Toward an OSGi-Based Infrastructure for Context-Aware Applications

Applications and services must adapt to changing contexts in dynamic environments. This OSGi-based infrastructure manages context-aware services reliably and securely and efficiently supports context acquisition, discovery, and reasoning. A formal, ontology-based context model enables semantic context representation, reasoning, and knowledge sharing.

Tao Gu and Hung Keng Pung
National University of Singapore

Da Qing Zhang
Institute for Infocomm Research

Emerging pervasive computing solutions provide "anytime, anywhere" computing by decoupling users from devices and viewing applications as entities that perform tasks on users' behalf.[1] Context-aware systems provide mechanisms for developing applications that are aware of their contexts and able to adapt to changing contexts seamlessly. A context-aware application uses an entity's context to modify its behavior to best meet the user's needs in that context. Such applications can be used in various application domains, such as smart homes. A smart home uses networked sensors, devices, and appliances to build an intelligent environment in which many features in the home are automated and where devices and services seamlessly cooperate to support household tasks. For example, in a smart-home environment, a context-aware application might alert a hospital if a person's blood pressure exceeds a certain threshold, or remind a family member to take medicine according to a doctor's e-prescription.

Context-aware computing has attracted much attention from researchers in recent years, and several context-aware systems have been developed to demonstrate the technology's usefulness. However, building context-aware applications is still complex and time-consuming due to inadequate infrastructure support.[2] In this article, we propose a context-aware infrastructure for building and rapidly prototyping such applications in a smart-home environment.

Our architecture

Before we can build a context-aware infrastructure, we must first establish a formal context model. Previous systems often presented context information as text strings or modeled it as software objects. We propose an ontology-based context model that leverages Semantic Web technology and OWL (Web Ontology Language).[3] OWL is an ontology markup language that enables context sharing and context reasoning.

Based on our context model, we also propose a service-oriented context-aware middleware (SOCAM) architecture, including a set of independent services that perform context discovery, acquisition, and interpretation. SOCAM's key feature is its ability to reason about various contexts. Through the reasoning process, high-level contexts can be derived from low-level ones, and implicit contexts from explicit ones. With these service components, we can build context-aware services easily by accessing various types of contexts with different levels of complexity.

The Open Service Gateway Initiative, or OSGi

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/04/$20.00 © 2004 IEEE

(www.osgi.org), is an emergent open architecture that lets us deploy a large array of wide-area-network services to local networks such as smart homes and automobiles. It offers the following benefits:

- Platform independence. OSGi specifications can be implemented on different types of hardware devices and operating systems because they're based on Java.
- Various levels of system security, allowing digital signing of downloaded services and object access control.
- Hosting of multiple services from different providers on a single gateway platform.
- Support for multiple home-networking technologies.

OSGi defines a lightweight framework for delivering and executing service-oriented applications. Its management functionalities include installing, activating, deactivating, updating, and removing services. Developing SOCAM on top of the service-oriented OSGi open standard can provide a robust and potentially interoperable infrastructure for building, provisioning, and managing context-aware services in smart homes and beyond.

Context modeling

In the artificial intelligence literature, an *ontology* is a formal, explicit description of concepts in a particular domain of discourse. It provides a vocabulary for representing domain knowledge and for describing specific situations in a domain. An ontology-based approach for context modeling lets us describe contexts semantically and share common understanding of the structure of contexts among users, devices, and services. This model's main benefit is that it enables a formal analysis of the domain knowledge, such as performing context reasoning using first-order logic.

We use OWL to describe our context ontologies because it's more expressive than other ontology languages (see the "Related Work" sidebar for information on other research).[4] For example, OWL supports cardinality constraints whereas RDF (Resource Description Framework) Schema does not. OWL has three sublanguages:[3]

- OWL Lite, designed for easy implementation, gives users a functional subset that gets them started using OWL.
- OWL Full contains all the OWL language constructs and provides a free and unconstrained use of RDF constructs.
- OWL DL (Description Logic) supports the same set of OWL language constructs as in OWL Full but places a number of constraints on their use.

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

- **subject** $\in S^*$: the set of subject names (for example, a person, location, or object)
- **Predicate** $\in V^*$: the set of predicate names (for example, is located in or has status)
- **value** $\in O^*$: the set of all values of subjects in S^* (for example, **living room**, **open**, **close**, or **empty**)

To illustrate,

- **Location(John, bathroom)** means that John is located in the bathroom.
- **Temperature(kitchen, 120)** means that the temperature in the kitchen is 120° F.
- **Status(door, open)** means that the door is open.

We can extend the basic model to form a complex context or a set of contexts by combining the predicate and Boolean algebra (union, intersection, and complement). For example, we can represent three family members' food preferences as

> The OSGi open architecture lets us deploy a large array of wide-area-network services to local networks such as smart homes and automobiles.

FoodPreference(familyMembers, foodItems) $\rightarrow$
FoodPreference(John, FoodList_1) $\vee$
FoodPreference(Alice, FoodList_2) $\vee$
FoodPreference(Tom, FoodList_3).

An ontology represents our context model's structure. The ontology is described in OWL as a collection of RDF triples, each statement being in the form of **(subject, predicate, object)**, where **subject** and **object** are the ontology's objects or individuals and **predicate** is a property relation defined by the ontology.

There's a great variety of context information. *Context* includes any information that can be used to characterize the situation of an entity, which can be a person, place, or physical or computational object (such as a person's name, role, current location, or a room's temperature) in any domain.[5] Because it's difficult to centrally manage and process a large amount of context knowledge in pervasive computing environments, we adopted a hierarchical approach, dividing our context ontologies into a common high-level ontology and domain-specific ontologies. The high-level ontology captures general information about the physical world in pervasive computing environments. Domain-specific ontologies define the

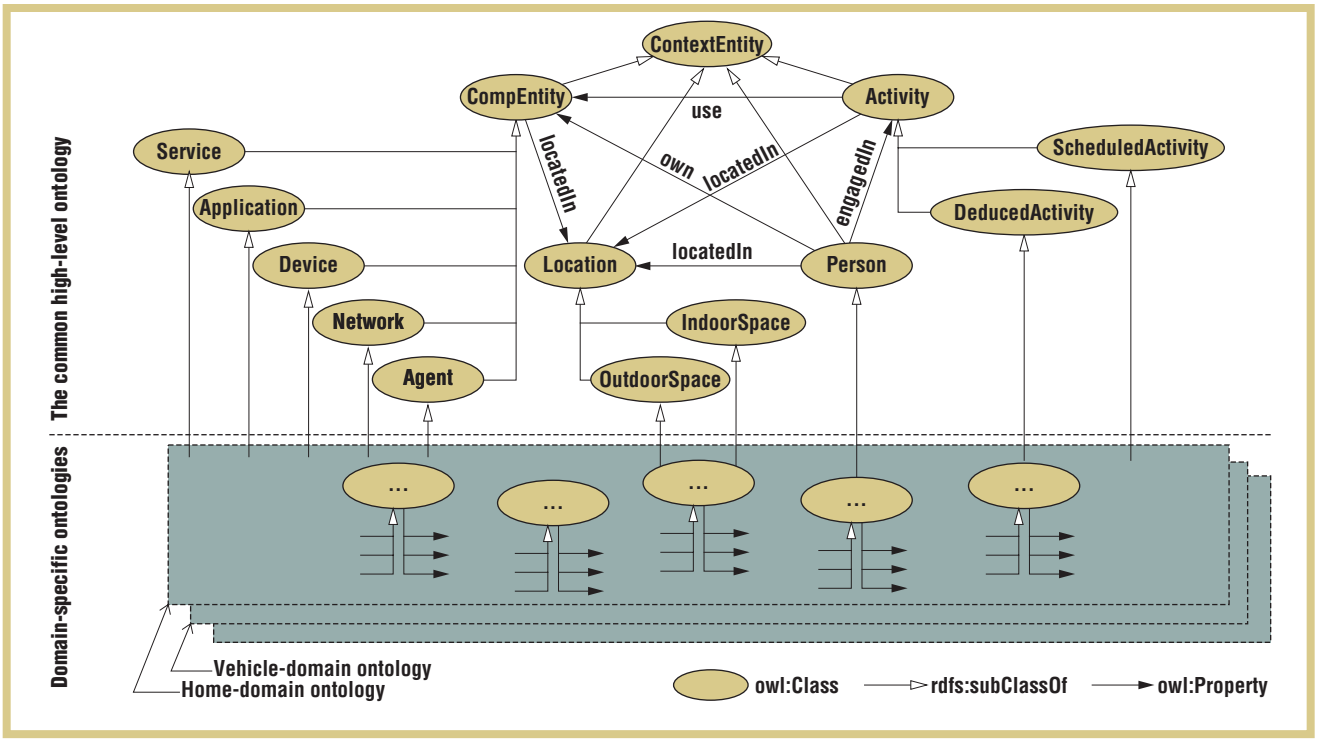

Figure 1. Class hierarchy diagram for our context ontologies.

details of general concepts and their properties in each subdomain, such as the home, office, or vehicle subdomain. Separating the domains significantly reduces the amount of context knowledge and relieves the burden of context processing.

The high-level ontology defines the basic concepts of person, location, computational entity, and activity, as Figure 1 shows. The class ContextEntity provides an entry point of reference for declaring the upper ontology. One instance of ContextEntity exists for each distinct user, agent, or service. Each instance of ContextEntity presents a set of descendant classes: Person, Location, CompEntity, and Activity. The details of these basic concepts are defined in the domain-specific ontologies, which can vary from one domain to another. We have defined all the descendant classes of these basic classes, and an associated set of properties and relationships, for a smart-home environment.[6]

The SOCAM architecture

We designed SOCAM, which is based on our context model, to enable rapid prototyping of context-aware applications (see Figure 2). SOCAM converts various physical spaces where contexts are acquired into a semantic space where context-aware applications can share and access them easily. It consists of the following components:

- *Context providers* abstract useful contexts from heterogeneous sources, external or internal, and convert them to OWL representations so that other service components can share and reuse them.
- The *context interpreter* provides logic reasoning services to process context information.
- The *context database* stores context ontologies and past contexts for a particular subdomain. Each domain has one logic context database.
- *Context-aware applications* use different levels of contexts and adapt the way they behave according to the current context.
- The *service-locating service* (SLS) provides a mechanism so that context providers and the context interpreter can advertise their presence and users and applications can locate these services.

SOCAM components are designed as independent service components that can be distributed over heterogeneous networks and can interact with each other. All SOCAM components are implemented in Java, so they are independent of underlying system platforms. For communication between various distributed components in SOCAM, we use Java RMI, which lets distributed objects invoke each other's methods. The requirement for running a SOCAM component is simply to have a Java Virtual Machine implementation. Because many pervasive devices are becoming Java enabled, our SOCAM middleware should have fewer deployment issues.

Context providers

Context providers provide context abstraction to separate low-level sensing

implementations from high-level context manipulation. Each context provider registers with the SLS in order to be discovered by others.

External context providers obtain contexts from external sources—for example, a weather information server might provide weather information on a particular place, or a location server might provide a person's outdoor location information. Internal context providers acquire contexts directly from ubiquitous sensors located in a subdomain (such as radio frequency identification-based location sensors in a smart home).

We deployed various sensors in our prototype system, including location sensors, lighting sensors, microphones, and video cameras. The advancement of video analysis technology gives us an opportunity to explore complex contexts such as human behaviors. We leveraged our video surveillance system to perform object tracking and human behavior analysis so as to provide a person's behavior and posture context. The system can collect raw video data and transform them into high-level descriptions of events. For example,

```
Event: "A person lies down on the bed"
Event: "A person falls down"
```

We've designed a set of procedures and APIs to support both context query and context event subscription. Users and services can either issue a query for a particular context of interest or subscribe to a context event from a context provider. When an event is triggered, the system returns the particular context to the subscriber in the form of OWL expressions, as the following example shows:

```
<socam:Person rdf:about="John">
   <socam:hasPosture rdf:resource=
     "http://lucan.ddns.comp.nus.edu.sg/
     octopus/posture#LIEDOWN"/>
</socam:Person>
```

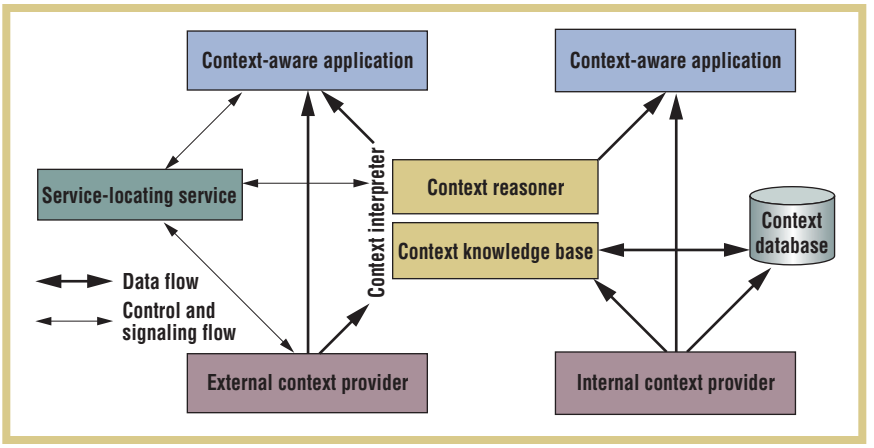

Figure 2. Overview of the SOCAM architecture.

Context interpreter

The context interpreter is also a context provider, as it provides high-level contexts by interpreting low-level contexts. It consists of a context reasoner and a context knowledge base.

The context reasoner provides deduced contexts based on direct contexts, resolves context conflicts, and maintains knowledge base consistency. To support various kinds of reasoning tasks, we can specify different inference rules, preload them into the appropriate logic reasoners, and then incorporate the reasoners into the context interpreter.

The context knowledge base provides a set of APIs for other service components to query, add, delete, or modify context knowledge. The knowledge base contains

- TBox information, which is the context ontology in a subdomain (that is, a smart home)
- ABox information, which consists of the instances of the context ontology
- Defined context instances, which users prespecify
- Sensed context instances, which are directly acquired from various context providers

We preload the first three items into the context knowledge base during system initiation; the system loads the fourth during runtime. To ensure freshness in the context knowledge base, we deploy an event-triggering mechanism to enable the system to update particular context instances. Different information requires different updating frequency. For example, defined context instances might require updating every month or year, whereas sensed context instances might need more frequent updating due to the highly dynamic nature of sensed data.

We adopted a rule-based approach based on first-order logic for reasoning about contexts. SOCAM currently supports two kinds of reasoning: ontology reasoning and user-defined rule-based reasoning.

Ontology reasoning. Our ontology's reasoning mechanism supports RDF Schema and OWL Lite. The reasoner supports all the RDF Schema specifications described by the RDF Core Working Group. The OWL reasoning system supports constructs for describing properties and classes, including relationships between classes (for example, disjointness); cardinality (for example, exactly one); equality; characteristics of properties (for example, symmetry); and enumerated classes. The RDF Schema rule sets are needed to perform RDF Schema reasoning—for example,

```
(?a rdfs:subClassOf ?b), (?b rdfs:subClassOf ?c) →
(?a rdfs:subClassOf ?c)
```

(?user rdf:type Elderly) ∧ (?user locatedIn Bedroom) ∧ (?user hasPosture LieDown) ∧ (Bedroom doorStatus Closed) ∧ (Bedroom lightLevel Low)
→ (?user status Sleeping)
(?user rdf:type Elderly) ∧ (?user locatedIn ?room) ∧ (TV locatedIn ?room) ∧ (TV status On) → (?user status WatchingTV)
(?user rdf:type Person) ∧ (?user locatedIn Bathroom) ∧ (WaterHeater status On) ∧ (Bathroom doorStatus Closed) → (?user status Showering)

Figure 3. Examples of a partial rule set based on the forward-chaining rule engine.

User-defined rule-based reasoning. This facility provides forward chaining, backward chaining, and a hybrid execution model. The forward-chaining rule engine is based on the standard Rete algorithm.[7] The backward-chaining rule engine uses a logic-programming engine similar to Prolog engines. A hybrid execution mode performs reasoning by combining both forward and backward chaining. The examples in Figure 3 illustrate a partial rule set based on the forward-chaining rule engine.

Context-aware services

These applications use different levels of context information, adapting the way they behave according to the current context. By querying the service registry provided by the SLS, context-aware services can locate all the context providers that provide interested contexts. To obtain contexts, a context-aware application either queries a context provider or listens for the events sent by a context provider.

A common way of developing context-aware applications is to specify actions in response to context changes with respect to the corresponding conditions and decision rules. In SOCAM, application developers can predefine rules and specify the methods to be invoked when a condition becomes true. All the rules are saved in a file and preloaded into the context reasoner. Developers can also modify the rule file and reload it during runtime. Table 1 gives three examples of context-aware behaviors specified by a set of rules.

Service-locating service

The SLS lets users, agents, and applications locate different context providers. The main features of the SLS mechanism include scalability, adaptability, and multiple matching capabilities.[8]

The SLS supports wide-area discovery of context providers, which might be located in internal or external networks. It tracks and adapts to changes that have been induced by adding or removing physical sensors or reconfiguring contexts. It also deploys a multiple matching mechanism that enables context providers to advertise their supporting contexts in different forms. A context provider can use a service template or OWL expressions to specify what kind of contexts it provides. An application that wishes to determine a context—for example, John's location context—will send the query **(locatedIn John ?x)** to the SLS. This service will first load the context ontologies stored in the database and the context instances advertised by different context providers, and then apply semantic matching to find out which context provider provides this information. If it finds a match, it sends the application the reference to the context provider.

TABLE 1
Rules for describing context-aware behaviors.

Application	Rule
Healthcare	**If** (John's blood pressure exceeds the threshold) ∨ (John's heartbeat is abnormal) ∨ (socam:temperature(John, greaterThan(101F)) **Then** alert hospital emergency department
Memory aid	**If** socam:status(E-prescription, VALID) ∧ (socam:time(Local, XX:YY) matches the time indicated in the E-prescription) **Then** prompt to take medicines
Energy saving	**If** (¬socam:locatedIn(John, Room) ∧ socam:hasLightinglevel(Room, HIGH) **Then** turn off the light

Infrastructure and design considerations

We are integrating SOCAM with the OSGi service platform to build a reliable, secure system that can deliver and manage context-aware services in a smart-home environment. Here, we briefly describe our OSGi-based context-aware infrastructure and design considerations.

System infrastructure overview

The system infrastructure (see Figure 4) consists of the following entities:

- *The OSGi-compliant residential gateway* enables and manages communications to and from various local networks. A Java embedded server connects both wide-area and local networks. Various types of networked devices, electronics appliances, and sensors can be attached to the residential gateway via different home networking technologies. It also hosts context-aware services that are downloaded from various context-aware service providers.
- *The Gateway operator* manages and maintains residential gateways and their services. Its functionalities

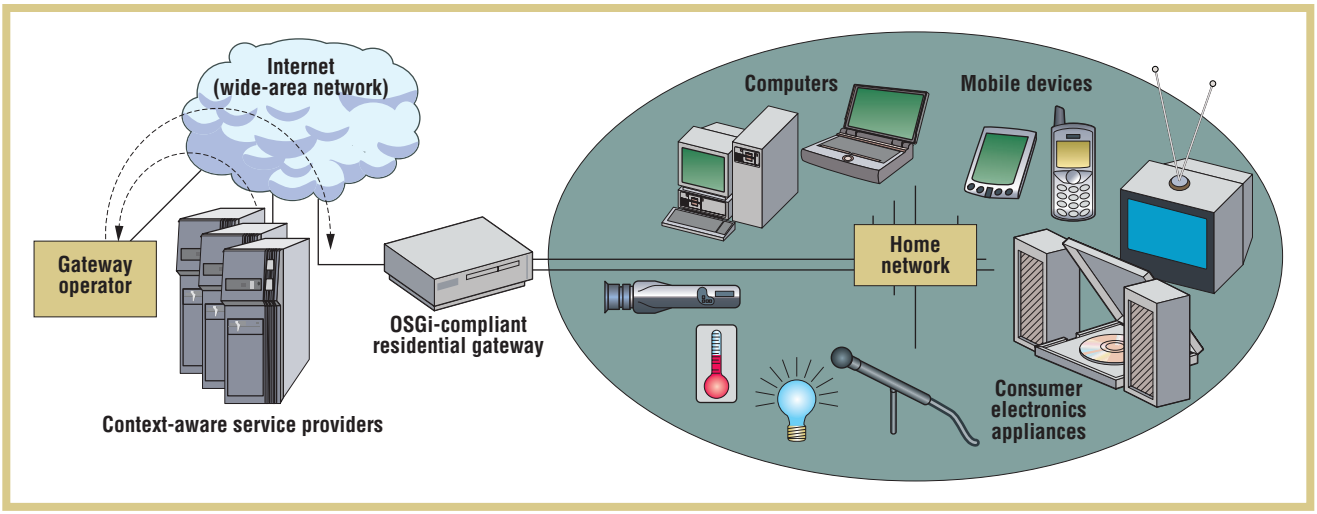

Figure 4. Overview of the OSGi-based context-aware infrastructure.

include monitoring the gateway to detect malfunctions and security attacks and installing services to and removing services from the gateway. It also serves as a service aggregator: it gathers all the services that are allowed to be loaded into the gateway and ensures that they are mutually compatible and their resource requirements don't conflict.

- *Context-aware service providers* offer a set of context-aware services to home users. Developers design a context-aware application as a set of services and combine them into bundles that can be downloaded to the gateway.

The OSGi service platform

The OSGi service platform refers to the software stack embedded in the OSGi-compliant residential gateway (see Figure 5). It provides a service-hosting environment as well as a set of common APIs to develop application bundles. It specifies several basic services such as configuration management, user management, permission administration, device management, and HTTP service. The SOCAM middleware components are built on top of the OSGi framework to facilitate context acquisition, discovery, and interpretation. Each SOCAM component can be constructed as an independent bundle—that is, an SLS bundle, a context interpreter bundle, and a context provider bundle.

The OSGi framework adopts the Java 2 security model. Many methods defined by the APIs require the caller to have certain permissions; services may also have specific permissions that provide finer-grained control over the operations they can perform. Each OSGi bundle is associated with a `Permission` class. When a permission must be verified, the checker creates an instance of the appropriate `Permission` class and calls the `SecurityManager` with this object. The gateway operator must configure the gateway's system policy to grant the required permissions to bundles that are to be installed and run in the gateway.[9]

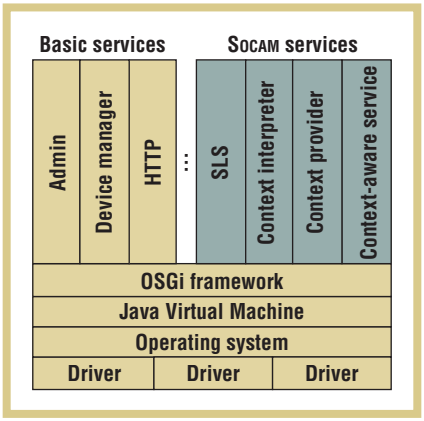

Figure 5. Software architecture for an OSGi-compliant residential gateway.

Experiments and performance study

We identified the context interpreter as a potential performance bottleneck, so we evaluated its feasibility and runtime performance (running on our residential gateway with a 600-Mbyte processor and 256 Mbytes of memory). We created a home-domain ontology consisting of 197 classes and instances. The context interpreter validated and parsed these OWL expressions into RDF triples and performed reasoning. It derived the right context based on the user-defined rule set. On average, it took about 1.7 seconds to load and merge different OWL files containing the context ontology and different context instances. The average runtime for the reasoning process was about 1.9 seconds, and the memory consumption was about 16.6 Mbytes. The interpreter was able to answer queries for derived contexts at an average rate of a few milliseconds per query.

We also conducted further performance evaluations.[10] Our results show that logic reasoning is a computationally intensive process and that it can become a bottleneck when applied to pervasive computing. However, it's acceptable for running non-hard-real-time context-aware applications in the prototype environment. The context interpreter per-

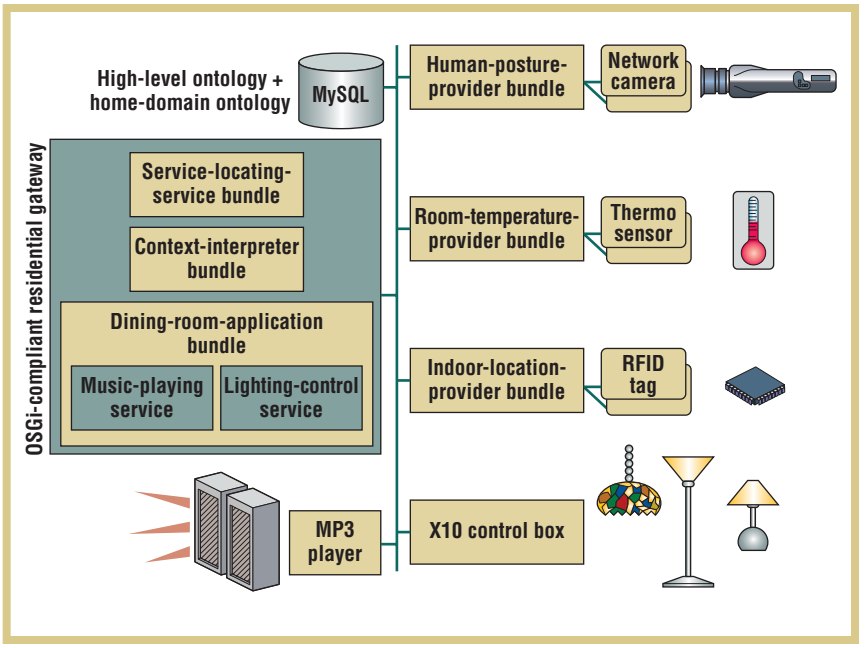

Figure 6. A dining room application.

```
∃ ?person locatedIn(?person, DiningRoom)
PlayMusic(welcome_music);
socam:hasActivity(DiningRoom, Breakfast) ∨ socam:hasActivity(DiningRoom, Lunch) ∨
   socam:hasActivity(DiningRoom, Dinner)
PlayMusic();
AdjustLighting(setting_1);
socam:hasActivity(DiningRoom, BirthdayParty)
PlayMusic(birthday_song);
AdjustLighting(setting_2);
```

Figure 7. The context configuration file for our dining room application.

formed fairly well when reasoning over small-scale context knowledge. By tailoring the high-level context ontology and domain-specific ontologies in our context model, we can control the total number of context classes and instances used in a domain. By decoupling the context-consuming process from the reasoning process, we can execute context-aware services on resource constraint devices and leave computationally intensive reasoning tasks to resource-rich devices such as residential gateways.

Developing context-aware applications

We are currently developing a smart-home prototype based on our proposed context-aware infrastructure. The prototype consists of an OSGi-compliant residential gateway, an AXIS 2100 network camera, a media server, and various sensors. Our residential gateway is built on an Intel Celeron 600 MHz processor and 256 Mbytes of memory and runs the embedded Linux (kernel 2.4.17) operating system and the Prosyst mBedded server as the OSGi service platform. It supports various wired and wireless network connections—such as Asymmetric Digital Subscriber Line, Ethernet, Home Phoneline Networking Alliance, Powerline, Bluetooth, and IEEE 1394—so that PCs, PDAs, network cameras, sensors, and various other devices can connect to it.

We're also developing a simple dining room application on top of SOCAM and OSGi for demonstration purposes (see Figure 6). The application plays music and adjusts lighting conditions when family members are having dinner in the dining room.

First, we need to create a number of context providers, such as an indoor location provider that would track a person's location at room granularity using RFID tags and transponders. We store the high-level context ontology and home-domain context ontology in a database using MySQL, and we implement the context interpreter using the Jena2 Semantic Web Toolkit (www.hpl.hp.com/semweb/jena2.htm). The ontology includes user contexts (such as whether it's a person's birthday) for each family member. Different context providers and the context interpreter are registered with the SLS. An application developer can then use the SLS to discover the services that are available in a smart home, and subscribe to them if desired.

In the dining room application, we choose the indoor location provider to determine whether a family member is in the dining room, and we choose the context interpreter to derive the room activities (for example, dinner) based on the number and identity of persons in the dining room, current time, and user context. We specify a set of methods for invocation within a set of contexts. When the context becomes true, the application calls the associated method. The contexts are specified in a context configuration file as shown in Figure 7.

In this application, different context providers, the context interpreter, and the SLS are packaged as OSGi bundles and published as OSGi services. We use Prosyst's framework package (www.prosyst.com/osgi.html) for editing and managing bundles, managing users and

Related Work

Many context-aware systems built in the past few years have focused on specific applications.

In the Massachusetts Institute of Technology's AIRE (*A*gent-Based *I*ntelligent *R*eactive *E*nvironments) project (www.ai.mit.edu/projects/aire/projects.shtml#835), an intelligent room equipped with computer vision and speech recognition systems was created to experiment with different forms of natural, multimodal human-computer interaction.

Hewlett-Packard's Cooltown project proposed a Web-based system for context awareness.[1]

Sumi Helal and his colleagues developed an indoor tracking system and OSGi (Open Service Gateway Initiative) services by using location context to assist elderly persons in a smart home.[2]

Using an object-oriented approach, the ContextToolkit provides a programming toolkit and a number of reusable components to support rapid prototyping of sensor-based context-aware applications.[3] However, it doesn't provide a common context model to enable knowledge sharing and context reasoning.

Recent research has focused on providing context-aware infrastructures that are built on a well-established, pervasive, reliable, and publicly accessible set of technologies for context-aware systems. To provide context abstraction, Jason Hong and colleagues took a database-oriented approach for their Context Fabric infrastructure by defining a Context Specification Language and a set of core services.[4] However, using proprietary context languages might lead to a lack of a common model and difficulty in sharing context.

Anand Ranganathan and colleagues developed a middleware for context awareness, in which they represented context in DAML+OIL.[5]

Harry Chen and colleagues proposed an agent-oriented Context Broker Architecture infrastructure for semantic context representation, knowledge sharing, and privacy control.[6]

Our context-aware infrastructure is integrated with an OSGi platform to support service delivery and provisioning, neither of which has been addressed in the systems just listed. As in any network service and application, security and privacy issues are also crucial for a pervasive deployment of context-aware services and applications. Existing security and privacy solutions for conventional applications and services are still evolving and are nowhere near maturity; they're unlikely to work well in context-aware pervasive computing environments due to limitations in device capability and the need for sensing and accessing privacy-sensitive context information. Little work has been reported in this area. Our work so far is also restricted to adapting OSGi's existing security features for our Service-Oriented Context-Aware Middleware (SOCAM).

REFERENCES

1. T. Kindberg and J. Barton, "A Web-based Nomadic Computing System," *Computer Networks*, vol. 35, no. 4, 2001, pp. 443–456.
2. S. Helal et al., "Enabling Location-Aware Pervasive Computing Applications for the Elderly," *Proc. 1st IEEE Pervasive Computing Conf.*, IEEE CS Press, 2003, pp. 531–538.
3. A.K. Dey, D. Salber, and G.D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," special issue on context-aware computing, *Human-Computer Interaction J.*, vol. 16, nos. 2–4, 2001, pp. 97–166.
4. J.I. Hong and J.A. Landay, "An Infrastructure Approach to Context-Aware Computing," *Human-Computer Interaction*, vol. 16, nos. 2–4, 2001, pp. 287–303.
5. A. Ranganathan and R.H. Campbell, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments," *ACM/IFIP/Usenix Int'l Middleware Conf.*, Springer-Verlag, 2003.
6. H. Chen and T. Finin, "An Ontology for a Context Aware Pervasive Computing Environment," *IJCAI Workshop on Ontologies and Distributed Systems*, IJCAI, 2003; www.cs.vu.nl/~heiner/IJCAI-03/Papers/Chen.pdf.

security, and so on. In this way, home users can access these OSGi services.

We are developing several context-aware applications to demonstrate our design's feasibility. In the near future, we'll also look into other mechanisms to reason about contexts and to solve context conflicts. Probabilistic logic, high-order logic, and Bayesian networks are promising techniques. P

REFERENCES

1. K. Henricksen, J. Indulska, and A. Rakotonirainy, "Infrastructure for Pervasive Computing: Challenges," *Proc. Informatik 01: Workshop on Pervasive Computing*, Univ. Vienna, 2001, pp. 214–222; www.dstc.edu.au/m3/papers/Informatik2001.pdf.
2. G. Chen and D. Kotz, *A Survey of Context-Aware Mobile Computing Research*, tech. report TR2000-381, Dept. Computer Science, Dartmouth College, 2000.
3. M.K. Smith, C. Welty, and D.L. McGuinness, "OWL Web Ontology Language Reference," World Wide Web Consortium (W3C) recommendation, Feb. 2004; www.w3.org/TR/owl-ref.
4. H. Chen and T. Finin, "An Ontology for a Context Aware Pervasive Computing Environment," *Proc. IJCAI Workshop on Ontologies and Distributed Systems*, IJCAI, 2003; www.cs.vu.nl/~heiner/IJCAI-03/Papers/Chen.pdf.
5. G.D. Abowd et al., "Towards a Better Understanding of Context and Context-Awareness," *Proc. 1st Int'l Symp. Handheld and Ubiquitous Computing*, Springer-Verlag, 1999, pp. 304–307.
6. T. Gu et al., "An Ontology-based Context

the AUTHORS

Tao Gu is a PhD candidate in the School of Computing at the National University of Singapore and at the Institute for Infocomm Research (I2R), Singapore. His 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. He is a member of the IEEE. Contact him at gutao@comp.nus.edu.sg; www.comp.nus.edu.sg/~gutao.

Hung Keng Pung is an associate professor in the Department of Computer Science at the National University of Singapore. He also heads the Network Systems and Services Laboratory and is a faculty associate at the Institute of Infocomm Research (I2R) in Singapore. His research focuses on adaptive network systems, protocol design and networking, quality of service, and mobile-commerce middleware. He received his PhD in electronics from the University of Kent at Canterbury, UK. Contact him at dcsphk@nus.edu.sg; www.comp.nus.edu.sg/~punghk.

Da Qing Zhang heads the Context-Aware Systems Department at the Institute for Infocomm Research (I2R), Singapore, where he has been leading the effort in connected home and context-aware systems. His research interests include pervasive computing, service-oriented computing, context-aware systems, and home networking. He received his PhD from the University of Rome's La Sapienza and the University of L'Aquila, Italy. Contact him at daqing@i2r.a-star.edu.sg.

Model in Intelligent Environments," *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conf.*, Soc. for Modeling and Simulation Int'l, 2004.

7. C.L. Forgy, "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, vol. 19, no. 1, 1982, pp. 17–37.
8. T. Gu, H.K. Pung, and J.K. Yao, "Towards a Flexible Service Discovery," to appear in *J. Network and Computer Applications*, 2004.
9. L. Gong, "A Software Architecture for Open Service Gateways," *IEEE Internet Computing*, vol. 5, no. 1, 2001, pp. 64–70.
10. T. Gu, H.K. Pung, and D.Q. Zhang, "A Service-Oriented Middleware for Building Context-Aware Services," to appear in *J. Network and Computer Applications*, 2004.