Service-oriented Continuous Queries for Pervasive Systems

Yann Gripay
Université de Lyon, INSA-Lyon, LIRIS – UMR 5205 CNRS
7 avenue Jean Capelle
F-69621 Villeurbanne, France
yann.gripay@liris.cnrs.fr

ABSTRACT

Computing environments evolve towards what is called pervasive systems: they tend to be more and more heterogeneous, decentralized and autonomous. From a data-centric point of view, applications in such environments have to use traditional databases alongside with non-conventional data sources like data streams and services.

In this research, we propose a framework that allows the development of applications for pervasive environments using service-oriented continuous queries over non-conventional data sources. This framework defines a homogeneous representation of those heterogeneous data sources to get a unified view of the environment. It allows declarative definitions of service-oriented continuous queries using a SQL-like language. We also tackle the design of a Pervasive Environment Management System that handles non-conventional data sources and whose overall objective is to make the development of pervasive applications easier.

1. INTRODUCTION

Computing environments evolve towards what is called pervasive systems [19]: they tend to be more and more heterogeneous, decentralized and autonomous. On the one hand, personal computers and other handheld devices are now democratized and take a large part of information systems. On the other hand, data sources may be distributed over large areas through networks that range from a world-wide network like the Internet to local peer-to-peer connections like for sensors.

In such environments, the set of available data sources and services is dynamic. Data sources are also dynamic, with relational tables that are frequently updated and with infinite data streams. Services represent an abstraction of the various functionalities of the heterogeneous devices in the environment, like interactions with the physical world through sensors and actuators [9], data retrieval or data transformations. From a data-centric point of view, we consider dynamic relations, data streams and services as non-conventional data sources.

Pervasive environments are in essence distributed over a network. Discovery techniques [23] are needed to automatically identify and use available services and data sources. We consider an environment where such techniques exist and allow an access to the heterogeneous services and data sources, *e.g.* with UPnP technology [18].

Applications in pervasive environments should then handle the dynamicity and heterogeneity of such environments. Pervasive applications can be viewed as continuous processes using those non-conventional data sources. Continuous queries [2, 6] are queries over dynamic relations and data streams continuously updating their results, and thus a sort of continuous process.

In this research, we focus on the following question: **How continuous query techniques over non-conventional data sources can make the development of pervasive applications easier?** We identify three major challenges to tackle this question:

1. **Definition of a homogeneous representation for non-conventional data sources** in pervasive environments, *i.e.* for dynamic relations, data streams and services. It is somehow a way to instantiate the general notion of Data Space [11, 15], or what could be called the Data and Service Space.
2. **Definition of a query language and optimization techniques for continuous queries over non-conventional data sources**, in order to get closer to the behavior of pervasive applications.
3. **Design of a Pervasive Environment Management System** extending DataBase Management Systems, that handles non-conventional data sources and continuous queries over those data sources. Such a system should manage both network issues and data management issues.

The rest of the paper is organized as follows. In Section 2, we situate our research problem within the related works. In Section 3, we expose in more details each three major challenges identified above. In Section 4, we present the prototype of the system that we have developed. We then conclude and discuss some perspectives in Section 5.

2. RELATED WORK

Pervasive systems have been studied by many works [3, 4, 9, 13, 14]. They address the problem of dynamic discovery [9] by an abstraction of the physical network and of the device features [3, 4]. Data and application sharing among distributed devices, including user interface devices, is simplified [14]. In a more user-centered fashion, [13] introduces the notion of task, *i.e.* the set of applications and data a user is currently using, that should be instantiated in every environment this user goes into. In our research, we focus on data management over pervasive applications that use databases, data streams and services on distributed devices. Up to our knowledge, bridging the gap between data management and pervasive applications has not been fully addressed yet.

A large bunch of work has been done in continuous query definition and processing [1, 2, 5, 6, 7, 12, 17, 22]. Most of works [2, 5, 12, 22] propose an extension of SQL in order to work with both relational databases and data streams. Some works [6] tackle continuous querying over distributed XML data sets and propose an extension of XML-QL. Others [1, 7] are based on a box representation of operators, expressing queries as a flow of tuples. However, when working with the data stream semantics mixed with the relational paradigm, SQL tends to be widely adopted as a base for query language extensions. Data streams are then represented using a relation schema. In our research, we also adopt this common representation and extend the notion of tables as a convenient way to represent the pervasive environment.

However, to the best of our knowledge, this notion of continuous query has had a few impacts on pervasive application development involving dynamic relations, data streams and services. In [21], continuous queries can implicitly interact with devices through an external function call. However, the relationship between functions and devices, as well as the optimization criteria, are not explicit and cannot be declaratively defined. In [16], the cleaning process for data retrieved from physical sensors is defined in a declarative way by a pipeline of continuous queries. It is however only a part of pervasive applications and does not involve services. The long term objective of our research is to define a homogeneous representation for those non-conventional data sources (dynamic relations, data streams and services) in order to define whole pervasive applications in a declarative way by a set of service-oriented continuous queries.

3. RESEARCH PLAN

In this research, we tackle the problem of developing pervasive applications, or parts of pervasive applications, by using continuous query techniques.

The *ad hoc* development of pervasive applications is replaced by a more flexible and adaptable way using declarative definitions and optimization techniques. The definition of those continuous queries relies on a homogeneous view of the computing environment abstracting the implementation details of data sources and services. The proposed model and architecture focus on the following goals:

- a seamless integration of heterogeneous distributed data sources along with traditional databases,
- an easy development of pervasive applications involving such data sources.

We adapt database principles in the context of pervasive systems. We aim at providing a sort of relational view representing the heterogeneous data sources of the pervasive environment, and a query model that remains compatible with traditional data sources and that handles those heterogeneous data sources. We study the limits of the relational model for this representation and propose some extensions for the data model and the query language.

In order to illustrate our work, we use a running example called the "Temperature Surveillance" scenario. Temperature sensors that are distributed in a building provide a data stream of temperature values. Cameras are present in the different rooms in order to take photos of the places when needed. The "Surveillance" consists in analyzing the temperature streams and sending alerts when the temperature exceeds a given threshold for a room. The generated alerts contain a photo of the room and are sent to the person in charge of that room.

In the following subsections, we describe our proposed approach for each of the three major challenges that we have identified:

1. **Homogeneous representation for heterogeneous non-conventional data sources** in pervasive environments, *i.e.* for dynamic relations, data streams and services.
2. **Query language definition and optimization techniques for continuous queries over non-conventional data sources**, also called Service-oriented Continuous Queries (SoCQs).
3. **Design of a Pervasive Environment Management System** (PEMS) that handles non-conventional data sources and continuous queries over those data sources.

3.1 Homogeneous Representation for Heterogeneous Non-Conventional Data Sources

In order to define continuous queries over heterogeneous and non-conventional data sources, we propose a homogeneous representation for dynamic relations, data streams and services at the logical level. As data sources are dynamic, the notion of time needs to be explicit, in contrast with the transactional paradigm. We represent time as a discrete and ordered domain of *timestamps* (*e.g.* positive integer values).

In order to homogeneously represent relations and streams, the notion of *dynamic relation* over a relation schema is proposed:

- **Dynamicity:** a dynamic relation is a continuous stream of timestamped insertions and deletions of tuples.

- **Instantaneousness:** at each timestamp, the *instantaneous relation* [2] of a dynamic relation is defined as the multiset of timestamped tuples that have been inserted until this timestamp included, and that have not yet been deleted.
- **Infinity:** a dynamic relation can be either *finite*, where tuples can be inserted and deleted (Figure 1), or *infinite*, where tuples can be inserted, but not deleted. A dynamic relation representing a data stream is infinite (Figure 2).

Timestamp	Name	Number
@10	Alice	+3369911####
@26	Charlie	+3369933####
@28	David	+3369944####

Figure 1: Example of a Finite Dynamic Relation "Phones" representing a standard relation at Timestamp @30

Timestamp	Temperature	Area
@16	17.5	A
@25	18.5	B
@25	16.0	C
@27	19.5	A
@27	15.0	D
@29	17.0	B
...	...	...

Figure 2: Example of an Infinite Dynamic Relation "Temperatures" representing a data stream from distributed sensors at Timestamp @30

In order to integrate smoothly the notion of services within the notion of dynamic relation, we propose the following definitions:

1. A *service interface* is a group of semantically related methods associated with a name. A *method* is associated with an *input schema* describing the set of its input parameters and an *output schema* describing the set of its output parameters. In Figure 3, an example of such an interface is described.
2. A *service* is an external entity (in regard to the query management system) that computes some methods. A service *implements* a service interface if it can compute every method from the service interface.

```
CREATE SERVICE INTERFACE iCamera {
  checkPhoto(target BYTE) :
    (coverage BOOLEAN, duration FLOAT)
  takePhoto(target BYTE) : (photo BLOB)
}
```

Figure 3: Example of a Service Interface "iCamera" with two methods: "checkPhoto" and "takePhoto"

We therefore extend the notion of dynamic relation with the following notions:

1. A *virtual attribute* is an attribute whose value is set during query execution time, *i.e.* is not set when the tuple is retrieved from the data source.
2. A *service reference attribute* is a non-virtual attribute that represents an identifier of a service registered in the query management system, *i.e.* whose methods can be invoked.
3. A *binding pattern* is a rule that indicates which method of a service interface has to be invoked using which service reference attribute in order to retrieve the values of some virtual attributes (the output parameters) when values are set for some other attributes (the input parameters). Input parameters may be virtual attributes or non-virtual attributes.

An *extended dynamic relation* is a dynamic relation which contains virtual attributes and service reference attributes in its schema. Such a dynamic relation is associated with binding patterns. It may be a finite or infinite dynamic relation. The *internal schema* of an extended dynamic relation is the schema containing all the non-virtual attributes of its schema (or *external schema*). The instantaneous relation of an extended relation is defined over its internal schema.

The data description language has not yet been carefully designed, but the basic idea we have in mind is to be able to write statements like described in Figure 4 for the definition of an extended dynamic relation "Camera" (also in Figure 3 for a service interface "iCamera").

```
CREATE FINITE EXTENDED DYNAMIC RELATION Camera {
  Service    SERVICE_REFERENCE,
  Location   BYTE     VIRTUAL,
  IsCovered  BOOLEAN  VIRTUAL,
  Latency    FLOAT    VIRTUAL,
  Photo      BLOB     VIRTUAL }
USING BINDING PATTERNS {
  iCamera(Service).checkPhoto(Location) :
    (IsCovered,Latency)
  iCamera(Service).takePhoto(Location) :
    (Photo)
}
```

(a) Table Definition

Timestamp	Service
@16	Camera0
@25	Camera2
@27	Camera3

(b) Instantaneous Relation at Timestamp @30

Figure 4: Example of an Extended Dynamic Relation "Camera"

Using an extended dynamic relation, a set of services implementing a same interface is represented as a set of tuples in this relation. The description of the services can be represented as attributes. Such an extended dynamic relation is then a dynamic representation of a part of the environment, as tuples are inserted into the relation for every newly available services and deleted from it when the corresponding services become unavailable.

Extended dynamic relations allow to homogeneously represent the three types of data sources: relations, data streams and services. This representation stays close to the relational model and abstracts the physical implementation of the data sources. Interactions with services, *i.e.* invocations of service methods, are masked by the virtual attributes and the binding patterns. Virtual attributes represent a possibility of interaction for the queries: the effective invocations occur only when needed during query execution time.

3.2 Service-oriented Continuous Queries

With the homogeneous view of the pervasive environment provided by extended dynamic relations, we propose to represent pervasive applications, or parts of pervasive applications, as continuous queries over those tables. From the application developer's point of view, combining relations and data streams and interacting with services can be declaratively expressed as continuous queries. Like in the CQL approach [2], we identify three categories of operators:

1. **Relation-to-relation** operators are *standard SQL operators*: selection, projection, join, aggregation. They can be applied to finite dynamic relations to produce other finite dynamic relations. We particularly study the aggregation operator: the declarative definition of top-K queries within tuple groups can be used as an aggregation function. It consists in extracting the tuple with the maximum value on an attribute within each group. It can be used to define an interesting applicative behavior like selecting the optimal actuator among a set of candidates.
2. **Stream-to-relation** operators are the *window operators*: they define a relation as a finite subset of tuples from a stream. Window operators may be of different types: sliding windows [2, 8], hopping windows [22] or more complex windows with independent lower and upper bounds [5]. Stream-to-relation operators can be applied to infinite dynamic relations to produce finite dynamic relations.
3. **Relation-to-stream** operators are the *streaming operators*: they define a stream as an append-only multiset of tuples from a relation. For each timestamp, tuples from the relation are inserted into the stream if they match a specific condition. Streaming operators can use different specific conditions: tuples that have been inserted, tuples that have been deleted, or every tuple. Relation-to-stream operators can be applied to finite dynamic relations to produce infinite dynamic relations.

As for the data description language, the query language has not yet been carefully designed. We are currently working on an extension of the relational algebra to formally define the expressive power of our language. For the time being, we aim at writing pseudo-SQL statements like described in Figure 5 for our "Temperature Surveillance" scenario. In Figure 5(a), a simple continuous query is expressed over a single infinite dynamic relation "Temperatures". In Figure 5(b), the query combines this dynamic relation with the extended relation "Camera". The predicate 'Camera.Location = Temperatures.Area' is here a mean to set a value to the virtual attribute 'Location', allowing to retrieve the virtual attributes 'Latency' and 'Photo'. The COLLAPSE clause is the extension of the aggregation operator discussed above.

```
    SELECT  Area, AVG(Temperature)
STREAMING   ON INSERTION
      FROM  Temperatures [60s]
  GROUP BY  Area
    HAVING  AVG(Temperature) > 35.0
```

(a) Stream of alerts for areas where the mean temperature exceeds a threshold

```
    SELECT  Area, Optimum.Temperature,
            Optimum.Photo
STREAMING   ON INSERTION
      FROM  Temperatures [now], Camera
     WHERE  Temperature > 35.0
       AND  Camera.Location = Temperatures.Area
  GROUP BY  Area
  COLLAPSE  Temperature, Photo
      INTO  Optimum
     USING  Latency ASC
```

(b) Stream of alerts with photos taken from the fastest camera for the areas where the temperature exceeds a threshold

Figure 5: Example of Continuous Queries

Using this declarative expression of continuous queries over extended dynamic relations, also called Service-oriented Continuous Queries (SoCQs), we study the query optimization process needed to produce an optimal executable form of the query. We explore the following points:

1. **Lexical and syntactic analysis:** apart from standard analysis (existence of tables and attributes, type checking...), virtual attributes and binding patterns introduce new constraints on the query. Virtual attributes used in a query involve binding patterns that require some input attributes: if those attributes are virtual, then they should be unified to a value (constant value or non-virtual attribute) or may be themselves output attributes of another binding pattern. The consistency of the query has to be checked.
2. **Logical query plan computation:** in addition to the logical query operators for continuous queries used in the query declarative definition, two logical operators need to be integrated in the query operator tree for each involved binding pattern. An *invocation operator* makes asynchronous calls to the service method associated with the binding pattern. A *binding operator* actually sets the requested values into the tuple virtual attributes. The logical query plan for the query in Figure 5(b) is shown in Figure 6.
3. **Query plan optimization:** query plans should be optimize at the logical level (reordering of operators...) and at the physical level (fusion of operators...). The optimization goal depends on the execution model of the query and on the associated costs. The integration of the invocation operator, the binding operator and the extended aggregation operator (with a top-K

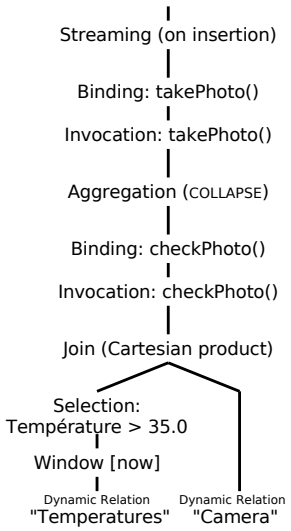

Figure 6: Example of Logical query plan

query behavior) leads to new costs models and optimization rules to be defined: reducing the number of service invocations, minimizing the execution idle time waiting for call results...

3.3 Design of a Pervasive Environment Management System

To support the homogeneous representation of the heterogeneous data sources and the optimization and execution of service-oriented continuous queries, a management system is needed. We study the requirements and challenges associated with the design of such a Pervasive Environment Management System (PEMS):

1. **Dynamic relations:** as dynamic relations are streams of timestamped insertion and deletion of tuples, the system should support streaming input from data sources and streaming output for data sinks. It requires flexible storage methods, indexing techniques, *etc.*

2. **Services:** services are external entities that the system interacts with. It should support service registrations and service interface registrations into the system catalog, in relation with service discovery techniques. It should also support service invocations during query execution.

3. **Queries:** the system should support a query optimization process for service-oriented continuous queries, and the effective continuous execution of the computed query plans. Efficient algorithms and data structures should be proposed for the different query operators, as well as scheduling techniques for real-time execution.

4. **Adaptation:** as queries are continuous, the system should gather on-line statistics on data sources in order to continuously optimize the query plans for the current context.

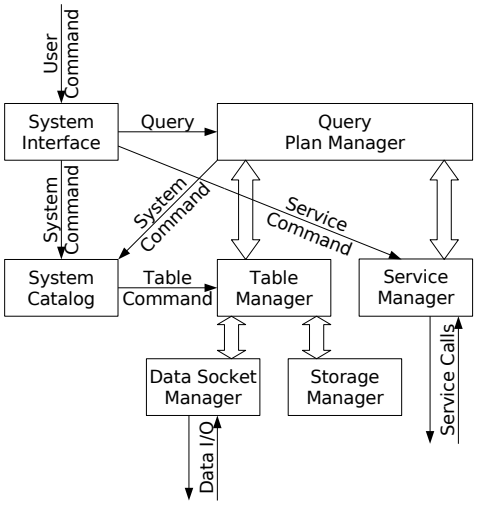

Figure 7: Architecture of the Pervasive Environment Management System

4. PROTOTYPE

In order to get a proof of concept and conduct some experiments, we are developing a prototype of the Pervasive Environment Management System. This prototype, currently called the SoCQ Processor, implements the base features presented in the previous section: it allows to register services, service interfaces, and extended dynamic relations, and to execute service-oriented continuous queries (query optimization is still a work-in-progress).

The architecture shown in Figure 7 has been implemented in C/C++ under Linux. User commands are entered using the command line or script files via the System Interface. Registrations are managed by the System Catalog. The Query Plan Manager handles the service-oriented continuous queries and executes them in a real-time fashion.

The Table Manager, along with the Data Socket Manager and the Storage Manager, is responsible for the management of the dynamic relations. Data from external data sources are received from TCP sockets over the network, using a simple dedicated protocol. External data sinks can also connect to the system to receive data from a dynamic table.

The Service Manager is responsible for the service invocations that occur during query execution. It calls the service methods with the input parameters given by the Query Plan Manager, and retrieve the output parameters from the call results. The services are physically represented by shell scripts, that can themselves execute various types of commands.

A basic user interface has been developed in Java (Figure 8): it enables to send data from a script file into a dynamic relation through a network connection (right side of the interface), and to retrieve data from a dynamic relation (left side), *e.g.* the resulting dynamic relation of a continuous query.

In order to use devices distributed in the pervasive environ-

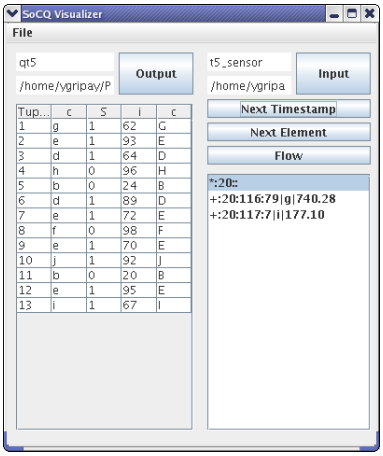

Figure 8: Snapshot of the basic User Interface

ment, we use the UPnP technology [18] that offers a solution to network issues. It enables the dynamic discovery of distributed devices and the remote invocation of methods on those devices. We use an implementation of the OSGi framework called Felix [10] that allows the development of UPnP components in Java. We have developed UPnP devices to control temperature sensors, a webcam, an IP camera and to interact with a instant messaging system (using Jabber/XMPP [20]).

We have developed tools that enable the SoCQ Processor to interact with those UPnP devices:

1. An UPnP Control Point dynamically discovers the temperature sensors and periodically retrieves the temperature values to feed the "Temperatures" data stream presented in Figure 2. We call it the "push mode" Control Point.
2. Another UPnP Control Point is used as a gateway to invoke services on the UPnP devices. We call it the "pull mode" Control Point. The "Camera" extended dynamic relation from Figure 4 can be created, and photos can be taken during the execution of the query presented in Figure 5(b). Interacting with the messaging system allows to define a query where alert messages are effectively sent to someone.

5. CONCLUSION

In this paper, we have proposed a framework that allows the development of applications for pervasive environments using service-oriented continuous queries over non-conventional data sources: dynamic relations, data streams and services. Our research plan focuses on three major challenges. First, the framework should define a homogeneous representation for those heterogeneous data sources to get a unified view of the environment. Second, it should allow the declarative definition of service-oriented continuous queries using a SQL-like language, and handle the optimization of such queries to compute continuous query plans. Third, we tackle the design of a Pervasive Environment Management System that handles the non-conventional data sources, and that optimizes and executes service-oriented continuous queries.

This research plan will also lead further with some long term perspectives. As a homogeneous representation for a pervasive environment, the framework can be a base towards the notion of DataSpace [11] where the users may discover available data sources before querying them. As a step further for query optimization, queries that handle distributed data sources can themselves be distributed on a peer-to-peer network of autonomous Pervasive Environment Management Systems.

6. REFERENCES

[1] D. J. Abadi et al. The Design of the Borealis Stream Processing Engine. In *CIDR 2005, Proceedings of Second Biennial Conference on Innovative Data Systems Research*, 2005.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, 2003.

[3] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM – A Component System for Pervasive Computing. In *PerCom'04, Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, page 67, 2004.

[4] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving: Technologies for intelligent environments. In *HUC 2000, Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing*, pages 12–29, 2000.

[5] S. Chandrasekaran et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research*, 2003.

[6] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 379–390, 2000.

[7] M. Cherniack et al. Scalable Distributed Stream Processing. In *CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research*, 2003.

[8] L. Ding and E. A. Rundensteiner. Evaluating Window Joins over Punctuated Streams. In *CIKM'04, Proceedings of the 13th ACM international conference on Information and Knowledge Management*, pages 98–107, 2004.

[9] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.

[10] Felix. Apache Felix Project. http://felix.apache.org/.

[11] M. Franklin, A. Halevy, and D. Maier. From Databases to Dataspaces: a new Abstraction for Information Management. *SIGMOD Rec.*, 34(4):27–33, 2005.

[12] M. J. Franklin et al. Design Considerations for High Fan-In Systems: The HiFi Approach. In *CIDR 2005, Proceedings of Second Biennial Conference on Innovative Data Systems Research*, 2005.

[13] D. Garlan et al. Project Aura: Toward

Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.

[14] R. Grimm et al. System Support for Pervasive Applications. *ACM Transactions on Computer Systems*, 22(4):421–486, November 2004.

[15] T. Imielinski and B. Nath. Wireless graffiti: data, data everywhere. In *VLDB'2002: Proceedings of the 28th international conference on Very Large Data Bases*, pages 9–19. VLDB Endowment, 2002.

[16] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative support for sensor data cleaning. In *Pervasive*, pages 83–100, 2006.

[17] F. Tian and D. J. DeWitt. Tuple Routing Strategies for Distributed Eddies. In *VLDB 2003, Proceedings of the 29th International Conference on Very Large Data Bases*, pages 333–344, 2003.

[18] UPnP. Universal Plug and Play. http://www.upnp.org/.

[19] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[20] XMPP. Extensible Messaging and Presence Protocol. http://www.xmpp.org/.

[21] W. Xue and Q. Luo. Action-Oriented Query Processing for Pervasive Computing. In *CIDR 2005, Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, 2005.

[22] Y. Yao and J. Gehrke. Query Processing in Sensor Networks. In *CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research*, 2003.

[23] F. Zhu, M. Mutka, and L. Ni. Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, 4(4):81–90, 2005.