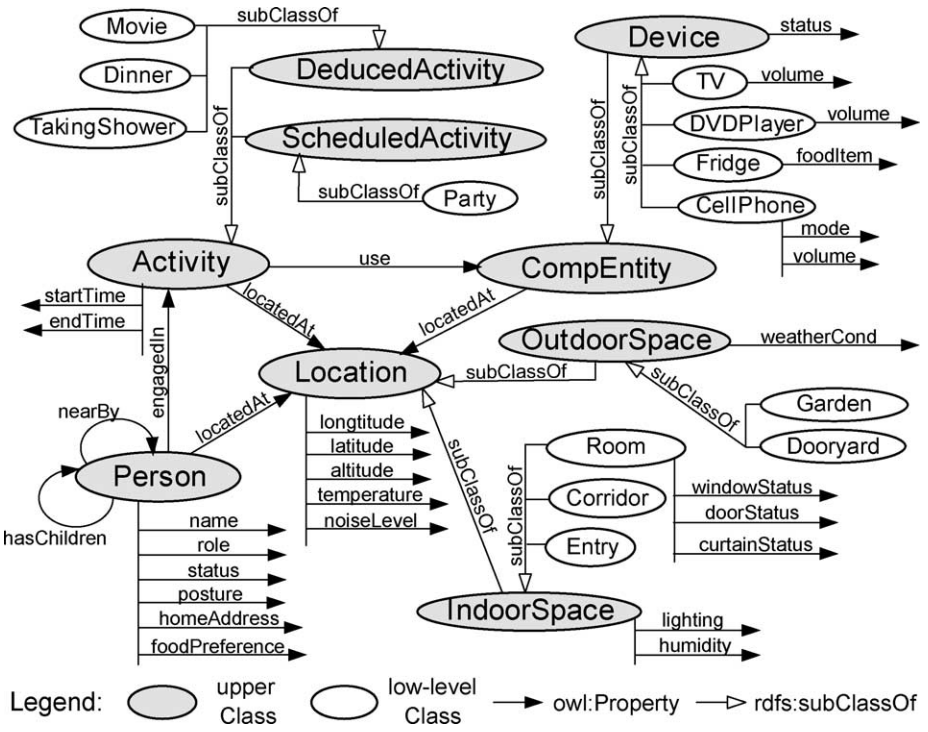

Fig. 3. A partial definition of a domain-specific ontology for smart home.

from his location (*Bathroom*), the status of the water heater (*On*) and the door's status (*Closed*).

Different types of contexts have different temporal characteristics, for example, sensed context is dynamic with the persistence of seconds to hours; defined context is most likely static over its lifetime. Contexts may be inconsistent or conflict with one another such as sensed contexts, for example, a RFID location sensor may sense a person is not present whereas a camera senses his/her presence due to imperfect sensing technology or processing delay.

By classifying various contexts and knowing their characteristics, we can perform context reasoning over different types of contexts to solve context conflict and maintain their consistency. For example, in general, defined context is more reliable as compared to sensed context.

To incorporate context classification information in the SOCAM context ontology using OWL, we introduce an additional property elements—*owl:classifiedAs*. This special element is able to capture the property of context classification associated with datatypes and objects. In our context ontologies, this additional property will have the value such as 'Sensed', 'Defined' or 'Deduced'.

Dependency is an important characteristic of context information. A dependency captures the existence of a reliance of property associated with one entity on another. To describe dependency information using OWL, we introduce an additional property

elements—*rdfs:dependsOn*. This special element is able to capture the dependency relationship of properties associated with datatypes and objects. For example, the status of a person may depend on his location context, his posture context, etc. The importance of context dependency is that we are able to incorporate probability and Bayesian networks to reason about uncertain contexts. We have extended the basic context model by attaching a probability value to each context predicate, i.e. *Prob(Predicate(subject, object))*. Based on the element—'*rdfs:dependsOn*', we are able to translate a RDF graph to a Directed Acyclic Graph (DAG) of Bayesian networks, each context predicate (represented as a RDF triple) is mapped into a node in the BN, and an arc is drawn between two nodes if and only if there exists a dependency relation between two context predicates. By training a set of data, we are able to obtain the probabilities of all root nodes and the conditional probabilities of all non-root modes. Hence, the probability distributions of various context events can be found. The details of the probability extension of our context model and reasoning uncertain contexts using Bayesian networks are included in Gu et al. (2004b).

4. The SOCAM architecture

Based on our context model, we design the SOCAM architecture (Fig. 4) which aims to provide an efficient infrastructure support for building context-aware services in pervasive computing environments. It is a distributed middleware that transfers converts various physical spaces from which contexts are acquired into a semantic space where contexts can be easily shared and accessed by context-aware services. It consists of the following components which act as independent service components.

- *Context providers*. They abstract useful contexts from heterogeneous sources—External or Internal; and convert them to OWL representations so that contexts can be shared and reused by other service components.
- *Context interpreter*. It provides logic reasoning services to process context information.

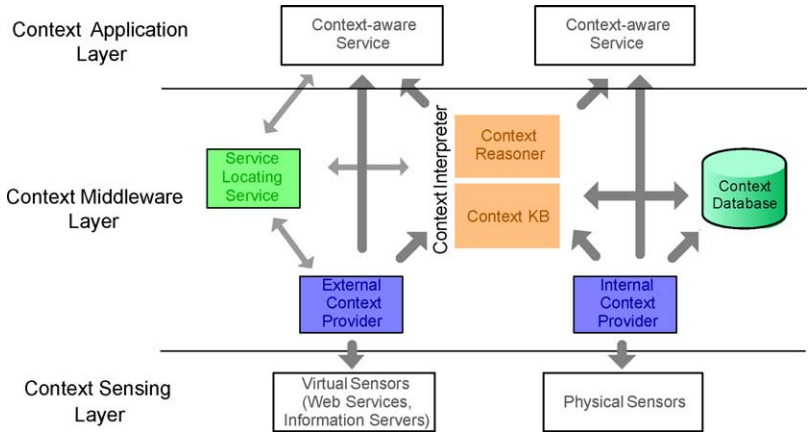

Fig. 4. Overview of the SOCAM architecture.

- *Context database*. It stores context ontologies and past contexts for a sub-domain. There is one logic context database in each domain, i.e. home domain.
- *Context-aware services*. They make use of different level of contexts and adapt the way they behave according to the current context.
- *Service locating service*. It provides a mechanism where context providers and the context interpreter can advertise their presence; it also enables users or applications to locate these services.

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 location server which provides outdoor location information for a person. Internal context providers acquire contexts directly from ubiquitous sensors located in a sub-domain, i.e. a smart home environment. For example, an indoor location provider can provide location information acquired from Radio Frequency Identification (RFID) based location sensors.

The advancement of video analysis technology provides us an opportunity to explore complex contexts, e.g. human's behavior. We leverage on our video surveillance system (Li et al., 2002) to perform object tracking and human behavior analysis, and provide behavior, posture context of a person. The system is able to collect raw video data and transform it into high-level descriptions of events. For example:

Event 1: a person lies down on the bed.
Event 2: an intruder detected in a private house.
Event 3: number of persons sitting down around a table in a meeting room.

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:

```
<socam:Adult rdf:about="John">
   <socam:hasPosture rdf:resource="#LIEDOWN"/>
</socam:Adult>
```

4.2. Context interpreter

The context interpreter is a component that is responsible for 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. It also acts as a context provider as it can provide deduced contexts.

It consists of a context reasoner and a context Knowledge Base (KB). The context reasoner has the functionality of providing deduced contexts based on direct contexts, detecting inconsistency and conflict in the context KB. Multiple logic reasoners can be incorporated into the context interpreter to support various kinds of reasoning tasks. Different inference rules can be specified and preloaded into various 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: a context ontology 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 ontology 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, defined context instances may require updating every month or year, whereas sensed context instances may need to be updated more frequently due to dynamic nature of the sensed data.

We adopt a rule-based approach based on first-order-logic for reasoning about contexts. In SOCAM, two kinds of reasoning are currently supported: ontology reasoning and user-defined rule-based reasoning.

4.2.1. Ontology reasoning

The ontology reasoning is responsible for checking class consistency and implied relationship, asserting inter-ontology relations when integrating or switching domain-specific ontologies. It is implemented using a rule-based reasoning engine.

The ontology reasoning includes RDFS reasoning and OWL reasoning. The RDFS reasoning supports all the RDFS entailments described by the RDF Core Working Group. The OWL reasoning supports OWL/lite (OWL Web Ontology Language Overview) which includes constructs such as relations between classes (e.g. disjointness), cardinality (e.g. 'exactly one'), equality, characteristics of properties (e.g. symmetry), and enumerated classes. A set of RDFS and OWL rules needs to be pre-specified, for examples

- subClassOf: (?A rdfs:subClassOf?A), (?B rdfs:subClassOf ?C)-> (?A rdfs:subClassOf ?C)
- TransitiveProperty: (?P rdf:type owl:TransitiveProperty), (?A ?P ?B), (?B ?P ?C)-> (?A ?P ?C)
- Disjointness: (?C owl:disjointWith ?D), (?X rdf:type ?C), (?Y rdf:type ?D)-> (?X owl:differentFrom ?Y)
- InverseOf: (?P owl:inverseOf?Q), (?X ?P ?Y)-> (?Y ?Q ?X)

Ontology reasoning is useful in SOCAM. For example, we can use the Transitive-Property rule to reason about physical location in a smart home. If an indoor location provider provides a context that *John* is currently located in his *LivingRoom*, the context interpreter can conclude that *John* is located in his home since *LivingRoom* is a part of

Home and the spatial property *locatedIn* exists the transitive property as illustrated in the following OWL descriptions.

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type="http://www.w3.org/2002/07/owl#TransitiveProperty">
</owl:ObjectProperty>
<socam:Room rdf:ID="LivingRoom">
  <socam:locatedIn rdf:resource="#Home"/>
</socam:Room>
```

4.2.2. User-defined rule-based reasoning

The user-defined rule-based reasoning provides forward chaining, backward chaining and a hybrid execution model. The forward-chaining rule engine is based on the standard RETE algorithm (Forgy, 1982). The backward-chaining rule engine uses a logic programming engine similar to Prolog engines. A hybrid execution mode performs reasoning by combining both forward-chaining and backward-chaining engines. The following examples show a partial rule set used in SOCAM based on the forward-chaining rule engine.

- (?user rdf:type socam:Person), (?user, socam:locatedIn, socam:Bedroom), (?user, socam:hasPosture, 'LIEDOWN'), (socam:Bedroom, socam:lightLevel, 'LOW'), (socam:Bedroom, socam:doorStatus, 'CLOSED')-> (?user socam:status 'SLEEPING')
- (?user rdf:type socam:Person), (?user, socam:locatedIn, socam:BathRoom), socam:-WaterHeater, socam:status, 'ON'), (socam:BathRoom, socam:doorStatus, 'CLO-CLOSED')-> (?user socam:status 'SHOWERING')]
- (?user rdf:type socam:Person), (?user, socam:locatedIn, ?room), (socam:TV, socam:-locatedIn, ?room), (socam:TV, socam:status 'ON')-> (?user socam:status 'WATCHINGTV')

4.3. Context-aware services

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

To construct context-aware services, a common way is to specify actions that triggered by a set of rules whenever the current context changes. In 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 an example configuration file which defines a set of rules.

Table 1
Rules for describing context-aware behaviors

Smart phone services	socam:status(John, SLEEPING) ∨ socam:locatedIn(John, socam:Bathroom)
	DivertCall(VoiceMail); //forward incoming calls to voice mail
	socam:locatedIn(John, Room) ∨ socam:hasNoiselevel(socam:Room, HIGH)
	VolumeUp(); //increase volume setting in his cell phone

Energy saving services	¬ socam:locatedIn(John, RoomA) ∨ socam:hasLightinglevel(RoomA, HIGH) OffLight(RoomA); //turn off the light

Happy dining services	∃ ?person locatedIn(?person, DiningRoom)
	PlayMusic(welcome_music); //play welcome music
	socam:hasActivity(DiningRoom, Breakfast) ∨
	socam:hasActivity(DiningRoom, Lunch) ∨ socam:hasActivity(DiningRoom, Dinner)
	PlayMusic(random); //play music randomly
	AdjustLighting(setting_1); //adjust lighting based on setting_1

4.4. Service locating service

The service locating service allows user and applications to discover contexts and locate context providers and context interpreters. We have developed the service locating service mechanism in Gu et al. (2003). The main features include scalability, dynamism, and the multiple-matching capability.

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 which it supports. 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, i.e. either using a service template or using OWL expressions to specify the kinds of contexts they provide. An application wishes to find out a context such as the location context of John will send the query '(*John socam:locatedIn ?x*)' to a Service Locating Service server. The server then loads the context ontologies stored in the database and context instances advertised by different context providers or the interpreter, and apply the semantic matching mechanism to find out which context provider provides this context. If a match is found, the reference to the context provider or the context interpreter will be returned to the application. For more details the interested reader is invited to see Gu et al. (2003).

4.5. Communication and interaction between components

For communication between various distributed components in SOCAM, we use Java RMI. The main advantage of Java RMI is interoperability between heterogeneous platforms; and it also offers a certain degree of security features. RMI allows distributed objects to invoke methods on each other even if such underlying platforms as processors, operating systems, and programming languages are heterogeneous. The requirements for implementing an RMI client or server are simply having an implementation of JVM; and it is easily met giving evidence that many pervasive devices with such capability are

available nowadays. We tailored the J2SE 1.3.1 RMI runtime classes to accompany the need of resource-constrained pervasive devices.

SOCAM components are designed as independent service components which may be distributed over heterogeneous networks and can interact with each other. Different context providers which reside in both an internal network and an external network register and advertise their services through the service locating service. The context interpreter or context-aware services are able to locate a context provider and obtain contexts. They 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 our 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 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 returned to the subscriber.

5. Performance evaluation

We have implemented the SOCAM middleware in Java using J2SE 1.3.1; and developed a prototype in a smart home environment which consists of an OSGi-compliant residential gateway (OSGI), network camera, mobile phone, laptop and various sensors, etc. The gateway was designed based on Intel Celeron 600 M CPU with 256 MB memory. It runs embedded Linux (kernel 2.4.17) operating system; and supports various wired and wireless network connections such as ADSL, Ethernet, HomePNA, WirelessLAN, Bluetooth and IEEE 1394 so that various devices such as PCs, PDAs and network cameras can be connected to the gateway.

We implemented the context interpreter using Jena2-HP's Semantic Web Toolkit (Jena 2—A Semantic Web Framework). The domain-specific ontologies in both home and vehicle have been developed in OWL. The home-domain ontology consists of 89 classes and 156 properties and the vehicle-domain ontology consists of 32 classes and 57 properties. We implemented a number of context providers such as an indoor location provider, an environmental context provider (temperature, noise level, light level, etc.), a weather information provider. Based on the SOCAM middleware, we developed a number of context-aware services in a smart home environment such as context-aware services for smart phone, home energy saving services, happy dining room services.

We evaluate our infrastructure in terms of overhead of the two-layer ontology design, context reasoning, context discovery and the results are shown in the following sections.

5.1. Overhead of the two-layer ontology design

To minimize the overhead imposed by 'binding' and 're-binding' between the generalized ontology and the different domain-specific ontologies, we adopt two-level hierarchy in our design. In each domain, the context interpreter which ran on the residential gateway loaded both the upper ontology and a domain-specific ontology and merged them together for context ontology reasoning. The operations of loading and merging involve checking the

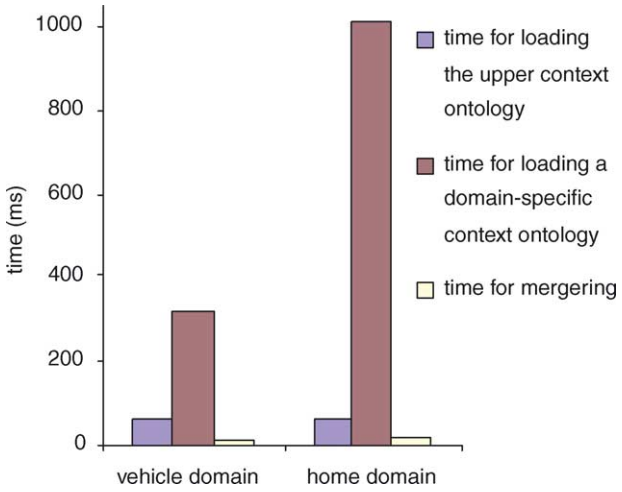

Fig. 5. Overhead of the two-layer ontology design.

ontologies for inconsistencies. We measured the loading time and the merging time for the upper ontology and domain-specific ontology in both home and vehicle scenarios. As Fig. 5 shows, the overhead—the time for loading and merging the upper ontology is low and it can be further reduced when a domain-specific ontology is extended.

5.2. Reasoning performance

In SOCAM, context reasoning is done in the context interpreter which ran on the gateway with 600 MHz CPU. In our prototype, the context interpreter takes about 521 ms to load 96 context instances from various internal context providers; and takes about 20 ms to merge these instances with the ontology. The context reasoning process takes about 1.9 s to derive high-level contexts. The context interpreter was able to answer queries for derived contexts at the average rate of a few milliseconds per query. The result shows that the logic reasoning is a computationally intensive process and it may become the bottleneck when it is applied to pervasive computing domain. However, it is acceptable for running non-time-critical context-aware applications as we mentioned in Section 4.3.

To study the reasoning performance over difference scales of context knowledge, we extended the home-domain ontology. In the experiment, we created five datasets with different sizes of classes and instances. The context interpreter ran on the gateway validated and parsed these OWL expressions into RDF triples and performed the reasoning task. For each dataset, we measured the average runtime as shown in Fig. 6. The result shows that the runtime is appropriately linear in the scale of context knowledge. Based on this observation, we are able to split the context reasoning process to improve the overall performance.

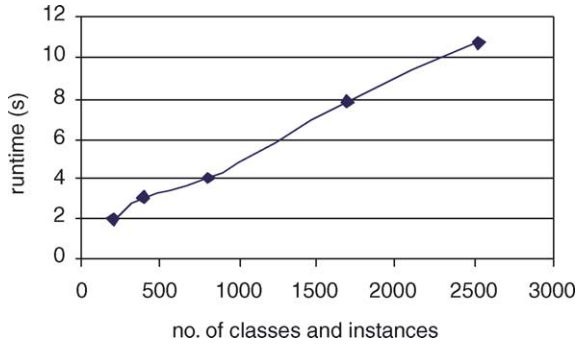

Fig. 6. The reasoning performance.

5.3. Computation split

As context reasoning is a time-consuming process, many current pervasive devices may not be able to run the context interpreter. In SOCAM, we decouple the context reasoning process and the consuming process. Context-aware services executed on a resource-constrained pervasive device only perform discovering contexts, querying contexts and subscribing a context event channel; the reasoning tasks are performed by resource-rich devices such as a residential gateway. Moreover, the separation of domain ontologies allows us to significantly reduce the context knowledge processed in each context interpreter, and hence improves the reasoning performance.

To achieve a better reasoning performance, we study the reasoning process by comparing the two kinds of reasoning which are currently supported in SOCAM-ontology reasoning and the user-defined rule-based reasoning. We measured the runtime of the two processes with respect to the different scale of context knowledge as shown in Fig. 7. The experiment result shows the time for the ontology reasoning is much more than the time for the user-defined rule-based reasoning. It is probably due to a large set of rules used in the ontology reasoning whereas the user-defined rule-based reasoning uses fewer rules.

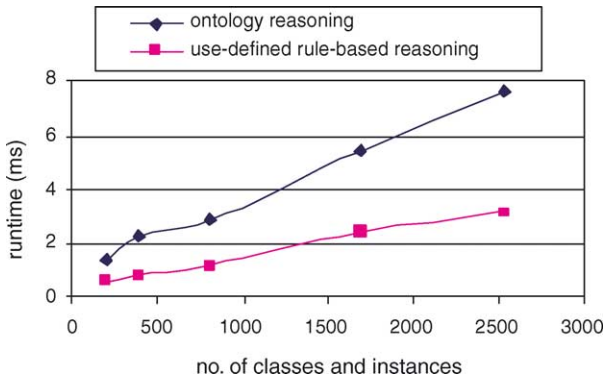

Fig. 7. Reasoning comparison.

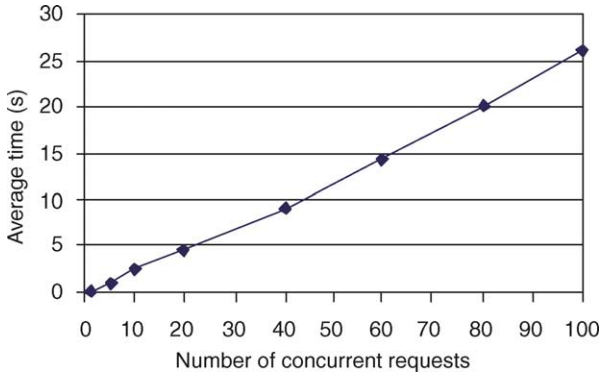

Fig. 8. Average time for concurrent requests.

As the rule set for ontology reasoning remains unchanged once defined, we are able to perform the ontology reasoning in advance of a context query.

5.4. Context discovery performance

We measured the performance for context discovery through the service locating service. We created and registered a number (100) of context providers and the interpreter in a service locating service server. Multiple mobile clients emulated using Pentium laptop with wireless LAN connectivity on separate machines made concurrent requests to a server. We measured the average time taken by our SLS system for different number of concurrent requests as shown in Fig. 8. We found that the average elapse time for each context discovery request is about 210 ms and is nearly proportional to number of concurrent requests. The context discovery mechanism performs well as number of requests increases. More performance results of the service locating service mechanism can be found in Gu et al. (2003).

6. Conclusion

In this article, we have presented a formal context model based on OWL to represent, manipulate and access context information. Based on our context model, the SOCAM middleware has been designed to support the building of context-aware services. The prototype system and evaluation results demonstrate a reasonable performance and the SOCAM middleware is able to meet the requirements of context-aware systems concerning limited memory and CPU resources in pervasive computing environments.

References

Berners-Lee T, Hendler J, Lassila O. The Semantic web.: Scientific American; 2001.

Brickley D, Guha RV. RDF Vocabulary description language 1.0: RDF Schema. World Wide Web Consortium; January 2003.

Chen G, Kotz D. A survey of context-aware mobile computing research. Technical Report TR2000-381. Dartmouth College; November 2000.

Chen H, Finin T. An ontology for a context aware pervasive computing environment IJCAI Workshop on Ontologies and Distributed Systems, Acapulco MX 2003.

Dey A, 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;2000.

Dey AK, Salber D, Abowd GD. 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. Hum–Comput Interact (HCI) J 2001;16(2-4):97–166.

Forgy CL. RETE: a fast algorithm for the many pattern/many object pattern match problem, artificial intelligence; 1982.

Gu T, Qian HC, Yao JK, Pung HK. An architecture for flexible service discovery in OCTOPUS Proceedings of the 12th International Conference on Computer Communications and Networks (ICCCN), Dallas, TX 2003.

Gu T, Wang XH, Pung HK, Zhang DQ. An ontology-based context model in intelligent environments In: Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, CA, USA 2004.

Gu T, Pung HK, Zhang DQ. A Bayesian approach for dealing with uncertain contexts. In: Proceedings of the Second International Conference on Pervasive Computing (Pervasive in the book, Advances in Pervasive Computing published by the Austrian Computer Society, vol. 176, ISBN 3-85403-176-9, Vienna, Austria 2004.

Henricksen K, Indulska J, Rakotonirainy A. Infrastructure for pervasive computing: challenges Workshop on Pervasive Computing INFORMATIK 01, Viena 2001.

Horrocks I. DAML+OIL: a Reason-able Web Ontology Language In: Proceedings of the Eighth International Conference on Extending Database Technology (EDBT), Prague 2002.

Hong JI, Landay JA. An infrastructure approach to context-aware computing In: Human–computer interaction, vol. 16 2001.

Jena 2—A Semantic Web Framework, http://www.hpl.hp.com/semweb/jena2.htm

Kindberg T, Barton J. A web-based nomadic computing system. Comput Networks (Amsterdam, Netherlands 2001;35(4):443–56.

Li L, Huang W, Gu IYH, Tian Q. Foreground object detection in changing background based on color co-occurrence statistics. Proceedings of the IEEE Workshop on Application of Computer Vision (WACV Orlando, USA, 3–4 December 2002.

OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features

Long S, Kooper R, Abowd GD, Atkeson CG. Rapid prototyping of mobile context-aware applications: the cyberguide case study Proceedings of the Second ACM International Conference on Mobile Computing and Networking (MobiCom'96), November 1996.

Ranganathan A, Campbell RH. A middleware for context-aware agents in ubiquitous computing environments In: ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 2003.

Shilit BN. A context-aware system architecture for mobile distributed computing. PhD Thesis. Department of Computer Science, Columbia University; 1995.

Smith M, Welty C, McGuinness D. Web Ontology Language (OWL) Guide, August 2003.

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

Want R, Hopper A, Falcao V, Gibbons J. The active badge location system. ACM Transactions on Information Systems 1992;10(1):91–102.

Tao Gu (http://www.comp.nus.edu.sg/~gutao) is a PhD candidate in the School of Computing at the National University of Singapore. His current research interests include pervasive computing, context-aware systems, service discovery, and peer-to-peer systems. He received his MS in Electrical and Electronics Engineering from Nanyang Technological University, Singapore. He is a member of IEEE. Contact him at gutao@comp.nus.edu.sg.

Dr Hung Keng Pung (http://www.comp.nus.edu.sg/~punghk) received his BSc in Communications Engineering and his PhD in Electronics from the University of Kent at Canterbury, UK, in 1981 and 1985, respectively. He is an associate Professor of the Department of Computer Science, National University of Singapore. He heads the Network Systems and Services Laboratory and a faculty associate of the Institute of Infocomm Research (I2R) in Singapore. His main research interest is on adaptive network systems, protocols design and networking, quality of service, and mobile commerce middleware.

Da Qing Zhang is the Head of the Context-Aware Systems Department at the Institute for Infocomm Research (I2R) in Singapore. He has been initiating and leading the effort in connected home and context-aware systems in I2R, Singapore. His research interests include pervasive computing, service-oriented computing, context-aware systems and home networking. Da Qing Zhang obtained his Ph.D. from University of Rome "La Sapienza" and University of L'Aquila, Italy in 1996.