Giuseppe Amato · Francesco Furfari · Stefano Lenzi · Stefano Chessa

Enabling Context Awareness through Distributed Query Processing in Wireless Sensor Networks

the date of receipt and acceptance should be inserted later

Abstract Wireless Sensor Networks (WSN) are an important technological support for smart environments and ambient assisted applications. Up to now most applications are based on ad hoc solutions for WSN, and attempts to provide uniform and reusable application are still in their youth. Among these attempts, those integrating database technologies and WSN appear the most promising. Under this trend of research we propose a general framework to be used in context aware architectures aimed at integrating wireless sensor networks by exploiting a distributed query processor approach. In the proposed approach the WSN can be programmed using a query language (MW-SQL) which offers constructs specialized for sensor networks. The query language is offered by a JDBC driver which is encapsulated within the OSGi framework.

1 Introduction

Smart environments unobtrusively assist people by predicting their needs. This prediction is often based on information implicitly acquired from the users, from their behavior and from the environment. To this purpose they exploit wireless networks of sensors [1] placed in the environment or on the users themselves. The interpretation and management of sensor data is offered to the applications through middlewares encapsulating context-aware engines [2], which mask the sensors to offer higher level services.

The first efforts to introduce context-awareness have been related to the localization of users [3],[4],[5],[6]. Localization is still one of the main building blocks of context-aware systems, although recently the concept of context-awareness has been enriched to take into account more general environmental parameters, where the meaning of the term ”environmental” is as broad as possible. Environmental parameters may refer to user physiological/emotional data, user actions/movements, user identity, status of the surrounding environment, location, time, profiles, agendas and data referable to the user and even presence and context of other users [2],[7],[8],[9]. Context aware systems are particularly useful to support mobile applications since the context may change rapidly with mobility (of the user and/or of the environment), and the system should react rapidly to such context changes.

In this paper we present a framework to be used in context aware architectures aimed at integrating wireless sensor networks by exploiting a distributed query processor approach. In the proposed framework the Wireless Sensor Network (WSN) can be programmed dynamically by means of an SQL-like language (MW-SQL) which offers constructs specialized for WSNs. The query language individually manipulates data sources consisting of specific transducers located on individual sensors. Queries can relate and compare data acquired by multiple (remote) nodes. Queries can also aggregate data in the spatial and temporal dimension. The interface to the query language is offered by means of a JDBC driver [26] designed for the MaD-WiSe system.

In order to enable a smooth integration of the JDBC driver within a context server, we foresee the use of the OSGi [21] platform which has already been successfully

Work funded in part by the EU FP6 projects PERSONA (contract N. 045459) and INTERMEDIA (contract N.38419).

G. Amato
ISTI - CNR, via Moruzzi 1, 56124 Pisa, Italy
E-mail: giuseppe.amato@isti.cnr.it

F. Furfari
ISTI - CNR, via Moruzzi 1, 56124 Pisa, Italy
E-mail: francesco.furfari@isti.cnr.it

S. Lenzi
ISTI - CNR, via Moruzzi 1, 56124 Pisa, Italy
E-mail: stefano.lenzi@isti.cnr.it

S. Chessa, Member, IEEE
ISTI - CNR, via Moruzzi 1, 56124 Pisa, Italy and
Department of Computer Science, University of Pisa, Largo Pontecorvo 3, 56127 Pisa, Italy.
E-mail: stefano.chessa@isti.cnr.it

used to build infrastructures for smart environments. The general context architecture is shown in Figure 1.

The architecture proposed in this paper is under the final stage of development in the MaD-WiSe system [17] using a WSN platform based on MICAz motes [18].

2 Related Work

Related works on context awareness - Context-aware systems can be implemented in many ways, leading to different architectures of the context-aware systems. The following classification (proposed in [13]) is based on the way the contextual information is collected:

- Direct sensor access: the access to the sensor is not structured, the driver of the sensors is directly encapsulated into the client application which access each individual sensors when it needs fresh data.
- Middleware infrastructure: the access to the sensors is mediated by a middleware which abstracts the sensors and provide a unified interface to access them
- Context server: the sensors are encapsulated within a context server and different client applications query the server to access contextual information.

Alternatively, [14] propose a classification based on the coordination of processes and components:

- Widget: is a software component that encapsulates a sensor and which provide a public interface for the sensor [15].
- Networked services: resembles the context server architecture: the context services are offered as a set of networked services [16].
- Blackboard model: represents a data-centric view. In this approach clients can subscribe events by posting a specific request to a shared media (the blackboard), and they are notified when such event occurs.

In general the architecture of a context-aware system includes the following layers: sensors, raw data retrieval, preprocessing, storage/management, and application.

The sensor layer include not only the hardware sensors (physical sensors) but also any data source providing context information, for instance virtual sensors which include data available from applications or services (e.g. data deducted by a specific use of a browser by the user) and logical sensors which combine information obtained from physical and logical sensors (e.g. the location of the user associated to an action on a browser). The raw data retrieval layer is responsible of the low level management of the sensors, for instance providing drivers for physical sensor access. This layer provides an abstraction of the sensor layer by providing more general primitives to access context information. The preprocessing layer can combine and preprocess data coming from several sensors to give aggregated context information to the upper layer programmers and to solve conflicts among data provided by different sensors. Storage and management layer organizes and stores the gathered data in a repository accessible to the client applications. At this level the clients can request for context data using both synchronous and asynchronous services. The application layer implements the client application.

Related works on WSN - An early attempt to provide a unifying organization of sensor networks is the Directed Diffusion paradigm [10] in which the sensor network is controlled by a special sensor node (called sink) which programs the network, collects the data and offers an interface to the application. This first effort has evolved into more advanced paradigms combining database technology with WSNs such as [11],[12]. In these paradigms the database to be queried is the physical environment where the WSN is deployed. These approaches abstract specific sensor features and offer to the application an SQL-like query language through which it can program the network and access the sensed data.

TinyDB [11] is currently considered the state of the art of these approaches. It translates a query into an (optimized) query execution plan, which is later sent by the sink to all sensors. The query plan dissemination sets up a routing tree (the Semantic Routing Tree) which is later used to collect data at the sink. A single (logical) table named Sensors contains data produced by the transducers of each node in the network. Time is divided into epochs wherein each node autonomously processes the query producing a new single logical record for the table. Table Sensors is distributed across all nodes of the network: each node can access only its own records and has no access to records produced by other nodes. During each epoch a node processes the query on its part of table Sensors and it accesses only the data produced in the current epoch. Records that qualify the query are sent toward the sink node along the semantic routing tree. TinyDB can process aggregate queries on data produced in the same epoch by several sensors (spatial aggregates).

TinyDB presents some limitations. It cannot execute queries that relate and compare data acquired by different nodes since table Sensors is distributed across all nodes of the network and each node can only access its own portion of the table. A node (leaving out of consideration hierarchical aggregation) has no knowledge of data acquired by other nodes and cannot compare its data with another node's data. Another disadvantage concerns query optimization. Given that the same query plan is processed autonomously at several nodes, the generation of an optimized query plan at the base station can only rely on global assumptions valid for all nodes. Specifically, it is not possible to exploit statistics of individual sensors (or groups of sensors). Although TinyDB is very efficient when several nodes (almost all nodes) have to perform the same task (process the same query), it is less effective when different portions of the network have to perform different tasks and when it is necessary to relate data from different zones of the network.

Related works on OSGi - OSGi [22] technology is gaining an increasing attention by open source

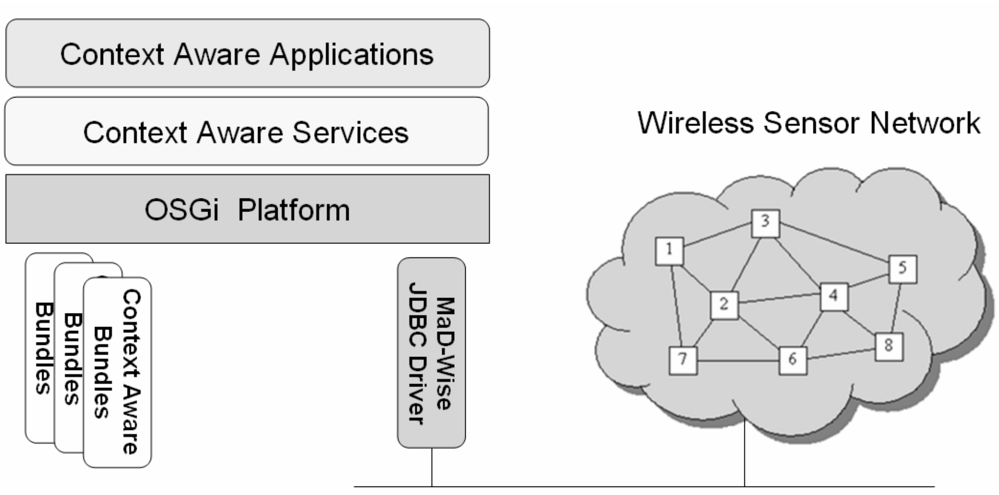

Fig. 1 The context aware architecture

communities [23], and industry market leaders like IBM, Siemens and others. At the beginning the OSGi specification addressed mainly networked device markets but, since OSGi enables the building of service and component oriented architectures and dynamic deployment of services, new and different markets are now adopting OSGi-based solutions, ranging from the enterprise market to the mobile, automotive and smart home markets [22]. OSGi has been adopted also in the research field of context-aware applications where different solutions have been proposed to build the infrastructures of Smart Environments. In [24] the authors show how OSGi framework can meet the requirements for pervasive computing spaces by providing a managed extensible framework to connect various devices and sensors and by defining an execution environment allowing the dynamic discovering of devices and services from different sources. In [21] the authors propose a middleware (SOCAM) in which context aware components are built on top of the OSGi framework to facilitate context acquisition, discovery and interpretation. These components are packaged as OSGi bundles that publish context aware services. A similar approach is used in [25], in particular the physical layer of the proposed middleware (FollowMe) uses Cricket sensors as location detectors and Mica2 as noise and light detectors.

3 Overview of the Proposed Approach

The MaD-WiSe [17] architecture comprises a set of modules running on the nodes of the WSN (network side), and a set of modules (context information provider) running on a PC attached to the WSN which is intended to offer access to the WSN services to the context server (see Figure 1).

The MaD-WiSe network side - The network-side consists of a set of modules that implement a distributed stream management system on a WSN. The network side of MaD-WiSe has been implemented over the MICAz devices [18]. It is organized into three layers, as depicted in Figure 2. The layers interact through well defined interfaces and are autonomous with respect to each other. The Network layer supports connection oriented communication between arbitrary pairs of nodes. The Stream System Layer offers abstraction mechanisms for data access by means of data streams. It can be thought of as the equivalent of a file system on a sensor network, the main difference being that, in the latter, data is continuously produced as a consequence of acquisition from transducers, communications between nodes, and data processing. The Stream System defines three types of streams: sensor, remote, and local streams.

A *sensor stream* is connected to a transducer and it carries data originated from the transducer. For this reason sensor streams are read-only.

A *remote stream* is a data channel between two distinct sensors: writing to a remote stream happens on one sensor while reading from the stream happens on the other sensor. Thus remote communication between different sensors is encapsulated within the stream system, which, under this respect, offers the transport layer functionalities [28].

A *local stream* is local to a sensor in the sense that writing to and reading from the stream can only be requested by code running on the same sensor. The Stream System offers functionalities to create/remove streams as well as read and write records from/to existing streams. Data rates can be associated with sensor streams as well as remote streams. In the first case data rates determine the activation frequency of transducers associated with sensor streams. In the second case, data rates are used by the network layer to optimize radio scheduling: the radio is switched on only when a piece of data must be sent through a remote stream [19]. Sensor streams can also be

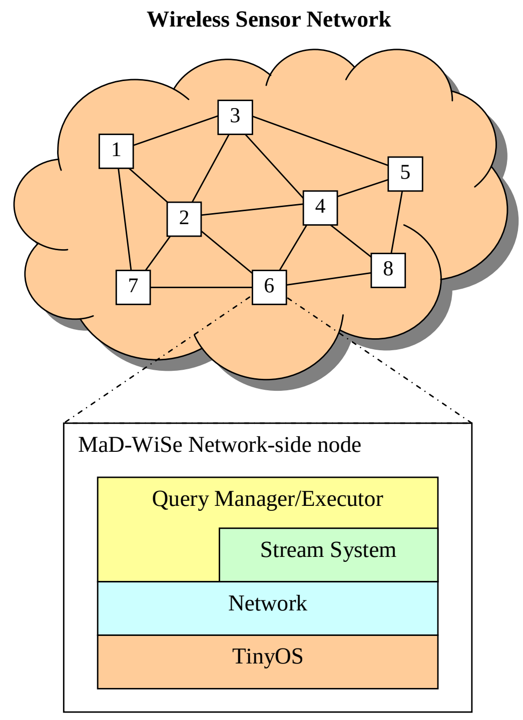

Fig. 2 The software of a network-side node.

on-demand. In this case transducers are activated only in response to an explicit read request on the stream.

The Query Processor Layer implements the query processor of a distributed data stream management system over the Stream System layer. It can be programmed by the client-side subsystem in order to take part in the execution of a distributed query. Queries are defined in terms of operations connected by streams. Operations are basically primitives of the query algebra which are applied to streamed relations implemented by the streams of the Stream System Layer. Note that in our model there are significant differences with respect to traditional relational algebra operations and relations. Relations (tables) are mainly static collections of records while streams are flowing records. Correspondingly, operators do not act on static relations but on continuously flowing data. In addition, given the limited resources available to sensor nodes (in terms of memory, processing power, and energy), data are processed on-the-fly when they arrive, using pipelined execution and avoiding as much as possible the storage of temporary data. This requires the use of non-blocking operations and exploits the inherent time ordering of data records. Although nodes could temporarily store data for later use, we avoid this to meet memory constraints.

The MaD-WiSe context information provider - The MaD-WiSe context information provider fits within the raw data retrieval layer of a context aware architecture. It comprises a low level module (composed of a query parser, an execution plan optimizer, and a query manager) and a higher level module, the JDBC driver which interacts with the low level module by means of the MW-SQL language. At the current stage of the project the JDBC driver is being encapsulated within an OSGi bundle.

In the low-level module, the query parser takes an SQL-like query and translates it into an initial distributed query execution plan. Operators of the query execution plan are allocated on the nodes involved in executing the query. The query optimizer then generates a semantically equivalent query execution plan which rearranges the nodes involved in the query execution, the operations to be executed, the transducer activations, and the radio communications in order reduce energy consumption and increase network lifetime. The query manager disseminates the optimized query execution plan in the network and receives the results obtained from in-network query execution.

4 Query Language

The query language used in MaD-WiSe is named MW-SQL and shares its basic constructs with SQL. However sensor network peculiarities and the distributive nature of the database implementation introduce some differences. MW-SQL allows users to express queries to manipulate, filter, and organize sequences of tuples generated by the sensors. MW-SQL relies on the concept of source to present the user with an abstraction of a sequence of tuples arriving from a precise origin.

MW-SQL queries are expressed through query statements having the form:

```
SELECT select-list
FROM source
[ WHERE condition ]
[ EPOCH samples [ SAMPLES ] ]
[ EVERY rate ]
```

A MW-SQL query selects the attributes (including temporal aggregates) specified by select-list from all tuples that satisfy a certain condition from the indicated source. Optionally, a query can request a sampling rate (rate) and epoch duration (samples). A sampling rate specifies at which rate transducers should acquire data. The epoch duration specifies how many samples are considered when processing temporal aggregation in queries. A detailed specification of MW-SQL can be found in [20].

The `FROM` clause in the MW-SQL query statement defines a source of data tuples to be considered when generating query results. The `SELECT` clause expresses which of the attributes of the source are relevant for the query as a comma-separated list of attribute names. As a special case `SELECT *` means that all attributes are significant and none must be discarded.

The ultimate data sources for any query computation are transducers. We call such elementary sources basic sources. Conceptually, for each transducer `TR` available on sensor `A` there exists a basic source named A.TR with two attributes named `Timestamp` and `A.TR`, where `A` is a numeric sensor id and `TR` identifies a transducer. Possible transducers include Light, Temperature, Magnetism, Humidity, etc..., subject to the actual availability on the sensor hardware. For instance, if a light transducer is available on sensor `1` the user may refer source `1.Light` with attributes `Timestamp, 1.Light` (note that `1.Light` is used both to denote the name of the basic source and the name of one of the attributes). Attribute `Timestamp` is a timestamp value for the reading contained in the other attribute. A complex source is a source constructed by combining together several other sources (basic or complex) by means of join, spatial aggregation, and union operations.

Join - Joining sources means combining their tuples on the basis of a common timestamp value. The resulting source has all the attributes of the component sources with the exception that attribute `Timestamp` is replicated only once. A complex source obtained by joining several sources can be expressed as a comma-separated list of the source names as in:

```
SELECT 2.Temp, 3.Temp
FROM 2.Light, 2.Temp, 3.Temp
WHERE 2.Light > 20
```

Note that the meaning of the above query is substantially different from standard SQL. In SQL the above query would have computed a simple Cartesian product, rather than a join on the Timestamp attribute, given that an explicit join condition is not defined.

Spatial aggregation - Aggregation of data produced by a group of sensors is a very significant capability in WSN since it reduces the amount of data sent to the sink.

In MW-SQL spatial aggregates are expressed by a functional notation in the `FROM` clause as the aggregation name (`max, min or avg`) such in `FROM avg(1.Light, 2.Light, 3.Light)`, which requests an average spatial aggregation of the light values from sensors `1`, `2` and `3`. The basic sources involved in the spatial aggregation must be of the same type i.e., they must all sample the same quantity (light in this case).

Aggregation is actually performed by computing the desired function (max, min or average) on the sampling attributes of the input sources, subject to the common timestamp constraint. The attributes of a spatial aggregation include the `Timestamp` and the name of the sampled attribute, deprived of any numeric prefix.

Union - Sometimes it is useful to sequentially merge into a single source data read by different sensors. This can be achieved by using unions of sources and can be specified in the `FROM` clause, as in `FROM union(1.Light, 2.Light, 3.Light)` which merges together light values from sensors `1`, `2` and `3`.

MW-SQL offers the functional `area().TR` (to be used in a `FROM` clause) to select sensors based on their position in the sensing field. Functional `area().TR` takes 4 arguments indicating the top-left and bottom-right corners of a rectangular region, while `TR` is a transducer type. It denotes all basic sources of the given type of all sensors located within the region.

MW-SQL supports four different temporal aggregates: max, min, average and count, indicated by the functionals `max(), min(), avg()` and `count()` respectively. The argument to the functionals must be one of the attributes from the source defined by the `FROM` clause. When temporal aggregates are requested the `SELECT` clause must contain a comma-separated list of aggregate functionals and optionally the Timestamp attribute. Different temporal aggregates can be requested on the same attribute. For instance the following query requests the minimum and maximum temporal aggregates over the light values from sensor `1` and the average temporal aggregate over the temperature from sensor `2`:

```
SELECT min(1.Light), max(1.Light),avg(2.Temp)
FROM 1.Light, 2.Temp
EPOCH 10 SAMPLES
EVERY 2000
```

while the following query requests the minimum and maximum temporal aggregates over the light values from sensor `1` and the average temporal aggregate over the temperature from sensor `2`:

```
SELECT min(1.Light), max(1.Light),avg(2.Temp)
FROM 1.Light, 2.Temp
EPOCH 10 SAMPLES
EVERY 2000
```

5 Applications

The MaD-WiSe system had been used to implement a prototype application providing remote monitoring of firefighters equipped with Totally Encapsulated Chemical Suits [27]. This application enable monitoring of parameters such as temperature, humidity and light within the coverall, physiological parameters such as heartbeat or breadth, and movements to detect if the firefighter is still active or if he is fallen. Using these information it is possible to raise alarms to be sent to the team leader in order to undertake the necessary actions, but they are also recorded to provide a history of the intervention for off-line analyses.

The system had been implemented on the MICAz devices [18]. The application exploits 5 sensors deployed on the arms, on the legs and on the chest of the operator.

We report a few samples of the queries used to implement the application.

1. Monitoring of the status of the operator using the sensor placed on the chest every one second:

```
SELECT *
FROM Chest.Light, Chest.Humidity, Chest.Temp,
Chest.AccelerationX
EVERY 1000
```

2. Monitoring of the operator movements using the sensors placed on the legs:

```
SELECT *
FROM LeftLeg.AccellerationX,
RightLeg.AccelerationX
EVERY 100
```

3. Monitoring of the temperature inside the suit and alarm for the temperature exceeding 35 degrees:

```
SELECT *
FROM Chest.Temp
WHERE Chest.Temp>35
EVERY 1000
```

6 Conclusions

We have presented a general framework to be used in context aware architectures to enable the use of wireless sensor networks in a sensor layer of a context server. The interaction between the context server and the wireless sensor network happens by means of an OSGi bundle encapsulating a JDBC driver. The JDBC driver interacts in turn with the sensor network by means of a general query language (MW-SQL) with constructs specialized for sensor networks. The query language individually manipulates data sources consisting of specific transducers located on individual sensors. Queries can relate and compare data acquired by multiple (remote) sensors.

References

1. P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. F. Hu, Wireless sensor networks: a survey on the state of the art and the 802.15.4 and zigbee standards, Computer Communications, 30:1655-1695, 2007
2. B. Schilit and M. Theimer, M., Disseminating active map information to mobile hosts, IEEE Network, 8(5):22-32, 1992
3. R. Want, A. Hopper, V. Falcao, and J. Gibbons, The Active Badge Location System. ACM Transactions on Information Systems, 10(1):91-102, 1992
4. G.D. Abowd, C.G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, Cyberguide: A mobile context-aware tour guide. Wirless Networks, 3(5),
5. Y. Sumi, T. Etani, S. Fels, N. Simonet, K. Kobayashi, and K. Mase, C-map: Building a context-aware mobile assistant for exhibition tours, Community Computing and Support Systems, Social Interaction in Networked Communities, 137-154, London, UK. Springer-Verlag, 1998.
6. K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou, Developing a context-aware electronic tourist guide: some issues and experiences, SIGCHI conference on Human Factors in Computing Systems, 17-24, 2000.
7. N. Ryan, J. Pascoe, and D. Morse, Enhanced reality fieldwork: The context-aware archaeological assistent. Computer Applications in Archaeology, 1997
8. A.K. Dey, Context-aware computing: The Cyber Desk project, AAAI Spring Symposium on Intelligent Environments, 1998
9. P.J. Brown, The stick-e document: A framework for creating context-aware applications, In Proceedings of the Electronic Publishing, Palo Alto, 259-272, 1996
10. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks, MOBICOM, 56-67, 2000.
11. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122-173, 2005.
12. Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. SIGMOD Record, 31(3):9-18, 2002.
13. H. Chen, T. Finin, and A. Joshi, An ontology for context-aware pervasive computing environments, Ontologies for Distributed Systems, Knowledge Engineering Review, 18(3):197-207, 2004
14. T. Winograd, Architectures for context. Human-Computer Interaction Journal, 16(2):401-419, 2001
15. A.K. Dey and G.D. Abowd, A conceptual framework and a toolkit for supporting rapid prototyping of context-aware applications, Human-Computer Interactions Journal, 16(2-4):7-166, 2001
16. J.I. Hong and J.A. Landay An infrastructure for context-aware computing,.Human-Computer Interaction, 16, 2001
17. MaD-WiSe: Management of Data in Wireless Sensor networks. http://mad-wise.isit.cnr.it.
18. Crossbow Technology Inc., http://www.xbow.com.
19. G. Amato, P. Baronti, and S. Chessa, Connection oriented communication protocol in wireless sensor networks. Technical Report 2005-TR-10, ISTI-CNR, 2005.
20. SensorViz/MaD-WiSe, version 1.3 edition, July 2006. http://mad-wise.isti.cnr.it/manual 13.pdf.
21. T. Gu, H. K. Pung, and D. Q. Zhang, Towards an OSGi-Based Infrastructure for Context-Aware Applications in Smart Homes. IEEE Pervasive Computing, 3(4):66-74, 2004
22. OSGi Alliance (http://www.osgi.org)
23. Apache Felix Project (http://felix.apache.org), Eclipse Equinox Project (http://www.eclipse.org/equinox/), Knopflerfish Project (http://www.knopflerfish.org/)
24. C. Lee, D. Nordstedt, and S. Helal, Enabling smart spaces with OSGi, Pervasive Computing, IEEE 2(3):89-94, 2003
25. J. Li; Y. Bu, S. Chen, X. Tao, and J. Lu, FollowMe: on research of pluggable infrastructure for context-awareness, AINA 2006, pp.18-20
26. JDBC - http://java.sun.com/jdbc
27. M. Iacono, P. Baronti, G. Romano, G. Amato, and S. Chessa, Monitoring Fire-Fighters Operating in Hostile Environments with Body-Area Wireless Sensor Networks, VGR 2006, Pisa, Italy, 17-19 October 2006
28. G. Amato, P. Baronti, S. Chessa, and V. Masi, The Stream System: a Data Collection and Communication Abstraction for Sensor Networks, IEEE Int. Conf. on Systems, Man, and Cybernetics, Taipei, Taiwan, 8-11 October 2006