Managing Pervasive Environments through Database Principles: A Survey

Yann Gripay, Frédérique Laforest and Jean-Marc Petit

Abstract As initially envisioned by Mark Weiser, pervasive environments are the trend for the future of information systems. Heterogeneous devices, from small sensors to framework computers, are all linked though ubiquitous networks ranging from local peer-to-peer wireless connections to the world-wide Internet. Managing such environments, so as to benefit from its full potential of available resources providing information and services, is a challenging issue that covers several research fields like data representation, network management, service discovery... However, some issues have already been tackled independently by the database community, e.g. for distributed databases or data integration. In this survey, we analyze current trends in pervasive environment management through database principles and sketch the main components of our ongoing project SoCQ, devoted to bridging the gap between pervasive environments and databases.

Key words: Pervasive environments, Databases, Continuous queries, Data streams, Services

Yann Gripay
Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France
e-mail: yann.gripay@liris.cnrs.fr

Frédérique Laforest
Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France
e-mail: frederique.laforest@liris.cnrs.fr

Jean-Marc Petit
Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France
e-mail: jean-marc.petit@liris.cnrs.fr

1 Introduction

As initially envisioned by Mark Weiser [61], pervasive environments are the trend for the future of information systems [43]. Heterogeneous devices, from small sensors to framework computers, are all linked though ubiquitous networks ranging from local peer-to-peer wireless connections to the world-wide Internet. Managing such environments, so as to benefit from its full potential of available resources providing information and services, is a challenging issue that covers several research fields like data representation, network management, service discovery...

In order to cope with the development of autonomous devices and location-dependent functionalities, an abstraction of device functionalities as distributed services allows the pervasive system to automate some of the possible interactions between heterogeneous devices. As devices may be sensors or effectors, services may represent some interactions with the physical environment, like taking a photo from a camera or displaying a picture on a screen. These interactions bridge the gap between the computing environment and the user environment, and can be managed by the pervasive system through such services. Many projects of pervasive systems have been devised, e.g. [8, 13, 45, 46, 57, 58].

In this setting, even data tend to change their form to handle information dynamicity. The relational paradigm widely adopted in DataBase Management Systems (DBMS) for many years is too restrictive to manage pervasive environments with emerging data sources such as data streams and services. Queries in traditional DBMS are "snapshot queries" expressed in SQL: a query is evaluated with the current state of the database, and the result is a static relational table. The "snapshot" term expresses that the result represents only the state of the database at the moment of the query, and is never updated. With dynamic data sources, "snapshot queries" may be not sufficient as it would be computation-expensive to periodically execute them and obtain up-to-date results.

Data streams open new opportunities to view and manage dynamic systems, such as sensor networks. The concept of queries that last in time, called *continuous queries* [17], allows to define queries whose results are continuously updated as data "flow" in the data streams. This kind of data sources has drawn the attention of the database community for many years. Data Stream Management Systems (DSMS) have been studied in many works, e.g. [3, 7, 14, 18, 26, 55, 63].

From a data-centric point of view, traditional databases [28, 44] have to be used alongside with non-conventional data sources like data streams and services to deal with new properties such as dynamicity, autonomy and decentralization. Query languages and processing techniques need to be adapted to those data sources. Data management systems tend to evolve from DBMS (DataBase Management System) or DSMS (Data Stream Management System) to a more general concept of DataSpace Support Platform (DSSP) [25]. A DSSP is intended to deal with "large amount of interrelated but disparately managed data". However, many issues have already been tackled in the field of databases to extend databases in this new setting, like distributed databases or data integration.

In this survey, we study pervasive computing from a data-centric point of view. Current trends in pervasive environment management can be related to, and enhanced by, current research in the database community. In this setting, we set up an ongoing project, called SoCQ, as our attempt to bridge the gap between pervasive computing and database principles.

In Section 2, we first give an overview of pervasive systems and of the many issues in this field. We then show how database principles have been leveraged to answer to new constraints in those environments in Section 3. In Section 4, we tackle enabling technologies for pervasive systems. We then discuss our approach to manage pervasive environment through database principles in Section 5. Finally, we conclude this survey and discuss some open issues in Section 6.

2 Overview of pervasive systems

Pervasive computing, or ubiquitous computing [61], is "a paradigm for the 21st century" [50] that tackles "connecting the physical world" [23] to a ubiquitous network and discovering available resources [64] in the environment. With such settings, applications like "Data Space" [38, 25] or "Programmable Pervasive Space" [34] can be realized.

Pervasive information systems can be analyzed as the interaction of three layers, each one interacting with the "individual layer" representing the user [41]:

1. the infrastructure layer, that represents the technical part for supporting pervasive systems inducing capabilities (and limitations) for the second layer;
2. the service layer, that represents applications that can be built in pervasive systems to answer to user expectations;
3. the social layer, that imposes some restrictions upon the behavior of applications to enforce social rules, like legal aspects and user privacy.

In this overview, the focus is put on the infrastructure layer and the service layer through the presentation of the principles of pervasive systems and the description of some projects of pervasive systems. The social layer is tackled in the conclusion of this section.

2.1 Principles of pervasive systems

> Most important, ubiquitous computers will help overcome the problem of information overload. There is more information available at our fingertips during a walk in the woods than in any computer system, yet people find a walk among trees relaxing and computers frustrating. Machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as taking a walk in the woods. *Mark Weiser [61]*

The idea of ubiquitous computing, or pervasive computing, was initiated by Mark Weiser in his famous article "The Computer for the 21st Century" [61] in 1991.

His vision of computers fully integrated in the human environment and gracefully providing information and services to users is still an open issue in computer science and computer engineering.

Pervasive computing results from the evolution of the computing paradigm from centralized mainframe computers with "dummy" terminals at an organizational level to more decentralized networks of personal computers at a user level, and toward the multiplication of "smart" small-scale appliances, e.g. hand-held devices like smart phones or PDAs, or embedded devices integrated in the surrounding environment, like autonomous sensors and actuators. In so-called pervasive information systems [42], those smart objects can benefit from wireless and wired networks to remotely access to powerful computing and large distributed databases, and to be remotely accessed by other smart objects, thus creating what could be called the "Internet of Things" [56].

This integration of "computerized artifacts" blurs the distinction between computers and other electronic devices [43], leading to new application models. From a user point of view, applications can be mobile, localized and personalized: new interaction possibilities can make applications go "off the desktop", i.e. applications can run in the background, using the user environment itself as an ubiquitous interface. From a system point of view, sensors and actuators can be distributed in the environment and autonomously gather data and execute actions with no or few human interactions.

As presented in [10], developing applications in such complex computing environments leads to the need for middlewares. Middlewares offer a unified representation and access to those distributed resources. The following requirements are detailed:

1. abstraction of devices (sensors, actuators, *etc.*);
2. loosely coupled communications, including discovery mechanisms;
3. context management;
4. application developer support.

2.1.1 Abstraction of devices

Pervasive systems are distributed systems of devices able to communicate with others through network links. Devices may range from isolated sensors to mainframe computers, including smart phones, PDAs, desktop computers, and may be embedded in the environment, mobile, handheld, or stationary. At an abstract level, devices can be viewed as entities providing some of the following functionalities:

- sensor: it can report one or more environment parameters or events;
- actuator: it can modify the environment through its actions;
- computation: it can compute some information given some input data;
- storage: it can store data and answer to queries about it.

Those devices are mainly represented by services distributed in the pervasive network. This abstraction enables interoperability between heterogeneous devices.

The service representation tends however to be divided in two categories: reactive services and autonomous components. Reactive services can be invoked and composed by a supervision system in order to create applications [9, 29, 12]. On the other hand, autonomous components [32] can decide themselves to collaborate with some others in order to create coherent processes.

As devices may be sensors or actuators [23], services may represent some interactions with the physical environment, like taking a photo from a camera or displaying a picture on a screen. These interactions bridge the gap between the computing environment and the physical environment, that can both be managed by the pervasive system.

In summary, the set of devices in pervasive systems can be abstracted as an environment of distributed services providing sensor, actuator, computation and storage functionalities, where some services may be autonomous. We call such an environment a *pervasive environment*.

2.1.2 Loosely coupled communications

A common representation for data and services is required for services to understand each others. Tuple representations like for databases or standardized languages such as XML are commonly used for data exchange between services. Services are represented by their interface: it provides a list of methods that can be invoked and potentially the types of events that the service may publish.

At a lower level, system functionalities are often accessed through proxies and wrappers that translate commands and data between platform-independent and platform-specific representations.

Service discovery is a common issue [64] in distributed systems (pervasive systems, grids, or even Internet). As services may enter or exit the pervasive environment at any time (e.g. services provided by mobile devices), the discovery should be dynamic in order to reflect the currently available services.

Remote invocation, or more generally communication between services, can not always rely on a stable network infrastructure in pervasive systems. Asynchronous messaging is then preferred to synchronous communications: asynchronous messaging can handle more gracefully network latency and failures in this dynamic setting.

Asynchronous messaging also enables event mechanisms through publish/subscribe systems: a service can subscribe to some events provided by another service, and the expected events are sent asynchronously when they occur.

2.1.3 Context management

Context management is a key element for dynamic adaptation of applications to their environment. As devices and services are spatially distributed in the environment and may be mobile, a strong need for localization appears in pervasive sys-

tems. A spatial indexation of the entities in the environment is necessary to allow location-aware processes.

In a more general way, the notion of context can be defined as "any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects." [22]. From [22], three layers of components are required to capture the context: *widgets* that acquire low-level information from sensors, *interpreters* abstracting this information and *aggregators* gathering information by entity. Applications can then use these components in order to provide context-aware behaviors.

A common representation for the context is also a requirement to enable interoperability. Whereas simple forms of context can be expressed using key-value pairs (e.g. `[name="carla", location="elysee"]`), more elaborate context models need graph-model representation like RDF (Resource Description Framework) or the more general concept of ontology (e.g. the Context Ontology Language (CoOL) [54]). Ontologies allow independent services to reason about the same concepts with a shared ontology or to agree about concepts with ontology alignments.

2.1.4 Application developer support

Distributed functionalities in a pervasive environment may appear or disappear dynamically. In order to make the development of applications easier, applications can be defined using abstract functionalities and dynamically linked to actual implementations at runtime, depending on the available resources. For instance, the OSGi "whiteboard pattern" [49] works as follows: service consumers use a given service interface and, at runtime, can search registered services that implement this interface, and then invoke their methods.

Middlewares like OSGi [48], combined with network protocols like UPnP [59] or DPWS [60], implement an abstraction of pervasive systems by providing a catalog of available services that are dynamically discovered, and by hiding communications details through unified interfaces to access to those services.

Interfaces with users or other software components try to hide the complexity of the pervasive system organization. Explicit interactions (reactivity) are often made with a declarative language using an abstract view of the environment, while implicit interactions (proactivity) provide useful automatically configured services to users depending on their context.

2.2 Projects of pervasive system

Giant research projects on pervasive systems have been conducted in the greatest universities throughout the world. Among them, we quote the Oxygen project [46]

of the MIT, the EasyLiving project [45] at Microsoft Research or the Aura project [13] of the Carnegie Mellon University. The examples they provide concern mainly intelligent workspaces and enhanced spaces (e.g. elderly homes). These projects encompass many research teams of different specialties (from hardware to software, including artificial intelligence, speech recognition and synthesis, multimodal and plastic user interfaces, local area networks, middleware, *etc.*), and have shown ambitious objectives in particular on the user interface.

In the Oxygen project [46, 27, 40], they have defined new devices for the end-user (called H21s) that include computing and communication facilities as well as multimodal user interfaces. They have designed a dedicated network technology. They have also studied the software level by defining a technique for the adaptation of software to the ever changing pervasive environment using a goal-oriented programming technique [51]. It decomposes applications in two levels: the goal level abstracts the end-user task, and the software components (called pebbles) level contains effective code realization. Pebbles are platform-independent software components, capable of being assembled dynamically by the goals planning mechanism in response to evolving system requirements. A subsystem of the Oxygen environment concerns the management of user knowledge. It is based on a RDF representation and a learning system gathers information on the user habits and preferences. Collaborative tools have also been proposed, like the annotation of web documents.

The EasyLiving project [45, 12] also works on intelligent environments (in-home or in-office). The context sensing and modeling has been highly studied (combination of multiple sensor modalities, automatic sensor calibration) as well as the interaction with the end-user (computer vision and visual user interaction, adaptation of user interfaces...). They have also defined device-independent protocols for communication and data. Like Oxygen, adaptation is based on an abstraction of users' tasks and on the discovery and composition of effective services in the environment. The originality of this project comes from their geometric model of the world. This model represents objects (of the real world or of the software world) as entities, and geometric relationships between entities as measurements. A measurement is a polygon that represents the object physical expanse, and can be associated with uncertainty values. The precise and complete geometric model allows to specify precise situations involving different objects (e.g. an object in a certain area, close to a certain software component).

The Aura project [13, 29] aims at providing each user with an invisible halo of computing and information services that persists regardless of location. They have deployed efforts at every level: from the hardware and network layers, through the operating system and middleware, to the user interface and applications. Their project ambition goes one step further compared to the others, as they want the user not to be restricted to classical devices, but should be able to interact continuously with his "aura" that follows him everywhere and at every time, using any available appliance, even the coffee maker while the user stands in front of it (as the video available on their web site [13] shows it).

Other big projects could be described. One can cite the following ones:

- The Portolano project [58] at the Washington University focuses on sensors management, networking, transparency to the end-user and trust. They have studied an infrastructure based on mobile agents that interact with applications and users. Data-centric routing automatically migrates data among applications on the user's behalf. Data thus becomes "smart" and serves as an interaction mechanism within the environment.
- The Endeavour project [57] at Berkeley University has studied a planet-scale, self-organizing, and adaptive Information Utility. Their main objective is to arbitrarily and automatically distribute data among Information Devices". Data are seen as software components that can advertise themselves, provide their own adaptable user interface and their own negotiation process for their integration in applications.
- The Sentient project [8] at AT&T Laboratories Cambridge is based on a device called a bat with a unique id, an ultrasound transmitter and a radio transceiver, 2 buttons and a beeper. It is located by a central controller, and the world model stores the correspondence between bats and their owners, applying algorithms to the bat location data to determine the location of the person or object which owns it.

All these projects focus on services and consider the environment as a halo of available services. The notion of data is not present in the front: data are embedded in software components. All information interesting the user or describing his way of working are represented in objects or services; the paradigm for the manipulation of artifacts are services or components. With the advent of the DataSpace notion [25], another vision has appeared, placing data at the centre of the pervasive system. Some projects have tried to focus on a data-like representation of the environment, including databases and data streams but also services. They have resulted in hybrid SQL-like systems that include remote services calls in queries (e.g. [31, 62]). They will be detailed in section 3.

2.3 Summary

In the previous section, we emphasized that pervasive systems need a certain degree of abstraction of devices about hardware, software and network capabilities. Middlewares and layered architectures are mainly used to achieve this level of abstraction, through a common representation of resources as services. Communications between services are often implemented as asynchronous messages that are independent of the underlying platform and network protocol. Context management is also a key element for dynamic adaptation of applications to their environment.

Among the restrictions imposed on pervasive applications by the social layer [41], a strong issue is to enforce security policy. As sharing of data and services among devices is one of the main point of pervasive systems, security is needed to protect access to resources and to ensure some level of user privacy. Despite many works on security for distributed systems, it remains an open issue in such complex envi-

ronments. Other restrictions may come from usability issues, aesthetics issues and environmental issues, in particular in term of energy consumption.

3 Related database research

Current trends in pervasive environment management can be analyzed as the leveraging of database principles, applied to a more dynamic and distributed setting. We first tackle the representation and management of streaming data sources. We then tackle data integration problems that occur in pervasive environment settings and describe the interplay between data and services.

3.1 Data streams

Pervasive systems often include services that periodically or occasionally generate data, be it events or sensor readings. Managing such data sources in programs (e.g. in a supervision system) can be complex as it implies asynchronous data handling. In order to cope with this complexity, database principles can be applied: data sources are represented in a way similar to relations in databases, and queries can be formulated in a declarative way using a SQL-like query language from which query optimization techniques can be applied.

Furthermore, data streams and relations may be handled in a homogeneous way so as to enable queries combining both types of data sources. Data streams represent the integration of dynamic data sources in databases, leading to the definition of continuous queries providing dynamic results that are continuously updated. Queries may also still be one-shot as standard SQL queries, i.e. their results are evaluated once and not updated.

Many projects have been launched on data streams, among which we quote NiagaraCQ, TelegraphCQ, Cougar, TinyDB, STREAM, the Global Sensor Network and Cayuga.

NiagaraCQ [17] introduces some definitions of continuous queries over XML data streams. Queries are defined as triggers and produced event notification in real-time. The TelegraphCQ system [14] proposes adaptive continuous query processing over streaming data and historical data.

Cougar [63, 11] and TinyDB [30] handle continuous queries over sensor networks with a focus on the optimization of energy consumption for sensors using in-network query processing. STREAM [7] defines a homogeneous framework for continuous queries over relations and data streams.

In those systems, continuous queries are defines using a SQL-like language. Another approach is tackled with Borealis [37, 3, 18]: a Distributed Stream Processing System (DSPS) enables to define dataflow graphs of operators in a “box & arrows” fashion, making distributed query processing easier.

Continuous queries can be used to define some parts of pervasive applications in a declarative way: in [26, 39], the progressive cleaning process for data retrieved from numerous physical sensors is defined by a pipeline of continuous queries declaratively defined in SQL. A complex event-processing using state-machine operator producing data streams is also proposed. In [4, 5], the Global Sensor Network, a middleware for sensor networks, enables to specify continuous queries as virtual sensors whose processing is specified declaratively in SQL, with a subquery for preprocessing each input stream.

Cayuga [20, 19] is a stateful publish/subscribe system for complex event monitoring where events are also defined by SQL-like continuous queries over data streams.

3.2 Data & services integration

In this section, we discuss the interplay between data and services, and possible optimizations for queries involving both types of data sources.

Data integration has been a long standing theme of research over the past 30 years. Now, the broader notion of dataspaces [25, 38] has appeared to provide base functionality over all data sources and applications, regardless of how integrated they are and without having a full control over the underlying data [25]. For example, to answer a query when some data sources are unavailable, the data accessible at the time of the query have to be used to propose the best possible results.

In the setting of data integration, the notion of *binding patterns* appears to be quite interesting since they allow to model a restricted access pattern to a relational data source as a specification of "which attributes of a relation must be given values when accessing a set of tuples" [24]. A relation with binding patterns can represent an external data source with limited access patterns in the context of data integration [24]. It can also represent an interface to an infinite data source like a web site search engine [31], providing a list of URLs corresponding to some given keywords. In a more general way, it can represent a data service, e.g. web services providing data sets, as a virtual relational table like in [53].

The SQL standard itself supports some forms of access to external functionalities through User-Defined Functions (UDF). UDFs can be scalar functions (returning a single value) or table functions (returning a relation). UDFs are defined in SQL or in another programming language (e.g. C, Java), enabling to access to any external resources. Table functions are a way to implement the notion of virtual tables, however limited to having only one binding pattern determined by the function input parameters. UDFs are also tagged as deterministic or non-deterministic: query rewriting may not change the number of invocations for non-deterministic UDFs. Abstract Data Types can also be used to get an object-oriented view of sensors, like in the Cougar project [11, 63].

Optimization of queries involving expensive functions or methods leads to the redefinition of cost models to integrate the estimated cost of computation. This issue

has been studied for standard databases [15, 16, 35, 36], and also for continuous query processing [21].

In a similar way to binding patterns, the ActiveXML language [1] allows to define XML documents containing extensional data, i.e. data that are present in the document, and intensional data, representing service calls that provide data when needed. Intensional data is close to the notion of virtual tables and binding patterns. ActiveXML is also a "framework for distributed XML data management" [6] and defines an algebra to model operations over ActiveXML documents distributed among peers, that enables query optimization.

In Aorta [62], 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 [5], the Global Sensor Network allows to define virtual sensors abstracting implementation details of data sources, and provides continuous query processing facilities over distributed data streams.

3.3 Summary

Database research that can be related to pervasive environments span across several issues. From a data-centric point of view, the management of pervasive environments is the management of distributed dynamic data sources and services that should be accessed through declarative queries: therefore, there is a need for integration of data streams, external methods and services, into relational or XML databases. Continuous or one-shot queries over such extended databases need to be declaratively defined, for example using a SQL-like language, optimized for this new setting, and processed (in real-time for continuous queries).

4 Enabling technologies

A pervasive environment is full of functionalities, but a user may be lost and not able to comprehend and optimally use all available data sources and services the environment can provide. Furthermore, applications are not easy to develop and maintain because of the heterogeneity and the dynamicity of the environment. Typically, low-level technical code using programming languages (Java, C#...) and network protocols has to be devised to come up with some pervasive applications.

In this section, we detail some technologies that enable to build pervasive environment systems. Those technologies tackle system problems like service discovery and remote invocation in a heterogeneous setting, but also some higher-level issues like a common data and service representation.

CORBA [47] (Common Object Request Broker Architecture) is an open architecture and infrastructure that enables applications to interoperate over network links.

It can be defined as an object bus: applications can access to local or remote objects without worrying about underlying network issues (including serialization issues). A lookup allows to search objects by name and get object references. Objects are defined using the platform-independent IDL (Interface Description Language) that can be used to generate stub and/or skeleton in many programming languages.

Some systems tackle the same issues, but are more platform- or language-dependent, like Microsoft DCOM (Distributed Component Object Model) or Java RMI (Remote Method Invocation).

Whereas those systems are a sort of object bus, other systems focus on a messaging protocol between services to achieve interoperability. Those protocol are more data-oriented. The open standard XML (eXtended Markup Language) is often used as the message format for such protocols, like for the simple yet efficient XML-RPC (XML - Remote Procedure Call) or its more complex but powerful evolution SOAP (Simple Object Access Protocol) for Web Services. REST (REpresentational State Transfer) relies on the HTTP API to transfer messages, but do not define a message format: it is rather an architecture style using the well-established HTTP protocol to simplify communications. For those protocols, service discovery needs to be done by external registries, like UDDI (Universal Description, Discovery and Integration) for Web Services.

UPnP [59] (Universal Plug and Play) and the more recent DPWS [60] (Device Profile for Web Services) are based on some messaging protocols and include automatic discovery mechanisms, using network broadcast facilities. UPnP/DPWS entities are devices that host several services providing methods and events.

JMX (Java Management eXtension) and OSGi [48] are two Java framework that can host some (potentially active) java objects as services and enable a local and remote access to them through various network protocols like RMI, Web Services, or even UPnP and DPWS if dynamic discovery is needed.

Nowadays, these relatively recent technologies have become mature, some of them being used in the industry (in particular for application servers). Focused on interoperability issues in a heterogeneous setting, they can be re-used in the context of pervasive environment management systems.

5 SoCQ: a comprehensive PEMS

Managing pervasive environments, in particular heterogeneous devices, remains a complex task: a certain level of abstraction and loosely coupled communications can be achieved with current middlewares, but application developer support still can not hide resource heterogeneity. However, with the adoption of a data-centered point of view, this heterogeneity can be further abstracted: devices can be represented as distributed data sources providing data, data streams and services that are manageable in a homogeneous way.

The **S**ervice-**o**riented **C**ontinuous **Q**uery project, or SoCQ project [33, 52], is devoted to making the development of pervasive applications easier through database

principles. It aims at contributing in the area of Dataspaces [25, 38] through a unified view of data and service spaces mandatory in pervasive environments.

We are currently working on the definition of an approach to homogeneously represent such pervasive environments through database principles. The basic idea is to present to application developers a database-like view of the environment resources, so that they can visualize this environment as a set of tables and launch declaratively-defined continuous queries involving available data sources and services. This approach is built on an extension of the relational model and uses a SQL-like query language.

DBMSs (DataBase Management Systems) provide a homogeneous view as well as storage and query facilities for relational data. DSMSs (Data Stream Management Systems) also provide a homogeneous view and query facilities for both relational data and data streams. We then call PEMS, for Pervasive Environment Management System, a system that manages in a similar way an environment containing data relations, data streams and services.

In this project, we tackle the following challenges:

- definition of a homogeneous representation for databases, data streams and services from the pervasive environment,
- definition of a query language over pervasive environments allowing to easily develop pervasive applications,
- design of a Pervasive Environment Management System (PEMS) supporting both the homogeneous representation and the query processing facilities.

In Figure 1, the different elements of a PEMS are shown. A distributed resource manager handles service discovery and remote invocations, with local resource managers as proxies for local devices that provide data, streams and services. An extended table manager builds a homogeneous representation of non-conventional data sources, and the query processor allows to define, optimize and execute queries.

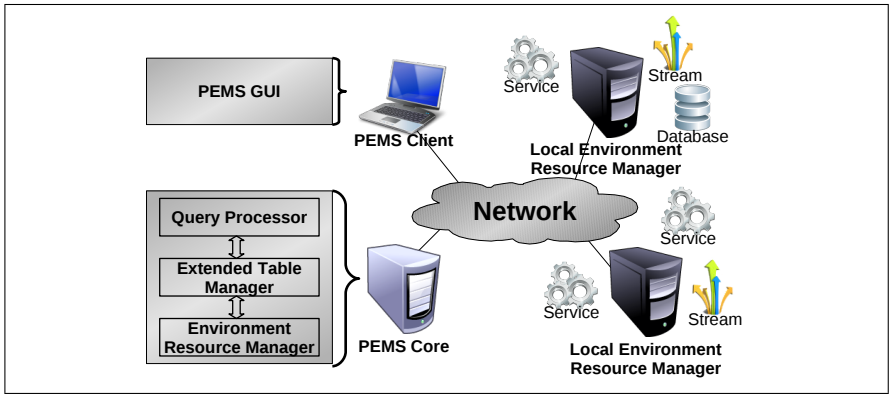

Fig. 1 Overview of a PEMS environment

5.1 Example scenario

In the example scenario, we monitor temperatures in an office building: when a temperature exceeds some threshold in a room, an alert message is sent to the manager of this room. A photo of the room can be joined to the message. We simulate an environment, illustrated in Figure 2, containing the following data sources and services:

- two data relations: one containing some information about the rooms (manager, temperature threshold...), the other one being a list of contacts (including contacts of the managers),
- some temperature sensors distributed in several rooms, providing data streams,
- some cameras installed in the rooms, providing photo services,
- some messenger services (by mail, instant message, SMS).

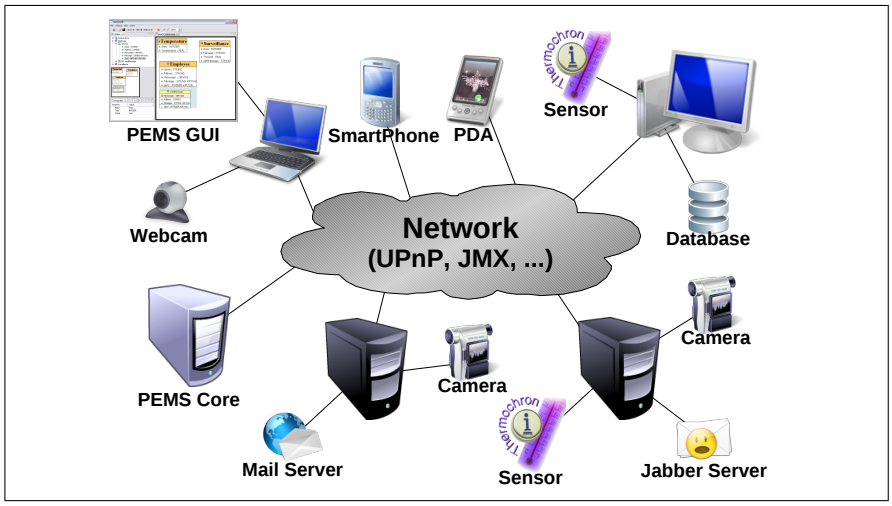

Fig. 2 Illustration of the scenario environment

This environment can be represented homogeneously with relations and streams extended with virtual attributes and binding patterns. Virtual attributes are attributes that do not have a value and may be provided a value through a query, due to binding patterns that indicate their relationship with method prototypes from services. We call such relations *XD-Relations*, standing for eXtended Dynamic Relations, and such environments *relational pervasive environments*. For the example scenario, we can view a DDL representation of the schema of this environment in Table 1.

With such environments, the use of distributed functionalities provided by services is declaratively specified in SQL-like queries by the virtual attributes that need to be realized, i.e. that need to be provided a value. In order to realize those

Table 1 DDL description of prototypes and XD-Relations for the environment of the example scenario

```
PROTOTYPE sendMessage( address STRING, text STRING ) :
  (sent BOOLEAN) ACTIVE;

PROTOTYPE takePhoto( ) :
  ( photo BLOB );

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

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

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

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

attributes, the corresponding binding patterns are invoked for every involved tuples, leading to several service invocations. We call these queries SoCQ, for Service-oriented Continuous Queries.

Over this environment, many different SoCQ queries could be launched. For example, the temperature monitoring can be declaratively defined as a continuous query. The three XD-Relations are joined (the stream "temperature" must be windowed) on the manager name and the area, the threshold is checked and the message body is set. The binding pattern "sendMessage" will be invoked in order to fetch a value for the virtual attribute "sent". The SQL-like query in Table 2 is a typical example of a pervasive application that is defined at the declarative level, without worrying about low-level technical considerations (programming languages, network protocols).

The role of a PEMS is to manage a relational pervasive environment, with its dynamic data sources and set of services, along with the execution of the continuous

Table 2 SoCQ query for the example scenario

```
SELECT surveillance.area, surveillance.manager, employees.sent
FROM   temperatures [now], employees, surveillance
WHERE  surveillance.manager = employees.name
  AND  surveillance.area = temperatures.area
  AND  surveillance.threshold < temperatures.temperature
  AND  employees.text IS surveillance.alertMessage
```

queries over this environment. In the following sections, we first sketch the data model that supports XD-Relations, and the algebra that enables SoCQ queries. We then give an overview of our PEMS implementation.

5.2 Modeling of Pervasive Environments

In order to homogeneously represent data sources and other resources from pervasive environments, we propose a model that integrate 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(text,address):(sent)` that takes two input attributes `text` and `address` and provides one output attribute `sent`. 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 service invocations can have an impact on the physical environment, e.g. invoking a service 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 (e.g. in Table 1, `sendMessage` is tagged as active). On the opposite, the impact of passive prototypes is non-existent or can be neglected, like reading sensor data (e.g. `takePhoto`).

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 one non-virtual attribute as the service reference attribute, the prototype and which attributes are linked with the prototype input and output attributes. For example, the

`employees` relation (see Table 1) 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.

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 value 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 `employees` 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 (*) symbol reminds that virtual attributes do not have a value.

name	address	messenger	text	sent
nicolas	nicolas@elysee.fr	mailer	*	*
carla	carla@elysee.fr	mailer	*	*
françois	francois@im.gouv.fr	jabber	*	*

Pervasive environments being dynamic, data sources may include streaming data. 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.

5.3 Service-oriented Continuous Queries

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, like in the example scenario. They are called Service-oriented Continuous Queries, or SoCQ queries. However, some queries may be snapshot queries, i.e. queries executed once that produce their results and do not maintain them, like standard SQL queries in DBMS.

SoCQ queries are based on the so-called Serena algebra (**Ser**vice-**ena**bled algebra) that defines query operators over XD-Relations. Standard relational operators are redefined over finite XD-Relations, and new operators are defined. Realization operators handle 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 and streaming operators handle 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 tuple inserted into the relation operand).

A SQL-like query language has been defined to declaratively express SoCQ queries. For example, for the example scenario, the query in Table 2 involves several operators: windows (the `[now]` is 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 actually send messages to area managers).

5.4 Implementation of PEMS

The PEMS core is composed of three logical layers (see Figure 1). A global resource manager handles service discovery and remote invocations, with local resource managers as distributed proxies for local devices that provide services. An extended table manager builds a homogeneous representation of non-conventional data sources, and the query processor allows to define, optimize and execute Service-oriented Continuous Queries. These layers are composed of several internal modules sketched in Figure 3.

The PEMS prototype is developed in the Java/OSGi framework [48]. Each module of the PEMS is an OSGi bundle and communicates with each other through the OSGi service life cycle management. The chosen network protocol for service discovery and remote invocations is UPnP [59]: the prototype uses the dedicated standard OSGi bundles for this protocol.

The PEMS GUI, shown in Figure 4, is also developed in the Java/OSGi framework, as an Eclipse RCP Plugin, i.e. the GUI is integrated in the Eclipse platform. It communicates remotely with the PEMS core through a JMX interface. It enables to visualize existing XD-Relations and their content, to add/alter/delete XD-Relations, and to launch/stop SoCQ queries.

6 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 databases, data streams and services is still lacking and is a major bottleneck toward the declarative definition of pervasive applications.

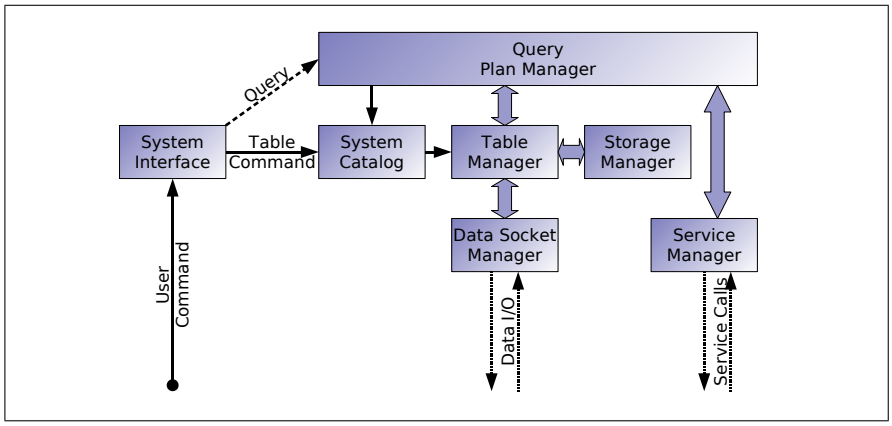

Fig. 3 Internal modules of the PEMS core

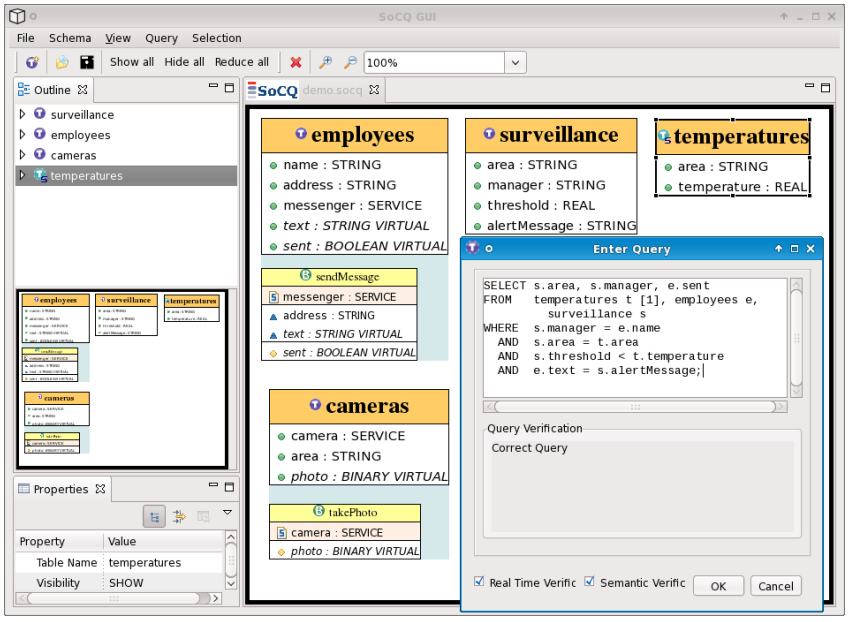

Fig. 4 The PEMS GUI

Pervasive environments are complex environments that raise issues in several research domains. Studying pervasive computing from a data-centric point of view raises some similarity with current database research like data streams, data and services integration, or distributed databases.

In this setting, the SoCQ project is our attempt to bridge the gap between pervasive computing and the database domain. It demonstrates the following points: 1) a homogeneous database-like view on pervasive environments containing dynamic

data sources and services is possible as a set of XD-Relations, through the notions of virtual attributes and binding patterns; 2) Service-oriented Continuous Queries (SoCQ queries) over relational pervasive environment allow to define pervasive applications combining data sources and services. A formal algebra has been devised allowing to apply query optimization techniques in pervasive environments. Such declarative definitions of SoCQ queries make the definition and the evolution of pervasive applications easier.

References

1. ActiveXML. `http://www.activexml.net/`
2. Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 3896. Springer (2006)
3. Abadi, D.J., et al.: The Design of the Borealis Stream Processing Engine. In: CIDR 2005, Proceedings of Second Biennial Conference on Innovative Data Systems Research (2005)
4. Aberer, K., Hauswirth, M., Salehi, A.: A middleware for fast and flexible sensor network deployment. In: VLDB 2006, Proceedings of the 32th International Conference on Very Large Data Bases (2006)
5. Aberer, K., Hauswirth, M., Salehi, A.: Infrastructure for data processing in large-scale interconnected sensor networks. In: MDM 2007, Proceedings of the 8th International Conference on Mobile Data Management (2007)
6. Abiteboul, S., Manolescu, I., Taropa, E.: A framework for distributed xml data management. In: EDBT [2], pp. 1049–1058
7. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., Widom, J.: STREAM: The Stanford Stream Data Manager. IEEE Data Engineering Bulletin **26**(1), 19–26 (2003)
8. ATT Laboratories, Cambridge: Sentient Computing Project. `http://www.cl.cam.ac.uk/research/dtg/attarchive/spirit/`
9. Becker, C., Handte, M., Schiele, G., Rothermel, K.: PCOM – A Component System for Pervasive Computing. In: PerCom'04, Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications, p. 67 (2004)
10. Biegel, G., Cahill, V.: Requirements for middleware for pervasive information systems, pp. 86–102. Vol. 10 of Kourouthanassis and Giaglis [42] (2007)
11. Bonnet, P., Gehrke, J., Seshadri, P.: Towards sensor database systems. In: MDM 2001, Proceedings of the Second International Conference on Mobile Data Management, pp. 3–14 (2001)
12. Brumitt, B., Meyers, B., Krumm, J., Kern, A., Shafer, S.: EasyLiving: Technologies for intelligent environments. In: HUC 2000, Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing, pp. 12–29 (2000)
13. Carnegie Mellon University: Project Aura, Distraction-free Ubiquitous Computing. `http://www.cs.cmu.edu/~aura/`
14. Chandrasekaran, S., 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)
15. Chaudhuri, S., Shim, K.: Query optimization in the presence of foreign functions. In: VLDB '93: Proceedings of the 19th International Conference on Very Large Data Bases, pp. 529–542. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
16. Chaudhuri, S., Shim, K.: Optimization of queries with user-defined predicates. ACM Trans. Database Syst. **24**(2), 177–228 (1999). DOI http://doi.acm.org/10.1145/320248.320249

17. Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 379–390 (2000)
18. Cherniack, M., et al.: Scalable Distributed Stream Processing. In: CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research (2003)
19. Demers, A.J., Gehrke, J., Hong, M., Riedewald, M., White, W.M.: Towards expressive publish/subscribe systems. In: EDBT [2], pp. 627–644
20. Demers, A.J., Gehrke, J., Panda, B., Riedewald, M., Sharma, V., White, W.M.: Cayuga: A general purpose event monitoring system. In: CIDR, pp. 412–422. www.crdrdb.org (2007)
21. Denny, M., Franklin, M.J.: Operators for expensive functions in continuous queries. In: ICDE '06: Proceedings of the 22nd International Conference on Data Engineering, p. 147. IEEE Computer Society, Washington, DC, USA (2006). DOI http://dx.doi.org/10.1109/ICDE.2006.110
22. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction **16**(2), 97–166 (2001)
23. Estrin, D., Culler, D., Pister, K., Sukhatme, G.: Connecting the Physical World with Pervasive Networks. IEEE Pervasive Computing **1**(1), 59–69 (2002)
24. Florescu, D., Levy, A., Manolescu, I., Suciu, D.: Query Optimization in the Presence of Limited Access Patterns. In: SIGMOD'99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, pp. 311–322 (1999). DOI http://doi.acm.org/10.1145/304182.304210
25. Franklin, M., Halevy, A., Maier, D.: From Databases to Dataspaces: a new Abstraction for Information Management. SIGMOD Rec. **34**(4), 27–33 (2005)
26. Franklin, M.J., 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)
27. Gajos, K., Fox, H., Shrobe, H.: End user empowerment in human centered pervasive computing. In: Pervasive 2002. Zurich, Switzerland (2002)
28. Garcia-Molina, H., Widom, J., Ullman, J.D.: Database System Implementation. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1999)
29. Garlan, D., et al.: Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing **1**(2), 22–31 (2002)
30. Gehrke, J., Madden, S.: Query processing in sensor networks. Pervasive Computing, IEEE **3**(1), 46–55 (2004). DOI 10.1109/MPRV.2004.1269131
31. Goldman, R., Widom, J.: WSQ/DSQ: A Practical Approach for Combined Querying of Databases and the Web. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 285–296 (2000)
32. Grimm, R., et al.: System Support for Pervasive Applications. ACM Transactions on Computer Systems **22**(4), 421–486 (2004)
33. Gripay, Y.: Service-oriented Continuous Queries for Pervasive Systems. In: EDBT 2008 PhD Workshop (2008). URL `http://liris.cnrs.fr/publis/?id=3428`
34. Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., Jansen, E.: The gator tech smart house: A programmable pervasive space. Computer **38**(3), 50–60 (2005)
35. Hellerstein, J.M.: Optimization techniques for queries with expensive methods. ACM Transactions on Database Systems **23**(2), 113–157 (1998). DOI http://doi.acm.org/10.1145/292481.277627
36. Hellerstein, J.M., Stonebraker, M.: Predicate migration: Optimizing queries with expensive predicates. In: SIGMOD'93, Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 267–276 (1993)
37. Hwang, J.H., Xing, Y., Cetintemel, U., Zdonik, S.: A cooperative, self-configuring high-availability solution for stream processing. In: ICDE'07, Proceedings of the 23rd International Conference on Data Engineering (2007)
38. Imielinski, T., Nath, B.: Wireless graffiti: data, data everywhere. In: VLDB'2002, pp. 9–19 (2002)

39. Jeffery, S.R., Alonso, G., Franklin, M.J., Hong, W., Widom, J.: Declarative support for sensor data cleaning. In: Pervasive, pp. 83–100 (2006)
40. Koile, K., Tollmar, K., Demirdjian, D., Shrobe, H., Darrell, T.: Activity zones for context-aware computing. In: Proceedings of UbiComp 2003, pp. 90–106. Springer-Verlag (2003)
41. Kourouthanassis, P.E., Giaglis, G.M.: The design challenge for pervasive information systems, pp. 29–85. Vol. 10 of *Advances in Management Information Systems* [42] (2007)
42. Kourouthanassis, P.E., Giaglis, G.M. (eds.): Pervasive Information Systems, *Advances in Management Information Systems*, vol. 10. M.E. Sharpe, Armonk, NY (2007)
43. Kourouthanassis, P.E., Giaglis, G.M.: Toward pervasiveness, pp. 3–25. Vol. 10 of *Advances in Management Information Systems* [42] (2007)
44. Levene, M., Loizou, G.: A Guided Tour of Relational Databases and Beyond. Springer-Verlag (1999)
45. Microsoft Research: EasyLiving. `http://research.microsoft.com/easyliving/`
46. MIT: Oxygen Project, Pervasive, Human-centered Computing. `http://oxygen.csail.mit.edu/`
47. OMG: CORBA. `http://www.corba.org/`
48. OSGi Alliance: `http://www.osgi.org/`
49. OSGi Alliance: Listeners Considered Harmful: The "Whiteboard" Pattern. Technical Whitepaper, `http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf` (2004)
50. Saha, D., Mukherjee, A.: Pervasive computing: a paradigm for the 21st century. Computer **36**(3), 25–31 (2003)
51. Saif, U., Pham, H., Paluska, J.M., Waterman, J., Terman, C., , Ward, S.: A case for goal-oriented programming semantics. In: UbiSys'03: Workshop on System Support for Ubiquitous Computing, 5th International Conference on Ubiquitous Computing (UbiComp 2003) (2003)
52. SoCQ Project: `http://socq.liris.cnrs.fr/`
53. Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query Optimization over Web Services. In: VLDB 2006, Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 355–366 (2006)
54. Strang, T., Linnhoff-popien, C.: Service interoperability on context level in ubiquitous computing environments. In: SSGRR2003w, Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (2003)
55. Tian, F., DeWitt, D.J.: Tuple Routing Strategies for Distributed Eddies. In: VLDB 2003, Proceedings of the 29th International Conference on Very Large Data Bases, pp. 333–344 (2003)
56. Union, I.T.: The Internet of Things. ITU Internet Reports. International Telecommunication Union (2005)
57. University of California, Berkeley: The Endeavour Expedition: Charting the Fluid Information Utility. `http://endeavour.cs.berkeley.edu/`
58. University of Washington: Portolano: An Expedition into Invisible Computing. `http://portolano.cs.washington.edu/`
59. UPnP Forum: Universal Plug and Play. `http://www.upnp.org/`
60. Web Services for Devices (WS4D): Devices Profile for Web Services (DPWS). `http://ws4d.org/`
61. Weiser, M.: The Computer for the 21st Century. Scientific American **265**(3), 94–104 (1991)
62. Xue, W., Luo, Q.: Action-Oriented Query Processing for Pervasive Computing. In: CIDR 2005, Proceedings of the Second Biennial Conference on Innovative Data Systems Research (2005)
63. Yao, Y., Gehrke, J.: Query Processing in Sensor Networks. In: CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research (2003)
64. Zhu, F., Mutka, M., Ni, L.: Service Discovery in Pervasive Computing Environments. IEEE Pervasive Computing **4**(4), 81–90 (2005)