# Web Services for the Dissemination of Ambient Information to I-centric Applications

Truong Ta

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

November 2005

# Abstract

Web Services for the Dissemination of Ambient Information to I-centric Applications

**Truong Ta**

Applications offered to end-users as value added services, or more simply services, are crucial to the success of future generations mobile communication systems. One important capability that will enable novel services is ambient awareness. The awareness of ambient information is usually acquired by a network of sensors and is related to end-users' context in terms of situation and environment. There exist several frameworks for the dissemination of sensor data to end-user applications. They range from low-level APIs to databases and include Web services. This thesis advocates Web services for the dissemination of ambient information to I-centric applications. It shows the shortcomings of the current dissemination frameworks and demonstrates the promises of Web services as a framework. High level of abstraction offered to application developers and ease of integration are among the key motivating factors. A set of Web services for bridging these applications and sensor networks are defined and implemented. They provide ambient information such as location, velocity, environmental data, physical presence and proximity. A generic functional architecture of the framework and a model for ambient information provide sensor interoperability and are implemented as part of the thesis. To ultimately show the feasibility of the Web service based framework, performance measurements are conducted with respect to network load and response time. The analysis shows that response time is increased while network load may decrease or increase depending on the type of ambient information requested. Ultimately, it is a small price to pay for benefiting from sensor interoperability and ease of application development.

**Acknowledgements**

I wish to express my sincere gratitude and appreciation to all the people who helped me made this master thesis possible. First and foremost, I dedicate special thanks to both my supervisors Dr. Ferhat Khendek and Dr. Roch Glitho who kept an eye on my progress and supported me with sound advices and discussions. Their words of wisdom always oriented me in the right direction or showed me different perspectives and I am very grateful for that.

I would like to further extend my gratitude to the Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson Research for their financial support. I also thank my fellow researchers who shared with me their past experiences, gave me constructive comments and helped me with my projects. I had great pleasure working with all of you and enjoyed being part of this research team.

Last but not least, my personal thanks go to my dearest friends and family who never ceased to give me moral support and were there when I needed them. You always understood and supported the choices I made during this long and personal journey.

**Truong Ta, December 2005**

# Table of Contents

# List of Figures

# List of Tables

# List of Acronyms and Abbreviations

| | |
|---|---|
| API: | Application Programming Interface |
| CORBA: | Common Object Request Broker Architecture (Object Management Group) |
| CPXe: | Common Picture eXchange Environment (International Imagery Industry Association) |
| DBMS: | Database Management System |
| IETF: | Internet Engineering Task Force |
| MMS: | Multimedia Message Service |
| MOD: | Moving Object Database |
| OASIS: | Organization for the Advancement of Structured Information Standards |
| OGC: | Open Geospatial Consortium |
| SMS: | Short Message Service |
| SOA: | Service Oriented Architecture |
| SOAP: | Simple Object Access Protocol |
| UDDI: | Universal Description, Discovery and Integration standard |
| VAS: | Value Added Service |
| WSDL: | Web Service Description Language |
| WSN: | Wireless Sensor Network |
| WWRF: | Wireless World Research Forum |
| W3C: | World Wide Web Consortium |
| XML: | eXtended Markup Language |

# Chapter 1 Introduction

## 1.1 The Research Domain

Recent studies showed that today's telecommunication network operators face a saturation of the voice market. The mobile penetration level is high in Europe and Asia and stagnation is setting in. These telecommunication network operators are shifting to Value Added Service (VAS) to increase revenues in both data and voice services. VAS are services that add extra value for the basic service offering or could be standalone services like Short Message Service (SMS). This opens up new opportunities and so far, the VAS segment is still growing in terms of profitability and in diversity of services offered.

Traditional telephony VAS include call forwarding, call screening, call transfer, call hold and conferencing; innovative VAS include Multimedia Messaging Service (MMS), presence and other lucrative VAS include downloading ring tones, movies, online gaming and tracking the scoreboard of sport events. A new emerging class of VAS are ambient-aware services. These VAS are sensitive to end-users' context in terms of situation and environment. Location-based services such as offering location-sensitive information and tracking services are a subset of ambient-aware VAS. This class of services is considered as a key VAS for telecommunication network operators to deliver in the coming years.

### 1.1.1 Ambient Awareness

Ambient awareness is one of the main features for enabling this class of services. It allows information about the individual user, his/her environment and situation to be known to applications in order to simplify interactions and to enrich communication. This user-centric view of computing leads to automatic, adaptive and personalized applications. Let us consider the example of a user trying to contact a colleague. An ambient-aware application may determine that the colleague is in a meeting, implied by the fact that other users are present in the same room. The user is notified and contact attempt is postponed until the colleague is alone. Once alone, the application may establish a video phone call or regular call depending on the nearest equipment available to the colleague.

### 1.1.2 Wireless Sensor Network

There are many ways to acquire information about the user situation and environment, or more simply his/her ambient information. Capturing this information is usually done by a sensor, or group of sensors. A typical sensor node is a small scale, multi-purpose sensing and computing device, with wireless connectivity, and is deployed in the environment. The area of application of this technology spans more than just providing ambient information for VASs. It is also used by the military, the scientific community, the health-care and industrial sector. Because of the wide area of applications, the sensors are very heterogeneous and as a consequence, there are several possible techniques to harvest information produced by sensors. These techniques range from low level APIs to databases, to mobile code and include Web services. There is an ongoing standardization however it is for very specific areas of application which are not necessarily applicable for VAS. Currently, there is no specific ways to model and disseminate this ambient information to VASs.

### 1.1.3 Web Services

Web services in its simplest definition are programmatic interfaces that allow application to application communication over a network. When properly designed, Web service interfaces provide a high level of abstraction as well as loose coupling between interacting software components. This ease the development of applications since developers are not required to have extensive domain-specific knowledge. Web services have been adopted in many application domains (telecommunications, digital imaging, e-commerce, etc), the reason to do so have mainly been ease of integration with other applications or with other business processes. The mentioned features make Web services an attractive solution to expose sensor capabilities while hiding their domain-specific details, to enable interoperability and to ease the development of VASs.

## *1.2 Problem Statement and Contribution of the Thesis*

Ambient-aware applications are slowly gaining momentum in the industry. Location-based services are gaining popularity as telecom operators, public LAN providers and enterprises are developing and deploying these services. The target audience for these services includes any mobile computing and communication device, such as cellular phone, PDA and laptop. Since service providers and service consumers are increasing in variety and volume, it is expected that rapid development and deployment of VASs will be a key strategy in the success of any service provider. However, from an application developer point of view, providing ambient awareness to applications and using it is not an easy task.

First, current data dissemination mechanism for sensors have several shortcomings that may hinder the application developer productivity. Some of these mechanisms are programming language dependant, others require extensive knowledge of sensors and its low level details, and a few of them do not have the right level of abstraction for building user-centric services (i.e. requires many methods to perform a simple ambient information request). In brief, it is difficult to integrate sensor technology in applications.

Second, since sensors may be embedded in the environment, application portability is a problem that end-user faces and application developers have to deal with. Because of the heterogeneous nature of sensors, there is no guaranty that an ambient aware application may work when the user is in a foreign environment with different sensors types, manufacturers, models or data dissemination mechanisms. Service continuity is an important aspect of customer satisfaction and hence, the application developer has to spend a portion of the development effort in dealing with sensor heterogeneity (write code to support different sensors).

This thesis proposes a high level and user-centric Web service interfaces and an accompanying framework to disseminate ambient information, more precisely environmental and spatial information. The goal is to allow application designers to quickly prototype ambient aware-applications, to concentrate on the high-level details of acquiring and acting on ambient information while letting the framework handle the time-consuming and tedious low-level details of ambient-aware computing. Hence, high level and user-centric Web services seem to be a promising approach to ease the development of ambient-aware applications.

As part of the research work, the proposed interfaces are implemented along with the generic framework. Commercially available sensors are integrated in the whole system and a simple prototype application has been created and tested. Performance measurements were taken and analyzed to evaluate the overhead of using the Web service middleware approach.

## 1.3 Organization of the Thesis

Chapter 2 gives and overview of technologies and concepts involved. It introduces the I-centric paradigm with a strong focus on ambient-awareness. It also outlines two other features related to ambient awareness: adaptability and personalization. Wireless sensor networks characteristics are described along with their logical topology, field of applications and their strengths and weaknesses. The last section defines Web services, their basic principles, architecture and application in the industry. Finally it concludes with the key benefits of Web services.

Chapter 3 defines key requirements for a data dissemination framework for wireless sensor network in an I-centric environment. An evaluation of those requirements for different data dissemination approaches follows before the concluding summary.

Chapter 4 starts with the description of the world model used for modeling ambient information. The Web service interfaces for disseminating ambient information is detailed and a generic functional architecture of the framework is outlined.

Chapter 5 describes the implementation of the framework including the sensors used, the mapping to the sensors and optimizations used. A section on performance measurements and analysis is included. A sample application prototype is described at the end of this chapter.

Chapter 6 concludes with the thesis contributions and potential future work.

# Chapter 2 Ambient Awareness, Wireless Sensor Networks and Web Services

This chapter introduces three main topics: ambient awareness, wireless sensor networks and Web services. The first section describes ambient awareness in I-centric communications and its role with respect to other features of I-centric communications. The second section discusses wireless sensor network, a technology that enables the capture of ambient information. Wireless sensor network characteristics, topology, their strengths and weaknesses are overviewed. The third section is about a technology that enables seamless application to application communication: Web services. It outlines Web services as well as its principles, architecture and standards.

## *2.1 I-centric Communications*

The Working Group 2 of the Wireless World Research Forum (WWRF) [14] has worked in the area of service architecture for future wireless communications systems. They envisioned a communication environment [6] where the individual end-user, 'I', is in the center of all activities a communication system has to perform. I-centric applications are a subset of context-aware applications. It focuses on end-users and their individual needs, preferences and surroundings in order to adapt to the changing situations. In I-centric communications, the user interacts with his or her communication space defined by related context and objects. This communication space support three features: personalization [41], adaptability [40] and most importantly ambient awareness [39]. The reference model is shown in Figure 2.1. Because I-centric applications should work under changing environmental conditions, these three features affect all layers of the communication. Hence a vertical approach is adopted and is provided by generic service elements, a set of common functions that ease the creation of I-centric services. Among the generic service elements, environment monitoring provides awareness of the user context to all layers. The service platform supports the development and operation of I-centric services (APIs to applications, runtime environment, etc) while the IP communication subsystem provides call control, session control and mobility management. The next sections elaborate on the three features.

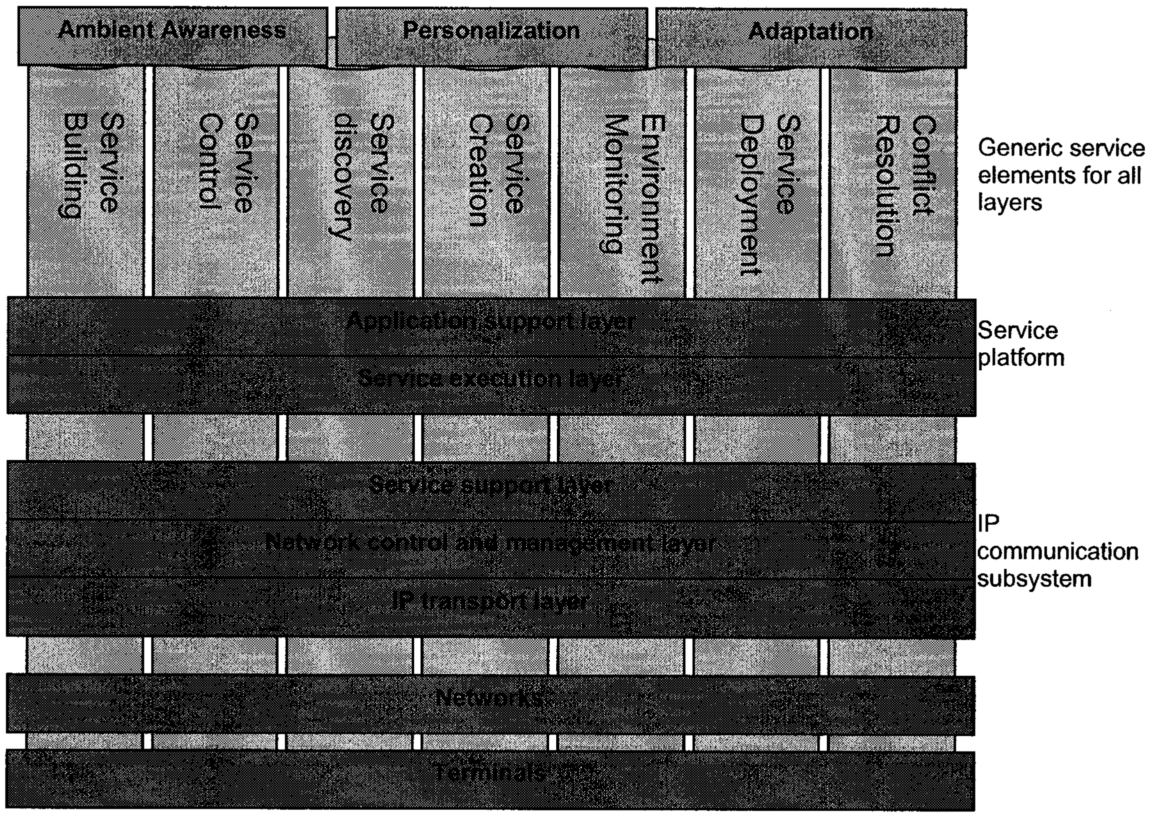


**Figure 2.1. Reference model of I-centric communications**

### 2.1.1 Ambient Awareness

Ambient awareness denotes the functionality of sensing and exchanging the ambient information of a user in his communication space. A communication space denotes the surrounding space of a user along with the objects he or she is communicating with. Ambient information is any information pertaining to the situational context in which an individual user is in. The situational context includes three main parts: spatial information, environmental information and physiological information. Spatial information is not limited to location but also includes orientation, speed, acceleration, surroundings (i.e. indoor/outdoor), proximity and physical presence. Environmental information includes temperature, air quality, humidity, dust level, luminosity and noise level. Examples of physiological information are blood pressure, heart rate, respiration rate, muscle activity and tone of voice. Besides the main situational contexts, there are also mental, social and task context. The goal of ambient awareness is to utilize ambient information to enable services to adapt actively to the changing context. Ambient information can also be combined with situational models (e.g. topology information) to produce higher level concepts (i.e. the user is in a room). There are countless ways to acquire ambient information, each have different format, reliability, applicability, uncertainty and so on. Ambient information can be acquired from an application, the network, sensors or other sources. For instance a networked calendar can give rough location information of a user given his time/activity schedule; IP addresses can give location and similar concepts can be applied to credit card/bank transactions to retrieve location. Of course, ambient information can also be retrieved by hardware sources such as sensors. Needless to say that a system that provides ambient awareness will need to retrieve diverse types of ambient information coming from various sources and to deliver to assorted devices or systems.

Application for ambient awareness can be much diversified. For security applications, one can create an application that detects an abrupt rise in ambient noise (such as a scream) in a school campus and notifies the nearest security guard [25]. For added security in corporate wireless access point, location information can be used to only permit network access to users who are within the physical boundaries of the corporate building. Ambient aware user interface is also a good area of application. For instance, the typical user interface of an email client is text-based, whereas it would become speech-based if the user is driving his/her car.

### 2.1.2 Personalization

Personalization provides means to model preferences and allows the individual user to manage his/her own information and communication space. These preferences allow the user to select, configure and arrange presented information individually to increase the usefulness and acceptance of digital information and applications, while filtering out unnecessary/irrelevant information. With personalization, products, services and data can be tailored to individual preferences and characteristics. Ambient information plays an important role in personalization of services. It dictates when (i.e. in which situation) a personalized service should be offered or adapted, it specifies how to personalize to the current situation and it is used to acquire preferences implicitly. While preferences can be specified explicitly (i.e. users input their preferred banking institution, set a bookmark, apply a 'Theme', etc), acquiring preferences implicitly can be done by Rule-based systems. Rule-based systems learn user preferences by observing their behavior (i.e. recurrence of a location or environmental conditions in which the user is consistently performing a task).

In order to share user preferences or profile information across different domains, it is important to formally define the semantics of this information, as well as trust and privacy considerations. Ongoing standardization efforts for handling profile information include IDsec [42] from the Internet Engineering Task Force (IETF), Composite Capability and Preferences Profiles [43] (CC/PP) and Platform for Privacy Preferences [44] (P3P) from the World Wide Web Consortium [49] (W3C).

### 2.1.3 Adaptability

Adaptability is defined as the ability of services and applications to change their behavior to reflect changes in the environmental conditions and/or individual preferences. By adapting, these services and applications meet current user needs and preferences. An example of adaptation can be a media streaming application that changes the sound output to text output on the display while the user moves from a silent environment to a noisy one. Other types of adaptation includes: altering the communication streams (bit rate) during transmission, adapting the presentation of information (changing from text output to sound output), altering the content of a message (adding / removing information) or modifying behavior (switching to another algorithm). Adaptability must be reactive as well as proactive. For instance, a mobile device that is merely reacting to a failing connectivity might still face service interruption. Anticipation is required for service continuity and implies predicting the near future. In all cases, adaptation requires constant environmental monitoring and event notification.

### 2.1.4 Business Model

Organizational and financial aspects are becoming more important in the development of telecommunication systems and hence the results of I-centric communication will be partly driven by business models and market demands as well as R&D outcomes. The WWRF identifies a flexible business model as a starting point to describe the roles, relationship and reference points of each party in the business community. This is done to motivate the participation of business partners as well as to provide a standardize point of contact (reference points) for the information exchange between business partners.

## *2.2 Wireless Sensor Networks*

A sensor, in a very generic definition, is a system that can sense a phenomenon and provide/display the sensed information. The sensed phenomenon can be of any type ranging form environmental conditions (temperature, magnetic fields, atmospheric pressure, humidity, etc), to chemical constitution and including electrical properties. A wireless sensor network [51] (WSN) is a collection of small-scale hardware sensor nodes capable of four main functions: sensing, powering, processing and communicating. The processing unit is typically an embedded microcontroller responsible for aggregating [26] [27], encrypting [28], storing the data as well as managing the collaboration with other nodes using power-aware routing algorithms. The communicating unit is responsible for the transmission and reception of data. The transmission media is typically done via radio communication but there are also instances of ultra-sound [2], optical communication

[36] and infrared. The sensor is powered by a power unit; it can operate on batteries, on an alternate power source such as solar cells or both in order to recharge the depleted batteries. Finally, the sensing unit might have more than one sensing capabilities as seen in Smart-Its [33].

A popular platform for developing WSNs is the mote platform from Berkeley University [45] and there have been many WSNs developed from it. The mica-mote [47], micadot-mote [48] and MIT crickets [2] are commercially available examples from Crossbow Technologies. Figure 2.2 and Figure 2.3 show the mica sensors next to golem-dust [Figure 2.4] a prototype sensor developed at Berkeley Sensor and Actuator Center [46].

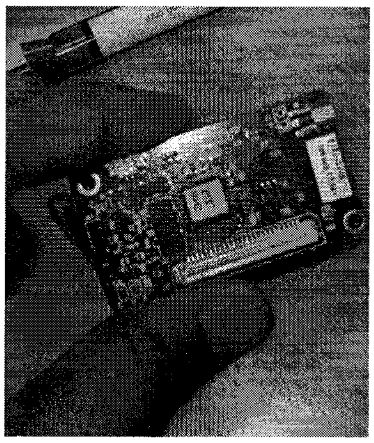


**Figure 2.2. Mica Mote**

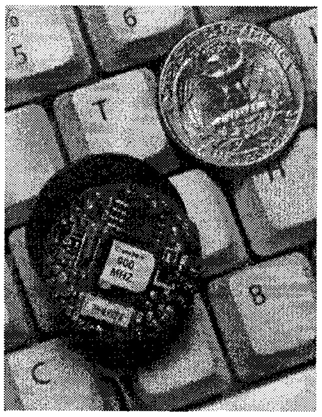


**Figure 2.3. MicaDot mote**

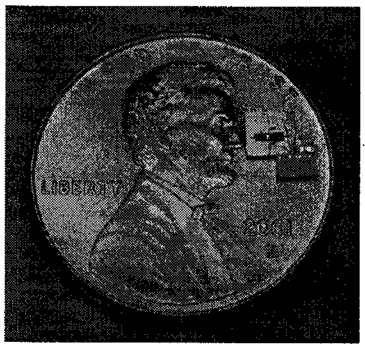


**Figure 2.4. Golem Dust**

## 2.2.1 Sensor Characteristics

Sensors have intrinsic and extrinsic characteristics [50]. Intrinsic characteristics are defined by the type of information they sense (or scientific requirements) and are not affected by the hardware design. The *number of data source* is the number of connected sources needed to make a meaningful measurement. For instance, measuring seismic activity requires a lot more data sources than measuring ambient temperature of a room.

This directly impacts the scalability of the data transport as well as management and administrative functions of the system. The *data rate* will depends on the type of collected data (i.e. numerical measurement, image data, video data, etc) and the sampling rate. The *timelessness* indicates if the collected data is to be use in real-time or for later analysis. The *value* of the data indicates the sensitivity to data loss. Systems collecting valuable data may incorporate fault-tolerance or redundancy mechanisms to avoid this loss. Extrinsic characteristics depend on the hardware design such as processing power, memory, available bandwidth, power consumption and uniqueness of design. Uniqueness of design is a characteristic of sensors that can only operate in certain environmental conditions (i.e. outer-space).

### 2.2.2 Wireless Sensor Network Topology

Typically, WSNs are densely deployed in a sensor field either by throwing them in a mass or placing them one by one. Several hundreds to thousands nodes are deployed in the field and the node density can be as high as 20 nodes/m$^3$ [51]. After initial deployment, the logical topology of the sensor network can rearrange itself due to many factors: node failure/malfunction, poor connectivity, insufficient energy/power or node mobility. Thus it is natural for a WSN to have self-configuration and self-organization capabilities. Furthermore, with the constant rearrangement of the topology and the redundancy in the WSNs nodes, global addressing is not appropriate in WSN. The importance of any one particular node is reduced as compared to traditional networks, hence addressing is done according to the attributes of the WSN or the data it provides. An example of data-centric addressing is geographical addressing, where data is retrieved based on a geographic 'bounding-box' or region.

In general, WSNs have a sink node in which external applications can interact with. The sink node is a special node with enhanced capabilities such as more processing power, storage, memory or a direct link to a gateway or the Internet. It may not have sensing capabilities like normal nodes, however it possess other roles such as the knowledge of the WSN (what data is available, how to communicate with the WSN, logical topology, etc). Of course, it also exposes WSN capabilities to the interested applications. Figure 2.5 shows the typical topology of WSNs and depicts a sink node that is not directly interacting with all the sensor nodes. Since data transmission is the most power-hungry operation in a sensor node, techniques such as multi-hop, aggregation and other power-aware routing protocols [29][30][31][32] are used to relay data to the sink node while minimizing power consumption. The aggregating nodes are connected to other sensor node(s), collect the data from the connected node(s) and forward it to the sink (or other closer node). Application interacting with the sink node might require low level or proprietary APIs. There are also other approaches such as databases, mobile code and Web service that will be discussed in Chapter 3.

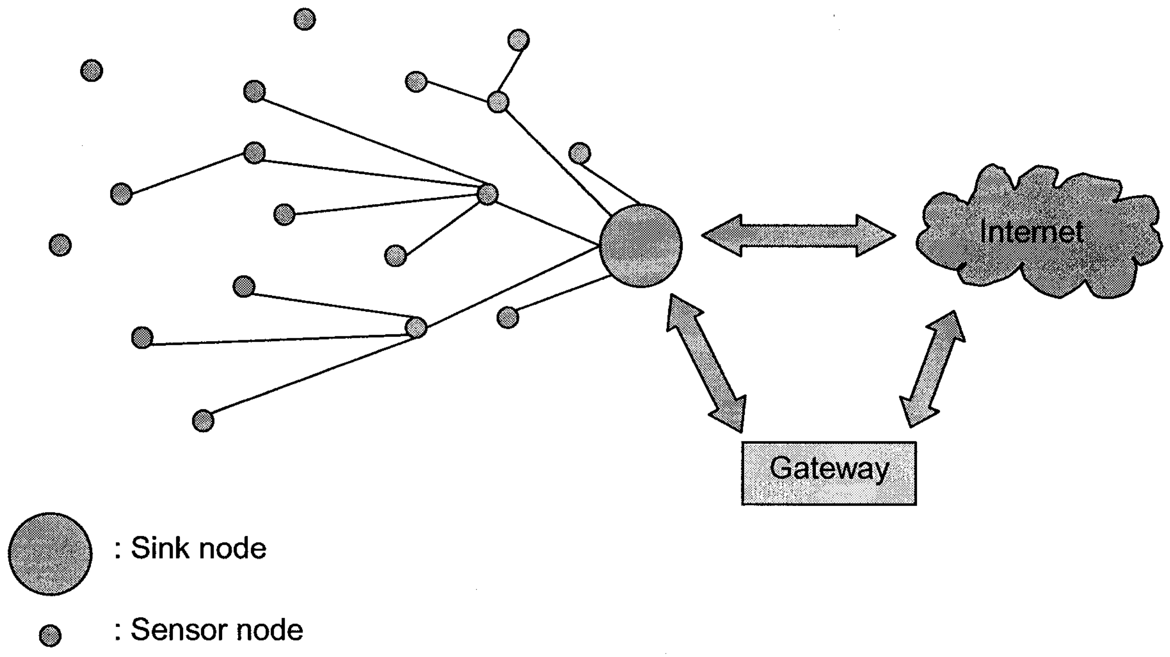


**Figure 2.5. WSN topology**

## 2.2.3 Field of Applications

WSNs can sense a wide variety of ambient information. With a wide variety of information one can expect a wide variety of application domains. Apart from ambient awareness applications discussed in Section 2.1.1, WSNs are also used in military applications, environment and habitat monitoring, health sector, home and other commercial areas.

Military applications benefit from the rapid deployment, self-organization and fault tolerance of WSNs. The destruction of some of these nodes by hostile forces does not affect much the operation of the WSN while providing friendly forces with intelligence, surveillance, reconnaissance and targeting data. Examples include monitoring friendly forces, equipment and ammunition; battlefield surveillance; reconnaissance of opposing forces and terrain, and targeting/guidance system improvement.

Environment and habitat monitoring [34] use WSN to study vegetation response to climatic trends and diseases. In the same context, acoustic and imaging sensors can identify, track and measure the population of many species. Flood detection [35], such as the ALERT system deployed in the United States, employ rainfall, water level and weather sensors. Other examples include forest fire detection and precision agriculture where pesticide level, soil erosion and air pollution are monitored in real-time.

In the health sector, human physiological data can be collected, stored and used for medical exploration. It can also detect elderly people's behavior (e.g. a fall), and help doctors to identify pre-defined symptoms earlier, as well as to track doctors and patients location inside the hospital. For home applications and commercial areas, there are intrusion detection systems, interactive museum, environmental control in office buildings, location tracking and many more.

### 2.2.4 Wireless Sensor Networks: Strengths and Weaknesses

While ambient information can be collected from other sources, it does not equal all the benefits WSNs can provide. The vast diversity, accuracy and availability (when densely deployed) of the collected information can easily be the deciding factors. It is a known fact that WSNs are suited for environmental information. For physiological information there seems to be no other obvious choices than sensors. Combine these benefits with their form-factor and their wireless feature, and it makes them even more attractive.

As for the weaknesses, sensor nodes have limited resources. There has been consistent research work in this area, notably in minimizing power consumption (i.e.

power aware routing algorithms, time-scheduling collection, low-power hardware design, etc). The fragile nodes are prone to failure, both in the node hardware or communication.

Using commands or low level APIs to access data via the sink node is also a limitation. This often requires technical knowledge of WSNs and their topologies when a data-centric addressing scheme is not available. Low level APIs are not suitable for rapid application development especially when developers do not have prior experience with WSNs.

WSNs are also heterogeneous; sensing one type of ambient information can be done by many sensor model/manufacturer. An application that needs to replace a sensor model/manufacturer by another model/manufacturer can be a difficult task. It will require modification of the application if the APIs are proprietary or use a different programming language/platform than the original one. Furthermore the problem of integrating and mixing heterogeneous types of sensors in an ambient-aware application can become an even more challenging task.

## 2.3 Web Services

Ambient awareness is the knowledge of ambient information and WSNs provides means to capture it. Web services deals with communicating information to applications. This sub-section defines and details Web services and its principles, architecture and standards involved. Some industry examples are discussed before concluding with a summary of the advantages of Web services.

### 2.3.1 Definition and Basic Principles

Adam Bosworth, one of the main contributors to the development and evolution of Web services, defines them as an architecture that allows applications to interact with each other over a network. There are three foundation principles [64] on top of which Web services are built: high level of abstraction, loose coupling and asynchronous communication.

High level of abstraction, also known as the coarse-grain approach, is to combine the steps involved in the communication process into one big operation. Effectively, this principle deals with communication efficiency by minimizing the interactions between the communicating components. There is also the added benefit of transactional integrity when dealing with atomic operations.

Loose coupling deals with minimizing interdependency of the communicating components. It is desirable to handle changes gracefully and reliably; a change in one component should not affect another component in the communication.

The third principle suggests that the mode of communication should be asynchronous. However synchronous mode is also supported. Asynchrony in the communication means that a component should not wait for the response of another component in order to continue its task. This is meant to handle unavailability in real-world application as they might go down, run into temporary bottlenecks or just take a large amount of time to complete (i.e. requires human intervention).

### 2.3.2 Service Oriented Architecture Definition, Roles, Operations and Internet Standards

A Service Oriented Architecture (SOA) is a software architecture that defines the use of services to make resources available to network nodes through a well-defined and standard interface. Web services implements this architecture.

The W3C defines the Web services architecture [Figure 2.6] by three entities [18] - the service requester, the service provider and the service registry - and their relationships: publish, find and bind. The service provider owns the service and publishes the service description in the service registry. The service description uses a standard format to describe the service and contains information on *what* is the service (i.e. inputs and/or outputs), *where* it can be located (i.e. network location of the service implementation) and *how* to use it. The service registry maintains a collection of service descriptions while providing search and publication operations to the service requester and service provider respectively. The service registry also contains general information about service providers, a classification of business and service types, and technical details of a service. The service requester finds the appropriate service by querying the service registry and binds to the service provider. While the service requester is the consumer of the Web service, the role of a service requester and service provider are not mutually exclusive. In fact, a service requestor can consume a Web service in order to provide a greater service to other service requestors.

Each of the three entities has two of the following behaviors: publishing a service description, finding a service description, and binding (or invoking) a service. Before a service description can be published, it needs to be created. This can be done by handcoding it, but there are more efficient ways such as software tools that generates them automatically from the programming model. The publication can be done via various mechanisms. The simplest case is the direct publish where the service provider sends the service description to the service requestor by common distribution scheme (i.e. email attachment, URL, CD-ROM distribution). In this case, it is assumed that the two parties have agreed on the business terms and/or the service requestor has paid the fees for using the service. Finding the service description can be done by either querying the service registry or retrieving it directly. Some find operations are used at design time for program development purposes while others are used at runtime for retrieving the service binding. Lastly, the bind operation involves locating, contacting and invoking the Web service at runtime using the information found in the service description.

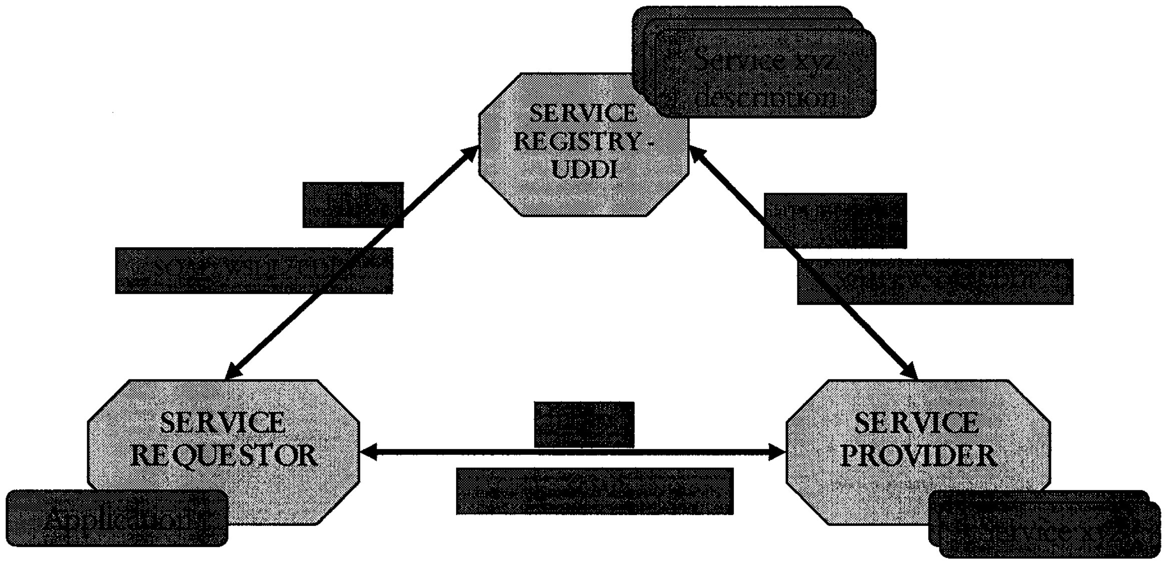


**Figure 2.6. Web service architecture**

The concept of SOA has been around for some time. Most standard distributed computing middleware such as Java Remote Method Invocation (Java RMI), the Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) and the Open Group Distributed Computing Environment (DCE) all implements SOA concepts. While detailing these middleware is out of the scope of this thesis, the general difference is that Web services do not rely on a new set of protocols, but rather build on top of existing standard Internet protocols. Table 2.1 shows these differences.

**Table 2.1. Service Oriented Architecture protocols**

| | Java RMI | CORBA | DCE | Web services |
|---|---|---|---|---|
| Data Format | Serialized Java | CDR | NDR | XML |
| Wire Format | Stream | GIOP | PDU | SOAP |
| Transport protocol | JRMP | IIOP | RPC CO | HTTP, SMTP,... |
| Interface description | Java Interface | CORBA IDL | DCE IDL | WSDL |
| Discovery mechanism | Java Registry | COS naming | CDS | UDDI |

### 2.3.2.1 XML: Extensible Markup Language

The fundamental building block is XML [65] which is a platform and programming language neutral mechanism to represent data in a structured fashion. It is human-legible and easily parsable by software for validation. It separates data representation from data content

### 2.3.2.2 SOAP: Simple Object Access Protocol

The de facto standard for the communication between all entities is the Simple Object Access Protocol [12] (SOAP). SOAP is an XML-based protocol fulfilling three main roles: it defines a framework for describing messages and how to process them, it specifies a set of encoding rules for expressing instances of application-defined datatypes and it provides a framework for transport protocol binding. The binding allows SOAP messages to be transferred using various transport protocols (HTTP/S, SMTP, POP3, IMAP, etc).

#### 2.3.2.3 WSDL: Web Service Description Language

The service description is expressed in a standard XML format: the Web Service Description Language [11] (WSDL). The WSDL service description contains an application-level service description and protocol-dependant instructions in order to access the Web service. The application-level service description is an abstract interface that describes *what* consist in the Web service. This includes the service type and information on the operations such as the input/output of the exchanged messages, the format of the messages as well as the individual datatypes of the messages elements. The protocol-dependant instructions contain information on *how* to communicate with the Web service and *where* to invoke it. The *how* details the binding of the abstract interface to a set of protocols. The bindings specifies which XML encoding style to use, which transport protocols to use, etc. The *where* details the service implementation as a collection of one or more related ports. A port is the access point of the actual service.

#### 2.3.2.4 UDDI: Universal Description, Discovery and Integration

The interface for publication and find operations is specified by the Universal Discovery, Description and Integration [13] (UDDI). An entry in the UDDI contains three types of information pertaining to Web services: white-pages information which includes a brief description, contact information and business identifiers; yellow-pages information which provides a classification of business and service type; and green-pages information which covers the technical details of a service such as compliance to certain technical specifications as well as a pointer to the WSDL document. The publication can be done to a private UDDI node hosted within an enterprise (i.e. for internal enterprise application integration), or to a public UDDI Operator node in hopes of being discovered by potential business partners. Searching the UDDI can be done by service type or by service provider. UDDI also defines a core set of taxonomies such as geographic location, product codes and industry codes to refine the search.

### 2.3.3 Web Services Industry Examples

Web services also allow corporate business models to extend by facilitating the development and deployment of new services, or by outsourcing services. New services can be easily created without requiring technical knowledge of the underlying technology. Parlay-X [7] is an example where a set of telecommunication capabilities (i.e. location, presence, multimedia messaging and call control) are exposed as Web services to ease the development of applications by developers that are not necessarily expert in telephony or telecommunications. The Common Picture Exchange environment (CPXe) [24] also uses Web services to facilitate the order, print, share and delivery of digital photographic images. CPXe provides standard Web service interfaces for business-to-consumer access services and wholesale business-to-business services to ultimately enable a digital picture to be printed and delivered at someone doorstep.

### 2.3.4 Web Services Key Benefits

This section outlines the key benefits the Web services.

- Loose coupling allows for two main advantages: interoperability and ease of integration. Interoperability is achieved because changes made to the internals of a service do not impact the service requester side as long as the service interface does not change. This allows applications to easily port from one underlying technology implementation to another (i.e. from one sensor network to another). Additionally, integration between disparate systems is as easy as agreeing to a standard format for the information exchange. Integration is also made easy since Web services are not tied to any particular programming language, platform and protocol.
- Open Internet Standards (XML, WSDL, SOAP, UDDI) allows for programming language, platform and transport protocol independence. Web services make use of existing ubiquitous transport protocols (HTTP) leveraging existing infrastructure and allowing ubiquitous access by a wide range of communication devices.
- Web services create new business opportunities by facilitating the development and deployment of new services, or by outsourcing services to third parties. Additionally, new business partnerships can be constructed dynamically by finding suitable services in the UDDI. The discovery/publication and security features provide supporting functionalities for a flexible business model. Any entity can become a service provider or requester and benefits from those features.
- Web services, being modular, can be reused easily and can extend the life and value of legacy system without much development effort. This entails that development effort using Web services is greatly reduced in terms of time and difficulty.

# Chapter 3 Data Dissemination Framework for Wireless Sensor Networks

The bridging element of WSNs and end-user applications is the data dissemination framework. There are different ways for WSNs to communicate the sensed information to external applications. Each has their advantages and their own area of application but ultimately is meant to simplify the development of applications. This chapter discusses the different data dissemination framework available for WSNs. The first section outlines some of the requirements of such framework for an I-centric environment. The following sections overviews the available frameworks. The last section summarizes the most promising framework.

## 3.1 Requirements for a Dissemination Framework for an I-centric Environment

The following requirements show what is needed in an I-centric dissemination framework from an application developer perspective.

*Provide a high level of abstraction* - The interface between the WSNs and the applications should be defined at the highest level of abstraction. To enable fast application development, the developer should not be aware of the low level details such as where the ambient information comes from (source of ambient information and technology used) and what approach underneath the framework is being used to disseminate/collect the data. As a direct implication, addressing sensors should be done as a whole and not on a node-by-node basis.

*Provide interoperability and ease of integration* - The software glue that connects heterogeneous sensors to applications on heterogeneous clients should make it easy for the developer to use and access ambient information. The developer is not expected to learn new programming paradigms, new programming languages or new technological concepts in order to use ambient information. Integration should be done with minimal effort. As a direct implication, the framework should be programming language independent. Providing a generic and flexible interface should insure interoperability.

*Support a flexible business model* - As mentioned in the Chapter 1, I-centric applications require a flexible business model. This adds security and publication/discovery requirements to the framework. The expected security features should include authentication and authorization of the entities involved in the business model; encryption and privacy mechanisms to protect sensitive data such as user location; non-repudiation as charging may be involved; and data integrity to prevent malicious tampering. Since adaptability plays an important role in I-centric communication, the framework should provide discovery/publication to cope with the changing environment and the reflecting availability of services or ambient information.

*Support for asynchronous and synchronous mode of communication* - When an application is requesting ambient information, it should not be blocked while waiting for the response of the framework. Thus the framework should support both mode of communication. More specifically to I-centric communications, the framework should support event notification to facilitate monitoring tasks and to promptly respond to changes in the environment.

*Introduce little overhead* - The framework should attempt to minimize the overhead related to network load and response time. While low overhead is important in some applications (i.e. location aware applications), it is not the case for all applications or all types of ambient data. Having a low overhead means that the framework can support a wider range of application requirements.

## 3.2 Data Dissemination Approaches for Wireless Sensor Networks

There are many dissemination frameworks available. This section classifies them by the programming paradigm used in conveying the sensed information to end-user applications. The following categories are evaluated: low level commands and APIs, databases, mobile code, Web services and others when the framework does not fit in the first four categories.

### 3.2.1 Low Level Commands and APIs

Almost every sensor has a low level command interface. The use of this low level command interface can serve several purposes including debugging, configuring it (e.g. crickets needs to set its mode of operation), upgrading its firmware or to retrieve data readings. This can be done via proprietary software provided by the sensor manufacturer or a standard text interfacing client (i.e. telnet, HyperTerminal). A general understanding of WSNs is not enough to fully use the commands but rather a full understanding of this particular instance of WSN. In the case of MIT Crickets, programming a sensor node requires the developer to connect it to a programming board; the programming board is then connected to the computer via a RS232 or USB interface; then after the installation of its libraries and its executing environment, the developer can send commands (via a telnet interface) to the software process that controls the sensor node. Even the commands themselves require the understanding of the technology/algorithms involved or the scientific nature behind the sensor themselves (i.e. the command <TM> in Figure 3.1). The advantage here is that the developer has full control of every aspects of the sensor as he works closely with the hardware. Figure 3.1 show an example of low level commands of Crickets

The general command format is: `<directive> <command> <parameters>`. In response to any command, the Cricket echoes the specified command followed by the result: `<command><result>`

`<directive>` The character "G" or "P" for "getting" or "setting" a value.

`<command>` The command to perform

`<parameters>` The argument(s) to the command.

Some cricket commands:

`<TM>` getting the uncorrected time-of-flight of the ultrasonic pulse

`<DB>` distance to beacon

`<SV>` save setting to flash

**Figure 3.1. Low level commands from Crickets**

This approach does not meet any of the requirements except for the low overhead. It has a low-level of abstraction for the reasons above along with treating the sensors on a node-by-node basis. It is difficult to integrate with other applications as it requires a steep learning curve from the low level details and the unique commands. Synchronous access is usually the only mode of communication available since commands tend to minimize complexity. The same reasoning can be applied to security / publication / discovery mechanism.

Fortunately, most WSNs also provide a basic set of APIs which builds on top of the primitive commands. The set of APIs might be based on high level programming languages like JAVA, C++, and C; or specialized programming languages like Nesc [1]. Some commercially available products use this approach: MIT provides Crickets with a JAVA API, ClientLib [2]; Ember [53] provides Embernet [54], a set of APIs and some development tools for Ember WSNs; Sensoria [55] provides sGate [56], a sensor gateway product, with C APIs as part of their development platform. While this approach allows for a higher level of abstraction than primitive commands, it is still considered as a low level approach since current APIs are still dealing with low level details of the WSNs. These details involve specifying IP addresses, port numbers, enabling special flag parameters, following a special chain of programming operations and requiring the use of certain programming constructs. With the support of high level programming languages, the mode of communication with the sink/gateway node can be both asynchronous and synchronous, and addressing the WSNs can be done as a whole or on a node-by-node basis. In Embernet security features like encryption is supported, however it is not the case for all available products. The overhead is not affected by much if not at all, and publication/discovery features are ignored. Integration is done more easily: the developer needs to include the library files in his/her programming environment to access the sensed information. It does still however tie the programming languages of the APIs to the application to be developed. This approach also does not necessarily store the data in a persistent storage, but rather focuses on current data. To avoid the loss of historical data, it is up to the application to retrieve and store them explicitly.

### 3.2.2 Databases

This approach is the most popular so far. It consists in treating the sensor network as a logical relational database. Applications send an SQL-like query to the sink/gateway node, which in turns propagates the query into the sensor network in the case of a distributed implementation, or processes it directly in the case of a centralized implementation (assuming that the database is co-located at the sink/gateway node). In many cases the query language is an extension of SQL because traditional query languages are not suitable for WSN. While traditional query languages tend to minimize computation time and disk accesses, query languages for WSN tend to minimize power consumption by using aggregation [26][27] techniques. The sensor nodes have a database processor which remembers all the long-running queries (i.e. the average temperature of an office every *t* seconds) in the WSN. This allows continuous update of data without having to send a separate query each time. Figure 3.2 shows a simple query.

```
SELECT AVG (temperature), room FROM sensors
   WHERE floor = 6
   GROUP BY room
   HAVING AVG(temperature) > 10
   EPOCH DURATION 30s
```

**Figure 3.2. Example of sensor query**

*Query processing in distributed database* - The application sends a query to the sink/gateway node. A spanning tree rooted at the sink node of the network is formed and maintained as the WSN topology changes. The query is flooded in a controlled fashion within the WSN to notify a newly active query task. A leaf node reads its sensor data, applies the query and sends it to its parent. The parent node receives the sensed data from all of its child nodes, combines it with its own sensor reading and applies the query. The parent sends the in-network aggregated data to its parent and the process goes on until the sink node receives the final query results. Finally, the application receives the results as if it was querying a centralized database. This approach is common to TinyDB [3], TASK (uses TinyDB) [70], COUGAR [31], ALERT [35], SINA [69], IrisNet [59], Mica2 and Mica2dot.

TinyDB is a simple query processing engine for extracting data from WSNs. It runs on TinyOS and it is designed to greatly simplify many data collection tasks, and includes support for power management, time synchronization, and in-network storage.

In TASK the sink/gateway node is the Sensor Network Appliance (SNA), it includes a DBMS accessed via ODBC to log all collected data. Data can be accessed via a web browser or dedicated client software. COUGAR and ALERT developed their own query language as well as their own query processing optimizations. SINA and Irisnet both use non-standard query languages. SINA developed the Sensor Query and Tasking Languages, while Irisnet uses XPATH to represent queries. However, Irisnet uses XML database to store the sensed data and XML schema to represent it. Mica2 and Mica2dot use a standard DBMS to log incoming data. However, application queries are not injected in the WSN, but rather the individual sensor nodes periodically reports its sensed data to the sink (i.e. there is no query processing done at the nodes). On each of these reports, data is logged in a central database and application queries are directed there.

The database approach offers a high level of abstraction since dealing with the WSN can be seen as interacting with a relational database system. The details of the WSN are hidden to the application and addressing the WSN can be done in a data-centric fashion. Database triggers allow for asynchronous notifications and integrating database connectivity in application is generally easy. It does however tie the application to a particular database access method (i.e. ODBC, JDBC, OLE DB, etc) that the sink node is using. This is usually not a problem since most high level programming languages supports a wide selection of database access method. On the same matter, many of the mentioned database approaches use their own query languages and compatibility with standard DBMS systems is not entirely clear. Therefore integration varies but tends towards being difficult for these particular cases. The overhead of the DBMS at the sink node is usually low. Publication and discovery are not supported; however security mechanisms such as encryption and authentication are supported in some DBMS implementation. Finally, traditional database systems are not suitable for all types of ambient data. In the case of highly dynamic data such as location of moving objects, there are many issues at hand [38]. Location data has to be modeled in a different way, query language has to be enhanced to support spatial and temporal constraints and intelligent indexing techniques have to be provided to increase performance of location updates.

### 3.2.3 Mobile Code

In mobile code we consider mobile agents and active networks as part of this approach.

*Mobile Agents* - A mobile agent is a composition of software and data, which is able to migrate from one network node (or sensor node) to another autonomously and to continue its execution (data collection) on the destination network node (or sensor node). The mobile agent may duplicate itself and move the execution to several other nodes while communicating with existing agents. The application developer injects the mobile agent into the WSN at the sink node. Eventually, each of the sensor nodes will have an agent that migrated and collected sensed data which will be carried or transmitted back to the sink node. This programming paradigm requires a mobile agent platform to be running on the sensors node. Agilla [52] uses this approach and is a middleware built on top of TinyOS. It supports geographical addressing (data-centric) however it does not scale well as each sensor node can only have up to four agents running concurrently.

*Active Networks* - In active networks, the network node (or sensor node) can perform customized computations on, and modify, the content of a message flowing through them in response to some event (network event, sensing event, timeout, etc). In contrast, traditional packet networks just pass the user data from one node to another without modification or examination with the exception of the packet header. Since the computation done are customized on a per-user or per-application basis, sensor nodes can be instructed on the operations to be done on the sensed data as well as the received data from adjacent nodes. These instructions are injected in the WSN as programs or scripts and command the sensor nodes to return the collected data to the application that made the injection. SensorWare [4] uses this approach and instructions are specified in Tcl, a procedural programming language. This entails a run-time environment to be installed on the sensor nodes such a script interpreter.

The mobile code approach provides a high level of abstraction since it treats the WSN as a whole by injecting high level instructions at the sink node. However, writing the scripts or programs is much more difficult than writing a simple query as seen in the database approach. Not only does it requires to learn a specialized language but also the low level details of sensors as well as the paradigm itself (i.e. when to migrate, duplicate, stop collecting data, communicate with other running scripts). Appendix A shows a sample SensorWare script whose role is to find the maximal value sensed in the WSN. Integration with external application is rather difficult: mobile code is not a commonly known programming paradigm and it requires the knowledge of specific scripting/programming languages. Furthermore this approach is not appropriate for all types of WSN since it requires a specific run-time environment to be present on each individual node. Not all types of sensor may be able to operate a run-time environment due to their limited resources. Because of the great level of control in this approach, asynchronous communication is achieved by programming the script to return on the occurrence of an event. Security in mobile code is problematic since the code needs to be protected as well as the node running the code. Security issues are not addressed in this approach. This also holds true for publication and discovery.

### 3.2.4 Web Services

The Web service approach has been adopted by the Open Geospatial Consortium [8] (OGC) in their Sensor Web vision. OGC is a non-profit and international organization that is leading the development of standards for geospatial and location based services. The Common Instrument Middleware Architecture [50] (CIMA) also uses Web services to bring scientific instruments and sensors on the Grid to allow different scientific communities to access the data as it is being collected. The Grid is a community of large scale, non-profit research projects of global significance involving the coordination and sharing of data, computing and network resources across different organizations. Tendril [72] and Ember are developing Web service support [73] for the Zigbee [71] protocol. Zigbee is a wireless standard-based radio-technology that addresses the unique needs of remote monitoring and control and sensor network applications.

One possible configuration with Web service as suggested in [61] is to model the sink node as the service provider and applications as clients of such services. The sink node also performs a secondary role: it acts as a service registry where sensor nodes publish their data and functionality to the sink node. Binary XML [75] is used to lighten the communication. The service registry contains two types of service description: one is provided by the sensor node for the sink node, the other is provided by the sink node for applications. Possible operations done at the nodes are as follow: Publish_content(), used by the sensor node to send a SOAP message containing its service description, Publish_data(), used by the sensor node to send a SOAP message containing the data, Subscribe_interest() used by an application to submit a query to a sink node, Subscribe_filter() used by an application to inject a filter in the WSN.

Subscribe_interest() must specify the sensor type, data type, geographical region of interest, acquisition interval/duration and real-time constraints (i.e. deadlines). Data disseminated to applications contains the data type, data value, sensor location, confidence data collected, timestamp and the current sensor amount of energy.

The dual role of the sink node is not used in CIMA or OGC Sensor Web. Instead, they simply expose the sensed data as Web services at the sink node. Nothing else is assumed about roles of the sensor node and sink node within the sensor network itself. The implementation of these Web services implies mapping of APIs or other framework approaches to the Web services.

The benefits of using Web services have been discussed in Chapter 2 and apply to this approach. It is evident that high level of abstraction, ease of integration, asynchronous mode of communication and publication/discovery mechanism are provided. There are several specifications to provide security mechanisms to this approach: XML encryption [67] from W3C, XML Signature [66] from W3C and IETF, Web service Security [5] (WS-security) from the Organization for the Advancement of Structured Information Standards (OASIS) and many more. The main drawback of this approach is its high overhead.

*OGC Sensor Web* - OGC Sensor Web is a web-centric, open, interconnected and dynamic network of sensors that presents a new vision on how we deploy, collect, fuse and distribute sensor information. OGC is defining the standard specifications [22] to enable the sensor web vision (Sensor Web Enablement Initiative). Sensor Web focuses on environmental data related to Earth's ecosystem (i.e. pollution, temperature, water levels). Their work aims at solving the interoperability of sensor networks by defining standard interfaces for sensor data and enabling their ubiquitous access through the Internet. Specification documents are being developed in the areas of information modeling and dissemination. Sensor Modeling Language (SensorML) provide a standard XML schema that describe sensors and sensor networks as functional models with inputs, outputs, parameters and observation methods. It describes the sensor identification (ID, name, sensor type), the observable that can be measured, the sensor response characteristics (sensitivity, range, quality), the information on the operator of the WSN and other metadata (manufacturer, model name). On the other hand, the Observations & Measurements specifications (O&M) provide the general models as well as the XML encodings of the sensor observations and measurements (i.e. value, coverage, timestamp, observation type, units, etc). The Sensor Observation Service (SOS) specifies the set of Web services involved in the dissemination of sensors observations.

GetObservation() defined in the SOS, returns the sensor data to the entity requesting it. GetObservation() addresses the WSN by geographical region with the aid of a rectangular bounding box covering the area to be sensed. It can also address them by sensorID. For geophysical applications, it is reasonable to address the WSN by geographical region in order to get the corresponding ambient data. For I-centric applications this is insufficient because addressing ambient information should be done in terms of individual user. Just like getting location information or presence information from an identity, ambient information should also be provided from an identity or user perspective. There are three reasons for doing so. First, an interface using SensorID or a bounding box implies that ambient information comes from sensors. A high level interface should not tie the source of ambient information solely to sensors (or a particular technology). Ideally the developer should not need to know or care about the source of this ambient information. Second, without the identity perspective, requesting ambient information from a user point of view requires two operations. It requires first knowing the location of the user, then using that geographic data to map to a bounding box or sensorIDs to retrieve ambient information perceived around that user. A high level interface should only abstract the dissemination in one operation. Third, the idea of a rectangular bounding box does not work well in an urban setting where areas of interest are being juxtaposed (i.e. multiple floors in a building). Hence OGC Web services are defined too specifically for geospatial applications. While the general approach seems promising, new interfaces should be specified to focus more on I-centric needs. To further motivate this need, OGC Sensor Web Enablement Initiative mainly focuses on environmental data and does not cover location data and its derivative. Spatial data is primordial to I-centric applications and should be part of the new interfaces.

### 3.2.5 Others

*Shaman* - Shaman [58] is an extendable java-based service gateway that integrates sensor-actuator modules, or simply sensors, into heterogeneous ad-hoc networks. Shaman supports multiple service wrappers (Jini wrapper, Universal Plug and Play (UPnP) wrapper, etc) allowing multiple service interfaces to be available to heterogeneous clients. Figure 3.3 illustrates Shaman main concepts. The boot protocol contains a simple discovery mechanism that lets the sensor find a gateway to deploy its proxy and publish its capabilities. The proxy translates client requests into sensor specific commands. The proxy can be provided by the sensor or the network (i.e. downloaded from an URL). The service wrapper exposes the sensor functionalities to the client.

Service wrappers provide a high level of abstraction, and depending on the wrapper available, it may ease integration with other applications. However Shaman only discusses Jini and UPnP wrappers. Applications can discover sensor services with the service manager. Security is not addressed and a drawback of this system is that sensors must implement a boot protocol and be able to deploy a java-based network proxy to a gateway host. Shaman has a high level of abstraction since it claims to have plug-and-work capabilities. Because of the presence of a gateway host and high level service wrappers, overhead is assumed to be high and asynchronous communication is assumed to be supported.

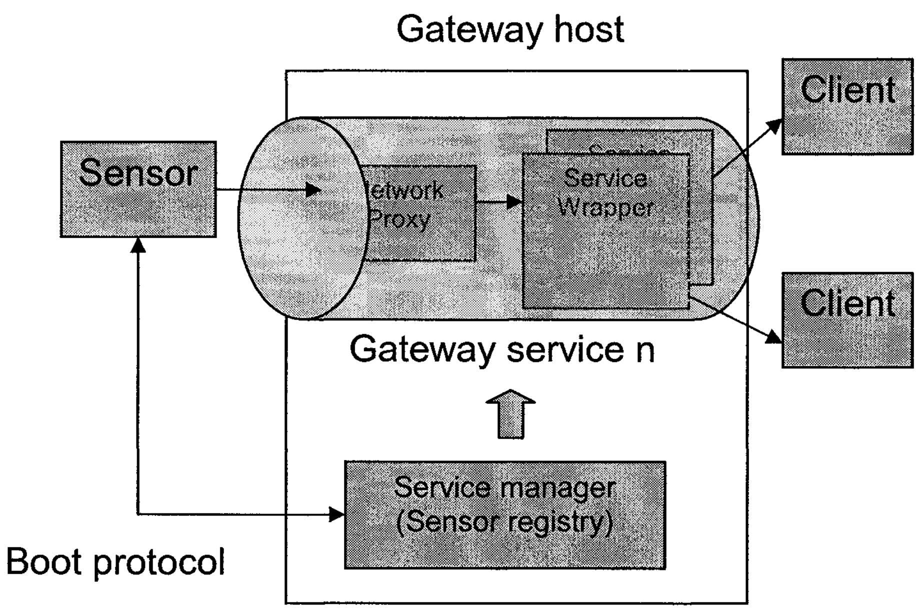


**Figure 3.3. Shaman service gateway**

*DSWare* - This [62] approach uses the notion of events. An application can subscribe to certain events and when they occur, the application is notified. Subscribing to events of interest is done in an SQL-like statement (Figure 3.4). It also allows the specification of compound events or a pattern of basic events to be detected. Once an event request is issued, DSware generates an execution plan and calls the corresponding methods for the event detection. DSware introduces novel ideas such as caching the most requested data and adding redundant storage within the nodes. The system also includes real-time aspects such as deadlines for reporting events and a window of validity for the events.

This approach has a high level of abstraction, a low overhead and supports asynchronous mode of communication. DSware is used with traditional database and with traditional query language. In this case, integration is assumed to be easy. Publication, discovery and security are not discussed but are assumed to be non existent.

```
INSERT INTO EVENT_LIST
      (EVENT_ID, RANGE_TYPE, DETECTING_RANGE,
            SUBEVENT_SET, REGISTRANT_SET, REPORT_DEADLINE,
            DETECTION_DURATION [, SPATIAL_RESOLUTION]
            [, ACTIONS])
VALUES ()
```

**Figure 3.4. A SQL-like subscribe statement**

*Mires* - Mires [60] is a message oriented middleware for WSN. The sensor nodes advertise their available topics (e.g. temperature, light, etc). The advertised messages are routed to the sink node where user applications can subscribe to topics that interest them. Then the sensor nodes continue to publish their sensed data to Mires and applications use APIs to communicate with Mires.

This approach has publication and discovery mechanisms. While it supports asynchronous mode of communication, security features are not discussed. Also, the nature of message-oriented middleware may implicate high overhead. The APIs provides a high level of abstraction but ties to a proprietary communication protocol/interface, a message format and a programming language.

*Mate* - Mate [63] is a virtual-machine (VM) based middleware for WSN. The byte-code interpreter is build on top of TinyOS. To use the VM, mate provides an instruction set reminiscent of the Assembly language. Mate provides an architecture that allows the development of higher level languages and programming models for application development. Using mate directly, the only requirement that can be met is good performance.

## 3.3 Data Dissemination Frameworks Using Other Technologies

This section discusses some of the work pertinent to the Web service approach and to ambient information but not necessarily in a WSN context. The work described is specific to outdoor location information only.

*Parlay-X* - Chapter 2 briefly discusses that Parlay-X [7] is a set of telecommunication capabilities exposed as Web services. Parlay-X defines a basic Web service interface to get the location of one or many terminals expressed in latitude, longitude and optionally altitude. The terminal location is provided immediately or periodically. It also support notification of a change in the location of a terminal to monitor if a terminal is entering or leaving a specified geographical region. Other than terminal location, Parlay-X also provides online presence.

*LORE* - LORE [15] is a framework for developing location-aware applications. It wraps heterogeneous location APIs into one consistent and simple interface. LORE focuses on large scale management of location data by including a Moving Object Database and a Query Engine. This enables LORE to handle complex services (i.e. proximity detection, physical presence) as queries. The framework also provide additional functionality such as specifying complex notifications (i.e. receive a notification only when a particular user is in a particular geographical region) and a service management framework (i.e. for privacy, billing, etc).

## *3.4 Summary*

There are many frameworks available for disseminating ambient information. Each of them adopts a different approach in the dissemination and hence has different characteristics. Table 3.1 summarizes these characteristics and is an extended version of the framework summary in [74]. None of these approaches fully satisfy all of the defined requirements. The low level commands and APIs are either too low level or programming language dependant. In database frameworks there is no discovery/publication mechanism. In mobile code frameworks, integration is difficult since it requires knowledge of WSN details, specialized programming languages and understanding mobile code paradigm. There is also no support for discovery/publication mechanisms. Other approaches exist but they lack security mechanisms or depend on a particular programming language. The Web service approach seems to be the most promising since it satisfies most of the main requirements. However, the current specifications (OGC sensor web) are not suited for an I-centric application development but rather a geophysical application development. So there is a need for new Web service interfaces for disseminating ambient information specifically targeted to I-centric applications.

**Table 3.1. Summary of the existing dissemination frameworks**

| | High level of abstraction | Interoperability and ease of integration | Flexible business model | | Synch / Asynch communication | Low overhead |
|---|---|---|---|---|---|---|
| | | | Security | Disc. / Pub. | | |
| Commands | No | No | No | No | No | Yes |
| APIs | No | No | No | No | Yes | Yes |
| Database | Yes | No | Yes | No | Yes | Yes |
| Mobile code | Yes | No | No | No | Yes | Yes |
| Web Service | Yes | Yes | Yes | Yes | Yes | No |
| Shaman | Yes | No | No | Yes | Yes | No |
| DSware | Yes | Yes | No | No | Yes | Yes |
| Mires | Yes | No | No | Yes | Yes | No |
| Mate | No | No | No | No | No | Yes |

# Chapter 4 Design of a Web Service Based Data Dissemination Framework for an I-centric Environment

The previous section showed that a Web service based framework is the most promising approach for the dissemination of ambient information. A Web service based framework provides language, platform and protocol independent interfaces to disseminate data. However, the existing Web service framework does not provide an interface that is sufficiently appropriate for I-centric applications. This chapter proposes a set of Web service methods to satisfy this need. The first section overviews the role model involved, followed by a definition of the world model and necessary concepts, and the proposed interfaces are detailed before the description of the framework architecture.

## *4.1 Role Model*

Since the framework is based on Web services, the entities involved would share similar roles to the Web service architectural model. The roles of the ambient information provider, ambient information requestor and ambient information provider registry map into the roles of service provider, service requestor and service registry respectively [Figure 4.1]. The ambient information provider can be an owner of a WSN and provides sensor data as services to I-centric applications. However the model is not restrictive to WSN and should be able to provide ambient data form other sources as well. The ambient information requestors, or simply I-centric applications, can search the ambient information provider registry for a suitable ambient information provider. In light of the requirements [Section 3.1], the ambient awareness framework will play the role of the ambient information provider and the ambient information provider registry.

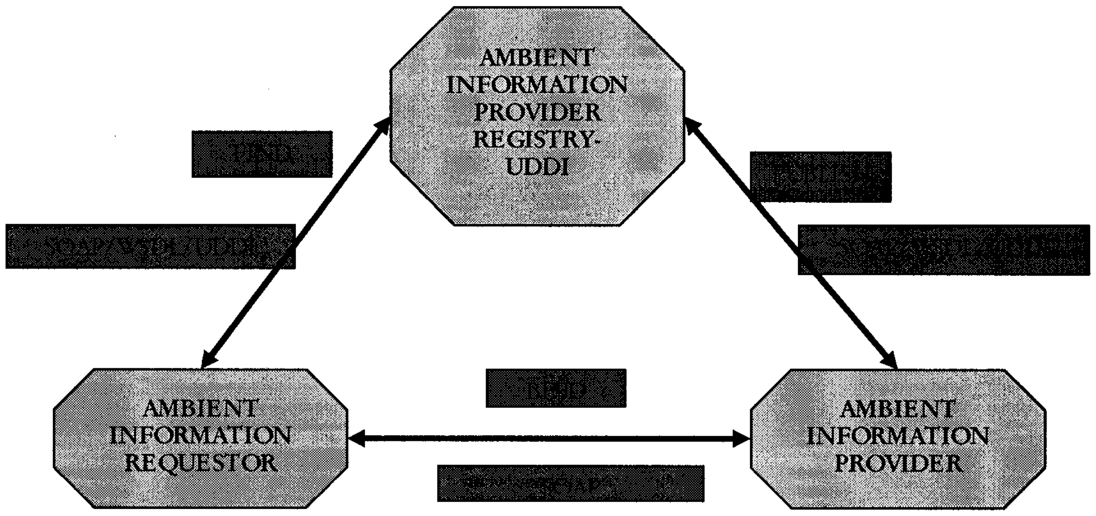

AMBIENT INFORMATION PROVIDER REGISTRY-UDDI
FIND
SOAP/WSDL/UDDI
SOAP/WSDL/UDDI
AMBIENT INFORMATION REQUESTOR
BIND
AMBIENT INFORMATION PROVIDER

**Figure 4.1. Role model for the ambient awareness framework**

## 4.2 The World Model

A world model is necessary in order to establish a common ground for the interpretation of spatial and environmental information. This preliminary step introduces simple concepts that will be used as parameters in the proposed Web service methods. *Space*, *Entity* and *Observable* are the three basic concepts that define the world model. A Space is a static three-dimensional expanse defined by a unique center, range (or radius) and a short description of the Space. As illustrated in Figure 4.2, an office room may have a space (SpaceID4) centered in the middle of the office and where the range is the distance to one of the walls. For describing more complex spaces such as the Electrical and Computer Engineering department of an educational institution or an 'L'-shaped room, the term *Area* is used to define a collection of Spaces. The center of the Space is defined in three dimensional Cartesian coordinates with a defined reference origin; however any coordinate system can be used. An Area has an address field and a short description. The address is defined to be the address of the establishment in which the spaces are located. Since this model deals with a three-dimensional coordinate system, it fits well in an urban setting and is not limited to the two-dimensional bounding box problem as outlined in section 3.2.4. An Entity is an object that can be located and that has an owner. An Entity has a type, such as laptop, PDA, printer or user; and a short description. While an Entity can be classified by its type, ownership information is used to further classify an Entity. An owner can be an individual or an organization. Ownership information can also be used as a part of a permission/authorization mechanism for getting location information. An Observable is a sensed and measured environmental phenomenon (i.e. temperature, the ambient light, etc). There are many ways to quantify and express an observable: ambient sound can be expressed as a value such as sound intensity (decibels), as a high level description such as "loud", as an audio sample of the ambient sound, or even as a video sample. In this world model, Observables are limited to measured values and high-level descriptions. An Observable should have a name, a description and the associated units. For instance, the name could be "Temperature", the description could be "ambient air temperature with humidity compensated" and units are "Celsius".

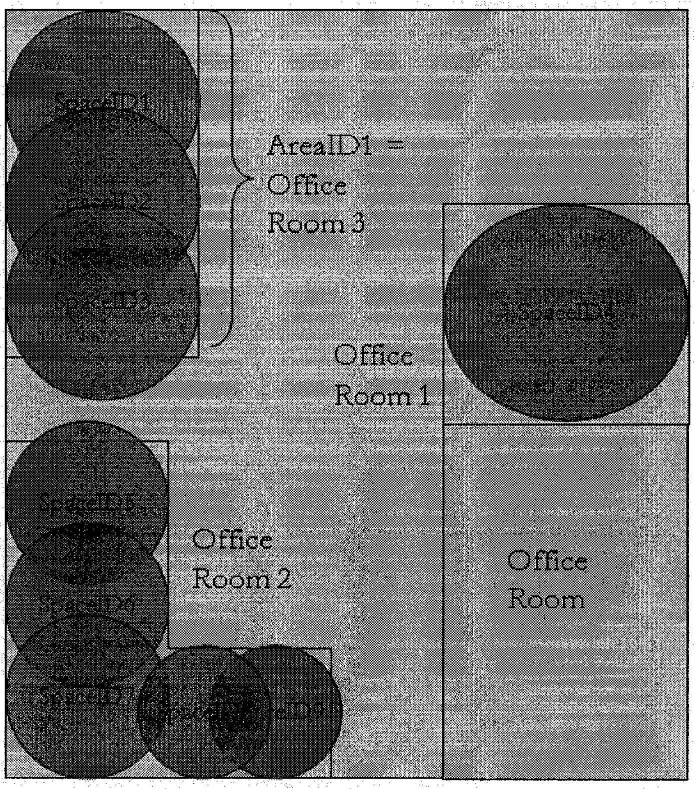


**Figure 4.2. Space and Area concepts**

## *4.3 Proposed Interfaces*

Six Web service methods are proposed in order to provide spatial and environmental information. Although physiological information is a subset of ambient information, it is not covered in the proposed interfaces. Instead, the proposed interfaces focus on commonly used and available ambient information. Figure 4.3 overviews the proposed Web service methods. Subscribe_Location() and Subscribe_AreaEnvironmentalData() are basic Web services, while the rest are complex Web services that may be reusing the core logic of the first two. Subscribe_Location() gives the location of an Entity while Subscribe_AreaEnvironmentalData() gives the environmental conditions of an Area. For complex spatial information, Subscribe_Velocity() gives the velocity of an Entity, Subscribe_Proximity() lists all Entities that are nearby a particular Entity, and Subscribe_PhysicalPresence() lists all Entities physically present in a confined Space. Subscribe_EntityEnvironmentalData() returns the environmental conditions perceived by a particular Entity. Applications can access both basic and complex services. For each of the defined methods, there is a corresponding unsubscribe() method which terminates the Web service, and a callback method, onUpdate(), which notifies the subscribed application of a change in the ambient information.

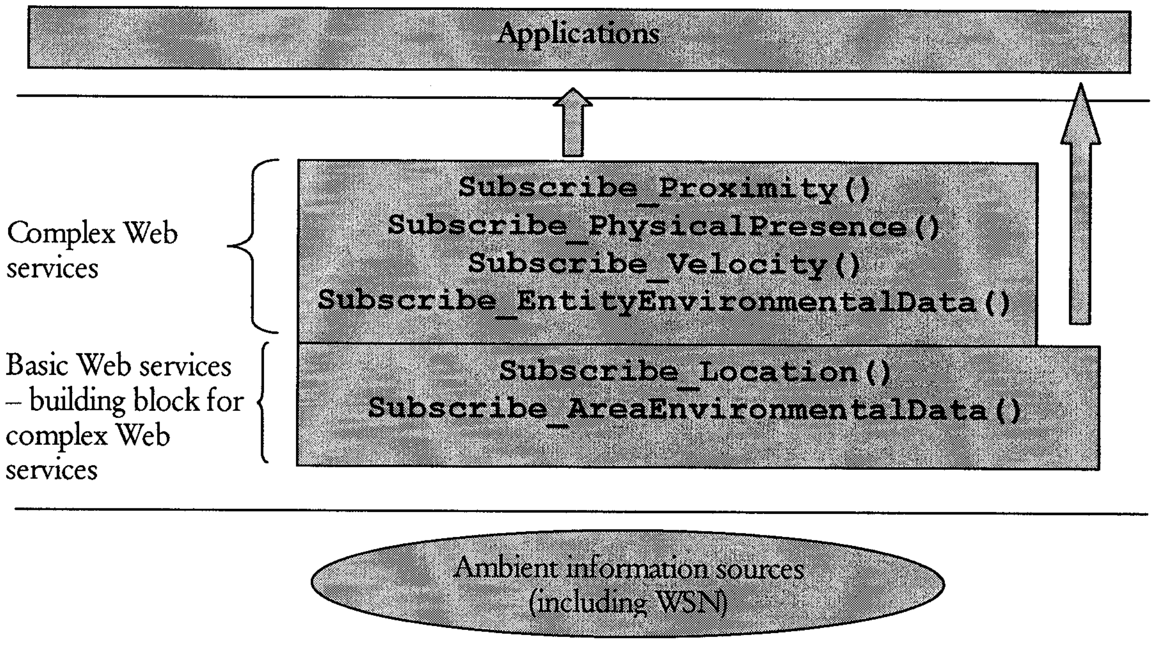

Applications
Complex Web services
Subscribe_Proximity()
Subscribe_PhysicalPresence()
Subscribe_Velocity()
Subscribe_EntityEnvironmentalData()
Basic Web services – building block for complex Web services
Subscribe_Location()
Subscribe_AreaEnvironmentalData()
Ambient information sources (including WSN)

**Figure 4.3. Web services interfaces overview**

## 4.3.1 Common Parameters of the Interfaces

The common parameters for these Web services are the service metadata parameters. *QualityOfContext* defines the desired minimum freshness of the sensed information in milliseconds. All sensed information is timestamped and thus the framework can determine the age of the sensed data. If the specified *QualityOfContext* is high, the framework can reused a previously sensed data as long as it is less than the specified *QualityOfContext*. If this is not the case, the framework will have to acquire fresh data from the WSN. The service response always contains the measured *QualityOfContext* in milliseconds. Because of the dynamicity of ambient information, *QualityOfContext* is a valuable data for determining exactness and reliability of the sensed information. *OnetimeOnly* specifies if the Web service should automatically unsubscribe after sending one response as opposed to continuously sending updates whenever they occur. *NotificationTrigger* specifies a condition, based on the value of the sensed data, in which a notification (or service response) is necessary. A *NotificationTriggeChangeSensitivity* is also provided to determine when the condition occurs: it specifies the minimum amount of variation in the sensed value that will trigger a notification. For instance, specifying that a notification is sent only if the ambient temperature changes by five degrees Celsius. In the example, *NotificationTrigger* is the 'change in temperature measurement' and *NotificationTriggeChangeSensitivity* is 'five degrees'. A *NotificationTrigger* allows filtering of uninteresting data and its event-driven approach ensures timely adaptation of applications. *RateOfNotification* indicates the maximum and minimum rate for the notifications or service responses. A maximum rate is needed to avoid overflowing the requester with responses. If a minimum rate is specified, it will supersede the *NotificationTrigger*: instead of having event-driven notifications, notifications are sent periodically. Although it is possible to have both approaches coexist, an exclusive approach is adopted for simplicity. The minimum and maximum rate is specified in number of notifications per minute. *Granularity* specifies the desired level of detail in the presentation of the ambient data. For instance, ambient temperature has a high granularity when expressed as 'HOT', and low granularity when expressed as a measurement in degree Celsius. *UnitsType* is simply the metric used in the representation of the sensed data in the request and the response.

### 4.3.2 Subscribe_Location()

*Subscribe_Location(int QoC, boolean OneTimeOnly, int MinRateOfNotification, int MaxRateOfNotification, String NotificationTrigger, int NotificationTriggerChangeSensitivity, String UnitsType, String Granularity, long EntityID, long AreaID)* – This method reports the location in terms of the current Space closest to the Entity or the current physical coordinates of the Entity. While location can be represented solely by physical coordinates, adding another representation to it, such as a Space, provides a more meaningful view of location to the user. A Space is expressed as a high level description such as "ECE-office512-EastWall" and the coordinate system is the same as the one in the world model. From the notifications, the method informs the invoker when the Entity moves from one Space to another or within the same Space.

Table 4.1 details selected parameters for this method.

**Table 4.1. Selected parameters of Subscribe_Location()**

| Name | Details | Possible values |
|---|---|---|
| NotificationTrigger | Triggers a notification based on changes in location coordinates or Space | "CoordChanges", "SpaceChanges" |
| NotificationTriggerChangeSensitivity | Valid only when the NotificationTrigger is "CoordChanges". It is the minimum distance change from one location notification to another | any |
| UnitsType | Valid only when NotificationTrigger is "CoordChanges", or when Granularity is "Coordinates". Units for location coordinates | "cm", "m", "inch" |
| Granularity | Location is presented in terms of coordinates or Space description | "Coordinates", "Space" |
| EntityID | Specifies the Entity which needs to be located | any |
| AreaID | Specifies that any notification should be sent only if EntityID is in AreaID | any |

### 4.3.3 Subscribe_AreaEnvironmentalData()

*Subscribe_AreaEnvironmentalData(int QoC, boolean OneTimeOnly, int MinRateOfNotification, int MaxRateOfNotification, WSEnvData[] envreq, long AreaID)* – This method reports the environmental conditions of a particular Area. Since there is much environmental information to sense, this method groups them into one enumerated parameter (envreq). Each WSEnvData entry contains all necessary specifications to request one type of Observable. This adds flexibility to the method since the invoker can select the ones it needs and allows future expansion of the supported Observables without modifying the interface definition. The information returned for each Observable is a single scientific measurement or a high level description (i.e. for ambient light there is "dark", "dim" and "bright"). It is assumed that the developer is not necessarily familiar with scientific measurements and their quantitative meaning (i.e. how much brightness is 90 lumens). Additionally sensors may have proprietary metrics for the measurements of data (i.e. use a scale of 0 to 100 for luminosity) and it is not obvious how to map them into standard metrics. These reasons motivate the addition of a high level description for environmental information. The reception of a notification can indicate three events: current data differs from previous by at least a certain value (NotificationTriggerChangeSensitivity), current data has exceeded or has gone below a specified threshold, or current data differs from previous data by its high level description (i.e. from 'dark' to 'bright').

Table 4.2 details selected parameters for this method.

**Table 4.2. Selected parameters of Subscribe_AreaEnvironmentalData()**

| Name | Details | Possible values |
|---|---|---|
| ObservableID | The type of monitored observation | any |
| NotificationTrigger | Triggers a notification based on a change in measured value, a reached threshold value or change in the description of the sensed information | "ValueChanges", "Threshold", "descriptionChange" |
| NotificationTriggerChangeSensitivity | Triggers a notification when the sensed data varies by at least this value | any |
| thresholdval | A notification is triggered if the sensed value exceed or goes below thresholdval | any |
| UnitsType | The type of units for NotificationTriggerChangeSensitivity, thresholdval and the service response | "Celsius", "Kelvin", "lumens", "Decibels" |
| Granularity | Environment data is presented in terms of high level description or scientific measurement | "highLevel", "ScientificMeasurments" |
| WSEnvData[] | An enumeration of observation to be monitored {ObservableID, NotificationTrigger, NotificationTriggerChangeSensitivity, UnitsType, Granularity, thresholdval} | any |
| AreaID | Area to be monitored | any |

### 4.3.4 Subscribe_Proximity()

*Subscribe_Proximity(int QoC, boolean OneTimeOnly, int MinRateOfNotification, int MaxRateOfNotification, String NotificationTrigger, String UnitsType, String Granularity, long EntityID, int MaxResults, int MaxRange, String[] Filters)* – This method reports a list of sorted Entities closest to the specified Entity and within a defined range. When the entity moves from one place to another, its proximity information changes. The resulting notifications may be that a new entity is detected in the circle of proximity, or an entity has left. The list may also reorder itself (entities are moving within the proximity circle) and result in a notification. Proximity information is useful to redirect an application session to a more suitable device (closer to the user, better computing resources, etc) or printing to the nearest printer. It provides awareness of all entities in the communication space of the user. In terms of information granularity, proximity is expressed as distance measurements (i.e. printer at 7 meters) or relative measurements (i.e. 'NEAR', 'FAR') with respect to the defined range. Mechanisms to limit the amount of proximity information are provided by specifying a maximum list size and by filtering the Entities by their type.

**Table 4.3. Selected parameters of Subscribe_Proximity()**

| Name | Details | Possible values |
|---|---|---|
| NotificationTrigger | Triggers a notification based on a new Entity entering/leaving or on changes in the list of closest entities | "EntityEnteringOrLeaving", "ListReorder" |
| UnitsType | Units for the proximity measurements | "cm", "m" , "inch" |
| Granularity | proximity measurements is presented in terms of distance or description | "distanceMeasurements" , "relativeMeasurements" |
| EntityID | Specifies the Entity which proximity information should be based on | any |
| MaxResults | The maximum number of results returned | any |
| MaxRange | The radius around the Entity which determine relevant proximity events | any |
| Filters[] | specifies which type of Entities should be included in the proximity list | "Laptops", "PDA", "Projector", "desktop", "printer", "users" |

### 4.3.5 Subscribe_PhysicalPresence()

*Subscribe_PhysicalPresence(int QoC, boolean OneTimeOnly, int MinRateOfNotification, int MaxRateOfNotification, String NotificationTrigger, String UnitsType, String Granularity, long AreaID, int MaxResults, String[] Filters)* – This method reports a list of all entities and their location in a specified Area. It is in some respect similar to Subscribe_Proximity(), except that it dissociates between an entity entering and leaving, and it reports a location instead of a distance. The granularity of the location information is similar to Subscribe_Location(). A problem with Subscribe_Proximity() is that it requires an Entity to detect other entities nearby. Subscribe_PhysicalPresence() does not have this problem since it is based on an existing Area. This method is useful for an I-centric service provider who wishes to offer services to Entities that are nearby a particular Area. For instance, an I-centric service provider can send targeted ads to users that are nearby a store section. The ads target the products within the vicinity of the user since the user may show interest in them by being in that store section.

**Table 4.4. Selected parameters of Subscribe_PhysicalPresence()**

| Name | Details | Possible values |
|---|---|---|
| NotificationTrigger | Triggers a notification based on a new Entity entering or an leaving the Area | “EntityEntered”, “EntityLeft” |
| UnitsType | Valid when Granularity is “Coordinates”. Specifies the units for location coordinates. | “cm”, “m” and “inch” |
| Granularity | Physical presence is expressed as location in terms of coordinates or Space description | “Coordinates”, “Space” |
| AreaID | Area in which physical presence is monitored | any |
| MaxResults | The maximum number of results returned | any |
| Filters[] | Specifies which type of Entity should be included in the physical presence information. | “Laptops”, “PDA”, “Projector”, “desktop”, “printer”, “users” |

### 4.3.6 Subscribe_Velocity()

*Subscribe_Velocity(int QoC, boolean OneTimeOnly, int MinRateOfNotification, int MaxRateOfNotification, String NotificationTrigger, int NotTriggerChangeSensitivity, String UnitsType, String Granularity, long EntityID)* – This method reports the velocity in magnitude and direction of an Entity. The magnitude is expressed a speed value (i.e. 1 m/s) while the direction is a vector in the coordinate system of the world model. Using different granularity, the velocity magnitude is expressed as a state such as ‘Stationary’, ‘walking’, ‘running’, and ‘driving’. As for the velocity vector, it can be mapped into cardinal directions (North/South/East/West) since the coordinate system is known. While velocity is information less often used than location, it has its meaningful use in some applications. As preference settings, the user should be able to disable incoming calls when he is running or driving. Alerts or application notifications should be postponed to not distract the busy user. In some other applications such as the targeted ad described in section 4.3.5, ads should be sent only if the user is in a stationary state within the store since it may show that the user is interested in the surrounding products.

**Table 4.5. Selected parameters of Subscribe_Velocity()**

| Name | Details | Possible values |
|---|---|---|
| NotificationTrigger | Triggers a notification based on changes in velocity magnitude or velocity state | "VelocityChanges" and "VelocityStateChanges" |
| NotificationTriggerChangeSensitivity | Valid only when the NotificationTrigger is "VelocityChanges". | any |
| UnitsType | Valid only when NotificationTrigger is "VelocityChanges", or when Granularity is "Velocity". | "cm/s", "m/s" and "inch/s" |
| Granularity | Velocity is presented in terms of speed measurements or high level description | "Velocity" and "VelocityState" |
| EntityID | Entity whose velocity is being monitored | any |

### 4.3.7 Subscribe_EntityEnvironmentalData()

*Subscribe_EntityEnvironmentalData(int QoC, boolean OneTimeOnly, int MinRateOfNotification, int MaxRateOfNotification, WSEnvData[] envreq, long EntityID)* – This method reports the environmental data seen or perceived around an Entity. It is the I-centric version of Subscribe_AreaEnvironmentalData(), instead of specifying a fixed area to monitor environmental conditions, it monitors the current surroundings of an Entity. All other parameters and returned data are the same. This method provides a higher level of abstraction: instead of requesting the Entity's location and then requesting environmental data based on that location, it does both steps in one method invocation.

**Table 4.6. Selected parameters of Subscribe_EntityEnvironmentalData()**

| Name | Details | Possible values |
|---|---|---|
| ObservableID, NotificationTrigger, NotificationTriggerChangeSensitivity, Thresholdval, UnitsType, Granularity, WSEnvData[] | See Table 4.2 for respective details | See Table 4.2 for respective details |
| EntityID | Entity whose environmental data should be monitored | Any |

## 4.4 *Functional Architecture of the Proposed Web Service Based Framework*

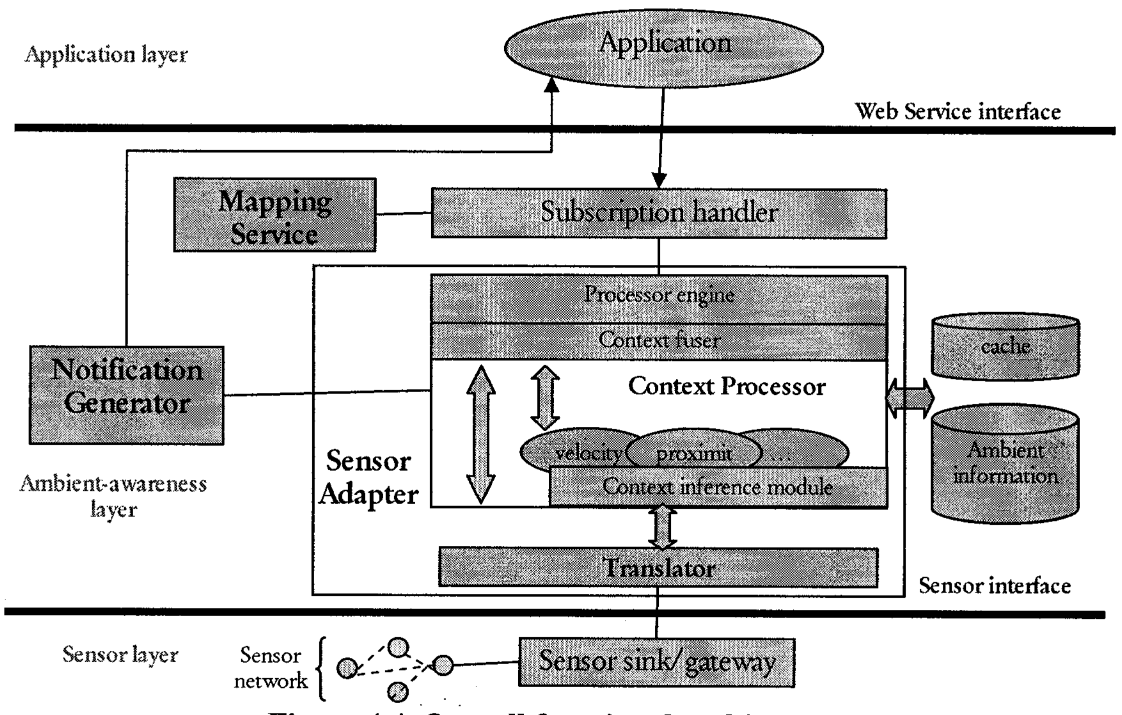


**Figure 4.4. Overall functional architecture**

The previous sections describe the interfaces; this section is about the framework that exposes these interfaces. The framework does more than just exposing the Web services to applications. Its main function is to wrap different data dissemination mechanisms for sensors (APIs, Databases, etc) into one consistent and simple Web service interface. Others functions include supporting the framework with an ambient information cache, logging sensed data for historical purposes, enhancing ambient information by fusing together different sources into a more coherent result, and adding ambient information by inferring from available sources. It is assumed that the raw data of sensors may not directly map into ambient information used by applications, hence most of the supporting functions relate to manipulating sensor data. Figure 4.4 shows the overall functional architecture. The main components are the Notification Generator, the

Sensor Adapter and the Mapping Service. The Notification Generator builds and sends the notifications to the application. It also keeps track of the previous notification sent so the Context Processor can determine if a NotificationTrigger is satisfied or not. The heart of the framework, the Sensor Adapter, is composed of a Translator and a Context Processor. Sensor-specific data is sent to the Translator and converted into a data format that can be used by the Context Processor. Same ambient information type but from different sensor model or manufacturer should be converted into this same format. Some ambient information may not be converted into this format directly (i.e. a proprietary scale for ambient sound). In this case, algorithms or approximations are used to convert ambient information into the desired format. The Context Processor encompasses three roles — performed by the processor engine, the context fuser and the context inference module. The context fuser retrieves ambient information from persistent storage and integrates it with the results from the Translator or the context inference module into one single, coherent and more reliable result. The role of the context fuser becomes more important as the number of sources of the same type of ambient information increases. Fusing should be considered as optional. For instance, an application requires a high degree of reliability that cannot be met from a single ambient information source. Each measurement has an error margin and multiple measurements should decrease this margin effectively. The processor engine monitors the ambient information and generates the notification events based on the specified NotificationTrigger. The context inference module is the logic responsible for computing high level ambient information based on low-level ambient information sources. For instance, location is most of the time inferred from proximity. From one Translator to another, different context inference modules are enabled depending on the type of ambient information available. Mapping the world model elements such as Entities, Spaces and Observables to elements in the sensor world is done by the Mapping Service. For instance, an Entity might map to a SensorNodeID or any other necessary data for the retrieval of the sensor reading. The Mapping Service also determines which Translator and/or which context inference module is needed to deliver the ambient information. In some cases it maps into more than one source, for example, velocity can be sensed by an accelerometer sensor or inferred by sampling several instances of location data at a known time interval. The Subscription Handler receives the Web service requests and, based on the mapping service, dispatches them to the Sensor Adapter with an appropriate Translator and context inference module.

## 4.5 Summary

Designed with a high level of abstraction in mind, six Web service methods are proposed for providing ambient information. These methods provide location, velocity, proximity, and environmental information of an Entity. They also provide physical presence and environmental information of an Area. Since the methods are centered on the concept of Entity which may be a user, they are highly applicable in an I-centric environment. A generic functional architecture is defined and will be used for the implementation section that follows.

# Chapter 5 Prototype Implementation and Performance Evaluation

This chapter is broken down into three sub-sections. The first talks about the implementation including the backend sensors, the mapping of the ambient awareness framework to those sensors and some optimization schemes used or proposed. The second sub-section deals with performance measurements and analysis. The last sub-section overviews an application implemented for a demonstration.

## 5.1 Prototype Implementation

All of the components in the functional architecture are implemented with the exception of the context fuser. The context fuser requires more than one source of the same type of ambient information in order to fuse them together. This functionality is judged to be too complex since it requires defining a methodology for fusing data in general (i.e. how to fuse 53°C from sensor X and 'HOT' from sensor Z, when to fuse, etc).

The system also implements a shared cache for optimizing (discussed in section 5.1.3) response time and a database for logging historical data. The historical database contains only spatial data. Every time sensor data is received, this data is updated in the cache as well as a temporary historical buffer in the main memory. Once the buffer is full, the system does a bulk write to add the new entries into the historical database. This is done to avoid continuous disk write operation each time there is new sensor data, thus increasing the overall response time. The historical database can be use to predict the mobility of an Entity in order to anticipate and adapt; or can be use for learning the Entity preferences. The historical database does not contain environmental data since it is already implemented as part of the software 'package' provided by the backend sensors.

A generic approach common to the implementation of the six Web service methods is to break them into two main JAVA classes. One that handles the high level details of the Web service [Figure 5.1] and another that handles the low level details of the sensors [Figure 5.2]. The figures describe ProximityWS and ProximityModuleImpl but the same logic is applied to all other Web services. For simplicity, the two classes are referred to as "higher class" and "lower class" for the rest of the explanation. One advantage of separating into two classes at that level is that the programming model creates a clear separation between how data is acquired and how it is used. Hence, adding a new WSN for implementing an existing Web service simply requires implementing a Translator or Context Inference Module since they map into the "lower class".

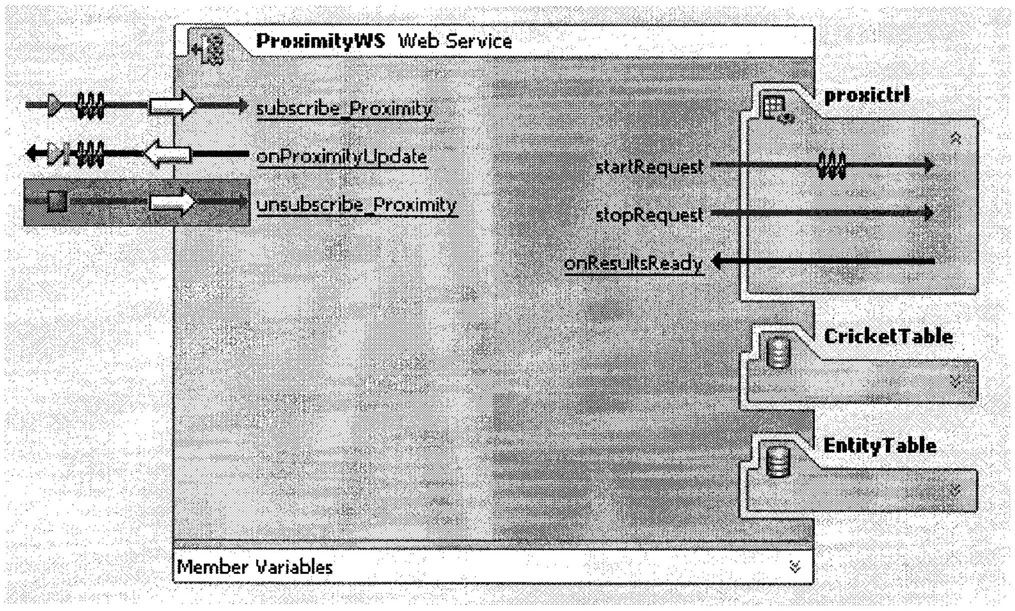


**Figure 5.1. 'higher class' for handling high level details of the Web service**

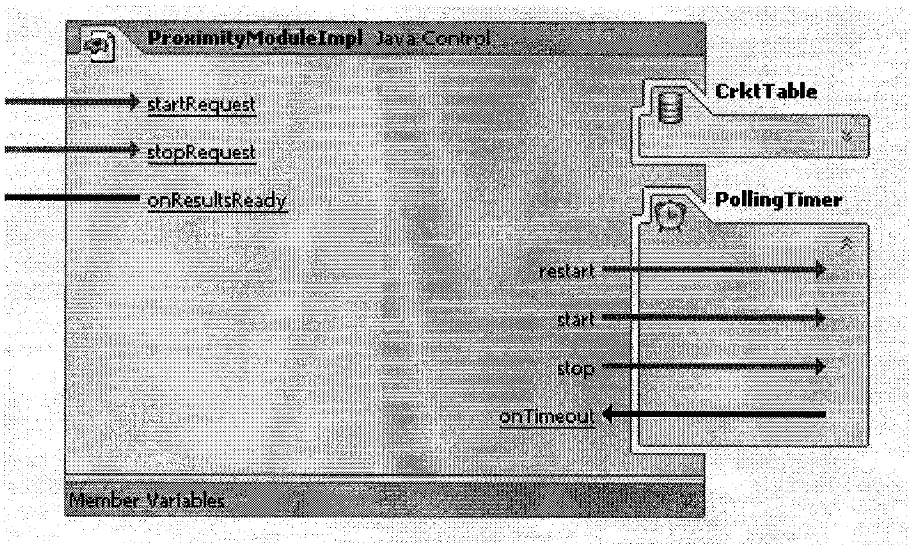


**Figure 5.2. 'lower class' for handling sensor specific details**

*Execution of a Web service* - The higher class implements the functionality of the Subscription Handler, the Processor Engine and the Notification Generator. When an application invokes a Web service, the Subscription handler validates the parameters and an error message is returned if an invalid parameter is detected. The Mapping Service (described in 5.1.2) is used to see what technology/sensor type is needed to get the desired ambient information. In other words, it is a look up table that finds a Translator or Context Inference Module for the task at hand. Parameters are validated again as capabilities differs from one WSN to another (i.e. not all location aware sensors supports location coordinates). Parameters are then passed to the lower class

The lower class implements the Context Inference Module for inferring proximity, other complex Web services do the same with their own Context Inference Module. Only the basic Web services implement the Translator functionality. While the complex Web services could be implemented as composed Web services involving the basic ones, the framework implementation does not offer this approach for performance reasons. Instead, the core logic of the Translator is simply reused in the logic of the inference module, thus avoiding overhead in response time. Once the lower class receives the parameters from the higher class, it starts a polling timer which returns the current sensed data (from the cache or the sensors) when a timeout occurs. The timer is restarted until the next timeout and so on.

Then, the higher class receives the ambient information from the lower class, and checks if it generates a notification (based on the NotificationTrigger criteria) and sends it to the application. Whether or not a notification is generated, the cache is updated and historical information is logged.

*Database Triggers* - The dissemination mechanism for the MTS300 sensors is based on databases. Therefore a natural choice for implementing the NotificationTrigger would be to use database triggers. However, they were not used. In general a database trigger can be set to execute when an SQL 'insert' statement occurs and when a condition is met. The trigger executes a stored SQL procedure. Usually DBMS systems allow these procedures to be written in different programming languages. However, at the time of the implementation, PostGreSQL 8.0 did not support JAVA procedure, only C, Tcl, Perl and Python were supported. In this case, the lack of support for JAVA means that it would be difficult to communicate with the framework that an ambient event has been detected.

### 5.1.1 Backend WSN

The ambient awareness framework uses MIT crickets sensors [2] for location data and Crossbow MTS300 [9] sensors for environmental data.

Crickets provide a sharp accuracy of 1-3 centimetres and have coverage of up to ten meters indoors. Location information can be expressed in terms of coordinates or space identifiers. Modeling location as spaces is common to many location aware sensors (Active Badges, Active Bats, RADAR), however using coordinates is less common. A cricket sensor is about the size of two AA batteries and can operate as a beacon or listener. Beacons are deployed on ceiling or walls and simultaneously broadcast radio signals and ultrasonic pulses at a semi-regular interval (750-1150 milliseconds). The listener sense both signals and pulses in order to determine the difference in time of flight. This difference is used to infer the approximate proximity to a beacon. The listener can be seen as a gateway/sink node and is attached to devices such as laptops or PDAs. Proprietary applications (CricketDaemon and cricketd) are running on a Cygwin platform on the host device. The roles of the two applications are to communicate with the sink, process triangulation algorithms and answer the requests of external application requiring the location of the host device. External applications use a JAVA API, ClientLib, to talk to CricketDaemon.

The MTS300 can sense ambient light, sound and temperature. The MTS300 operates with MICA motes [Figure 2.2] to provide processing and wireless communication capabilities. The MTS300 uses a proprietary scale for measuring light, temperature and sound. However, formulas to convert the proprietary values to standard values (i.e. Celsius degrees) are provided. Data collected is routed to the sink node and stored in a local PostGreSQL database on the host gateway. For remote access, a proprietary application can be used to view current readings. Alternatively, database access methods (i.e. ODBC) can also be used. The database is updated at a period of 10-15 seconds; this represents the minimum quality of context that is supported by this particular WSN. MoteView is the required application that writes the MTS300 updates into the PostGreSQL database.

Figure 5.3 shows the physical deployment of the ambient awareness framework along with both backend WSNs. The framework is part of the Web service gateway for sensor and is described in more details in section 5.2.2. The cricket beacons are densely deployed (at least three for triangulation) in order to get location coordinates, on the other hand, only one MTS300 is required per room.

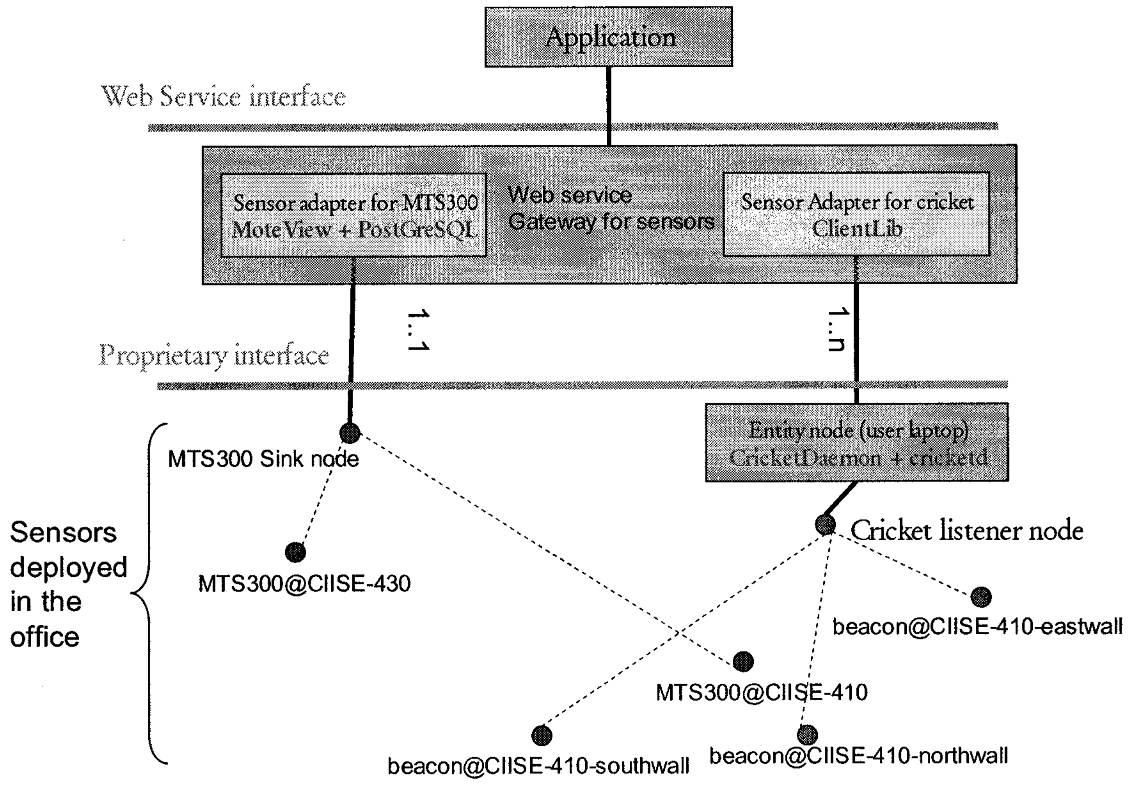
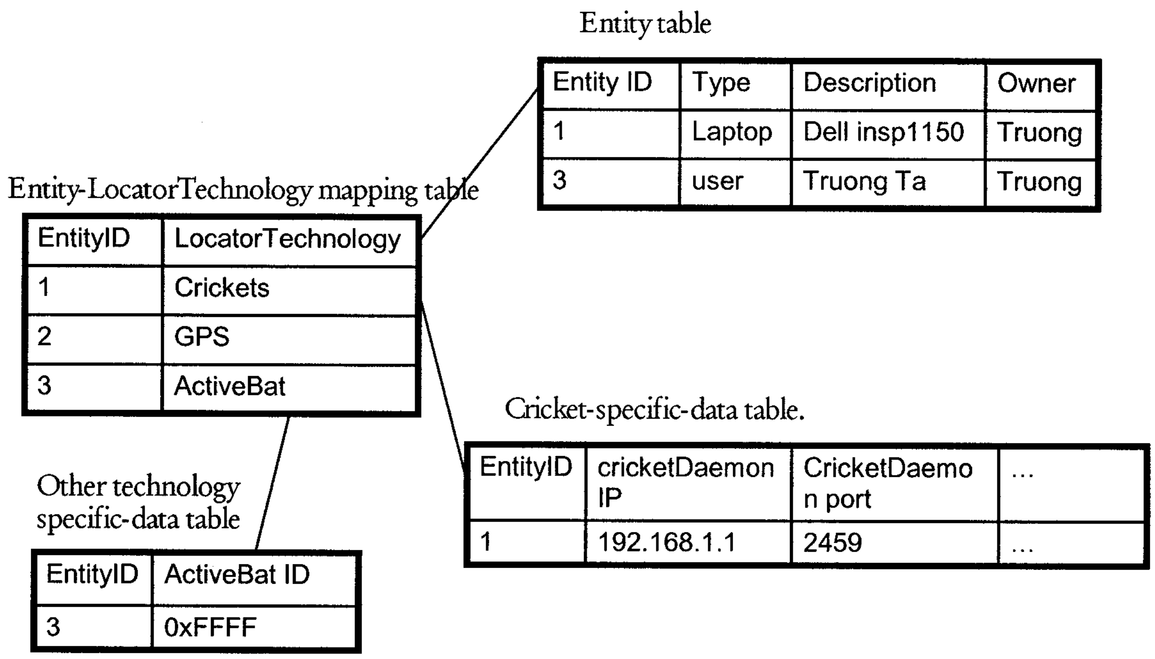

Application
Web Service interface
Sensor adapter for MTS300 MoteView + PostGreSQL
Web service Gateway for sensors
Sensor Adapter for cricket ClientLib
1..1
1..n
Proprietary interface
MTS300 Sink node
Entity node (user laptop) CricketDaemon + cricketd
Cricket listener node
Sensors deployed in the office
MTS300@CIISE-430
beacon@CIISE-410-eastwall
MTS300@CIISE-410
beacon@CIISE-410-southwall
beacon@CIISE-410-northwall

**Figure 5.3. Physical deployment view**

## 5.1.2 Mapping to Sensors

The mapping functionalities in the Mapping Service [Figure 4.4] are defined as relational databases. This mapping is necessary to bridge the connections between different sensors (including various models by different manufacturers) into an Entity that needs to be located or any other service that needs to be fulfilled. From the framework point of view, this mapping answers the following questions: which translator or inference module needs to be used and what are the required parameters for accessing the sensors. For instance, cricket sensors require individual IP address and port information in order to retrieve location, while another might use sensor node identifiers. Additionally, the framework should not discard the case where an Entity can be located using different technologies. In Figure 5.4, the framework is asked for the location of an Entity and looks up the Entity-LocatorTechnology table to find that Entity '1' is associated with crickets. The framework then uses the Cricket-specific-data table to retrieve the necessary parameters to query the location sensors. That particular table indicates that the framework should use a Cricket Translator along with the specified parameters to get location. However, the implementation only uses one location-awareness technology and consequently the Entity-LocatorTechnology table maps to the same LocatorTechnology.



Entity table

| Entity ID | Type | Description | Owner |
|---|---|---|---|
| 1 | Laptop | Dell insp1150 | Truong |
| 3 | user | Truong Ta | Truong |

Entity-LocatorTechnology mapping table

| EntityID | LocatorTechnology |
|---|---|
| 1 | Crickets |
| 2 | GPS |
| 3 | ActiveBat |

Cricket-specific-data table.

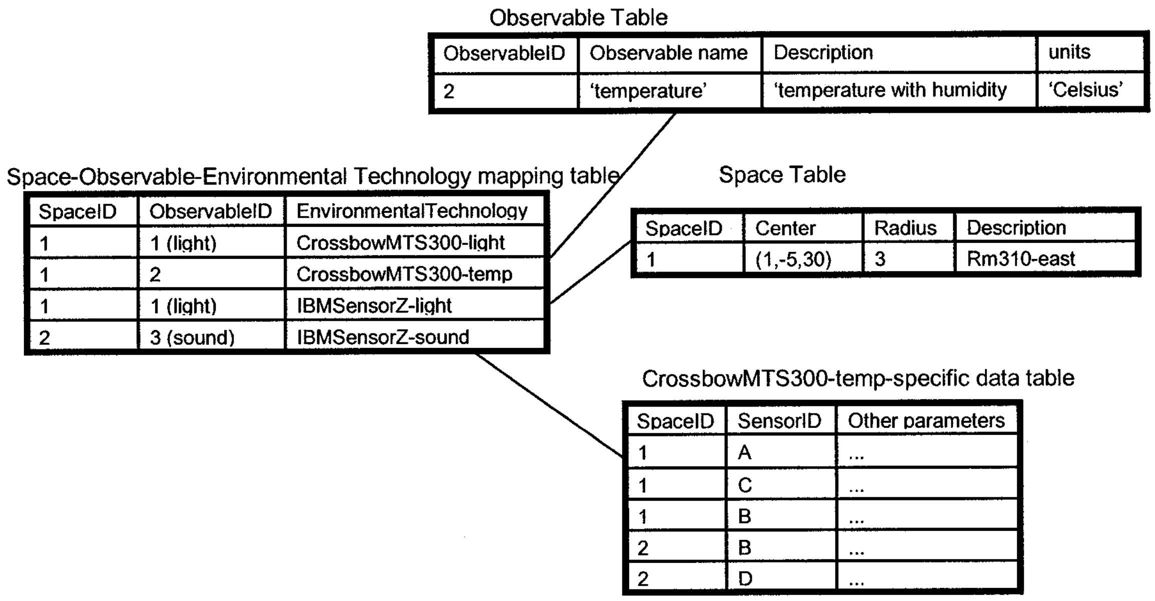
| EntityID | cricketDaemon IP | CricketDaemo n port | ... |
|---|---|---|---|
| 1 | 192.168.1.1 | 2459 | ... |

Other technology specific-data table

| EntityID | ActiveBat ID |
|---|---|
| 3 | 0xFFFF |

**Figure 5.4. Mapping Service for location data**

In other words, the mapping table provides a function $f_{\text{Location}}$(Entity) = {Technology-Specific-data-Table$_1$, Technology-Specific-data-Table$_2$, ...}. The same logic is applied for the other Web services. For environmental data, the function is $f_{\text{EnvironmentalData}}$(Space, Observable) = {Technology-Specific-data-Table$_1$, Technology-Specific-data-Table$_2$, ...} and Figure 5.5 illustrates the above relationship. It is assumed that the MTS300 sensors are deployed at a fixed location in the environment. For this reason, a Space is associated with an Observable which an MTS300 node can provide. For basic Web services, the mapping is simple and uses $f_{\text{Location}}$ and $f_{\text{EnvironmentalData}}$ directly. On the other hand, complex Web services are implemented as inference module using a combination of spatial and/or environmental data. The mapping here is more complex, as it requires finding a combination of parameters as well as performing additional logic. Ultimately $f_{\text{Location}}$ and $f_{\text{EnvironmentalData}}$ are used in the Context Inference Modules as well.



Observable Table

| ObservableID | Observable name | Description | units |
|---|---|---|---|
| 2 | 'temperature' | 'temperature with humidity | 'Celsius' |

Space-Observable-Environmental Technology mapping table

| SpaceID | ObservableID | EnvironmentalTechnology |
|---|---|---|
| 1 | 1 (light) | CrossbowMTS300-light |
| 1 | 2 | CrossbowMTS300-temp |
| 1 | 1 (light) | IBMSensorZ-light |
| 2 | 3 (sound) | IBMSensorZ-sound |

Space Table

| SpaceID | Center | Radius | Description |
|---|---|---|---|
| 1 | (1,-5,30) | 3 | Rm310-east |

CrossbowMTS300-temp-specific data table

| SpaceID | SensorID | Other parameters |
|---|---|---|
| 1 | A | ... |
| 1 | C | ... |
| 1 | B | ... |
| 2 | B | ... |
| 2 | D | ... |

**Figure 5.5. Mapping Service for environmental data**

## 5.1.3 Optimizations

*Cache* - The cache is for spatial data only. It aggregates velocity, space and coordinates of an entity into one table entry. The idea is to minimize requests to the crickets sensors if the requested QualityOfContext is superior to the one in the current cache. By avoiding this request, it increases the overall response time as location is retrieved locally. The cache contains the latest spatial information of all Entities; it stores the spatial data and associated timestamp. It is constantly updated and indexed by Entity for fast searching. The cache supports different searching functions, such as searching by Space(s), by velocity and by coordinates range. A caching scheme was not implemented for environmental data since the gain in response time would not benefit much this type of ambient information. Furthermore, environmental data is generally considered as static and is limited by the long period of update of the MTS300.

*Subscribe_Velocity()* - The work described in [57] shows how to quickly find a path based on location coordinates samples. The problem of finding accurate velocity direction or path is also a problem in *Subscribe_Velocity()*. Sampling only two location coordinates to infer velocity is not enough and leads to significant inaccuracies. Based on the author's observations and the analysis in [57], the proposed solution is to sample at least three location coordinates and use linear regression to effectively get the velocity direction. Also, the sampling coordinates must be separated by a minimum distance. There are a lot of similarities in [57] in terms of tools, settings and goals, thus the proposed solution can be applied to *Subscribe_Velocity()*.

*Subscribe_PhysicalPresence() and Subscribe_Proximity()* - In both of these services, the framework needs to know the location of many Entities in order to generate the proximity or physical presence information. It is desired to have access to this information without querying all the sensors in the WSN. One possible solution is to use a Moving Object Database [38] (MOD). The literature behind MOD suggests that current DBMSs are not suitable to handle highly dynamic data such as location. MOD solves the main issues related to location, such as handling the dynamicity of the data by modeling location differently (i.e. location is stored as trajectories), providing a flexible query language for formulating complex queries over a space and time, and most importantly, indexing dynamic attributes to allow range queries without going through all the objects in the database.

### 5.1.4 Development Platform

For the development platform, refer to section 5.2.2 on the testbed. It is the same software and hardware used for the Web service Gateway for sensors.

## *5.2 Performance Evaluation*

This section talks about the performance measurements of the framework followed by a short analysis of the results.

### 5.2.1 Performance Metrics

There are two main quantitative metrics that are considered for the performance analysis: network load and response time. It is assumed that these parameters are important from the network provider and the user points of view, respectively. For the user, a short response time means that the application is highly responsive, or that an adaptation is made just at the right moment. Additionally, knowing that these applications will run on thin devices and that wireless communication is a power hungry operation; measuring the network load in this context make a lot of sense. The tests are performed by writing a small dummy application that uses each of the individual Web service methods. This approach is also used to measure the performance of the proprietary interfaces. The resulting measurements from using both interfaces are compared for the purpose of the analysis.

Since the Web service responses are based on NotificationTriggers, a minor modification was made in order to get accurate response time measurements. The framework will immediately send the response without waiting for a NotificationTrigger to occur. The measurements take into account two cases: when the framework is retrieving the sensed data from the sensors and alternatively, from the cache. Response time is calculated from timestamps embedded in the client application. It starts when a request is issued and stops when a response is received. For the network load, a protocol-analyzer is running during the tests; hence the average network load generated from a pair of request-response can be calculated.

### 5.2.2 Testbed

The hardware specification of our testbed is as follows:

- Web service Gateway for sensors: Pentium 4 2.4GHz / 512MB / 100Mb(Ethernet) / ATA133, 7200rpm, 60GB, MTS300 sink node
- Web service Client and Proprietary Client: Mobile Pentium 4 1.6GHz / 512MB / 802.11g(WiFi) / 5400rpm, 40GB, cricket location aware sensor (listening mode)
- Miscellaneous: Linksys 802.11g in infrastructure mode, cricket location aware sensor (beacon mode) deployed in two offices, a single MTS300 node deployed in one office

As for the software used in the testbed: the Web Service Gateway for sensors is deployed on BEA Weblogic 8.1.3sp3 while running MoteView 1.0/PostGreSQL 8.0.0 for the MTS300 sensors; PointBase DBMS is used for all other database tables (Mapping Service); The Web service Client and Proprietary Client are deployed on BEA Weblogic 8.1.3sp3 and Jbuilder 9, respectively; Cygwin 1.3, Cricket API v2.3.0 and firmware v2.3.1 are used on both proprietary and Web service client applications. Each of the machines has Sun JDK 1.4.2 and WindowsXP home sp2 installed, and is networked in a closed test environment. Etherpeek NX 2.0 is used to capture all packets for the average network load calculation. For each Web service methods, at most five runs are executed, and each run has at least 100 measurements (request-response pair).

### 5.2.3 Results and Analysis

Table 5.1 shows the total network load for both cases. The column on network load efficiency is the ratio of the network load difference and total network load of the Web service interface (or total network load of the proprietary interface in the case of an overhead). This ratio represents the percentage of request-response pair one interface can do more than another. At first glance, the network load for the proprietary interfaces is much higher than for the Web service interfaces. Closer inspection shows that this is due to the large amount of low-level information exchanged between the sensor network and the client such as statistical information (i.e. minimum / maximum / mean / median distances for each beacon that can be heard). This is not the case for less verbose interactions such as Area environmental where the location data is not retrieved from the sensors. In this case, the overhead using the Web service is 321% compared to the proprietary interface. In all of the other cases, Web service efficiency over the proprietary APIs ranges from 81% to 249% with a mean of 171%. This percentage seems to scale with the level of abstraction of the operations. The size of the SOAP envelope is included in Table 5.1 for comparison and, in all cases, the SOAP message takes from 47% to 69% of the Web service total message size. This percentage increases as the number of parameters increases in the response and request messages.

**Table 5.1. Average network load per measurement, sent and received from the client application**

| | Proprietary interface | Web service interface | | Network load differen ce | Network load efficiency for Web service in terms of request-response pair |
|---|---|---|---|---|---|
| | Total network load | Total network load | SOAP envelope message size | | |
| | (bytes) | (bytes) | (bytes) | (bytes) | (%) |
| Location | 11865 | 6551 | 3190 | 5314 | 81 |
| Velocity (3 samples) | 19060 | 6056 | 2858 | 13004 | 215 |
| Physical presence (2 entities) | 24277 | 7961 | 4495 | 16316 | 205 |
| Proximity (2 entities) | 24277 | 6956 | 3559 | 17321 | 249 |
| Area environmental | 2058 | 8681 | 5842 | -6623 | 321 (overhead) |
| Entity environmental | 17156 | 8366 | 5798 | 8790 | 105 |

**Table 5.2. Average response time per measurements for proprietary and Web service interfaces**

| | Proprietary interface | Web service interface | | Overhead | | | |
|---|---|---|---|---|---|---|---|
| | | Normal operation | Cached operation | Normal operation | | Cached operation | |
| | (ms) | (ms) | (ms) | (ms) | (%) | (ms) | (%) |
| location | 486 | 1817 | 1597 | 1331 | 273 | 1111 | 228 |
| velocity (3 samples) | 494 | 1851 | 1667 | 1357 | 274 | 1173 | 237 |
| physical presence (2 entities) | 503 | 1967 | 1752 | 1464 | 291 | 1249 | 248 |
| Proximity (2 entities) | 909 | 2053 | 1784 | 1144 | 125 | 875 | 96 |
| area environmental | 323 | 1078 | n/a | 755 | 233 | n/a | n/a |
| Entity environmental | 694 | 1936 | 1648 | 1242 | 179 | 954 | 137 |

As shown in

Table 5.2, it is no surprise that the response times for the Web service framework are longer than for proprietary frameworks. This extra delay can be attributed mainly to SOAP serialization/deserialization. The delay is significant even when considering the cache optimization. A head-to-head comparison shows that the overhead ranges from 125% to 291% when considering normal operations. While the cached operation decreases the response time by an average of 235ms, the relative overhead is still large and ranges from 96% to 248% with a mean of 189%.

On the other hand, it is expected that the footprint of the Web service client application would be greatly reduced. In fact, proprietary clients have a footprint varying from 90 loc to 150 loc, while for Web service clients, which consist of one single atomic operation, it is less than 5 loc.

## 5.3 *Application: SenseCall*

SenseCall is an application that establishes a 3rd party call between two users when they are both in their respective office space. For instance, it can notify an employee when his colleague is back from a long meeting and automatically establish a call. SenseCall, shown in Figure 5.6, uses the framework to retrieve the location of both users and a Web service built on top of a Parlay gateway [10] to deliver the 3rd party call while providing a high-level of abstraction for call control. Parlay [20][21] is a set of telecommunication APIs that allow 3rd party application providers to access telecommunications capabilities via a controlled and secured manner. The high level APIs are programming language and network-technology independent (fixed, mobile or

IP networks). There are two types of APIs, service APIs, which cover call control, messaging, presence, terminal location and account/policy management services, and framework APIs, which cover authentication, authorization and discovery of network service capabilities. SenseCall uses Subscribe_Location() and its goal is to test and demonstrate the ambient awareness framework. Although current specifications of Parlay-X already support 3rd party call control capabilities, the Web service Parlay gateway provides a high level of abstraction for call control as well as support for conferencing capabilities (which was not available with Parlay-X at the time of SenseCall implementation). Also, the Web service Parlay gateway was product of a previous project.

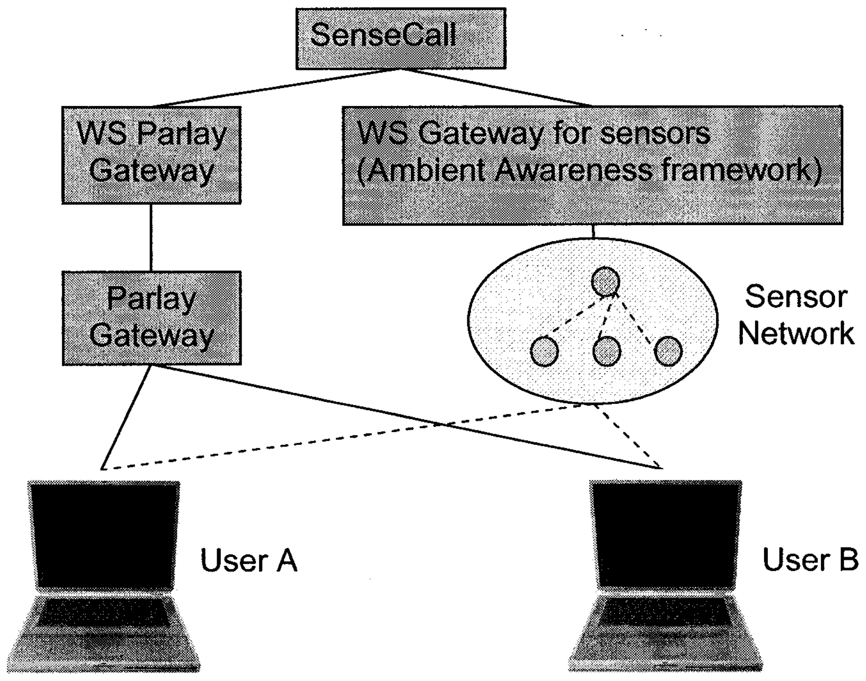


**Figure 5.6. The overall architecture of SenseCall**

# Chapter 6 Conclusion

This chapter concludes by presenting the main contributions and the lessons learned. Potential future work is also outlined.

## *6.1 Thesis Contributions*

In ambient-aware applications, a bulk of the development effort is spent building mechanisms to disseminate ambient information collected from sensors. Many frameworks have been proposed to disseminate ambient information to applications. However, they are not completely suitable for application development in an I-centric environment. The work described in this thesis investigates the use of Web services to ease the development of I-centric applications.

As part of the thesis contributions, criteria for an ideal framework from a developer's point of view were outlined and existing frameworks were evaluated and analyzed with respect to these criteria. The reviewed frameworks were based on API, mobile code, database or Web service. With the exception of the Web service based framework, the reviewed frameworks are programming language dependant, require extensive knowledge of sensors, or do not support a flexible business model. On the other hand, current Web service based framework does not have the right level of abstraction for building I-centric services.

This thesis established that a Web service-based framework is the most promising. When well defined, Web services provide a high level of abstraction and integrate easily with other applications while being programming language and platform neutral. Web services also provide needed security features as well as the capabilities to publish and discover the sensing services to applications. These benefits lower the threshold for the developers as they do not need to know the low level details of sensors or the details of integration.

As the core contribution, a set of Web services for providing ambient information is defined and implemented in the scope of this ideal framework and its overhead with respect to network load and response time has been evaluated. This time, the Web services were defined at the right level of abstraction for I-centric application development. As a part of these Web services, a generic framework was defined and implemented to provide performance improvements (i.e. caching), ambient information enhancements (i.e. Quality of Context, high level representation of ambient information) and mechanisms to consolidate with different types of sensor (Mapping Service).

Performance wise, Web services are found to be lagging by 189% in terms of response time, while network load is a mixed bag of results. For verbose interactions between the sensor network and the Web service gateway, Web services reduce the network load by 171% by only disseminating the required high-level information. On the other hand, terse interactions results in a network load overhead of 321% when using Web services.

## 6.2 Future Work

One of the biggest drawbacks of Web services is performance in terms of response time and in some cases, network load. Knowing that the bottleneck resides in the SOAP serialization and deserialization, one possible future work is to investigate the different mechanisms to accelerate or to avoid the serialization/deserialization of SOAP. Some of the proposed mechanisms include dynamic multi-protocol framework and differential serialization. Another future work is to investigate the use of Moving Object Database to store location data and its impact on the overall performance and scalability. It would be interesting to re-evaluate the framework performance in light of aforementioned enhancements. Additionally, the test environment could be improved. Thin clients like PDAs and cellular phones could be used as the testing platform, thus providing more realistic performance evaluation.

Aside from performance considerations, another open issue is the notion of context fusing. Potential work in this area could be to model a methodology for fusing ambient information.

# References

[1] Gay, D., Levis, P., Behren, R., Welsh, M., Brewer, E., Culler D.: The nesC Language: A Holistic Approach to Networked Embedded Systems. Proceedings of the ACM SIGPLAN 2003, San Diego, California, USA, pp. 1–11, 2003

[2] Cricket Version 2 User manual, MIT Computer Science and Artificial Intelligence Lab, available at http://nms.csail.mit.edu/projects/cricket/v2man-html/, Jan 2005

[3] Madden, S., Franklin, M. J., Hellerstein, J. M. and Hong, W.: Tinydb: An acquisitional query processing system for sensor networks. Transactions on Database Systems (TODS), 2005.

[4] Athanassios, B., Chih-Chieh, H., Mani, B. S.: Design and implementation of a framework for efficient and programmable sensor networks. Proceedings of the 1st international conference on Mobile systems, applications and services, San Francisco, California, pp. 187 – 200, 2003

[5] Specification: Web Services Security (WS-Security), available at: http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[6] Arbanowski, S., Ballon, P., David, K., Droegehorn, O., Eertink, H., Kellerer, W., Van Kranenburg, H., Raatikainen, K., Popescu-Zeletin, R.: I-centric Communications: Personalization, Ambient Awareness, and Adaptability for Future Mobile Services. IEEE Communications Magazine pp. 63-69, Sept 2004.

[7] Parlay-X specifications Version 2, available at http://www.parlay.org/specs/

[8] Open Geospatial Consortium, available at http://www.opengeospatial.org

[9] Crossbow Technology, Mica2 Multi-Sensor Module, available at http://www.xbow.com/Products/productsdetails.aspx?sid=75

[10] Torreira da Silva, J.S., Hassan, K., Glitho, R., Khendek, F.: Web services for Conferencing in 3G Networks: A Parlay based implementation. Proceedings of ICIN'2004, Bordeaux, France, October 2004

[11] Web Services Description Language (WSDL) 1.1, available at: http://www.w3.org/TR/wsdl

[12] SOAP Version 1.2, available at: http://www.w3.org/TR/soap

[13] UDDI Version 3 Specifications, available at: http://www.uddi.org/

[14] Wireless world research Forum, available at: http://www.wireless-world-research.org/

[15] Chen, Y., Chen, X.Y., Rao, F.Y., Yu, X.L., Li, Y., Liu, D.: LORE: An infrastructure to support location-aware services. IBM Journal of Research and Development, Vol. 48. No 4/6, pp. 601-615, 2004

[16] Alonso, Casati, F., Kuno, H. and Machiraju, V.: Web Services Concepts, Architectures and Applications, Chapter 5. Data-Centric Systems and Applications, Springer Verlag, 2004

[17] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. and Weerawarana, S.: Unraveling the Web Services Web An Introduction to SOAP, WSDL, and UDDI, IEEE Internet Computing, March 2002

[18] Gottschalk, K., Graham, S., Kreger, H., Snell, J.: Introduction to Web services architecture. IBM SYSTEMS JOURNAL, VOL 41, 170 NO 2, 2002

[19] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. W3C Working Group Note, available at: http://www.w3.org/TR/ws-arch, Feb 2004

[20] Yates, M. J. and Boyd, I.: The Parlay network API specification. BT Technology Journal, Vol 18, No 2, April 2000

[21] Stretch R. M.: The Parlay API — allowing third party application providers safe and secure access to network capabilities. BT Technology Journal, Vol 21, No 3, July 2003

[22] Botts, M., Percivall, G.: Sensor Web Enablement. Open Geospatial Consortium white paper, Sept 2005

[23] Web services toolkit for mobile devices, available at: http://www.alphaworks.ibm.com/tech/wstkmd

[24] Thompson, T., Weil, R., Wood, M. D.: CPXe: Web Services for Internet Imaging. IEEE Computer Society, pp. 54-62, Oct 2003

[25] Cnet news: GPS-enabled school uniforms hit Japan, available at: http://ecoustics-cnet.com.com/2061-10790-5671524.html, April 2005

[26] Krishnamachari, B., Estrin, D., Wicker, S.: Impact of Data Aggregation in Wireless Sensor Networks. International Workshop on Distributed Event-Based Systems 2002, Oct 2002

[27] Madden, S., Franklin, M. J., Hellerstein, J., & Hong, W.: TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. Symposium on Operating Systems Design and Implementation, Dec 2002

[28] Slijepcevic, S., Potkonjak, M., Tsiatsis, V., Zimbek, S., & Srivastava, M.: On Communication Security in Wireless Ad-Hoc Sensor Networks. Proceedings of the Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE’02), 2002

[29] Intanagonwiwat, C., Govindan R. and Estrin, D.: Directed diffusion: A scalable and robust communication paradigm for sensor networks. Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'00), Boston, MA, August 2000

[30] Lindsey, S. and Raghavendra, C. S.: PEGASIS: Power Efficient Gathering in Sensor Information Systems. Proceedings of the IEEE Aerospace Conference, Big Sky, Montana, March 2002

[31] Yao, Y., Gehrke, J.: The Cougar Approach to In-Network Query Processing in Sensor Networks, Department of Computer Science Cornell university, SIGMOND Record. Vol 31, No. 3, September 2002

[32] Braginsky D. and Estrin D.: Rumor Routing Algorithm for Sensor Networks. Proceedings of the First Workshop on Sensor Networks and Applications (WSNA), Atlanta, GA, October 2002

[33] Beigl, M., Zimmer, T., Krohn, A., Decker, C. and Robinson P.: Smart-Its – Communication and Sensing Technology for UbiComp Environments. Telecooperation Office (TecO), University of Karlsruhe

[34] Steere, D., Baptista, A., McNamee, D., Pu, C. and Walpole, J.: Research challenges in environmental observation and forecasting systems. Proceedings of the 6th Int. Conf. Mobile Computing and Networking (MOBICOMM), pp. 292–299, 2000

[35] Bonnet, P., Gehrke, J. and Seshadri, P.: Querying the physical world. IEEE Personal Communications, pp. 10–15, Oct 2000

[36] Warneke, B., Last, M., Leibowitz, B. and Pister, K. S. J.: Smart Dust: Communicating with a Cubic-Millimeter Computer. Computer Magazine, January 2001

[37] Berkeley Robotics and Intelligent Machines Laboratory: Golem Dust, available at: http://www-bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust/

[38] Wolfsony, O., Xuz, B., Chamberlainx, S., Jiang, L.: Moving Objects Databases: Issues and Solutions. Proceedings of the 10th International Conference on Scientific and Statistical Database Management, pp 111-122, 1998

[39] Working Group 2 Service architectures for the wireless world: White Paper on Ambient Awareness. Wireless World Research Forum, Dec 2003

[40] Working Group 2 Service architectures for the wireless world: White Paper on Service Adaptability. Wireless World Research Forum, Dec 2003

[41] Working Group 2 Service architectures for the wireless world, White Paper on Service Personalization. Wireless World Research Forum, Dec 2003

[42] IDSec, available at: http://idsec.sourceforge.net/

[43] Composite Capabilities/Preference Profiles available at: http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/

[44] Platform for Privacy Preferences available at: http://www.w3.org/P3P/

[45] Berkeley University of California available at: http://www.berkeley.edu/

[46] Berkeley Sensor and Actuator Center available at: http://www-bsac.eecs.berkeley.edu/

[47] Crossbow Technology, mica-mote available at: http://www.xbow.com/Products/productsdetails.aspx?sid=72

[48] Crossbow Technology, micadot-mote available at: http://www.xbow.com/Products/productsdetails.aspx?sid=73

[49] World Wide Web Consortium available at: http://www.w3.org/

[50] Chiu, K., Bramley, R., Huffman, J.C., Huffman, K., McMullen, D.F.: Instruments and Sensors as Network Services: Making Instruments First Class Members of the Grid. Indiana University Computer Science department Technical report 588, 2003

[51] Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A Survey on Sensor Networks. IEEE Communications Magazine, pp. 102-114, August 2002

[52] Fok, C-L., Roman, G-C., Lu, C.: Mobile Agent Middleware for sensor networks: an application case study. Washington University in St-Louis

[53] Ember Corporation available at: http://www.ember.com/index.html

[54] Ember Corporation , EmberNet technical brief available at: http://www.ember.com/downloads/pdfs/embernet_techbrief.pdf

[55] Sensoria Corporation available at: http://www.sensoria.com/

[56] Sensoria Corporation, sGate wireless sensor gateway available at: http://www.sensoria.com/pdf/sGate-Brochure.pdf

[57] Gupta, S., Das, S. R.: Tracking Moving Targets in a Smart Sensor Network. University of Cincinnati

[58] Schramm, P., Naroska, E., Resch, P., Platte, J., Linde, H., Stromberg, G., Sturm, T.: A Service Gateway for Networked Sensor Systems. IEEE Pervasive Computing, pp. 66-74, January-March 2004

[59] Nath S., Deshpande A., Ke, Y., Gibbons, P. B., Karp, B., Seshan, S., IrisNet: An Architecture for Internet-scale Sensing Services. In proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

[60] Souto E., Guimaraes, G., Vasconcelos, G., Vieira, M., Rosa, N. and Ferraz, C.: A Message-Oriented Middleware for Sensor Networks. Federal University of Pernambuco, 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing, Toronto, Ontario, Canada, October 2004

[61] Delicato, F. C., Pires, P. F., Pirmez, L., Rust da Costa Carmo, L. F.: A Flexible Web Service based Architecture for Wireless Sensor Networks, Proceedings of the 23rd International Conference on Distributed Computing System Workshops (ICDCSW'03), 2003

[62] Li, S., Lin, Y., Son, S. H., Stankovic, J. A., Wei, Y.: Event Detection Services Using Data Service Middleware in Distributed Sensor Networks, Department of Computer Science, University of Virginia, 2003

[63] Levis, P., Culler, D.: Mate: A Tiny Virtual Machine for Sensor Networks, Computer Science Division of University of California Bekerley and Intel Research

[64] McKusick, K.: A conversation with Adam Bosworth. ACM Queue vol. 1, no. available at http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=29, March 2003

[65] XML available at: http://www.w3.org/XML/

[66] XML signature available at: http://www.w3.org/Signature/

[67] XML encryption available at: http://www.w3.org/Encryption/2001/

[68] The Internet Engineering Task Force available at: http://www.ietf.org/

[69] Shen, C-C., Srisathapornphat, C. and Jaikaeo, C.: Sensor Information Networking Architecture and Applications, IEEE Personal Communications, August 2001

[70] Buonadonna, P., Gay, D., Hellerstein, J. M., Hong W. and Madden, S.: TASK: Sensor Network in a Box, Intel Research Berkeley and UC Berkeley, 2005

[71] Zigbee Alliance, Zigbee specification 1.0, June 2005, available at: http://www.zigbee.org/en/index.asp

[72] Trendil Corporation, available at: http://www.tendrilnetworks.com/

[73] SOA Web Services News Desk: Tendril Creates Web Services "Server Broker" for Zigbee, available at: http://webservices.sys-con.com/read/48248.htm

[74] Ta, T., Othman, N. Y., Glitho, R. and Khendek, F.: Bridging End-User Applications and Wireless Sensor Networks with Web Services: A Promising Approach. Work under review at Ad Hoc Networks, Elsevier, October 2005

[75] W3C note on WAP Binary XML Content Format, available at: http://www.w3.org/TR/wbxml/

## Appendix A - sample SensorWare script

```
set need_reply_from [ replicate -m]
set maxvalue [ query sensor value ]
if {$need_reply_from == ""} { send $parent $maxtemp; exit }
else { set return_reply_to $parent }
set first_time 1
while {1} {
        wait anyRadioPck // "anyRadioPck" is a predefined eventID
        if { $msg_body ==add_user } {
                if { $first_time == 1 } {
                        send $parent $msg_body
                        set first_time 0
                }
                set return_reply_to "$return_reply_to $msg_sender"
        }else {
                set maxvalue [expr {($maxvalue<$msg_body) ? $maxvalue
: $ msg_body }]
        set n [lsearch $need_reply_from $ msg_sender]
        set need_reply_from [lreplace $need_reply_from $n $n]
        }
        foreach node $return_reply_to {
                if { ($need_reply_from=="")||($need_reply_from==$node)} {
                        send $node $maxvalue
                        set n [lsearch $return_reply_to $node]
                        set return_reply_to [lreplace $return_reply_to $n $n]
                }
        }
        if {$return_reply_to==""} {exit}
}
```