International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 20, No. 4 (2006) 585–607

World Scientific
www.worldscientific.com

SUPPORTING DEVELOPMENT OF CONTEXT-AWARE APPLICATIONS USING SEMANTIC SPACE TOOLKIT

DAQING ZHANG*

Context-Aware Systems Department
Institute for Infocomm Research, Singapore
**daqing@i2r.a-star.edu.sg*

ZHIWEN YU

School of Computer Science
Northwestern Polytechnical University, P. R. China
zhiwenyu@nwpu.edu.cn

XIAOHANG WANG

School of Computing, National University of Singapore, Singapore
xwang@nus.edu.sg

MATTHEW Y. MA

IPVALUE Management, Inc., USA
mattma@ieee.org

In order to facilitate rapid development of context-aware applications, there is a need for architectural support in the entire context processing flow, and improved programming abstractions that ease the prototyping. In this paper, a toolkit called Semantic Space, is proposed to support rapid prototyping of context-aware applications via a set of programming abstractions on context processing. The functionality encapsulated in the toolkit handles the common, time-consuming and low-level details in context acquisition, aggregation, storage and inference. Architectural design and implementation issues of the Semantic Space toolkit are discussed in detail. Finally, a case study on building a mobile situation aware phone is described to illustrate the validity of our approach and usability of the toolkit.

Keywords: Context-awareness; ontology-based context modeling; context middleware; context toolkit; mobile applications.

1. Introduction

With the recent convergence of ubiquitous computing and context-aware computing, there has been a considerable rise in interest in context-aware applications. These applications exploit various contexts to offer services, present information, tailor application behavior and trigger adaptation.

However, due to the lack of generic mechanisms for supporting context-awareness, context-aware applications remain difficult to build. Developers must often deal with a wide range of details related to the processing of context, which include context representation, aggregation, storage, reasoning and query.[12] These tedious tasks are of much distraction for developers to concentrate on building an application's functionalities. Therefore, there is a need for improving programming abstractions that can ease prototyping of applications. To allow developers build ubiquitous computing applications in a simple and efficient manner, we need to provide system-level support for context representation, acquisition, inference, delivery and usage. Particularly, we envision a simple and systematic application development process in using a context toolkit.

This paper attempts to build a new context toolkit, called Semantic Space,[24] for the rapid prototyping of context-aware applications. The Semantic Space is an architecture that hides the low-level context processing details from the user. It offers high-level programming abstractions that provide application designers with a flexible mechanism to build sensor wrappers and context-aware applications. One of Semantic Space's novelties is the utilization of Semantic Web[2] technologies for explicit representation, expressive querying and flexible reasoning of contexts in smart spaces. Via the use of Semantic Web technologies, we present a novel modeling approach that allows context to be explicitly described. In order to achieve interoperability between heterogeneous data sources and applications, Web Ontology Language (OWL[17]) is adopted as the representation language. With Semantic Space, application designers can concentrate on the development of application logic, whereas only minimal effort is required to access and manipulate context.

The rest of this paper is organized as follows. In Sec. 2, we describe related work in context toolkit and highlight the distinctive aspects of our approach. In Sec. 3, the design and implementation details of Semantic Space are presented following the requirement analysis of context toolkit. Section 4 proposes a four-step development process for context-aware applications leveraging Semantic Space. A context-aware mobile application, called *SituAwarePhone*, is presented as a case study to illustrate the development process. Finally, Sec. 5 concludes our paper.

2. Related Work

Quite a number of systems have been developed in the past to provide generic architectural support for efficient development and deployment of context-aware applications. In a broad sense, those systems fall into two categories: context processing oriented and service-oriented architecture.

Context processing oriented systems focus more on the context processing process and mechanisms. They usually provide programming abstractions that separate the different context processing stages. For instance, the Context Toolkit[6] provides application developers a set of programming abstractions that separate context

acquisition from actual context processing and usage, it uses the simple name-value pair to model context. The Solar system[5] also adopts the simple name value pair as the context model, but it develops a graph-based programming abstraction for context aggregation and dissemination. Both Gaia[20] and CoBrA[4] use Ontology based approach to model context, they can thus leverage on the techniques in Semantic Web to process context. The difference of the two systems is that Gaia adopts a middleware architecture to manage ubiquitous resources whereas CoBrA proposes an agent based architecture to support context-aware applications. The European Smart-Its project presents a programming abstraction for raw data sensing, feature extraction (cues), and high-level context abstraction from cues.[8] Harter *et al.* proposed a three-tier context architecture,[11] which consists of location context sensing, an Oracle database for context storage, update and query. Although all the abovementioned systems provide certain level of context processing support, none of them provides a complete solution for dynamic discovery of context sources, efficient context inference, expressive context query and scalability.

Service-oriented context-aware systems emphasize mainly on separating context processing and usage. They usually use service as the abstraction unit to represent each functional module and facilitate development of context-processing service and context-aware applications. For example, the Context Fabric[12] provides two fundamental built-in services, namely event service and query service, to support the acquisition and retrieval of context data. Judd and SteenKiste[13] introduced a Contextual Information Service (CIS) that enables the dynamic composition of context query results via a virtual database. Grimm *et al.* presented a system architecture with discovery and migration services, called one.world,[9] which provides an integrated and comprehensive framework for building pervasive applications that can be adapted to context change. Some other interesting context-aware systems and context-processing methods can be found in Refs. 1, 7, 15, 16 and 22.

Our work falls into the category of context processing oriented systems and it overcomes the drawbacks in previous work in several aspects. Firstly, by adopting the Semantic Web technologies for context representation, aggregation, inference and query, we developed a toolkit with a generic mechanism for querying contexts using a declarative language and inferring higher-level contexts based on predefined rules or opportunistic events.[21] This facilitates developers' work because they can realize expressive context querying and flexible context reasoning without programming. Unlike Gaia[20] and CoBrA[4] which are also leveraging the Semantic Web, we support automatic discovery of new context sources in the context toolkit and adopt an open-standard based service framework, which makes the toolkit and context-aware application development flexible and scalable. Secondly, while existing work does not elaborate on the detailed design process of context-aware applications, we propose a four-step development process for building context-aware applications leveraging Semantic Space, and use a concrete example to illustrate the usability of our context-aware toolkit for building a mobile application prototype.

3. Semantic Space Toolkit

3.1. *Toolkit requirement*

Context toolkit is designed to provide application developers with mechanisms supporting context-awareness. By analyzing the context processing flow and application developer's needs, we establish a generic set of architecture requirements for supporting context-aware application development. These requirements include:

- *R1: Context Capture* — The first step to use context is to capture context from the contextual environment. This requires the support of a wide variety of context sensed from both physical and virtual worlds. Physical context includes anything that can be sensed by hardware devices such as Radio Frequency (RF) devices or environment sensors. Virtual context refers to context obtained through the use of software components, for example, monitoring keyboard activity, device status or processor load.
- *R2: Explicit Representation* — Raw context obtained from disparity sources comes in heterogeneous formats, which cannot be used by applications without prior knowledge of its representation. Therefore, to support interoperability, context's meanings (or semantics) need to be explicitly represented so that independently developed applications can easily understand them.
- *R3: Context Inference* — For applications to successfully utilize context in a meaningful way, inference of higher-level context is required. Higher-level context (e.g. What is the user doing? What is the activity in the room?) augments context-aware applications by providing summarized descriptions about a user's state and surroundings. As sensor devices cannot directly recognize such context, the context toolkit should provide a support for applications to infer this information from basic sensed context.
- *R4: Expressive Query* — Whereas the contextual environments maintain a large amount of context, a particular context-aware application may only need to selectively access a subset of the context. The context toolkit should be able to answer expressive queries that can well specify application's context need — for example, "Who is in the room with the user?", "When will the meeting the user is attending end?". A query mechanism involves application developers defining their context need using declarative query specification.
- *R5: Continuous Delivery* — In a highly dynamic ubiquitous computing environment, the delivery of context based on request-response mode is not able to continuously feed applications with up-to-date context; application developers have to handle this problem by polling context sources in an *ad hoc* manner. To advance this matter, the context toolkit should support continuous context delivery mechanism in which an application registers query specifications, and a continuous query engine filters the query result to deliver streaming context to the application.
- *R6: Dynamic Discovery* — The dynamism of ubiquitous computing environment calls for the need to support discovery and configuration of context sources

(or their software wrappers). When a new context source joins the contextual environment, the context toolkit and applications should be able to locate and access it, and when the context source leaves the contextual environment, applications should be aware of its unavailability to avoid stale information.

- *R7: Persistent Storage* — Exploiting context requires persistently gathering useful information from the contextual environment and storing it for later retrieval. Consider a tour guide application, location of users can be persistently stored and later utilized to determine the popular sites within the city and information that is often requested at those sites.
- *R8: Programming Abstraction* — Context-aware applications must be able to utilize a wide range of computing devices, communications and sensing technologies and implement their own functionalities irrespective of the underlying toolkit. To ease the development work, a system support for context-awareness needs a set of programming abstractions to decouple enabling mechanisms from an application's functionality. These programming abstractions allow developers to implement context-aware applications and to integrate sensor components in a simple way without worrying about the low-level processing.

3.2. *Toolkit design*

The above-mentioned requirements are identified for supporting context-aware applications within ubiquitous computing environment. To address these requirements, we design a toolkit called Semantic Space.[24] Semantic Space provides a generic architecture support which allows context to be represented as semantic markups, enables applications to access context using queries, and supports the inference of higher-level context from basic context using logic rules.

Semantic Space toolkit consists of several components, as shown in Fig. 1. Context Wrappers obtain raw data from software and hardware sensors, and transform this information into semantic representation. When wrappers are instantiated, they announce their availability and publish context such that other components can select and subscribe to the wrappers with required context. Context Aggregator discovers distributed wrappers, gathers context from them and updates Context Knowledge Base (KB) asynchronously. Context KB dynamically links context into a single coherent data model, and provides interfaces for Context Reasoner and Context Query Engine to manipulate stored context. Context Reasoner is a rule-based inference engine that can infer higher-level context from stored context. Context Query Engine is responsible for handling queries about both stored context and inferred, higher-level context.

There are two basic approaches an application can use in order to retrieve context they need. Applications which make use of context with simple form can directly contact wrapper(s) using subscription-based interface. In this way, applications should specify their contextual interests in the form of triple patterns, based

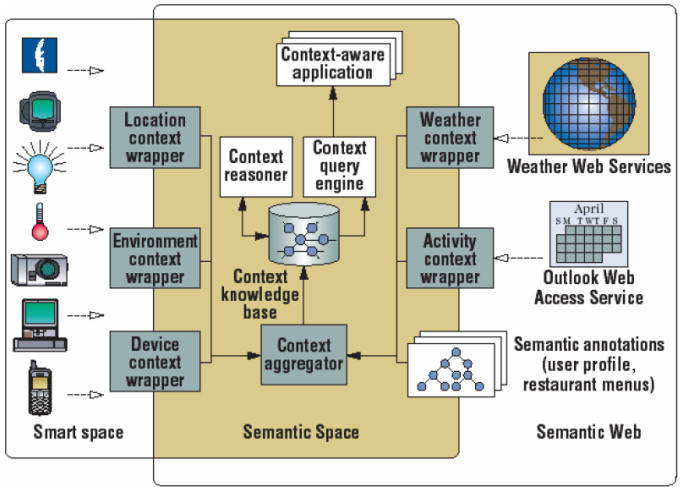


Fig. 1. Semantic space architecture.

on which pattern matching is performed to select the appropriate wrapper. Alternatively, applications with complex context needs, such as the expressive query about correlated context and inference of higher-level context, may access the functionality provided by the Semantic Space toolkit by registering inference-enabled continuous query and receiving updated query result from the Semantic Space asynchronously.

With Semantic Space, most of the context processing and management tasks are incorporated into the toolkit. It allows context-aware applications to run on thin mobile clients effectively.

3.2.1. *Context wrapper*

Context wrappers obtain raw context information from various sources such as physical devices and software programs and transform them into context markups. Physical context wrappers involve hardware sensors deployed in the smart space. They include the location context wrapper (which reports user or device location through GPS or RFID), the environment context wrapper (which gathers environmental information such as temperature, noise and light from embedded sensors), and the door status context wrapper (which reports the open or closed status of doors in each room). Software-based context wrappers include the activity context wrapper, which extracts schedule information from Microsoft's Outlook 2000; the application context wrapper, which monitors the status (idle, busy, closed) of applications such as JBuilder, Microsoft Word, and RealPlayer from their CPU usage; and the weather context wrapper, which periodically queries a Weather Web Service (www.xmethods.com) to gather local weather information.

All context wrappers are self-contained and self-configured components that support a unified interface for obtaining context from sensors and providing context markups to applications and aggregator. We implemented these wrappers as

Universal Plug and Play (UPnP)[23] services that can dynamically join a local network, advertise their presences and allowed actions, discover devices/sensors and other services. Context wrappers can publish context, and applications can register to be notified of context changes detected by the wrapper.

The use of context wrapper provides programming abstraction for context acquisition — it helps to (i) hide the specifics of physical sensors and information processing from the application developer; (ii) allow changes with minimal impact on applications; and (iii) provide reusable building blocks. Besides the abstract programming model, the role of context wrapper in transforming raw context into semantic markups is crucial in that independently developed applications are able to understand and process context based on its semantics.

Location and other spatial-temporal attributes of people or devices are important in intelligent mobile context-aware applications. Bluetooth mobile phone tracking system and RFID user tracking system can be deployed as context wrapper to provide such context.

3.2.2. *Context Aggregator*

Context Aggregator discovers context wrappers and gathers context markups from them. The need for aggregation comes partially from the distributed nature of context, as context must often be retrieved from distributed sensors via software components. Rather than each individual context-aware application having to access multiple distributed wrappers, aggregator gathers all context markups thus distributed context is made available within a single point. The use of aggregator provides an additional separation of concerns between context acquisition and actual context use. In addition, context aggregation is critical for supporting knowledge-based management and processing tasks, such as expressive query and logic inference of context.

We implemented the Context Aggregator as an *UPnP control point* that inherits the capability to discover wrappers and subscribe to context changes. Once a new wrapper is attached to the smart space, aggregator will discover it and register to published context. Whenever a wrapper detects the change of context, aggregator is notified and then asserts updated context markups into Context KB.

3.2.3. *Context Knowledge Base*

Context KB provides support for scalable storage and knowledge management of context. A Context KB stores the extended context ontology for that particular contextual environment and context markups that are either provided by users or gathered from distributed wrappers. It dynamically links context ontology and context markups into a single semantic model and provides an interface for the query engine and reasoner to manipulate correlated context.

It is important to note that Context Knowledge Base is different from a relational database that purely supports storage and query. From a knowledge

management perspective, "data" of context is low-level facts provided by input sources. When data is connected based on relations, it can answer the "who", "when", "what", "where" questions of context, for example, "where is the user", "who is in the room", "when will the activity begin". The Context Knowledge Base allows the inference of implicit information from the explicit one, for example, if the Context KB (coupled with inference engine) knows "the user is in the meeting location" and "the current time is within the meeting's scheduled interval", it can deduce that "the user is at the meeting".

Contexts in ubiquitous computing environment exhibit high degree of dynamics, so the Context Aggregator must regularly update the Context KB with up-to-date values. The scope of contexts that the Context KB manages also changes depending on the availability of wrappers. Application developers can add a new wrapper to expand the scope of context in a contextual environment or remove an existing wrapper when the contexts it provides are no longer needed. The aggregator monitors the wrappers' availability and manages the scope of contexts in the Context KB. When a Context Wrapper joins the smart space, the Context Aggregator adds the provided context to the Context KB, and when the wrapper leaves, the aggregator deletes the contexts it supplied to avoid stale information.

3.2.4. *Context Reasoner*

The Context Reasoner is responsible for inferring higher-level contexts from basic sensed contexts stored in the Context KB. Because Semantic Space explicitly represents context, existing general-purpose reasoning engines can directly process this information. This makes it easy for developers to realize application-specific inferences simply by defining heuristic rules. The use of Context Reasoner helps to separate the implementation of context inference from individual applications, thus frees application developers from writing code to perform reasoning.

A most important feature of ubiquitous computing applications is customizability. In the same smart space, different applications may define dissimilar (sometimes conflicting) rules for inferring a given type of higher-level context. Since application-specific rules may generate conflicting results, the Context Reasoner does not assert inferred contexts into the Context KB, thus avoiding conflict in the coherent model. When an application needs certain higher-level context, it submits a set of rules to the Context Reasoner, which applies them to infer higher-level context on the application's behalf, then keeps the newly inferred context in a temporary model without storing them in the Context KB. The temporary model can be accessed by the Context Query Engine to provide higher-level context to the requesting application.

Our current system applies Jena2[3] to support forward-chaining reasoning over the OWL-represented context. To perform context inference, an application developer needs to provide horn-logic rules for a particular application based on its needs. The Context Reasoner is responsible for interpreting rules, connecting to Context KB, evaluating rules against stored context, and providing interface for the query

engine to access inferred result. For example, developers can define application-specific rules to infer a user's likely situation based on the context about the user, activity and location. The following rule examines whether a given person is currently engaged in a meeting on the basis of location and schedule. If the person is in the meeting location and the current time (returned by currentDateTime()) is within the meeting's scheduled interval, he is likely to be at the meeting.

```
type(?user,User) ∧ type(?event,Meeting) ∧ location(?event,?room)
∧ locatedIn(?user,?room) ∧ startDateTime(?event,?t1)
∧ endDateTime(?event,?t2) ∧ lessThan(?t1,currentDateTime())
∧ greaterThan(?t2,currentDateTime())⇒situation(?user,AtMeeting)
```

3.2.5. *Context Query Engine*

Context Query Engine is responsible for handling expressive queries from applications and updating applications with up-to-date query result on a continuous basis. Unlike request-response paradigm where an application poses a query and a query engine generates a finite result set, Context Query Engine allows applications to register logical specifications of interest over changing context and receive streaming results asynchronously.

Furthermore, an application may seek higher-level context (e.g. user situation, room activity) that is not directly available in Context KB. In this case, Context Query Engine interfaces with Context Reasoner to derive higher-level context. For instance, an application seeking user's situation context needs to provide Context Query Engine with two parameters — the query statement specifying application's contextual interests and the logic rules defining the desired way to derive user situation. Upon request, Context Query Engine will add the application's request in its registration table. To answer the query, the engine first triggers Context Reasoner to generate the inferred result, from which it extracts user situation by query processing.

In Semantic Space, we name the above-described support as *Inference-enabled continuous query*, which provides a federated support for expressive query, continuous delivery and context inference.

3.3. *Context model*

We use ontologies to model contexts in Semantic Space. Within the domain of knowledge representation, the term ontology refers to the formal and explicit description of domain concepts, which are often conceived as a set of entities, relations, instances, functions, and axioms.[10]

In Semantic Space, we have defined the Upper Level Context Ontologies (ULCO) to provide a set of basic concepts commonly across different smart space environments. By taking an object-oriented modeling approach, we have identified three classes of physical entities (user, location, computing entity) and one class

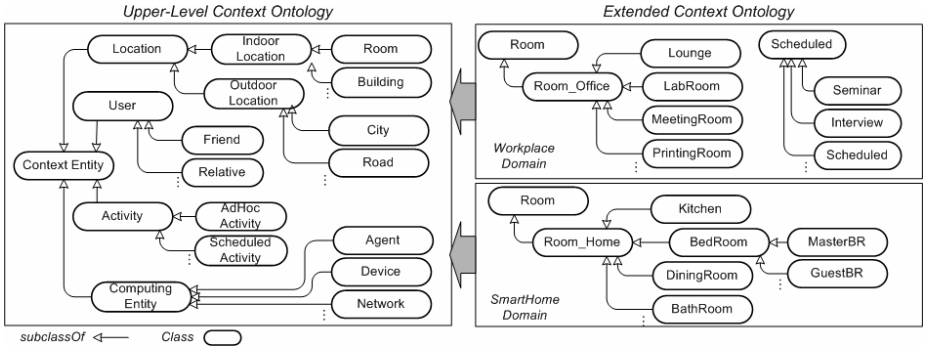


Fig. 2. Context ontology in semantic space.

of conceptual entity (activity) that play a critical role in characterizing a ubiquitous computing environment (see Fig. 2). Properties of these entities, as well as the relationships between them, form the skeleton of a general contextual environment. Furthermore, primitive contextual entities can provide indices into associated context, for example, given a location, we can acquire related context such as the temperature and noise level of it, people and activity inside it, and so on. The identification of primary contextual entities forms the basis for our ontology-based context model.

To let developers customize the context model for a particular ubiquitous computing environment, ULCO allow the definition of new concepts in terms of subclasses to complement the upper-level classes. A new application that needs additional classes can obtain them by inheriting from the ULCO classes, and forming an Extended Context Ontology, as shown in Fig. 2. This allows application developers to easily build detailed context models for a new contextual environment. Moreover, by providing shared terms and definitions for context, an ULCO supports better interoperability among extended ontologies.

We use OWL (Web Ontology Language)[17] to express ontologies of the context model. Using the context model, we represent context as ontology instances and associated properties (context markups), which applications can semantically understand and process.

Context often originates from diverse sources, leading to dissimilar approaches to generating context markups. Static context such as a person's name and scheduled seminar time have relatively slow change rates, and users often supply this information. We have developed a Web-based application that let users create online profiles based on the ontology class User defined in the context model. The following context markup describes static context about user `John`. Each OWL instance has a unique URI, and context markups can link to external definitions that are not directly included through these URIs. For example, the URI `www.i2r.a-star.edu.sg/SemanticSpace #John` refers to the user we just defined,

and the URI `www.i2r.a-star.edu.sg/SemanticSpace#Room209` refers to a specific room defined elsewhere.

```
<User rdf:about="John">
  <name>John</name>
  <mbox>john@i2r.org.sg</mbox>
  <homepage rdf:resource="www.i2r.org.sg/~john"/>
  <office rdf:resource="#Room209"/>
  <mobilePhone>6789</mobilePhone>
  <supervisorOf rdf:resource=#George"/>
  <supervisorOf rdf:resource=#Mary"/>
  <!--More properties not shown in this example-->
</User>
```

3.4. *Implementation*

The Semantic Space consists of a set of well-defined APIs for supporting the integration of sensors and the development of context-aware applications.

In the Semantic Space implementation, the classes are grouped into three categories: classes for context wrappers, classes for server architecture, and classes for context-aware applications. Components therefore are grouped into three packages: (i) `org.xyz.semanticspace.wrapper`, (ii) `org.xyz.semanticspace.server` and (iii) `org.xyz.semanticspace.application`. Among them, package wrapper consists of API for implementing software wrapper that automatically integrates context sources into Semantic Space; package `server` contains server-side components that provide essential context functionalities (aggregation, storage, inference, query, etc.) together with API used to interact with wrappers and applications; package `application` consists of APIs that enable distributed applications to access server architecture and wrappers. These components make up the programming abstraction for context-aware application development, allowing developers to think of building context-aware applications in terms of independent logical blocks.

We adopt OSGi (Open Service Gateway Initiative)[19] based service framework to implement Semantic Space toolkit. OSGi defines a lightweight framework for delivering and executing service-oriented applications. It also delineates API standards for the execution environment of services. These APIs address service life cycle management, inter-service dependencies, data management, device management, resource management and security.

The OSGi service platform is composed of two key components: service framework and service bundles. Service framework provides a service-hosting environment as well as a set of common APIs to develop application bundles. The Semantic Space components and context-aware applications were constructed as independent bundles on top of the OSGi framework. In the OSGi environment, bundles are the entities for deploying Java-based applications. A bundle is a Java Archive (JAR) file that comprises Java classes and other resources.

3.4.1. *Wrapper*

Wrappers acquire captured data from context sources, transform them into semantic markups and publish them for distributed components to access. Applications can search for particular wrappers based on their contextual interests, subscribe to the wrappers and get updated context information asynchronously. This approach can avoid explicit binding of the application to a particular underlying context sources technology.

To support explicit representation of context, wrappers need to transform sensor data into context markups based on shared ontologies. In Semantic Space, context markups are described in ontology instances, and are published in the form of triples. For example, a piece of context markup expressing the weather forecast of a city is serialized into triple format as follows:

```
(<http://...#CityZ>  <http://...#highTemperature> "36")
(<http://...#CityZ>  <http://...#lowTemperature> "28")
(<http://...#CityZ>  <http://...#weatherType> <http://...#Sunny>)
```

Other components can search for wrappers based on the matching of triple patterns. A triple pattern is in the form of (subject, predicate, object) that is comprised of named variables and Resource Description Framework (RDF) values (URIs and literals).[14] To explicitly describe a wrapper's capability, each wrapper is associated with one or more triple patterns to specify the types of provided context. These triple patterns will be used as service description in wrapper advertisement and discovery. For example, the weather wrapper can be specified by multiple triple patterns as below:

```
(?city, <http://...#highTemperature>, ?high_temp)
(?city, <http://...#lowTemperature>, ?low_temp)
(?city, <http://...#weatherType> ?weather_type)
```

Once a wrapper is started, it periodically sends advertisement message (with triple patterns) on the local network. Due to multicast and the periodic messages, applications are notified about the presence of a wrapper, followed by the process of triple pattern matching and context subscription.

The `org.xyz.semanticspace.wrapper` package provides a set of classes and abstract interface that can be used by application developers to implement a wrapper in a highly-structured, object-oriented way. The Class `Wrapper` represents the key object to construct a wrapper. A `Wrapper` object instance is associated with a set of `ContextTriple` objects specifying the provided context and an `UpdateHandler` object implementing actions for context update. Because `Wrapper` is designed as a subclass from standard Universal Plug and Play service, it automatically inherits the communication and discovery functionalities, and is therefore able to support the advertisement and removal of wrappers from the network.

For applications to access a wrapper, it first needs to be located. `org.xyz.semanticspace.application` package provides the Class `Discoverer`

which enables applications to discover context wrappers without requiring *a priori* knowledge about the wrapper's existence. This class inherits the functionality of UPnP control point to listen for the advertisement message sent by context wrappers and perform matchmaking on triple patterns to find required context triples. The discovery of context is based on context triples. There are two ways to search for context triple: a specific context triple or all context triples. Typically, an individual application uses the method `searchContextTriple()` to search for specific triples that match its context needs. Similar to the specification of context triples, application's context needs is also specified by triple pattern, type of subject and type of object. Because the server architecture needs to find all context within the contextual environment, this class provides the method `searchAllContextTriples()` for the Context Aggregator to discover all available context triples. Upon successful discovery, method `getUpdatedTripleProxy()` is called to retrieve the latest context triple provided by the identified wrapper.

3.4.2. *Semantic Space server*

The server architecture supports rich functionality, including context aggregation, persistent storage, context inference, expressive query and continuous delivery, to complement individual wrappers. It consists of four collaborating components: Context Aggregator, Context Knowledge Base, Context Reasoner and Context Query Engine.

The interface between applications and server is query based. To give the details of the support for inference-enabled continuous query, we describe the typical process for an application asking for higher-level context from the server (see Fig. 3). The example application (*SituationQueryer*) needs to be notified when the situation of *UserX* changes. It expresses this context needs as the RDQL[18] statement and a set of inference rules. One of the rules examines whether a given person is engaged in a meeting on the basis of location and meeting schedule — if a person is in the meeting location and the current time is within the meeting's scheduled interval, she is likely to be at the meeting.

The following lists the interaction between the example application and different components. (1) To utilize server functionality, the application first registers the query specification with the server architecture. (2) When a relevant wrapper (e.g. the location wrapper) has new incoming data from the sensor system, for example, "*UserX is present at the meeting location Room233*", it updates the context triple that represents this sensor data. (3) The wrapper then notifies all subscribing components (Context Aggregator and individual applications) with the updated context triple. (4) When Context Aggregator receives the location change, it submits this information to Context KB for update. (5) Once Context KB finishes the location update in its persistent storage, it notifies Context Query Engine with a KB update event. (6) Because the application needs higher-level context, query engine calls Context Reasoner to apply context inference. (7) Context Reasoner

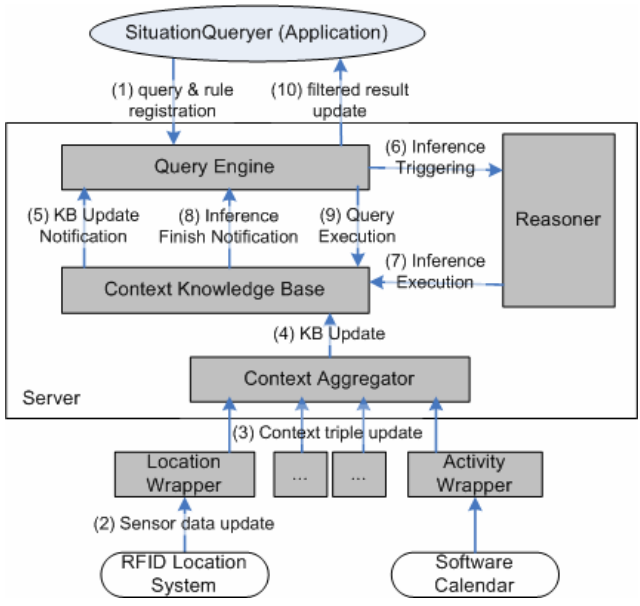


Fig. 3. Process for inference-enabled continuous query.

evaluates the application-supplied rule set to generate an inferred result, and keeps it in a temporary KB model. In this case, the situation of user *UserX* is determined to be *AtMeeting*. (8) Context Query Engine will be notified when the inference finishes. (9) It then extracts the desired context (UserX's situation) by querying the temporary KB model. It is also responsible for destructing the temporary KB model after the query finishes. (10) Finally, Context Query Engine compares the new result with the old one, and only sends the changed query result back to the application.

Semantic Space logically encapsulates underlying processes into a single architecture and provides applications with a simple interface to access server functionality. An application uses the method `submitQuery(Query query) to` register a query with server. Class `Query` is used to specify the application's context needs. Semantic Space supports two distinct query modes: synchronous mode is used to extract context from server in a query-response manner; and asynchronous mode (or inference-enabled continuous query) is used for server to push streaming results to the application on a continuous basis.

The Semantic Space server architecture provides a level of programming abstraction and flexibility to application developers; therefore, reduce the burden of developers from dealing with context. It also supports the following features: to facilitate the inference of higher-level context, to selectively access context using expressive query, and to support multiple applications accessing the server simultaneously.

3.4.3. *Application*

As described earlier, the `org.xyz.semanticspace.application` package offers complementary programming abstractions for applications to use context — triple subscription at the wrapper, and query registration at the server. Typically, the application with simple context needs may discover and subscribe to appropriate triples published by individual wrappers. If the application deals with expressive query or higher-level inference, it can contact the server architecture to utilize inference-enabled continuous query. We have implemented two set of Java APIs (wrapper access API and server access API) to support the two programming abstractions respectively, such that application programmers can prototype a context-aware application in a highly-structured way. The details are illustrated in the next section.

4. Context-Aware Mobile Application Development

This section illustrates how the context model and Semantic Space toolkit described in the previous chapters can be used to author a novel prototype context-aware application. Firstly, we will present a general process for context-aware application development using Semantic Space. Then, we will show, step-by-step, how a real context-aware mobile application, called *SituAwarePhone*, is built.

4.1. *Application development process using Semantic Space*

Given that Semantic Space is deployed, the process of developing context-aware applications is as simple as four general steps (see Fig. 4).

- **Step 1:** The development process begins by determining the collection of context that is required by the context-aware application to fulfill its functionality. If the Upper-Level Context Ontology (ULCO) in the Semantic Space Context Model is not sufficient to model all features of the required context, the application developer can extend ULCO with application-specific ontology to model the features.
- **Step 2:** Context wrappers, if any, are built for generating the required context from the various physical and software sensors. Wrapper creation comes in two stages: defining context triples and building wrapper using the provided APIs.
- **Step 3:** Queries for the required contexts are specified and connection channels are established. Triple patterns are specified for discovery and subscription

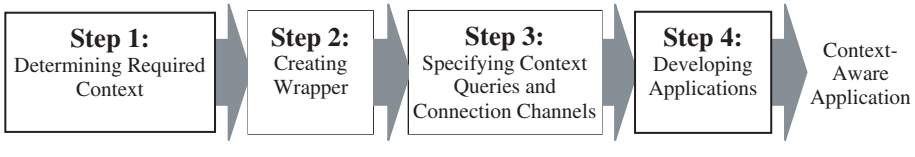


Fig. 4. Context-aware application development process using Semantic Space toolkit.

of wrappers; RDQL queries and relevant application-specific inference rules are generated for registering continuous query with Semantic Space server; Communication abstractions are provided to hide the complexity of wired or wireless links between applications and the Semantic Space.

- **Step 4:** The application logic and interaction UI are developed. Context-aware behavior is implemented, with access to context through submitting queries to the corresponding listeners.

4.2. *Prototyping SituAwarePhone*

Through the implementation details in prototyping *SituAwarePhone* application, we provide insights into the fourth-step context-aware application development process using Semantic Space, to illustrate the feasibility and usability of our toolkit.

4.2.1. *SituAwarePhone overview*

The widespread use of mobile phones makes voice communication available anytime, anywhere. However, it also raises many social problems when, for example, phones ring during meetings or important face-to-face conversations. Normally, users often have to configure the settings of mobile phones according to their circumstances to avoid inappropriate usage. Such manual configuration causes frequent interactions with mobile phone, imposing significant user distractions. To advance this matter, we developed a context-aware application, *SituAwarePhone* (Situation-Aware Phone), which automatically adapts mobile phone profiles to the changing situations. In this case, user's situation is viewed as a form of higher-level context to adapt the behavior of the mobile phone. For example, when a user is determined to be in a meeting, the mobile phone is automatically switched to silent mode and all incoming calls are diverted to voice mail.

To access the situation context, SituAwarePhone registers query with Semantic Space toolkit, which in turn notifies the phone with the changes of user's situation on a continuous basis. SituAwarePhone may also query other types of context that helps in adaptation. For example, it queries the end time of the meeting the user is engaged in to schedule a callback to the caller.

One of the key requirements of SituAwarePhone is the support for user customizability. Customizability is achieved not only by allowing the users to specify how the mobile phone should respond to the incoming call in different situations, but also by allowing them to define their own situation inference rule set for the specific contextual environment and application scenarios.

4.2.2. *Determining required context*

SituAwarePhone prototype is currently designed for use in our workplace. As a result, we need to add additional classes and properties to ULCO in order to model our workplace contextual environment. As shown in the top right of

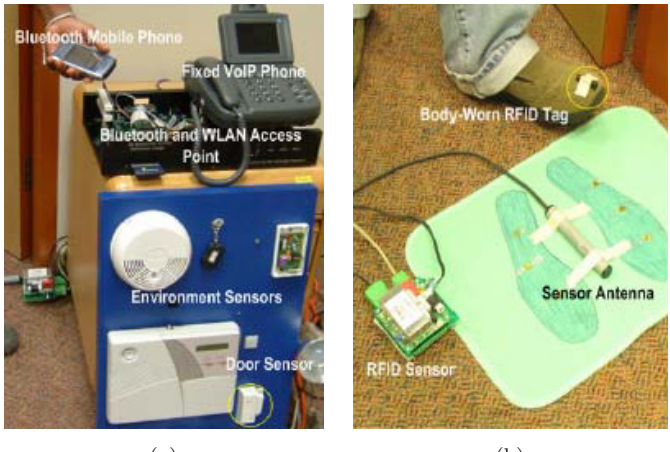
Fig. 2, the extended context ontology includes the following features. The class *Room* is classified into detailed types including *LabRoom*, *MeetingRoom*, *Lounge* and *PrintingRoom*. To describe typical activities that happen in our workplace, the abstract class *ScheduledActivity* has concrete sub-classes such as *Meeting*, *WeeklyDiscussion*, *Seminar* and *Interview*. Similarly, the class *AdHocActivity* is sub-classed by *MeetingSupervisor*, *AdHocDiscussion*, *TakingPhoneCall* and so on. To take into account the context about devices and utilities, we create sub-classes of *ComputingEntity*, such as *MobilePhone*, *FixedPhone*, *MS_PointPoint*, *MS_Word* and *Borland_JBuilder*.

The hierarchical structure of ontologies makes it easy for developers to add application-specific concepts into the context model. When the application evolves and needs more context types, we can fulfill the application's modeling requirement by extending the abstract classes in ULCO.

4.2.3. *Creating wrapper*

There are several types of sensor systems that need to be integrated into Semantic Space for providing the required context identified in Step 1. As shown in Fig. 5, the deployed sensor systems in our workplace include X-10 door sensors, noise sensors, Bluetooth mobile phone tracking system, and RFID user tracking system. Accordingly we created context wrappers which transform sensor data into context triples for Semantic Space toolkit and context-aware applications to use.

Semantic Space allows developers to use an efficient and highly-structured way to write context wrappers via its wrapper APIs. Figure 6 shows a wrapper GUI used by users to edit calendar event for providing activity-related context. The standard wrapper APIs makes it very easy to change the underlying technology used in



(a) (b)

Fig. 5. Physical deployment of SituAwarePhone: (a) networked sensors and devices; (b) RFID user tracking system.

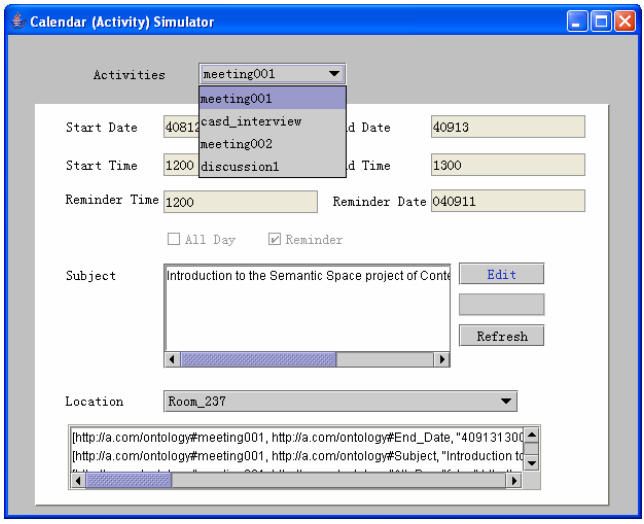


Fig. 6. GUI for supplying activity-related context.

context sensing. We were able to swap the implementation of the calendar wrapper entirely from simulated software sensor to Outlook calendar service. This ability allows us to easily evolve our systems and to prototype with a variety of sensors.

4.2.4. *Specifying context queries and connection channels*

The most important context used by SituAwarePhone is the situation of a user, which is a higher-level context that has to be inferred from other basic context obtained from sensors. In order to be notified with the changing situations, the application needs to register an inference-enabled continuous query with Context Query Engine, specifying the query statement, situation inference rule set, and temporal information.

A simple specification language for developers to create inference-enhanced continuous query is defined in the following form:

```
{CREATE query}
[TRIGGER [rule 1][rule 2]...[rule n]]
[START start]
[EVERY interval]
[EXPIRE expiration]
```

Application developers can define such queries by combining an ordinary query with the inference rule set and additional temporal annotations. The query will become effective at the time given by *start*. The parameter *interval* indicates how often the query is to be executed. If the value is zero, the inference and query will be triggered whenever Context KB is updated. Queries will be deleted from

the Context Query Engine automatically after their expiration time indicated by *expiration*. The query specification can be associated with logic rules (given by TRIGGER) for inferring higher-level context based on the application's need.

Besides continuous query, SituAwarePhone also uses simple (synchronous) queries to get other context that is useful in profile adaptation. For example, below shows the query used to get the end time of the meeting the user is currently engaged in:

```
SELECT (?endTime
WHERE (?event, <rdf:type>, <ss:Meeting>),
      (?event, <ss:hasLocation>, ?room),
      (<ss:UserX>), <ss:locatedIn>, ?room),
      (?event, <ss:start>, ?startTime),(?event, <ss:end>, ?endTime)
AND} (startTime < currentDateTime() && endTime > currentDaytime())
USING} ss FOR} <http://www.xyz.org/semanticspace{\#}>,
       rdf FOR} <http://www.w3.org/1999/02/22-rdf-syntax-ns{\#}>
```

As the SituAwarePhone connects to the Semantic Space through Bluetooth, we develop a Mobile Device Widget (MDW) within the query engine of Semantic Space and a Context-Aware Mobile Application API (CAMAPI) within mobile client, as shown in Fig. 7. The MDW allows applications residing on the mobile phone (client) to send context queries to the Semantic Space (server) using Bluetooth and receive the required contexts whenever they are updated. The MDW is also responsible for hiding the complexity of underlying communication protocols and multiple device handling. Access to the Semantic Space through a MDW by mobile applications is achieved by invoking CAMAPI on the client side. This API abstracts tasks such as initiating/terminating connections and sending/receiving queries in the mobile phone.

The CAMAPI together with the MDW in the Semantic Space provide the following tools to context-aware mobile application developers:

(1) A simplified version of the Java Bluetooth API (JSR-82) to connect to the Semantic Space infrastructure.

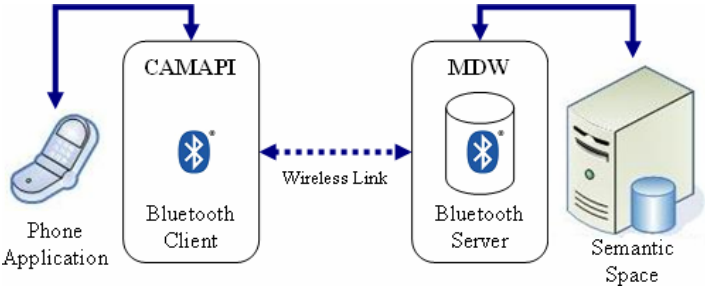


Fig. 7. Connection between mobile application and Semantic Space.

(2) A set of predefined queries for developers to obtain context information from Semantic Space.
(3) A standard query and inference rule designing process that offers application developers to design and publish their own queries for mobile applications to use.
(4) Autonomous multiple mobile device handling capability.

4.2.5. *Developing application*

We are now ready to develop the SituAwarePhone application. Besides the application logic that determines the functionalities of the SituAwarePhone, we emphasize on the acquisition of context from Semantic Space.

SituAwarePhone acquires high-level situational context, such as user's current event, and activities in room, from the Semantic Space server. The implementation of `SituationQueryer` (see Sec. 3.4.2) can be leveraged by SituAwarePhone to query and acquire the user's situation from Semantic Space. By making use of the situation context acquired from Semantic Space, SituAwarePhone enables the mobile phone to execute relevant situation-aware behavior. Mobile phone user can customize the situation-aware behavior via interactive UI. Figure 8 shows the snapshots of the prototype implementation.

The implementation amounts to approximately 800 lines of Java code, most of which deals with the MIDlet GUI and different response modes. Only about 20 lines of application-side code deal with contexts (to import libraries, issue queries, perform reasoning, parse returned models and handle exceptions), some of which are shown as follows:

```
SemanticSpace(String ServerURL) throws InstantiationException;
//Instantiate a client object of the context infrastructure
RDFModel SemanticSpace.ContextQueryEngine(String Query) throws
    QueryException;
```

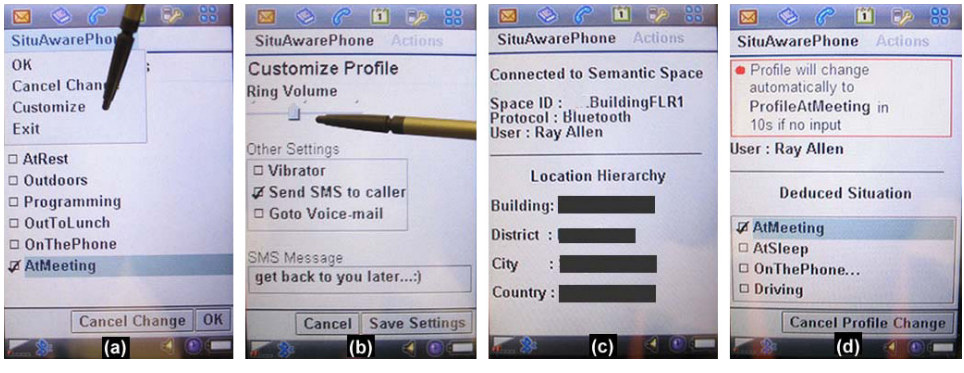


Fig. 8. Snapshots of *SituAwarePhone* running on Sony-Ericsson P900 mobile phone: (a, b) profile customization for each situation; (c) connection establishment with Semantic Space toolkit; (d) automatic profile adaptation in situation change.

```
//Query contexts from the context infrastructure using the statement defined in
Query
RDFModel SemanticSpace.ContextReasoner(String RuleSet) throws
   ReasoningException
//Request Context Reasoner to perform inference using the rules defined in
RuleSet
```

The *SituAwarePhone* application ran on a Sony Ericsson P900 smart phone. The application was developed following the Mobile Information Device Profile (MIDP) provided by J2ME. The smart phone was connected to the Semantic Space server through Bluetooth. The discovery and event notification mechanism was implemented using Siemens UPnP SDK v1.01.

5. Conclusion

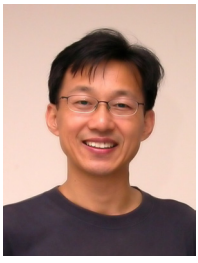
This paper proposes a toolkit approach for rapid development of context-aware applications. The contribution of this paper is twofold: first, we present a novel toolkit, called Semantic Space that leverages Semantic Web and component-based programming model. It offers a unified interface for gathering context from sensors and disseminating context to applications, which ease the development of context-aware applications. Secondly, we show how rapid development of context-aware applications can be achieved with the support of Semantic Space. Using a real context-aware mobile application, namely SituAwarePhone, we demonstrate the proposed four-step development process and the benefits offered by the Semantic Space toolkit. The Java code for dealing with Semantic Space toolkit are only about 20 lines on the client side.

Even though our context toolkit Semantic Space leverages general requirements of applications in smart spaces and supports the SituAwarePhone quite well, there is still much work to do to develop more context-aware applications using the toolkit and improve the design. We are developing different applications, e.g. healthcare, entertainment, and e-learning, to understand the real needs and improve the user interface of Semantic Space.

References

1. C. Becker, M. Handte, G. Schiele and K. Rothermel, PCOM — a component system for pervasive computing, in *Second IEEE Int. Conf. Pervasive Computing and Communications (PerCom'04)*, Orlando, Florida (14–17 March, 2004), pp. 67–76.
2. T. Berners-Lee *et al.* The semantic web, *Sci. Amer.* **284**(5) (2001) 34–43.
3. J. Carroll *et al.* Jena: implementing the semantic web recommendations, *Proc. 13th Int. Conf. World Wide Web.* New York, USA (17–22 May, 2004), pp. 74–83.
4. H. Chen, T. Finin and A. Joshi, Semantic web in the context broker architecture, *Second IEEE Int. Conf. Pervasive Computing and Communications (PerCom'04)*, Orlando, Florida (14–17 March, 2004), pp. 277–286.
5. G. Chen and D. Kotz, Design and implementation of a large-scale context fusion network, in *1st Int. Conf. Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous'04)*, Boston, Massachusetts, USA (22–26 August, 2004), pp. 246–255.

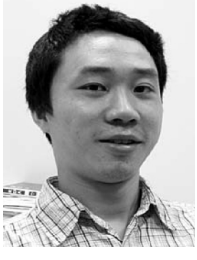
6. A. K. Dey and G. D. Abowd, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human-Computer Interaction (HCI) J.* **16**(2–4) (2001) 97–166.
7. C. Efstratiou, K. Cheverst, N. Davies and A. Friday, An architecture for the effective support of adaptive context-aware applications, in *Second Int. Conf. Mobile Data Management (MDM2001)*, Hong Kong, China (January, 2001), pp. 15–26.
8. H. Gellersen, G. Kortuem, A. Schmidt and M. Beigl, Physical prototyping with smart-its, *IEEE Pervasive Comput.* **3**(3) (2004) 12–18.
9. R. Grimm, J. Davis, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble and D. Wetherall, System support for pervasive applications, *ACM Trans. Comput. Syst.* **22**(4) (2004) 421–486.
10. T. Gruber, A translation approach to portable ontology specifications, *Knowl. Acquisition* **5**(2) (1993) 199–220.
11. A. Harter, A. Hopper, P. Steggles, A. Ward and P. Webster, The anatomy of a context-aware application, in *Proc. MobiCom1999*, Seattle, USA (August, 1999), pp. 59–68.
12. J. I. Hong and J. A. Landy, An infrastructure approach to context-aware computing, *Human-Computer Interaction (HCI) J.* **16**(2–4) (2001) 287–303.
13. G. Judd and P. Steenkiste, Providing contextual information to pervasive computing applications, in *First IEEE Int. Conf. Pervasive Computing and Communications (PerCom'03)*, Fort Worth, Texas (23–26 March, 2003), pp. 133–142.
14. G. Klyne and J. Carroll (eds.), Resource description framework (RDF): concepts and abstract syntax, W3C Recommendation (2004).
15. A. Kofod-Petersen and M. Mikalsen, Context: representation and reasoning — representing and reasoning about context in a mobile environment, *Revue d'Intelligence Artificielle* **19**(3) (2005) 479–498.
16. H. Liu and P. Singh, ConceptNet: a practical commonsense reasoning toolkit, *BT Technol. J.* **22**(4) (2004) 211–226.
17. D. L. McGuinness and F. van Harmelen, OWL web ontology language overview, W3C Recommendation (2004).
18. L. Miller, A. Seaborne and A. Reggiori, Three implementations of SquishQL, a simple RDF query language, in *Proc. 1st Int. Semantic Web Conf. (ISWC 2002)*, LNCS 2342, Sardinia, Italia (9–12 June, 2002), pp. 423–435.
19. Open Service Gateway Initiative (OSGi), http://www.osgi.org.
20. M. Roman, C. Hess, R. Cerqueira, K. Nahrsted and R. H. Campbell, Gaia: A middleware infrastructure to enable active spaces, *IEEE Pervasive Comput.* **1**(4) (2002) 74–83.
21. J. Tan, D. Zhang, X. Wang and H. Cheng, Enhancing semantic spaces with event-driven context interpretation, in *Third Int. Conf. Pervasive*, LNCS 3468, Munich, Germany (May, 2005), pp. 80–97.
22. H. Tokuda, K. Takashio, J. Nakazawa, K. Matsumiya, M. Ito and M. Saito, SF2: smart furniture for creating ubiquitous applications, in *2004 Sym. Applications and the Internet (SAINT 2004)*, Tokyo, Japan (26–30 January, 2004), pp. 423–429.
23. Universal Plug and Play (UPnP), http://www.upnp.org.
24. X. Wang, D. Zhang, J. Dong, C. Chin and S. R. Hettiarachchi, Semantic space: an infrastructure for smart space, *IEEE Pervasive Comput.* **3**(3) (2004) 32–39.



Daqing Zhang is a Principal Investigator in Context-Aware Infrastructure at the Institute for Infocomm Research (I2R) in Singapore. He obtained his Ph.D. from the University of Rome "La Sapienza" and University of L'Aquila, Italy in 1996.

Dr. Zhang was the Program Chair of First International Conference of Smart Home and Health Telematics (ICOST2003) in Paris, France. He served as the General Co-Chair of ICOST2004 (Singapore) and ICOST2005 (Canada), respectively. Dr. Zhang has been a frequently invited speaker in various international events such as Net@Home, OSGi World Congress, etc.

His research interests include pervasive computing, service-oriented computing, context-aware systems and home networking.



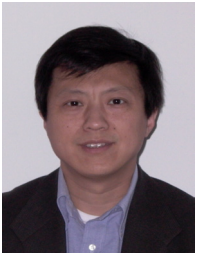
Xiaohang Wang received his B.S. in computer science from Huazhong University of Science and Technology, China. He obtained his M.S. in computer science at the National University of Singapore.

His research interests include pervasive and context-aware computing, the Semantic Web, and home networking.

Zhiwen Yu received his Ph.D. in computer science from Northwestern Polytechnical University, P. R. China in 2005.

He is a member of the IEEE. He has joined Nagoya University, Japan as a postdoctoral researcher since February 2006.

His research interests include context-aware computing, intelligent information technology, and personalization.



Matthew Y. Ma received his Ph.D. in electrical and computer engineering at Northeastern University, Boston, Massachusetts. His M.S. and B.S. degrees are both in electrical engineering from State University of New York at Buffalo and Tsinghua University, Beijing, respectively. Dr. Ma recently joined IPVALUE Management Inc. as a subject matter expert in image science. Prior to that, he worked as a Senior Scientist at Panasonic R&D Company of America for 11 years, where he managed a research group which focused on Panasonic's document and mobile imaging business. Dr. Ma has 11 granted US patents and is the author of several dozens conference and journal publications. He is also actively involved in the pattern recognition community.

He is the associate editor of the *Int. J. Pattern Recognition and Artificial Intelligence* (IJPRAI). He currently serves as Demo Program chair and program committee member of the *International Conf. of Pattern Recognition* 2006. Dr. Ma is an affiliated professor at both Northeastern University and Beijing Institute of Technology, China.

His primary research interests include image analysis, pattern recognition and natural language processing, and their applications in home networking and ambient intelligence of smart appliances.