Gateways of Physical Spaces in Context-Aware Computing

Wenwei Xue, Hung Keng Pung, Wen Long Ng, Chee Weng Tang
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{dcsxw, dcsphk, dcsnwl, dcstcw}@nus.edu.sg

Tao Gu
Institute for Infocomm Research
#21-01 Connexis, South Tower
1 Fusionopolis Way, Singapore 138632
tgu@i2r.a-star.edu.sg

Abstract—**Applications in context-aware computing continuously change their operations based on sensor data or services provided by various physical spaces in the real world, such as persons, shops and homes. We propose a gateway framework for physical spaces in this paper, which serves as a middleware platform for context-aware applications to access data from sensor networks deployed in these spaces. The gateway contains a number of components that facilitate the sensor data acquisition, processing and communication in context-aware applications. These include diversified services with plug-in management, a query processing engine with reasoning capability, and the global connection and local detection between gateways.**

Keywords-context-aware; middleware; physical spaces; physical space gateways; services

I. INTRODUCTION

Context-aware applications [17] in pervasive computing use networks of sensors installed in real-world physical spaces to obtain the dynamic context of these spaces. Such context data is utilized to drive the automatic adaptation of operations in the applications. We define a *physical space* as an entity in the physical world that provides *context* data, which can be any information used to characterize the situation of the entity [5]. Example classes of physical spaces are persons, shops, homes and offices. A physical space contains many different sensors scattered throughout the space that produce various kinds of context data for the space. As examples, a person can have body sensors to monitor *heart beat* and *blood pressure*, a shop can use video sensors to inspect its current *crowd level* and use temperature sensors for *fire alarm*.

Multiple physical spaces may employ a large number of different sensing technologies to provide a sheer diversity of exploitable context data. This heterogeneity issue makes the applications difficult to access context data from these spaces, since an application must get familiar with the specific sensor hardware and software used in every space it wants to acquire data from. To address this problem and ease the development of context-aware applications, we propose the framework of a *physical space gateway* (*PSG*) in this paper.

A PSG in our work is a logical software module that can run on any computer chosen by the administrator of a physical space. For instance, the gateway of a person space can locate at the person's PDA or laptop, and a home space gateway can be the owner's PC in the house. The PSG acts as a middleware between the physical space and the applications. It provides a single, uniform interface for applications as well as other PSGs to acquire context data from the sensor network in the space or invoke diversified services provided by the space.

We have designed a generic PSG framework for all classes of physical spaces, which is shown in Fig. 1. The framework contains a number of functional components that manage and manipulate the context data or services in a physical space. The components of a PSG can be mainly divided into the following three categories:

1) Service management. A PSG maintains different kinds of services using its service manager. The core of the services is a set of context data services encapsulating various operations of context data acquisition in the sensor network, similar to the ODBC and JDBC APIs [16]. Each service can be registered to an HTTP server at the PSG and exposed as web services for remote application invocation via SOAP [18] and WSDL [20]. The plug-in manager at a PSG allows applications to deploy and share libraries of reusable plug-in modules. A plug-in is designed to be able to work with the web services so that the processing burden of applications can be alleviated.

2) Query processing. Applications can issue declarative, SQL-based queries to a PSG. The syntax of the queries is based on a context schema kept by the schema manager that specifies all context attributes the physical space currently provides. We define a *context attribute* as a kind of context data produced by a type of sensor, such as *temperature*, *light* and *noise*. The queries are executed by a query processing engine that the PSG is equipped with. The engine employs a context reasoner to deduce high-level context attributes from low-level ones. A local context database is used by the engine to store historical context data as well as by the reasoner to store the training datasets and reasoning results.

3) Data communication. The connection manager links a PSG to a number of other PSGs to form a global Peer-to-Peer (P2P) network. The network enables the inter-gateway data communication between physical spaces and is managed by a few pre-dedicated data servers. A PSG can dynamically detect neighboring PSGs in the same local area network and establish connections with them for data exchange.

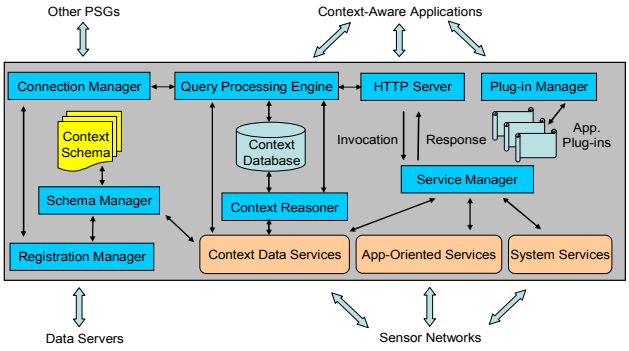

Figure 1. Framework of a physical space gateway.

Our PSG design is aimed to simplify and accelerate the development of context-aware applications in various aspects. For instance, a shopping application can invoke the virtual shopping cart service or plug-in provided by a bookstore PSG to locate the books that the application user wants to buy and to make the order. The bookstore PSG can also recommend other books to the user by querying the person's preference or by reasoning on the current purchase list. In a personalized healthcare application that runs on the PDA of a patient, the PSG on the same PDA can automatically set up a connection with the PSG of a clinic when the person enters the clinic. The clinic PSG can then issue a query to the patient PSG to obtain a historical medical record and display the record to a doctor.

The PSG framework we present in this paper is a building block of a context-aware sentient computing infrastructure [3] called *Coalition* we are developing in our research project. In addition to the PSGs, Coalition contains a data management layer consisting of several data servers to enable the effective organization and efficient context lookup over the PSGs. The infrastructure further includes a service management layer for applications to search and compose context-aware services.

The rest of the paper is organized as follows. We present technical details of the components for service management, query processing and data communication at a PSG in Sections II-IV, respectively. We evaluate the performance of our PSG in Section V. Section VI discusses the related work and Section VII concludes the paper.

II. SERVICE MANAGEMENT

A PSG is a provider for context data and services. A service can be any software program registered to the service manager with a service template. There are three categories of services at a PSG: (i) context data services, (ii) system services, (iii) application-oriented services. The first two are developed by the PSG administrator while the last by the application developers.

A. Context Data Services

We specify a number of *context data services* to encapsulate the different operations of context data acquisition at a PSG, which include sampling, filtering, aggregation, reasoning and event notification. These services are the sole methods for the PSG to acquire data from the sensor network in the physical space. They provide a high-level, service-oriented abstraction that shields the underlying details of context data acquisition to the applications as well as other PSGs.

The specification of the context data services defines the input/output behaviors rather than the implementation details of the services. It is not specific to any particular programming language. A kernel of these services must be implemented by the PSG administrator before the PSG can start to function, whereas the other parts can be optionally implemented. The implementation of a context data service replies on the sensor hardware and software used in the particular physical space. As examples, a number of context data services in the kernel are:

- *getContext(attrName)* samples and collects the current value of a context attribute named as *attrName* from the corresponding type of sensor and returns the value to an application.
- *filterContext(attrName, op, val)* returns the current value *v* of an attribute *attrName* if the value satisfies the condition *v op val*. *op* is a comparison operator such as ">", "<" and "=". *val* is a constant value.
- *subscribeEvent(e)* subscribes an application to an event *e* of interest in the physical space and notifies the application whenever the event occurs. An *event* is a deduced attribute with "true" and "false" semantics output from the context reasoner, such as *isFire* for an office and *isEating* for a person.

We have designed the context data services to incorporate many typical features of sensor data acquisition required by the context-aware applications. These include pull or push-based, one-time or continuous, blocked or unblocked acquisition. For instance, the service *getContextCont(attrName, sp, T)* pushes the real-time values of an attribute *attrName* to an application continuously every time interval of *sp* lasting a total period of *T*. The application can resume its processing immediately after the service invocation and need not to block itself to wait the entire data collection process to be completed.

Data privacy is an important issue for context-aware applications. Private context of a physical space, such as the location and friend list of a person, must only be revealed to applications that have been authorized. The optional part of our context data services includes a simple extension of the kernel to support data privacy. The extension is based on assigning a public key certificate in cryptography [14] and a corresponding private key to each authorized application. An application must first submit its public key certificate when it connects to the PSG for data acquisition. The PSG will use the public key to encrypt all data that it returns to the application. The application then uses its private key to decrypt the data before usage.

B. System Services

The administrator of a PSG may optionally develop several system services to monitor the status and health of the sensor network as well as to control and alter the operations on the sensor nodes. Examples of these system services are:

- *packetLossMonitoring()* monitors the packet loss rate in the sensor network and identifies the nodes that are responsible for heavy packet loss. These nodes should be replaced in order to enhance the performance of data transmission in the network.

- *reprogram()* installs a new piece of OS code on a subset of sensor nodes in the network.
- *scheduleTransmission()* synchronizes the time as well as fixes the data transmission and sleeping intervals on the sensor nodes to save their energy consumption.

Unlike the context data services, the set of system services provided by the PSGs can be totally different in both service specification and implementation.

C. Application-Oriented Services

A PSG can contain user-provided services that encapsulate common processing tasks the class of physical spaces involves. An example of these application-oriented services is the virtual shopping cart service at a shop PSG described in Introduction. Another example in the healthcare domain is a service at a person PSG that periodically updates the health profile of an elderly person staying at home and makes the profile available for remote real-time access from the doctor. The health profile can include various medical context data of the elderly person such as blood pressure, blood glucose, body temperature, pulse and heartbeat.

D. Service Invocation

All categories of services at a PSG provide useful tools for the applications to flexibly manipulate data acquisition from the physical space in different ways. Applications can invoke these services using an HTTP web server deployed at the PSG. The server publishes the WSDL [20] files of the services for open application access via a web interface. Based on the WSDL file of a service, an application can compose a SOAP [18] request message for service invocation and send the message to the server. The server forwards the received request to the service manager, which parses and validates the request. The manager then invokes the service. The result is replied to the application via a SOAP response message by the server.

The server essentially exposes the services at a PSG as web services for public access. If the PSG administrator does not want to expose a service of restricted access, such as a system service, the administrator can simply remove the WSDL file of the service from the server.

E. Plug-in Management

We have newly equipped our PSG framework with a plug-in manager that allows applications to store and share reusable plug-in libraries. The purpose is to enable the modularization and division of application features and to make the application development faster and easier for user maintenance. A plug-in here can be regarded as a special system service that needs to be installed in rather than be invoked by an application.

We are currently using Adobe Flex [1] to implement the plug-in manager in our PSG framework. A plug-in kept by the manager consists of a code block for a software module and an XML template that describes the module's ID, name, version, functionality and dependency on the other plug-ins. To add a plug-in from the manager, the user of an application just has to place it in the plug-in folder of the application. The application will automatically detect the new plug-in and look for its XML template to check the dependency. If there is some missing dependent module in the application, the plug-in can not be run and the application will alert the user; otherwise it will install and execute the plug-in at run time.

A plug-in can invoke the web services at the PSG via the HTTP server. This integrates the multiple system components to further facilitate the application development.

III. QUERY PROCESSING

In addition to using the services, applications can specify SQL-based queries as an alternative way to acquire context data from the physical space. In this section, we describe a number of mechanisms we design at a PSG to support the processing these context queries.

A. Context Schema

Our PSG framework represents context data using a simple attribute-value model similar to the relational database model [16]. Given this model, all context attributes available from a physical space are encapsulated into a single virtual table at the PSG, the same as prior work on sensor databases [13][22]. The context schema is the schema of this virtual table. It specifies the table name as well as the names of the context attributes in the table and their data properties, such as the types and value ranges. The schema is composed by the PSG administrator and is maintained by the schema manager. When new sensors are added into the physical space or old sensors are removed, the context schema needs to be updated to reflect the corresponding addition or deletion of context attributes if any.

B. Declarative Query Interface

The declarative context query interface for applications at a PSG supports two classes of queries: *data collection queries* and *event subscription queries*. An example query in each class is shown as follows.

```
SELECT      crowd_level   FROM   BORDERS
SUBSCRIBE   isMeeting     FROM   KEITH
```

The semantics of these SQL-based queries in the interface are self-explanatory. The expressiveness of the queries stems from the context data services and the syntax of them depends on the context schema. A PSG administrator can freely pick the name of its virtual context table and those of the attributes in the table, as we can see from the two example queries.

An optional CONT keyword is used in our query interface to specify the continuous context data acquisition [13][22]. The sample interval and the total period of the acquisition can be specified using the new SAMPLE INTERVAL and LIFETIME clauses. The following query gives an example.

```
SELECT    CONT       blood_pressure, blood_glucose
FROM                 PATIENT
SAMPLE INTERVAL   30 mins     LIFETIME   2 weeks
```

Local applications for a PSG can issue queries to a query analyzer, which is a sub-component of the query processing engine. Remote applications in the Internet can issue queries via a public web interface at the HTTP server.

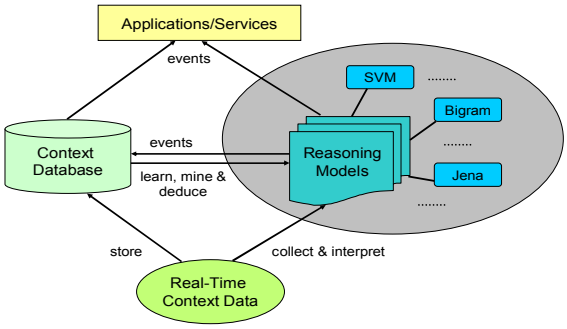

Figure 2. Context reasoning model at a PSG.

C. Query Execution

The query processing engine parses each query string from an application and examines its syntax based on the context schema. If the query has an invalid syntax, an error message is sent to the application and the query will not be further processed.

For each query with a correct syntax, the engine generates an evaluation plan for the query. The plan is similar to a relational query plan [16] and contains scan, selection, projection and aggregation operators. Each operator invokes the corresponding context data services for data collection and processing. For instance, a projection operator will invoke the *getContext()* or *subscribeEvent()* service to obtain the attribute values.

If the query is continuous, a repetitive timer is created for the query plan. The plan is executed every time the timer is fired. The length of the timer is decided by the value in the SAMPLE INTERVAL clause of the query. The query plan is terminated when the duration specified in the LIFETIME clause has passed.

A query can request the storage of historical context data within a period. In this case, the plan will periodically push the data into a query result table in the context database rather than sending it directly to the application. This result table can be subsequently queried or deleted by the application.

D. Context Reasoning

We are developing a reasoning model in the context reasoner to deduce high-level events from low-level simple attributes at a PSG. The model is depicted in Fig. 2. The reasoner implements a set of pre-defined machine learners, probabilistic models and rule-based engines as building blocks, such as Bigram, Support Vector Machines (SVM), Dynamic Bayesian Networks (DBN) [2] and Jena [9]. An event is predicted by one of these reasoning blocks or a combination of multiple given a user-provided event specification. A new block for some other statistical model can be flexibly added into the reasoner any time without affecting the correct functioning of existing blocks. An existing block can as well be removed from the reasoner suppose it is not used to deduce any event at the moment.

In the reasoner, the real-time context data required for every reasoning block is collected via the context data services. A block can further require a certain amount of historical data to be stored in the context database for offline processing in batch. If a training dataset is required for a particular block, the dataset is first sampled and stored in the context database, and then used to initially construct and incrementally update the model parameters in the block. The output events of the reasoning blocks are directly reported to the applications whenever they are detected, or are stored in special event result tables in the context database for future probing via queries.

IV. DATA COMMUNICATION

We have designed a connection manager to enable the effective data communication among PSGs. The connection manager at a PSG includes two sub-components: (i) a P2P manager, (ii) a mobile neighbor locator.

A. Connecting to Global Peers

The P2P manager connects the PSGs for the same class of physical spaces, such as those for all shops or persons, into a global P2P network. It is a component required by our Coalition infrastructure [3] for the data servers to effectively organize and efficiently search data over all physical spaces. In our current implementation, we use Gnutella [6] version 0.4 as the P2P protocol. When a PSG is registered to Coalition, its initial P2P neighbors are selected by the data servers. The registration request of a PSG contains its IP address and context schema. The request is sent by the registration manager at the PSG.

The global P2P connection enables a context query sourced from a PSG to be disseminated to and executed at remote sink PSGs. The WHERE clause of a query can specify which PSGs should process a query. For instance, the following query asks the locations of all Borders bookstores.

SELECT location FROM BOOKSTORE
WHERE name = "Borders"

The query exchange among PSGs brings the problem of how to match the context schemas of two autonomous PSGs in the same class. We have developed an adaptive and lightweight algorithm for context schema matching [21] in our Coalition infrastructure, whose details are beyond the scope of this paper.

B. Locating Mobile Neighbours

A PSG may run on a mobile device such as the PDA of a person and the mini-computer in a car. Such mobility makes the PSG dynamically join and leave multiple wireless networks. When one mobile PSG moves into the same local area network as the other, it is useful to detect, establish and maintain this dynamic neighboring relationship between the two PSGs. The healthcare application described in the Introduction illustrates the usefulness of mobile neighbor detection between PSGs.

The mobile neighbor locator enables PSGs in a local area network to automatically connect each other and transmit data. Whenever a PSG newly joins a wireless network, its mobile neighbor locator broadcasts a linkup message to the whole local network via pre-defined socket ports. The locators at all existing PSGs in the network listen to and receive the message, knowing a new LAN neighbor has appeared. A connection can then be established between the mobile PSG and any other PSG in the local network based on a mutual agreement for data and query communication in a way similar to the global P2P connection.

The dynamic neighboring relationship of a PSG can be modeled as special context attributes in the context schema and

be exposed for application utilization. For instance, the attribute *currentNeighbors* returns the IP addresses of all PSGs that have currently established a local connection with the PSG through the mobile neighbor locator.

V. PERFORMANCE EVALUATION

We present initial results of performance evaluation on our PSG framework in this section.

A. System Toolkit and Testbed

We have developed a system toolkit for our PSG framework that encapsulates the implementation of most components in Fig. 1. For the components that must be externally generated, implemented and linked, such as the services, plug-ins and context schemas, the toolkit provides a number of manuals to guide the PSG administrator how to set up these components, as well as a few examples for each component. Specifically, the example context data services we implement in the toolkit work on a network of Crossbow motes [4]. Most parts of the toolkit are implemented in Java.

The administrator of a physical space can download and install the toolkit on any PC of choice in the space to make it the PSG, as long as the PC has connection to the Internet. Fig. 3 shows a snapshot of the toolkit GUI for a PSG administrator to configure and manage the PSG. We have installed the toolkit on several computers in our research lab to form a testbed for our performance evaluation. The testbed includes five physical spaces of different classes: a home, office, shop, clinic and person. We use a laptop for the person PSG to enable mobility while we use desktop PCs for the other PSGs. Each space has a network of motes to provide real-time sensor data.

B. Response Time of Queries

We first examine the response time of a context query at a PSG, which is defined as the interval between the time when the query is issued from an application and the time when the application receives the query result. Table I shows the time breakdown of a query in our testbed. Each value in the table is the average of 30 independent experimental runs using various queries at different PSGs. This is the same for every value in the figures and tables we show in the following of this section.

In the table, "query parsing" is the time the query processing engine takes to parse and validate the query string, "in-network sensor data collection" is the time spent on getting the context attribute values from the network of motes in the space, "PSG-side post-processing" is the time the PSG spends on smoothing and packaging the collected attribute values, and "result reply" is the time required to send the query result to the application.

In the table we see that, the total response time of a query in our testbed was less than one second. The collection of context data from the sensor network was the most time-consuming task among the four. It is expected to become the bottleneck of query processing when the network size grows up. The time for PSG-side post-processing of sensor data was also considerable while those of other two tasks were negligible.

Fig. 5 shows the average response time of multiple queries issued to a PSG simultaneously. The result indicates that the response time of a query increased when the total number of concurrent queries a PSG needs to process increased. Although the increase curse is very close to linear, it is still sub-linear as implied by the response time numbers labeled in the figure.

C. Response Time of Services

We continue to evaluate the response time of one-time service invocation at a PSG, which is defined as the interval between the time when an application sends a SOAP service request to the HTTP server and the time when the application receives the response message from the server.

The time breakdown of a service invocation in our testbed is shown in Table II. In the table, "service lookup" is the time the server takes to check which service in the service manager to be invoked, "service execution" is the running time of the code block of the service, "result packing" is the time for the server to create the response message in SOAP format, and "result reply" is the time for the server to reply the message to the application. As expected, the table shows that the service execution time dominated while those of other three tasks were relatively tiny. The total time for service invocation was less than one second as that for query processing.

TABLE I. TIME BREAKDOWN OF A CONTEXT QUERY

Item	Time (Milliseconds)
Query Parsing	15
In-Network Sensor Data Collection	442
PSG-Side Post-Processing	370
Result Reply	16
Total	843

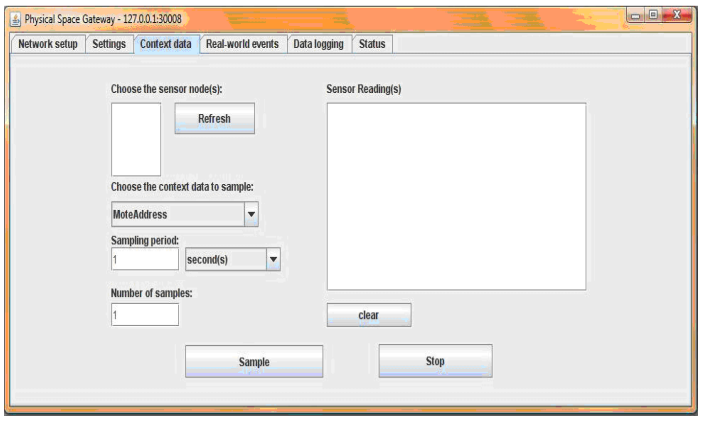

Figure 3. GUI for the PSG toolkit.

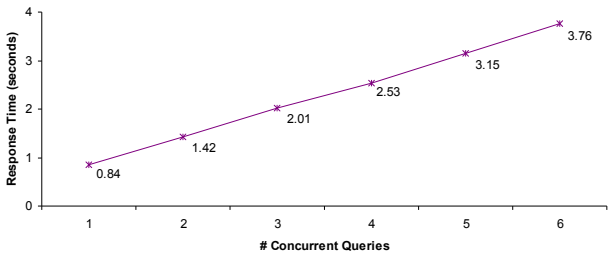

Figure 4. Average response time of multiple concurrent queries.

TABLE II. TIME BREAKDOWN OF SERVICE INVOCATION

Item	Time (Milliseconds)
Service Lookup	30
Service Execution	848
Result Packing	14
Result Reply	68
Total	960

VI. RELATED WORK

There have been many recent publications on the development of middleware systems that manage sensor networks for context-aware computing. The SOCAM architecture proposed an ontology-based approach to model, reason and query context data that is collected from heterogeneous sensors via multiple context provider modules [7]. The system provides various context-aware services to support the fast application building, which can all be located using a service locating service.

Henricksen and Indulska [8] designed a set of conceptual models and developed a corresponding infrastructure to support the software engineering process of context-aware applications. Judd and Steenkiste [10] presented a contextual information service that provides a virtual database view of sensor networks to the applications and a SQL-based query interface to access the data. The PICO middleware framework [11] uses software communities to exchange sensory data among multiple devices of different types and to provide services on these devices.

Lombriser et al. [12] proposed the e-SENSE middleware for distributed context data processing in sensor networks. Sensor nodes sharing the same context are clustered in the lower layer of the middleware to form a topology with little changes, which serves as the basis for the service-oriented processing at the upper layer. Such processing is organized by distributing subsets of each service task graph onto individual sensor nodes in the clusters for execution. AirSenseWare [15] is a sensor network system for the inter-application information sharing. It supports concurrent execution of multiple applications by mechanisms for abstract modeling and event delivery.

In comparison with these existing middleware systems, our PSG framework provides both service and query based context data access from a physical space. It applies a context reasoning model that contains various machine learning methods. We have also studied the automatic detection of local neighboring PSGs as well as the P2P connection of PSGs in a global infrastructure.

VII. CONCLUSION

We have presented the framework of a physical space gateway in this paper. The gateway acts as a middleware for context-aware applications to access context data and services from a physical space. A PSG provides different categories of services, including context data services, system services and application-oriented services. All these services can be invoked via an HTTP server using SOAP. Applications can also share reusable plug-ins at a PSG.

A PSG has a query processing engine that receives SQL-based queries from applications and executes these queries. A context reasoner at the PSG deduces events in the physical space for applications to subscribe to. The PSGs can be globally or locally interconnected for data communication.

We have developed a system toolkit for the proposed PSG framework and evaluated the framework's performance using a testbed deployment in our research lab. Initial experimental results of us show that the query processing time and service invocation time at a PSG in our testbed are reasonably small.

Our future work includes adapting the PSG framework to specific application domains and conducting more extensive performance evaluation of the framework based on multiple types of sensors besides the motes.

ACKNOWLEDGMENT

Funding for this work is provided by the Science and Engineering Research Council (SERC) of Singapore under the research grant no. 0521210083.

REFERENCES

[1] Adobe Flex, http://www.adobe.com/products/flex/.

[2] E. Alpaydin, Introduction to Machine Learning. MIT Press, 2004.

[3] Context-Aware Middleware Infrastructure for Sentient Computing, http://lucan.ddns.comp.nus.edu.sg/researchContextAware.aspx.

[4] Crossbow Inc., http://www.xbow.com.

[5] A.K. Dey, "Understanding and using context," Personal Ubiquitous Computing, vol. 5, no. 1, pp. 4-7, February 2001.

[6] Gnutella, http://rfc-gnutella.sourceforge.net/.

[7] T. Gu, H.K. Pung, and D. Zhang, "A service-oriented middleware for building context-aware services," Journal of Network and Computer Applications, vol. 28, no. 1, pp. 1-18, January 2005.

[8] K. Henricksen and J. Indulska, "Developing context-aware pervasive computing applications: Models and approach," Pervasive and Mobile Computing, vol. 2, no. 1, pp. 37-64, Feburary 2006.

[9] Jena Semantic Web Framework, http://jena.sourceforge.net/.

[10] G. Judd and P. Steenkiste, "Providing contextual information to pervasive computing applications," Proc. PerCom, pp. 133-142, March 2003.

[11] M. Kumar et al., "PICO: A middleware framework for pervasive computing," IEEE Pervasive Computing, vol. 2, no. 3, pp. 72-79, July-September 2003.

[12] C. Lombriser et al., "Organizing context information processing in dynamic wireless sensor networks," Proc. ISSNIP, pp. 67-72, December 2007.

[13] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," ACM Trans. Database Syst., vol. 30, no. 1, pp. 122-173, March 2005.

[14] A.J. Menezes, P.C. Oorschot, and S.A. Vanstone, Handbook of Applied Cryptography. CRC Press, 1996.

[15] K. Muro, T. Urano, T. Odaka, and K. Suzuki, "AirSenseWare: Sensor-network middleware for information sharing," Proc. ISSNIP, pp. 497-502, December 2007.

[16] R. Ramakrishnan and J. Gehrke, Database Management Systems. McGraw-Hill, 2002.

[17] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," Proc. WMCSA, pp. 85-90, December 2004.

[18] SOAP Specifications, http://www.w3.org/TR/soap/.

[19] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a sensor network expedition," Proc. EWSN, pp. 307-322, January 2004.

[20] Web Service Definition Language, http://www.w3.org/TR/wsdl.

[21] W. Xue, H.K. Pung, P.P. Palmes, and T. Gu, "Schema matching for context-aware computing," Proc. Ubicomp, pp. 292-301, September 2008.

[22] Y. Yao and J. Gehrke, "Query processing for sensor networks," Proc. CIDR, January 2003.