Spatio-Temporal Sensor Data Management for Context-Aware Services

Designing Sensor-Event Driven Service Coordination Middleware

Akio Sashima
ITRI, AIST / CREST, JST
1-18-13 Sotokanda,
Chiyoda-ku Tokyo, 101-0021, Japan
sashima-akio@aist.go.jp

Yutaka Inoue
ITRI, AIST / CREST, JST
1-18-13 Sotokanda,
Chiyoda-ku Tokyo, 101-0021, Japan
yutaka.inoue@aist.go.jp

Koichi Kurumatani
ITRI, AIST / CREST, JST
1-18-13 Sotokanda,
Chiyoda-ku Tokyo, 101-0021, Japan
k.kurumatani@aist.go.jp

ABSTRACT

How various kinds of sensor devices are handled, and how numerous lower-level sensor data are managed and integrated into higher-level context representations are important issues to realize context-aware services. We have been developing Sensor-Event-Driven Service Coordination Middleware (SENSORD) to fill coordination gaps between higher-level services and lower-level sensors. The SENSORD system obtains and stores sensor data into an in-memory data container to achieve fast, complex analysis of the sensed data. The facility of real-time analysis, which includes periodical evaluations of spatio-temporal conditions, enables application developers to outsource context-aware mechanisms. As described in this paper, we first examine middleware of context-aware services and our approaches. We then show the outline of SENSORD and an exemplary application: an indoor emergency response system in our laboratories. Preparing for emergency situations, such as fire emergencies, it manages sensor devices (thermometers, hygrometers, vision systems, microphone arrays, etc.) to detect emergent situations.

Categories and Subject Descriptors

H.3 [**Information Systems Applications**]: Miscellaneous; H.2.8 [**DATABASE MANAGEMENT**]: Database ApplicationsSpatial databases and GIS

General Terms

Design

1. INTRODUCTION

Important applications of ubiquitous computing for users in the real world are context-aware services. Such systems are intended to recognize users' contexts (e.g., locations, profiles, current activities) based on sensor data, and provide suitable services according to users' contexts. Although various context-aware systems have been proposed, most of them fill this gap in an ad-hoc manner. Their context representations are bound strongly to the specific sensor data; the services and the sensor devices are tightly coupled within this system. Therefore, the manner in which various kinds of sensor devices are handled, and the management and integration of numerous lower-level sensor data into higher-level context representations are open problems that remain to be considered.

2. MIDDLEWARE FOR CONTEXT AWARE SYSTEMS

Middleware is a software that mediates between application programs and low-level software or even hardware infrastructure. It encapsulates complexity and validity of low-level interfaces of the infrastructure, and provides abstract common interface layers to the applications. Middleware is not merely a platform or a framework that is intended to provide programmers with high-level APIs because middleware is used to extract essential functions that have been commonly used by the application systems, such as DBMS. The extracted functions are outsourced to the middleware, thereby simplifying the application programs. Therefore, the questions are what functions of context-aware systems should be extracted and outsourced, and what middleware is required.

In this study, we specifically consider the following features of middleware: *handling various sensor data, spatio-temporal sensor data management, and real-time sensor data analysis*. Although, we also assume that the context-aware middleware should be heterogeneous, distributed, and secure computing systems as with most modern middleware system, this paper does not specifically address those common features.

2.1 Handling Various Sensor Data and Interfaces

Some sensors, such as those in video surveillance systems, continuously send many data for a long time; others, such as fire alarm systems, only send data when they function. In addition, some sensor devices only have poor low-level interfaces such as serial-interfaces; others have high-level network interfaces such as web services. Therefore, middleware must handle various kinds of data and interfaces of sensors in ubiquitous computing environments.

2.2 Spatio-Temporal Sensor Data Management

In this study, we specifically examine management of sensor data, including locations and histories. We call these functions spatio-temporal sensor data management. Context-aware systems require some sensors to be aware of users' contexts. Any sensor data can be comprehensible with spatio-temporal data. Such data shows when and where the sensor data is obtained. Consequently, spatio-temporal management is an inevitable function of context-aware systems. It provides spatio-temporal APIs for accessing the sensor data. The standardized formats of locations and times should be desirable.

In addition, it requires functions that store, retrieve, and analyze spatio-temporal trajectories of moving objects, such as humans' trajectories in indoor spaces. Although spatio-temporal databases are proposed before, how the database stores and retrieves such trajectories remains an open problem.

2.3 Real-time Sensor Data Analysis

Location-aware content delivery systems, such as guide services in museum and navigation services, require real-time processing that infers spatio-temporal relations of humans and other objects (e.g. pictures). In particular, for emergency-response systems based on sensing environmental status, such as fire alarm services, immediate processing of the spatio-temporal relations is desirable because quicker response might prevent great damage. For that reason, the facility of the real-time spatio-temporal processing for sensor data is a very important feature of the context aware systems.

3. SENSOR-EVENT-DRIVEN SERVICE COORDINATION MIDDLEWARE

To fill those requirements, we have been developing *Sensor-Event-Driven Service Coordination Middleware* (SENSORD), a middleware for sensor data analysis, which enables context-aware application service access sensor data. It analyzes results in a coordinated and systematic manner. First, application services can access sensor data by its spatio-temporal attribute, which indicates the time and place where data are detected. Second, services can make use of sensor data through statistical and logical analyses of the data. Moreover, sensor-data-related services can be described uniformly using *SENSORDScript*, which indicates the condition for automatic control of activating, terminating, and executing the services either in sequence or in parallel.

3.1 Overview

As described above, SENSORD is a middleware coordinating lower-level sensor device with higher-level applications. Therefore, SENSORD functions cooperatively with other software and sensors modules or programs within an application system. A typical application system with SENSORD consists of the following four modules: sensor devices, sensor servers, SENSORD, and SENSORDScript (see Figure 1.)

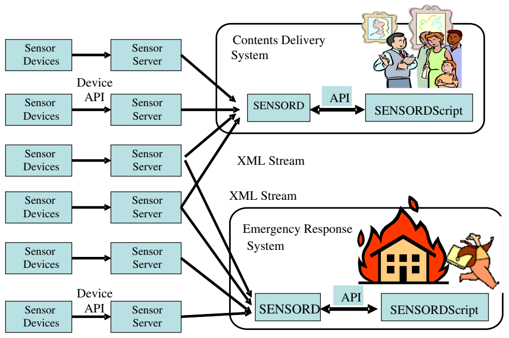

Figure 1: Applications of Sensor-Event-Driven Service Coordination Middleware

3.1.1 Sensor Devices

Sensor devices are placed in a ubiquitous computing environment. They capture users' context and monitor environmental information using thermometers, hygrometers, vision sensors that tracks humans, microphone arrays, and so on.

3.1.2 Sensor Servers

A sensor server mediates between sensor devices and SENSORDs. We assume the servers are developed by application programmers so as to connect with the sensor devices and SENSORDs. The servers periodically seek or receive the sensor data through low-level data channels, and send the data to the SENSORD. In order to encapsulate the validity of the sensor data, the servers wrap up them in XML documents; they then send the XML documents to SENSORD through TCP/IP communication channels. A sensor server can serve multi-SENSORD clients; the server opens TCP/IP connections to more than one SENSORD client.

The following XML data are exemplary of the sensor event of a USV Camera System (see section 4) that tracks users' locations. This kind of frame data is always sent when a sensor event occurs. The data are a part of an XML data stream; the whole data stream can be interpreted as a valid XML document.

```
<frame no="39267">
            <line id="103"> -45 55 167</line>
            <line id="104"> 105 35 107</line>
</frame>
```

The data show an event that detects two persons in a monitoring area. In the line tag, an "id" attribute value is used for identifications of the users in the monitoring area, and line values represent three-dimensional locations of users. The sensor server repeatedly sends this kind of XML data to the SENSORD as an XML data stream.

3.1.3 SENSORD

A SENSORD obtains and stores sensor data sent from sensor servers into its in-memory data container to realize fast, complex analysis of the data. The facility of the real-time analysis includes periodical evaluations of spatio-temporal conditions, and enables application developers to outsource the context-aware mechanisms.

SENSORD also provides various APIs (e.g., sensor record management, spatio-temporal access methods to various sensor data, synchronization, etc.), which are used for developing sensor-based systems such as context-aware systems.

The spatio-temporal query API is used to store and retrieve sensor data. For example, the programmers can ask SENSORD to retrieve sensor data using something like the following query: “retrieve all sensor data of sensor devices placed within 5 meters of me for the past 10 minutes.”

In addition, the spatio-temporal query API provides facilities that store and retrieve the contents with spatio-temporal features as with a spatio-temporal database. For example, a photograph taken at a certain time and place can be stored in a database in a relational manner along with its spatio-temporal attributes.

The synchronization API provides interfaces for data synchronization for multi-SENSORD scenarios; sensor data detected by other SENSORDs, e.g., data detected by simplified and smaller versions of SENSORDs on portable telephones, laptop computers, or personal data assistants at a certain time and place, can be stored in a database in a relational manner with spatio-temporal attributes of the detected time and place.

3.1.4 SENSORDScript

As mentioned earlier, SENSORDScript is a program written by users to use facilities of SENSORD and call the SENSORD API according to application logic, as with Java Servlets that are used in application servers. They can program the SENSORDScript to outsource the validation to the SENSORD if programmers require continuous validation of spatio-temporal relations to realize some context-aware services. For example, SENSORD validates a rule like “when a user is near a picture in a museum, send the guide information of the picture to the user” instead of user programs.

The SENSORDScript API provides the following capabilities:

- Template definitions of service plug-ins, which are provided as abstract classes (template definitions) to access various remote services invocation interfaces, e.g., RMI, Web-Services, other SENSORDScript, and SENSORD API.
- Execution Control API: It includes interfaces for rule-based execution of services realized by rule-description classes, which describe maps between conditions and actions. SENSORD activates and terminates the actions (services) when conditions are true. The condition can be ranges of sensor data, their statistical analysis results, and logical conjunctions. The API also includes interfaces of sequential and parallel execution of services.

From the programmer’s point of view, SENSORDScript API provides a script-like programming style to realize context-aware systems, by which they can write sensor-data-related service execution control easily using a SENSORDScript framework.

To start new context-aware services, programmers require the following steps:

- writing a SENSORDScript to process sensor data to realize context-aware services;
- registering it to SENSORD;
- setting up SENSORD to receive and process the required sensor data from appropriate sensor servers;
- requesting SENSORD to start the sensor recording processes and the required SENSORDScript.

3.2 Architecture

Figure 2 depicts the SENSORD architecture. SENSORD consists of some functional modules: Sensor Data Recorder Module, Data Container Module, and Service Manager.

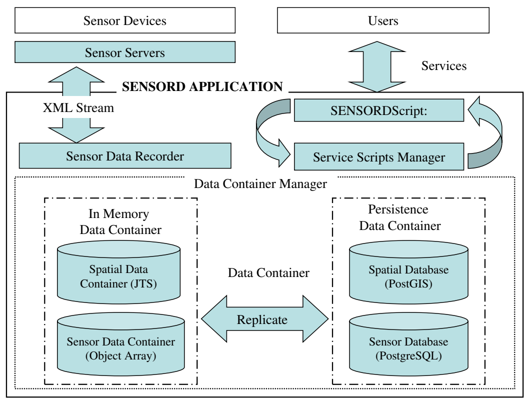

Figure 2: Architecture of SENSORD

3.2.1 Sensor Data Recorder

Each sensor data recorder connects to a sensor server. A SENSORD has sensor data recorders as multi-thread processes that access the sensor server, and receive sensor data through XML data streams from sensor servers. Receiving and parsing XML streaming data on TCP/IP communications, the recorder binds them to Java objects, which is a process known as *XML/Object Binding*. For that purpose, we use XML Pull Parser [10], a streaming pull XML parser, and JiBX [11], a framework for binding XML data to Java objects. The mapping processes are not only highly configurable, but they are very fast. Thus, using these modern Java-XML technologies, SENSORD can efficiently receive various kinds of sensor data.

3.2.2 Data Container Manager

Data container managers manage data containers that are stored with sensor data and spatio-temporal information, and provide spatio-temporal API for application programmers. Currently, we have developed four types of data containers: an in-memory sensor data container, an in-memory spatial data container, a persistent sensor data container, and a persistence spatial data container.

The *in-memory sensor data container* is stored with the objects converted by sensor data recorder. The container is a ring-buffer where sensor data are stored in chronological order. SENSORD overwrites the oldest data in it with new data if it is full. The container is implemented by Java object *Collection* classes.

The *in-memory spatial data container* is stored with spatial or geometric information, specifically locations of objects, humans and sensor devices. SENSORD overwrites information of areas that have not been accessed for a long time with new information if it is full. The information is described by XML configuration files1. The locations are represented as using a widely used coordinate system, "Longitude / Latitude (WGS 84)." The container is implemented as collections of geometry objects of JTS Topology Suite (JTS) [12]. Also, JTS is an API of 2D spatial predicates and functions. It conforms to the Simple Features Specification for SQL published by the Open GIS Consortium, and provides a two-dimensional spatial query API.

The *persistent sensor and spatial data containers* are copies of the in-memory sensor and spatial data containers. The data container manager replicates the in-memory data to persistent data containers implemented by relational databases when load-averages of SENSORD are not high, such as at midnight. We have implemented persistent data containers by PostgreSQL [3] and PostGIS [9]. PostgreSQL is a well-known open source object-relational database. Furthermore, PostGIS adds support for geographic objects to the PostgreSQL. PostGIS also conforms to the Simple Features Specification for SQL. Thus, sensor data and spatial data are stored as tables in the PostgreSQL database extended by PostGIS.

3.2.3 Service Manager

Service Manager manages life-cycles of the SENSORDScripts, such as registration, starting, stopping, and deregistration of the SENSORDScripts. The manager initiates SENSORDScripts based on XML configuration files. We have implemented Service Manager as a lightweight container of SENSORDScripts.

Figure 3 illustrates management architecture of the SENSORDScript and their rules. The SENSORDScripts are managed by "Service Manager", execute spatio-temporal queries programmed by the users, and send the results to the users. Executing the SENSORDScript with rules, such as spatio-temporal service rules, SENSORDScript registers the rules to "Rule Manager" at the initial stage. The rule manager repeatedly evaluates the registered rules with new sensor data, and notifies the client scripts of the matched rules.

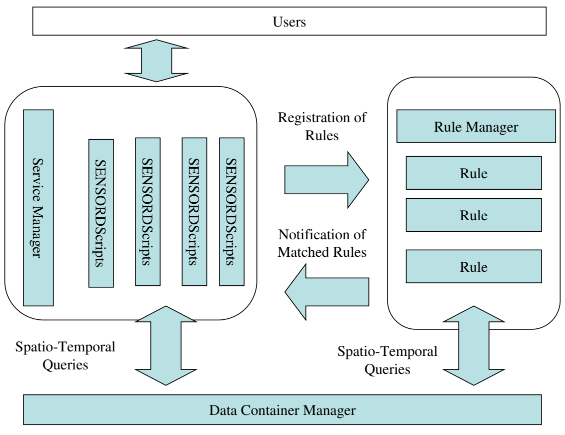

Figure 3: SENSORDScript and Rule Manager of the SENSORD

4. EXPERIENCES: INDOOR EMERGENCY RESPONSE SYSTEMS

We have been developing SENSORD and its application, an indoor emergency response system, in our laboratories[6]. The emergency response system has been prepared for emergency situations. It manages sensor devices (e.g., thermometers, hygrometers, video surveillance systems, and microphone arrays) and monitors their respective environmental statuses to detect abnormal events, such as fire emergencies. It then counts the people remaining in an area (floor of a building) to provide navigation and evacuation services for them.

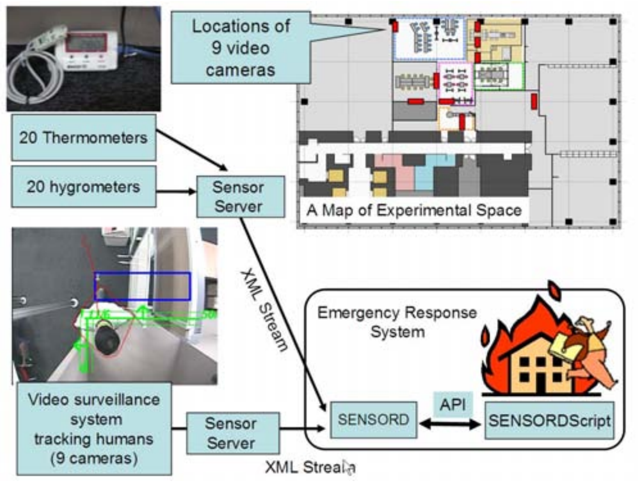
Locations of 9 video cameras
20 Thermometers
20 hygrometers
Sensor Server
A Map of Experimental Space
XML Stream
Emergency Response System
Video surveillance system tracking humans (9 cameras)
Sensor Server
SENSORD
API
SENSORDScript
XML Stream

Figure 4: Overview of the indoor emergency response system

Figure 4 displays an overview of the system developed for a floor of our laboratory building. The experimental area of the floor is about 700 square meters. There are 20 thermometers and hygrometers [1], and 9 video surveillance camera systems that count humans who cross predefined border lines in the monitoring areas 5. We use three video surveillance systems: Vitracom SiteView EP[2], Ubiquitous Stereo Vision System (USV)[3], and IBS Counter [4]).

The SENSORD system communicates with these sensors, receives the sensor-events (e.g. the events detecting passers in the monitoring area) from them, and infers the status of the environment (e.g. the number of people staying in closed areas) by analyzing the events.

In our experiments, the SENSORD system counted the number of people staying in four different rooms, such as a "meeting room". Figure 6 shows a snapshot of the system GUI. In the figure, a 2D Map at the center shows our office floor; a table on the right-hand side shows the number of people staying in each room. A graph in the lower part shows histories of temperature and humidity of a selected

[1]TR-72W; T&D Corp. `http://english.tandd.com/product/tr_7w/tr_7w_01feature.html`

[2]Vitracom SiteView EP: `http://www.vitracom.de/downloads/siteview-st-ep-en.pdf`

[3]Ubiquitous Stereo Vision: `http://staff.aist.go.jp/i-yoda/usv/index.html`

[4]CED System, Inc. IBS Counter: `http://www.ced.cp.jp/IBS_CT_CAT.pdf`

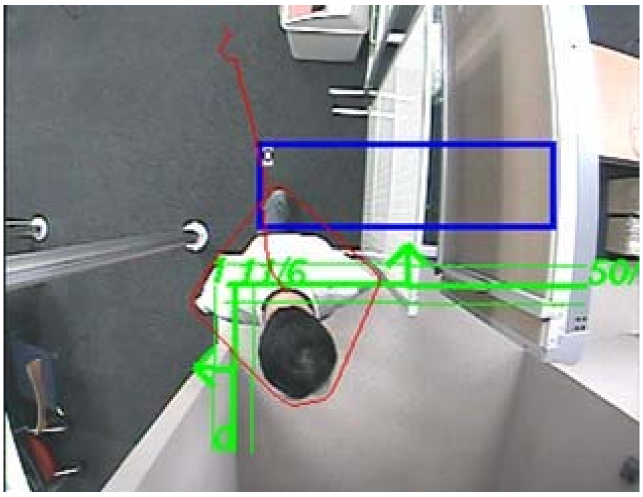

Figure 5: Raw image of the video surveillance system (SiteView EP)

sensor device.

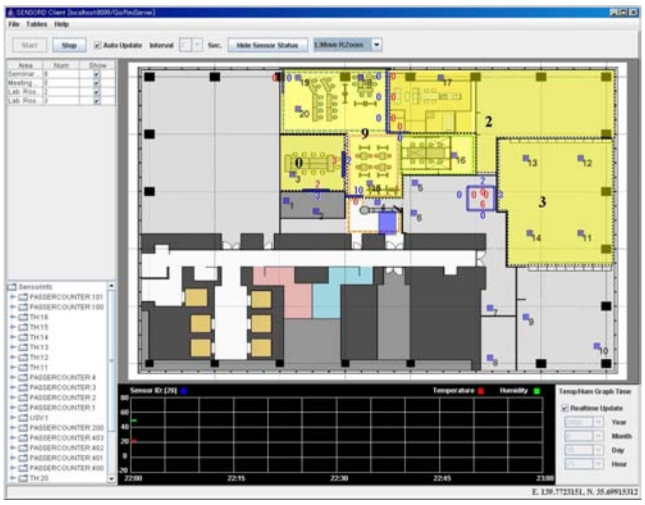

Figure 6: Snapshot of the Indoor Emergency Response System. No one occupies the meeting room at the left side of the floor map.

A video surveillance camera system, Ubiquitous Stereo Vision (USV), has facilities by which it tracks locations of users and records their trajectories [14][15]. The SENSORD communicates with USV systems and stores the data into the data containers of the SENSORD. Figure 7 shows a snapshot of a graphical viewer for the stored human trajectories obtained by USV. The viewer has a simple interface to retrieve sensor data obtained in a specific period. The retrieved trajectories are played back in chronological order.

5. RELATED WORK

Various types of databases related to this research, spatio-temporal databases [7][13], in-memory databases [1], and sensor databases [4], are proposed under the context of sensor networks. However, they did not provide service coordination facilities of SENSORDScript.

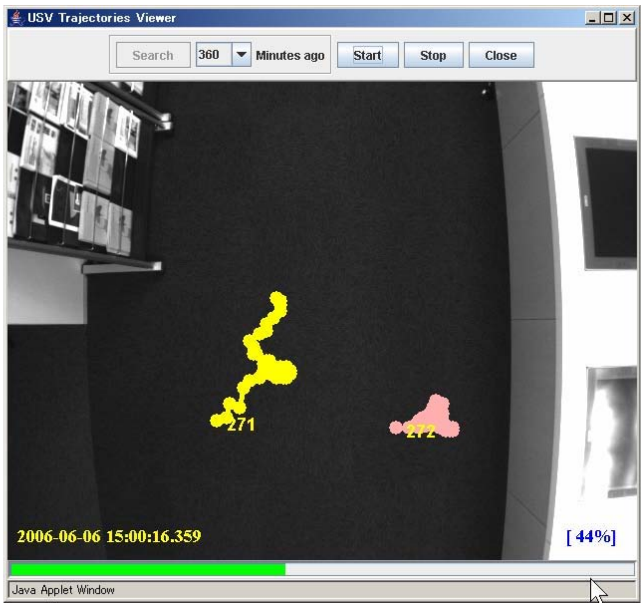

Figure 7: A snapshot of the human trajectory viewer. It shows two stored trajectories retrieved from the system. In the figure, a trajectory at the right side shows a man has been staying there. The other shows a man has been moving through the center area.

In ubiquitous computing, infrastructure for real-world service systems is drawing the attention of researchers. Those systems are developed as a service infrastructure for context-aware services. For example, QoSDream [2] coordinates multimedia contents for various user devices. GAIA [8] is intended to provide various services for ubiquitous computing. Solar aggregates sensor information for context-aware services. Context Toolkit [5] is intended as a widget for developing ubiquitous computing services. However, these systems have no spatio-temporal sensor data management facilities that we have proposed in this paper.

6. CONCLUSION

In this paper, we have described the requirements of middleware for context-aware systems in the context of handling sensor data. We have proposed a spatio-temporal in-memory sensor data management system (SENSORD) that facilitates use with context-aware application services. It helps their implementors to access sensor data, its statistical and logical analysis, and automatic control of execution of application services by providing a plug-in called SENSORDScript. We have described SENSORD and an exemplary application, an indoor emergency response system in our laboratories. Preparing for emergency situations, such as fires, it manages sensor devices, monitors environmental status, and counts people staying in an area to provide them navigation and evacuation services.

7. REFERENCES

[1] Oracle Times Ten In-Memory Database. `http://www.oracle.com/database/timesten.html`, 2001.

[2] QoSDream. http://www-lce.eng.cam.ac.uk/qosdream/, 2003.

[3] PostgreSQL. http://www.postgresql.org/, 2006.

[4] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. The cougar project: a work-in-progress report. *SIGMOD Rec.*, 32(4):53–59, 2003.

[5] A. K. Dey. *Providing architectural support for building context-aware applications*. PhD thesis, 2000. Director: Gregory D. Abowd.

[6] Y. Inoue, A. Sashima, and K. Kurumatani. Indoor navigation system for emergency evacuation in a ubiquitous environment. In *Ubicomp 2006 (Poster)*, to appear. ACM Press, 2006.

[7] D. H. Kim, K. H. Ryu, and C. H. Park. Design and implementation of spatiotemporal database query processing system. *Journal of Systems and Software*, 60(1):37–49, 2002.

[8] M. Roman, C. Hess, A. Ranganathan, P. Madhavarapu, B. Borthakur, P. Viswanathan, R. Cerquiera, R. Campbell, and M. D. Mickunas. GaiaOS: An infrastructure for active spaces. Technical Report UIUCDCS-R-2001-2224 UILU-ENG-2001-1731, University of Illinois at Urbana-Champaign, 2001.

[9] S. Santilli, C. Hodgson, P. Ramsey, and J. Lounsbury. PostGIS. http://www.postgis.org/, 2006.

[10] A. A. Slominski. MXP1: XML Pull Parser 3rd Edition (XPP3). http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html, 2006.

[11] D. Sosnoski. JiBX: Binding XML to Java Code. http://jibx.sourceforge.net/, 2006.

[12] JTS Topology Suite. http://www.postgresql.org/, 2006.

[13] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pp. 111–122, 1998.

[14] I. Yoda, D. Hosotani, and K. Sakaue. Ubiquitous stereo vision for controlling safety on platforms in railroad stations. In *Proceedings of the Sixth Asian Conference on Computer Vision (ACCV 2004)*, vol. 2, pp. 770–775, 2004.

[15] I. Yoda and K. Sakaue. Concept of ubiquitous stereo vision and applications for human sensing. In *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA2003)*, pp. 1251–1257, 2003.