# SoCQ: a Framework for Pervasive Environments

Yann Gripay, Frédérique Laforest, Jean-Marc Petit
INSA-Lyon, LIRIS, UMR5205
Université de Lyon, CNRS
Lyon, France
{yann.gripay,frederique.laforest,jean-marc.petit}@liris.cnrs.fr

***Abstract*—Querying non-conventional data sources is recognized as a major issue in new environments and applications such as those occurring in pervasive computing. A key issue is the ability to query data, streams and services in a declarative way. In this paper, we propose a framework that defines a data-centric view of pervasive environments: the classical notion of database is extended to come up with a broader notion, defined as relational pervasive environment, homogeneously integrating data, streams and active/passive services. It allows declarative definitions of service-oriented continuous queries using a SQL-like language, based on the so-called Serena algebra. We also tackle the design of a Pervasive Environment Management System that handles non-conventional data sources and service-oriented continuous queries.**

***Index Terms*—databases; data streams; services; continuous queries; pervasive environments;**

## I. Introduction

Querying non-conventional data sources is recognized as a major issue in new environments and applications such as those occurring in pervasive systems. In such environments, available data sources and functionalities are dynamic and heterogeneous: distributed databases with frequent updates, data streams from logical or physical sensors, and services providing data from sensors or storage units, transforming data or commanding actuators [1]. These data sources and functionalities are however not homogeneously manageable in today's systems, which is an issue when building pervasive applications.

Pervasive environments are in essence distributed over a network. Discovery techniques [2] 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 [3].

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 (e.g. [4]) are queries over dynamic relations and data streams continuously updating their results, and thus a sort of continuous process.

In this paper, 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) **Modeling pervasive environments integrating non-conventional data sources**, i.e. defining a homogeneous representation for dynamic relations, data streams and services; it is somehow a way to instantiate the general notion of Data Space [5], [6], or what could be called the Data and Service Space;
2) **Defining 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) **Designing a Pervasive Environment Management System** extending DataBase Management Systems, that handles non-conventional data sources and continuous queries over these 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 II, we position our research problem within the related work. In Section III, we expose our contributions to the first two major challenges identified above. In Section IV, we tackle the third and last identified major challenge and present the prototype of the system that we have developed. We then conclude and discuss some perspectives in Section V.

## II. Related Work

With the development of autonomous devices and location-dependent functionalities, information systems tend to become what Mark Weiser [7] called ubiquitous systems, or pervasive systems. Pervasive systems (e.g. [1], [8], [9]) are distributed systems of devices able to communicate with others through network links. They offer to users access to devices and control over their environment through various types of interfaces. The abstraction of device functionalities allows the system to automate some of the possible interactions between heterogeneous devices, in order to facilitate the use of the whole system, e.g. dynamic discovery of devices [1], data and application sharing among devices [8], [9]. In our framework, we focus on data management for pervasive applications that use databases, data streams and services from distributed devices. As far as we know, bridging the gap between data management and pervasive applications has not been fully addressed yet.

A great number of works have been realized in continuous query definition and processing. Most of works (e.g. [4], [10], [11]) propose an extension of SQL in order to handle both relational databases and data streams, where data streams

are represented using relation schemas. In [12], 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 [13], 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. To the best of our knowledge, query processing techniques over non-conventional data have had few impacts on pervasive application developments involving dynamic relations, data streams and services. Through a homogeneous representation for non-conventional data sources, we claim that pervasive application development is indeed possible at the declarative level using service-oriented continuous queries.

## III. Data Model

Our overall objective is to make the development of pervasive applications easier through database principles. We propose a framework that allows to develop pervasive applications, or parts of pervasive applications, 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 propose a data model, namely the **SoCQ** data model (standing for **S**ervice-**o**riented **C**ontinuous **Q**uery), to represent the heterogeneous data sources of the pervasive environment, and to build queries over this environment that remain compatible with traditional data sources. We extend the relational model with new notions to handle this representation, called the *relational pervasive environment*. We also define the so-called *Serena algebra* as the query language over this relational pervasive environment, that can be expressed using a SQL-like 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 (by mail, SMS, instant messaging...).

In the following subsections, we describe our contributions for the first two major challenges that we have identified (the third and last one being tackled in Section IV):

1) **Modeling pervasive environments through database principles**, to build the notion of relational pervasive environment integrating data, streams and services;
2) **Defining a query language over pervasive environments**, to build Service-oriented Continuous Queries (SoCQ queries) combining data, streams and services.

### A. Modeling pervasive environments

In order to homogeneously represent data sources and other resources from pervasive environments, we propose a model that integrates distributed functionalities of resources within data sources. Our model, based on the relational model, is built on the following notions: prototypes, services and extended relations with virtual attributes and binding patterns.

Distributed functionalities can be represented as *services* implementing *prototypes*. For example, a webcam and an IP camera are two services from the environment that implement a prototype `takePhoto():(photo)` that takes zero input attribute and provides one output attribute `photo`; a mail server, an instant messaging server and a SMS gateway are three services that implement a prototype `sendMessage(address,text):(sent)` that takes two input attributes `address` and `text` and provides one output attribute `sent`; and temperature sensors are services that implement a prototype `getTemperature():(temperature)`.

Invoking a prototype on a service realizes the implied actions, like taking a photo for a camera and sending a message to the given address for the mail server. As invocations can have an impact on the physical environment, e.g. invoking a prototype that sends a message, we need to consider two categories of prototypes: *active prototypes* and *passive prototypes*. Active prototypes are prototypes having a side effect on the physical environment that can not be neglected. On the opposite, the impact of passive prototypes is non-existent or can be neglected, like reading sensor data.

Those services and active/passive prototypes can be described as follows, using a pseudo-DDL:

```
PROTOTYPE sendMessage( address STRING, text STRING ) :
    (sent BOOLEAN) ACTIVE;
PROTOTYPE takePhoto( ) : ( photo BLOB );
PROTOTYPE getTemperature( ) : ( temperature REAL );

SERVICE email    IMPLEMENTS sendMessage;
SERVICE jabber   IMPLEMENTS sendMessage;
SERVICE camera01 IMPLEMENTS takePhoto;
SERVICE webcam17 IMPLEMENTS takePhoto;
SERVICE sensor01 IMPLEMENTS getTemperature;
SERVICE sensor06 IMPLEMENTS getTemperature;
SERVICE sensor22 IMPLEMENTS getTemperature;
```

Prototypes can be integrated into data relations schemas through virtual attributes and binding patterns. Virtual attributes are attributes from the relation schema that do not have a value at the tuple level. They represent input and output attributes of prototypes. A binding pattern is associated with a relation schema and specifies a prototype, a non-virtual attribute as the service reference, and which attributes are linked with the prototype input and output attributes. For example, the `contacts` relation has five attributes: `name`, `address`, `messenger`, `text` and `sent`, the two last attributes being

virtual attributes; it is associated with one binding pattern that uses the prototype `sendMessage`, the service reference attribute `messenger` and that links the attributes `address` and `text` with the prototype input attributes, and the attribute `sent` with the prototype output attribute. Output attribute should be virtual attributes, whereas input attributes can also be real (i.e. non-virtual) attributes, like the attribute `address` in this example.

```
RELATION contacts (
  name      STRING,
  address   STRING,
  messenger SERVICE,
  text      STRING   VIRTUAL,
  sent      BOOLEAN  VIRTUAL
)
USING BINDING PATTERNS (
  sendMessage[messenger] ( address, text ) : ( sent )
);
```

We call such relations, *X-Relations*, standing for *eXtended Relations*. Virtual attributes represent possible interactions with services: when a query needs the virtual attribute `sent`, a value is required for the virtual attribute `text` due to the binding pattern (the attribute `address` being real), and it implies an invocation of the prototype `sendMessage`. The required values for input attributes should be provided by the query itself. The services on which the prototype is invoked are defined by the value of the service reference attribute (here, attribute `messenger`), at the tuple level.

In the following table, an example of content for the X-Relation `contacts` is presented. The constants "mailer" and "jabber" are two service references, the former for the mail server, the latter for the instant messaging server. The star ($\star$) symbol reminds that virtual attributes have no value.

| name | address | messenger | text | sent |
|---|---|---|---|---|
| nicolas | nicolas@elysee.fr | mailer | $\star$ | $\star$ |
| carla | carla@elysee.fr | mailer | $\star$ | $\star$ |
| françois | francois@im.gouv.fr | jabber | $\star$ | $\star$ |

Pervasive environments being 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. integer values). We extend our model to integrate data sources like data streams. We call *XD-Relations*, for *eXtended Dynamic Relations*, X-Relations that are time-dependent: XD-Relations can be either *finite* (relations where tuples can be inserted and deleted) or *infinite* (append-only relations, i.e. data streams). An environment represented by a set of XD-Relations is defined as a *relational pervasive environment*.

For the whole "Temperature Surveillance" scenario, the relational pervasive environment contains the following finite XD-Relations: `contacts` as the contact list (described in the previous examples), `surveillance` containing some information about the rooms (manager, temperature threshold…), `cameras` and `sensors` representing all services in the environment that implement prototypes `takePhoto` and `getTemperature` (respectively); and also one infinite XD-Relation `temperatures` representing a stream of temperatures (with their location) periodically sent by sensor services.

```
RELATION cameras (
  camera  SERVICE,
  area    STRING,
  photo   BINARY   VIRTUAL
)
USING BINDING PATTERNS (
  takePhoto[camera] ( ) : ( photo )
);

RELATION sensors (
  sensor      SERVICE,
  area        STRING,
  temperature REAL     VIRTUAL
USING BINDING PATTERNS (
  getTemperature[sensor] ( ) : ( temperature )
);
```

```
RELATION surveillance (
  area         STRING,
  manager      STRING,
  threshold    REAL,
  alertMessage STRING
);

STREAM temperatures (
  area        STRING,
  temperature REAL
);
```

### B. Defining a query language over pervasive environments

Queries over relational pervasive environments allow to define interactions between dynamic data sources and services, i.e. pervasive applications. Such queries are defined to be continuous queries, i.e. queries that are executed continuously to maintain their results up-to-date. They are called Service-oriented Continuous Queries, or SoCQ queries. However, some queries may be one-shot queries, i.e. queries executed once that produce their results and do not maintain them, like standard SQL queries in DBMS.

*1) Serena algebra:* SoCQ queries are based on the Serena algebra (**Ser**vice-**ena**bled relational algebra) that defines query operators over XD-Relations. Standard relational operators (projection, selection, renaming, natural join) are redefined over finite XD-Relations, and new operators are defined. Realization operators (assignment $\alpha_{A:=B}$, invocation $\beta_{\langle p,S\rangle}$) handles the transformation of virtual attributes either by providing them a value (a constant or the value of another attribute) or by invoking a binding pattern. Window operators $\mathcal{W}_{[size]}$ and streaming operators $\mathcal{S}_{[event]}$ handles infinite XD-Relations: window operators transform an infinite XD-Relations into a finite XD-Relations (e.g. a relation that contains the tuples inserted during the last 5 minutes into the stream operand), and streaming operators transform finite XD-Relations into infinite XD-Relations (e.g. a stream of the tuples inserted into the relation operand).

For the "Temperature Surveillance" scenario, the behavior *send the message "Hot!" to all contacts except Carla when a temperature exceeds 35.5°C* can be expressed by the following Serena expression using the finite XD-Relation `contacts` and the infinite XD-Relation `temperatures`:

$$\beta_{\langle sendMessage,messenger\rangle}(\alpha_{text:=\text{"Hot!"}}(\sigma_{name\neq\text{"carla"}}(contacts) \bowtie \sigma_{temperature>35.5}(\mathcal{W}_{[1]}(temperatures))))$$

*2) Query equivalence:* Without formal semantics, it is hard to prove correctness of query formulations and query optimization is *de facto* limited, operators being often seen as "black boxes". Logical query optimization is possible in our setting as we can define query equivalence for SoCQ queries. Three issues need to be addressed in the context of pervasive environments: time-dependence, service determinism and impact of service invocations on the physical environment.

As a pervasive system is a dynamic system, the same service invoked with the same input, but at two different instants in time may lead to two different results (e.g. a service that takes a photo). As a consequence, the same query $q$ over the same relational pervasive environment may lead to different

results if $q$ is evaluated at different instants. In order to define query equivalence, we consider a discrete time domain and we assume that query evaluation occurs at a given time instant. As a consequence, all service invocations in a query occur simultaneously, from a theoretical point of view. We also consider that services are deterministic at a given instant, so that the invocation order has no impact on invocation results. For example, a service that returns the number of times it has been invoked should still return the same value for all invocations at a given instant.

In order to reflect the impact of a query on the environment, we define the notion of *action set* induced by a query against a relational pervasive environment as the set of invocations of *active binding patterns* triggered by this query, i.e. invocations of active prototypes. Invocations of passive prototypes are not taken into account. For instance, considering the previous Serena expression, we want to capture the set of messages sent by the execution of this continuous query for each time instant. An *action* is described by an active prototype, a service reference and an input data tuple for the prototype. For the previous Serena expression, the action set at instant $\tau_1$ could be:

$$Actions(\tau_1) = \{ \\ \langle sendMessage, email, \langle \text{nicolas@elysee.fr}, \text{"Hot!"} \rangle \rangle, \\ \langle sendMessage, jabber, \langle \text{francois@im.gouv.fr}, \text{"Hot!"} \rangle \rangle \}.$$

Using this model, the evaluation of a query over a relational pervasive environment is unambiguously defined. Two queries over a given relational pervasive environment are *equivalent* if and only if their evaluations at the same discrete time instant lead to the same result, i.e. the same content in the resulting XD-Relation, and the same action sets, i.e. the same set of invocations of active binding patterns, although they may imply different invocations of passive binding patterns.

Based on this query equivalence, *rewriting rules* can be applied to queries expressed in the Serena algebra. Some well-known rewriting rules of the relational algebra are still pertinent and allow to reorganize the order of operators in queries. Concerning realization operators, they can also be reorganized, except for invocation operators associated with active binding patterns. Invocations of binding patterns can be reorganized only for passive binding patterns.

For example, consider the selection operator $\sigma_{name \neq \text{"carla"}}$ in the previous Serena expression: if it is moved after the natural join, the query result and the action set are still equal, thus forming an equivalent query; if it is moved after the invocation operator $\beta_{\langle sendMessage, messenger \rangle}$, then additional invocations of the active binding pattern `sendMessage` may occur, resulting in a different action set, thus a non-equivalent query. In the latter case, the action set could be:

$$Actions(\tau_1) = \{ \\ \langle sendMessage, email, \langle \text{nicolas@elysee.fr}, \text{"Hot!"} \rangle \rangle, \\ \langle sendMessage, email, \langle \text{carla@elysee.fr}, \text{"Hot!"} \rangle \rangle, \\ \langle sendMessage, jabber, \langle \text{francois@im.gouv.fr}, \text{"Hot!"} \rangle \rangle \}.$$

*3) Query optimization:* With rewriting rules based on the query equivalence, *cost models* dedicated to pervasive environment can now be defined. One standard optimization goal is to reduce the size of intermediary relations in queries. Standard optimization rules, e.g. pushing selections down, can be applied. Moreover, assignment operators and invocation operators can be pushed up in the query tree so that the attributes they realize remain virtual until the latest possible level.

However, invocation operators lead to a new optimization variable: the number of invocations. It depends on the cardinality of their input relation. As invocations imply costly I/O accesses (remote invocations of services), this goal should have a higher priority than reducing the size of intermediary relations. Nevertheless, invocation operators for active binding patterns limit this possibility of optimization.

*4) SQL-like language:* A SQL-like query language, based on the Serena algebra, has been defined to declaratively express SoCQ query. For example, for the "Temperature Surveillance" scenario, the following query involves several operators: windows (the notation `[now]` denotes a window of size 1 applied on the stream `temperatures`), selections, joins, realizations, streaming. This query produces a stream of alerts (when a threshold is exceeded) while invoking the `sendMessage` prototype when needed (to send messages to area managers).

```
SELECT surveillance.area, surveillance.manager, contacts.sent
STREAMING UPON insertion
FROM   temperatures[now], contacts, surveillance
WITH   contacts.text := surveillance.alertMessage
WHERE  surveillance.manager = contacts.name
  AND  surveillance.area = temperatures.area
  AND  surveillance.threshold < temperatures.temperature
USING contacts.sendMessage
```

SoCQ queries can also be service discovery queries: they are dedicated queries that update an XD-Relation so that it represents the set of all available services implementing some given prototypes (and having some properties, e.g. the property `area` for sensors). Those XD-Relations then contain one corresponding tuple for each matching services. For example, the XD-Relations `cameras` and `sensors` are continuously updated by the following service discovery queries:

```
// XD-Relation "cameras"
DISCOVER SERVICES PROVIDING
  METHOD takePhoto ( ) : ( photo BLOB ) ;

// XD-Relation "sensors"
DISCOVER SERVICES PROVIDING
  PROPERTY area STRING,
  METHOD getTemperature ( ) : ( temperature REAL ) ;
```

## IV. Implementation

In this section, we tackle our third and last identified major challenge: **designing a Pervasive Environment Management System** that handles a relational pervasive environment and enables SoCQ queries.

In order to validate our approach and conduct some experiments, we have designed and developed a prototype of a Pervasive Environment Management System (PEMS). The role of a PEMS is to manage a relational pervasive environment, with its dynamic data sources and set of services, and to execute continuous queries over this environment. We also have defined a Data Description Language for XD-Relations (the Serena DDL) along with a SQL-like query language for continuous queries over XD-Relations (the Serena SQL), in order to interact with the PEMS without worrying about low-

level technical considerations like programming languages or network protocols.

### A. Design

The PEMS architecture is composed of three core modules (Environment Resource Manager, Extended Table Manager, Query Processor) and several distributed modules (Local Environment Resource Managers): the deployment of the different modules and their interactions are illustrated in Figure 1.

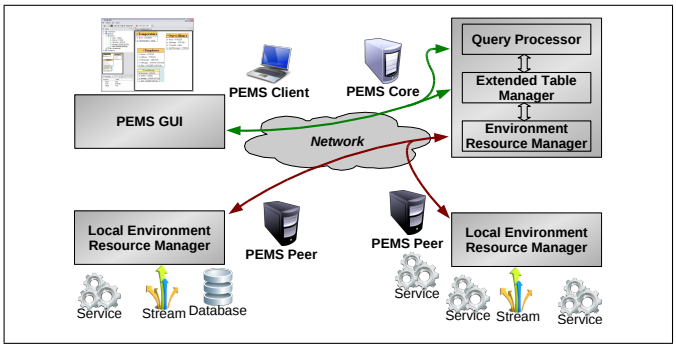


Fig. 1. Overview of the PEMS Architecture

The core Environment Resource Manager handles network issues for service discovery and remote invocation, as well as input of data from remote sources (data relations, data streams). It discovers and communicates with Local Environment Resource Managers that are distributed in the network. Services simply register to their Local Environment Resource Manager, and are then transparently available through the core Environment Resource Manager.

The Extended Table Manager allows to define XD-Relations from Serena DDL statements, and to manage their data (insertion and deletion of tuples).

The Query Processor allows to register queries using the Serena SQL and to execute them in a real-time fashion. It implements all relational operators and realization operators, as well as the Window and Streaming operators for continuous queries. Service invocations are handled asynchronously by the invocation operator, relying on the core Environment Resource Manager for actual invocations. The Query Processor also handles service discovery queries: it continuously updates some specific XD-Relations so that they represent the set of services (implementing some given prototypes) that are available through the core Environment Resource Manager, like for the XD-Relations `cameras` and `sensors` from the "Temperature Surveillance" scenario.

### B. Prototype

A prototype of the whole architecture has been developed in Java using the OSGi framework [14], including UPnP technologies [3] for network issues. The three core modules and the distributed modules are packaged as OSGi bundles.

A GUI (illustrated in Figure 1) has also been developed in Java as an Eclipse RCP Plugin, in order to control the PEMS (through the JMX network protocol): it allows to visualize XD-Relations and their content, and to launch queries.

### C. Experimentation

In order to experiment the "Temperature Surveillance" scenario, we have developed an experimental environment composed of several services: physical or simulated temperature sensors (*Thermochron iButton DS1921*), webcams (from *Logitech*), instant messaging server (*Openfire* server from *Jive Software*), (gateway to) SMS gateway (commercial service from *Clickatell*), (gateway to) mail server. Those *distributed functionalities* were wrapped as services and registered to their Local Environment Resource Manager.

An instant messaging client (*Psi*) and a mail client (*Mozilla Thunderbird*) are also used to receive messages, along with a smart phone for SMS. Through the PEMS GUI, XD-Relations have been created on the Extended Table Manager using the Serena DDL, and continuous queries have been registered to the Query Processor using the Serena SQL.

For the temperature surveillance scenario, four XD-Relations have been created: three finite XD-Relations, `surveillance` (indicating who is the "manager" of which area), `contacts` (with an additional attribute allowing to send a picture with a message) and `cameras`; and one infinite XD-Relation `temperatures`. The binding pattern `sendMessage` of `contacts` is active, whereas the binding pattern `takePhoto` of `cameras` is passive. The continuous query combining these four XD-Relations has been executed: when temperature sensors (physical or simulated) are heated over the threshold specified in `surveillance`, alert messages start to be sent to the "manager" of the associated area, by mail, instant message or SMS. Using an additional service discovery query, new temperature sensors have been dynamically discovered and integrated in the temperature stream without the need to stop the continuous query execution.

We have also experimented another scenario with RSS feeds. A wrapper service transforms RSS feeds into real streams so that a tuple is inserted in the stream when a new item appears in the RSS feed (that is periodically checked). We have tested continuous queries providing the last RSS items containing a given word (e.g. "Obama"), with a one-hour window, from several national and international information websites (french newspapers "Le Monde" and "Le Figaro", and also from "CNN Europe"). The resulting table has been continuously updated, when news of interest appeared and when one-hour-old news expired. Combining this table with the previous finite XD-Relation `contacts`, those news of interest can be sent as messages to a contact.

Those two scenarios (temperature surveillance, RSS feeds) have been successfully tested, showing the feasibility of our approach, as well as its adaptation to different kind of data sources and services. Further experiments need to be conducted to assess the scalability and the robustness of our proposal. Note that in the context of pervasive environment, this is not a trivial issue since, to the best of our knowledge, no benchmark can be used for that purpose.

## V. Conclusion

Pervasive systems intend to take advantage of the evolving user environment so as to provide applications adapted to the environment resources. As far as we know, bridging the gap between data management and pervasive applications has not been fully addressed yet. A clear understanding of the interplays between relational data, data streams and services is still lacking and is the major bottleneck toward the declarative definition of pervasive applications.

We have presented an extension of the relational model that provides a homogeneous view on all available conventional and non-conventional data sources, i.e. databases, data streams and services. The integration of services into relations allows to use a different service for each tuple (e.g. a different messaging service for each contact in a contact list) through the key notions of prototype, service reference, virtual attribute and binding pattern. The SoCQ data model includes a formal model of such extended relations, along with the **Ser**vice-**ena**bled algebra (Serena algebra) on top of it. The issue of side effect of service invocations has also been considered to define query equivalence and optimization rules. A prototype of a Pervasive Environment Management System has been devised, demonstrating the feasibility of our approach.

Future works in this project concern the query optimization process, including a formal definition of a cost metric specific to pervasive environments. We are also investigating a new notion of *streaming binding pattern* to homogeneously integrate in our model streams provided by services.

We also aim at developing a benchmark for pervasive environments to evaluate the performance of "hybrid queries" involving data and services with objective indicators. This benchmark is part of a French National Research Agency (ANR) project called OPTIMACS, started in December 2008.

## References


[1] D. Estrin *et al.*, "Connecting the Physical World with Pervasive Networks," *IEEE Pervasive Computing*, vol. 1, no. 1, pp. 59–69, 2002.

[2] F. Zhu, M. Mutka, and L. Ni, "Service Discovery in Pervasive Computing Environments," *IEEE Pervasive Computing*, vol. 4, no. 4, pp. 81–90, 2005.

[3] UPnP Forum, http://www.upnp.org/.

[4] J. Widom *et al.*, "STREAM: The Stanford Stream Data Manager," *IEEE Data Engineering Bulletin*, vol. 26, no. 1, pp. 19–26, 2003.

[5] M. Franklin, A. Halevy, and D. Maier, "From Databases to Dataspaces: a new Abstraction for Information Management," *SIGMOD Record*, vol. 34, no. 4, pp. 27–33, 2005.

[6] T. Imielinski and B. Nath, "Wireless graffiti: data, data everywhere," in *Proceedings of VLDB'02*, 2002, pp. 9–19.

[7] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94–104, September 1991.

[8] D. Garlan *et al.*, "Project Aura: Toward Distraction-Free Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 22–31, 2002.

[9] R. Grimm *et al.*, "System Support for Pervasive Applications," *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 421–486, November 2004.

[10] S. Chandrasekaran *et al.*, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," in *Proceedings of CIDR'03*, 2003.

[11] Y. Yao and J. Gehrke, "Query Processing in Sensor Networks," in *Proceedings of CIDR'03*, 2003.

[12] W. Xue and Q. Luo, "Action-Oriented Query Processing for Pervasive Computing," in *Proceedings of CIDR'05*, 2005.

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

[14] OSGi Alliance, http://www.osgi.org/.