SStreaMWare: a service oriented middleware for heterogeneous sensor data management

Levent Gürgen*, Claudia Roncancio†, Cyril Labbé †, André Bottaro* and Vincent Olive*
* Orange Labs, France Telecom R&D – Grenoble, France
Email: firstname.surname@orange-ftgroup.com
† LIG Laboratory – Grenoble, France
Email: firstname.surname@imag.fr

Abstract— **Smart sensors are already being used in various application domains such as medical, environmental, urban, domestic and industrial. These applications mostly need data from sensors of different types (temperature, pressure, location, camera, etc.) that may be managed by different software, e.g., proprietary software from manufacturers. Heterogeneous distributed sensor data should then be aggregated in order to obtain more accurate and complete *information* on real world events. This paper proposes SStreaMWare, a service-oriented middleware for heterogeneous sensor data management. SStreaMWare's simple data schema allows data representation of various types of sensors in a common generic way. Declarative queries can then be formulated according to this schema. Thanks to the service-oriented approach of SStreaMWare, heterogeneity of sensor software is hidden by generic query services, which can be discovered and used dynamically.**

I. Introduction

Along with the technological advances, sensors are more and more intelligent, autonomous, small and cheap. Their small size, as well as their communication and computing autonomy, allow them to be everywhere and to make information omnipresent. They are already being used in various application domains such as medical, environmental, urban, domestic and industrial. Sensors form a virtual skin between the real and the digital world. They measure various real world conditions such as temperature, pressure, humidity, chemical, location, audio or visual information. Measured events are processed by applications which react, for instance, by triggering alarms, controlling actuators or applying business rules.

Sensors have come with new challenges to many areas of computer science such as networks, databases, embedded operating systems and more generally, ubiquitous computing [1]. In particular, with the multitude of "sensor-based" applications, a need for a more generic, reusable way of sensor querying is raised. A generic solution would be preferred to application specific data querying and processing scripts. To fulfill this need, the "database approach" is now adopted by a large number of proposals [2], [3], [4], [5], [6], [7]. The reason of this evolution is similar to that of emergence of DBMS a couple of decades ago, i.e. transition from application dependent data files to application independent database systems.

The "database approach" allows different applications to use a same sensor set accessed via declarative queries. However, a given application may also need to access different types of sensors or sensors of the same type but managed by different proprietary software. The former is related to **data heterogeneity** and the latter to **software heterogeneity**. Currently, most of the existing sensor data management solutions suppose homogeneous sensors generating data according to a same data schema, and being managed by known common software. Nevertheless, the trend is to have numerous and heterogeneous sensors getting involved in applications. In order to obtain valuable information on a given physical environment, such applications has to filter, aggregate, merge, fuse heterogeneous data streams.

This paper proposes SStreaMWare [8], a service-oriented sensor mediator middleware. SStreaMWare is placed between applications and heterogeneous sensor farms. It provides a query service supporting a declarative query language to access data streams issued by heterogeneous sensors. SStreaMWare adopts a mediator-adaptor approach [9] to cope with data heterogeneity and uses a service-oriented architecture to handle software heterogeneity and to be able to operate upon a dynamic set of distributed sensors.

SStreaMWare components can be deployed according to a hierarchical organization to fit the natural hierarchy of geographic locations [10]: a monitored zone is divided into regions; each one can be divided in their turn depending on the scale of the environment to be monitored. User queries are submitted to a top-level control-site of SStreaMWare. Service orientation brings loose coupling and dynamic discovery to SStreaMWare network organization. Sensors concerned by a query are identified dynamically by a service discovery mechanism. A query is then rewritten and distributed to be evaluated by the appropriate gateways in charge of the relevant sensors. Sensors form the lowest level of the hierarchy. Generic sensor services provide a common way of access data streams of heterogeneous sensors. In other words, these *adapter services* form an interface between the sensors (or sensor specific software) and SStreaMWare.

As we will see in the following, main design choices in SStreaMWare are motivated by the will 1) to provide declarative querying of large scale sensor farms and 2) to manage sensor heterogeneity and dynamicity.

The paper is organized as follows. Background and related work are presented in section II. The architecture of SStreaMWare is detailed in section III. Section IV introduces the data representation adopted by the mediator to provide a homogeneous view over heterogeneous sensor data. It also introduces query capabilities supported by SStreaMWare. Section V gives a detailed description of the middleware and its implementation whereas section VI presents its experimentation in different contexts. Section VII gives our concluding remarks and future work.

II. Related Work

Existing sensor data management systems can be classified into three categories according to the place where query processing takes place: purely distributed in sensor networks [2], [3], [11]; centralized evaluation in data stream management systems [4], [5], [6]; and hybrid systems [12], [13], [14], [15], [16], [7], [8]. Following subsections discuss each category, its advantages and drawbacks, and their positioning with respect to scalability, heterogeneity and query support.

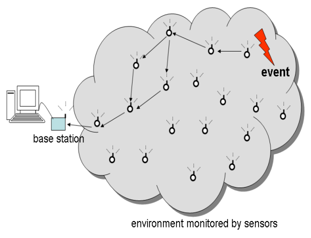
A. Sensor Networks

Sensor networks [2], [3], [11] provide a **distributed** evaluation of queries **at the sensors** by exploiting their computing capacity. The main objective is to limit message transmissions in order to save energy, precious resource for battery operated sensors. In-network aggregation [17] is another technique for energy saving which allows calculating some aggregation operators in the network. It takes advantage of multi-hop routing protocols used in sensor networks. Sensors are dispersed in the environment to measure physical quantities or to detect events. They are auto-organized in an ad-hoc manner. Queries on sensor data include statements to reflect their temporal and continuous nature. They are mostly declarative and formulated by a SQL-like language [3]. Query answers are typically sent to a base station which forms the interface between the applications and the sensors (see Figure 1(a)). Many sensor networks are already deployed in academic and industrial contexts.

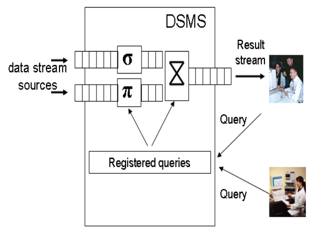
The purely distributed infrastructureless approach of sensor networks allow them to **scale well** against increasing number of involved sensors. However, sensors are usually supposed to be **homogeneous**, with the same communication protocol, same query processor, same data schema, etc. This is a strong restriction to deploy large scale networked sensing systems. In addition, queries are supposed to be relatively **simple** due to the limited resources of sensors and to the high distribution in the network.

B. Data Stream Management Systems (DSMS)

DSMS [4], [5], [6] are considered as **centralized** as they use central servers to evaluate continuous queries on data streams issued by sensors or other data stream sources (see Figure 1(b)). The fundamental difference between traditional DBMS and DSMS is that, in DBMS data are persistent and queries are transitory (they are evaluated and dropped), while in DSMS queries are persistent (evaluated continuously) and data are transitory (they are used to evaluate queries and then dropped). Data streams are considered as infinite sequences of tuples ordered by their timestamp. Queries are formulated by using declarative query languages [18], [6]. They are continuously evaluated on, possibly infinite, data streams or on finite *data windows* by blocking operators [19]. DSMS are ideal for applications in which data processing time is critical, i.e., which need to process large amounts of frequent data in order to react as soon as possible. Typical examples include financial applications, sensor based applications, transactional log analysis, etc.



(a) In sensor networks, data are processed in the network



(b) In DSMS, data are processed in a centralized server

Fig. 1. Two distinct approaches for sensor data management

As DSMS are centralized systems, despite various techniques (load shedding, approximate query answers, resource sharing [19]) proposed to resist against increasing rate and number of data to process, **bottlenecks** are inevitable with high charge. **Heterogeneity** is not much explored in DSMS. Data are supposed to be in a format comprehensible by the DSMS. Some proposals use adapters to translate data formats [6]. Queries may be **expressive** since the centralized query evaluation allows supporting complex operators such as join, aggregation on sliding windows, etc.

C. Hybrid Approaches

SStreaMWare is a hybrid solution that aims to take advantages of each category presented above and proposes a solution with particular focus on issues such as scalability, sensor heterogeneity management and complex query evaluation. Several similar hybrid solutions are recently proposed in the literature. This section briefly describes existing work and gives the positioning of SStreamWare according to different aspects such as architecture, heterogeneity management, query representation, etc. More detailed analysis can be found in [20].

1) **Existing Work***:* This section gives a very brief survey of some of the existing work related to SStreaMWare.
Fjords [12] is a proposition of distributed infrastructure on which can be constructed sensor data management systems. Sensor networks and a central query processor are interfaced by *proxies*. Queries on both persistent and stream data are supported.
Borealis [21] is a distributed data stream management system. One part of the project concerns integration of sensor networks [13]. Sensor networks are interfaced with Borealis nodes by intermediary *proxies*. Each sensor network provides an *adapter* in order to provide common information to the Borealis node.
IrisNet [14] envisions a sensor network at the Internet scale. It provides software components to facilitate the deployment of sensor services. The objective is to allow users to find and

use different kinds of sensor services available in a large scale network.

Hourglass [15] proposes, similarly to IrisNet, a service infrastructure to interconnect sensors and applications via *services*. Based on a publish-subscribe mechanism, producers publish their services and consumers subscribe to interesting services.

GSN [16] is conceived for a fast and flexible deployment of applications which need integration of data from heterogeneous sensors. "Virtual sensors" abstract the sensors from the physical implementations and provide a homogeneous view of sensor data.

HiFi [7] is based on a hierarchical, location-based organization for sensor data processing. Sensors form the leaves of the hierarchical tree and the intermediary nodes are relatively powerful entities performing different operations such as filtering, data cleaning, aggregation and join.

2) **Synthesis***:* Above-mentioned proposals share similar aspects on some points and differ on some others. Below is a discussion on these points, as well as the positioning of SStreaMWare among these proposals. Figure 2 synthesizes the discussion.

Architecture

At the architectural level, the proposals can be divided into two groups: hierarchical [7], [21], [14], [12] and peer-to-peer (P2P) [16], [15]. P2P architectures provide scalable and flexible (easy node addition/suppression) deployment features. However, this approach suffers from limited control on the nodes and difficulties for maintaining a directory of available nodes. On the other hand, hierarchical architectures may keep partial control at different levels of the architecture. In addition, this kind of architectures seems natural when dealing with sensor data which is strongly associated with location information. For these reasons, SStreaMWare also adopts a hierarchical approach while providing dynamic service discovery at every level of the architecture. Service orientation allows dynamic addition, removal and modification of sensor services in the architecture.

Heterogeneity

Each proposal deals with heterogeneity issues at a certain degree by using *adapters* – although they use different terms. This adapter layer provides an abstraction of sensor data access, and performs necessary translation operations for data formats or method calls. Some proposals [15], [16], [14], just as SStreaMWare, adopt a service-oriented approach which offers better adaptability thanks to its loosely-coupled approach.

Query expression

Utility of declarative queries for sensor querying is now well admitted. For query formulation, various techniques are adopted by existing proposals: SQL-like query languages [16], [7], XML derived languages [15], [14] or graphical interfaces [21]. SStreaMWare provides a graphical interface and also supports simple textual SQL-like queries.

Query evaluation

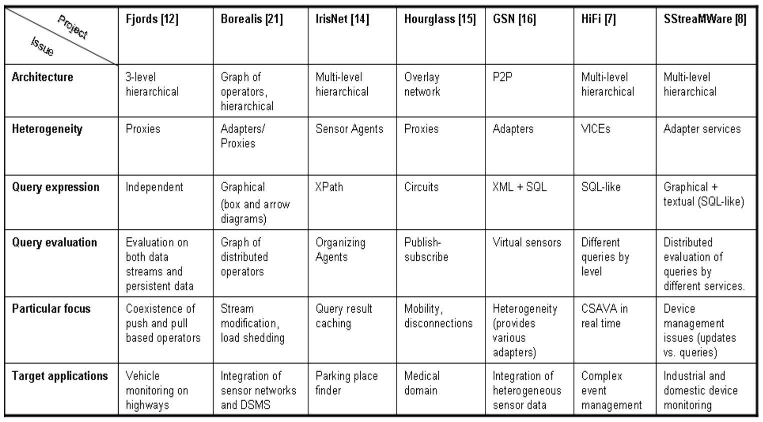
All the proposals mentioned above propose a distributed query evaluation mechanism. Different parts of queries are evaluated at the nodes distributed either with a hierarchical or a peer-to-peer manner. Queries involve operators on data streams and/or windows. SStreaMWare queries are composed of sub-queries evaluated at different levels in its hierarchical architecture and support both stream and window operators.

Particular focus

Besides sensor data querying functionalities, each proposal deals with a particular problem identified in its context; e.g., the coexistence of push and pull based operators [12], load shedding [22], semantic cache [23], sensor mobility [15], sensor heterogeneity [16], real-time event management [24]. SStreaMWare is particularly interested in *device management* issues for sensors. It aims to deal with the dynamicity caused by various operations (e.g., updates) on sensors and their effect on concurrent continuous queries. However, this subject is out of the scope of this paper. More information on this issue can be found in [25].

Target applications

Although each proposal argues that it is generic enough to be applied in various contexts, it targets more specifically an application or a family of application domain; e.g., vehicle traffic monitoring [12], integration of sensor networks and DSMS [13], parking place finder application [14], medical applications [15], heterogeneous data integration [26], complex event management generated by RFID readers [24]. SStreaMWare is conceived to be sufficiently generic in order to be applied to a large range of applications that monitor, in real-time, a physical environment with sensors possibly heterogeneous and belonging to different organizations. SStreaMWare is used to monitor electric power materials in the context of a French government funded project [27]; and its utility in various other contexts is also demonstrated [28].



Project / Issue	Fjords [12]	Borealis [21]	IrisNet [14]	Hourglass [15]	GSN [16]	HiFi [7]	SStreaMWare [8]
Architecture	3-level hierarchical	Graph of operators, hierarchical	Multi-level hierarchical	Overlay network	P2P	Multi-level hierarchical	Multi-level hierarchical
Heterogeneity	Proxies	Adapters/ Proxies	Sensor Agents	Proxies	Adapters	VICEs	Adapter services
Query expression	Independent	Graphical (box and arrow diagrams)	XPath	Circuits	XML + SQL	SQL-like	Graphical + textual (SQL-like)
Query evaluation	Evaluation on both data streams and persistent data	Graph of distributed operators	Organizing Agents	Publish-subscribe	Virtual sensors	Different queries by level	Distributed evaluation of queries by different services.
Particular focus	Coexistence of push and pull based operators	Stream modification, load shedding	Query result caching	Mobility, disconnections	Heterogeneity (provides various adapters)	CSAVA in real time	Device management issues (updates vs. queries)
Target applications	Vehicle monitoring on highways	Integration of sensor networks and DSMS	Parking place finder	Medical domain	Integration of heterogeneous sensor data	Complex event management	Industrial and domestic device monitoring

Fig. 2. Comparison of proposals according to different aspects

III. Architecture

As shown in the preceding section, hybrid approaches have emerged for heterogeneous large scale sensor information systems. SStreaMWare is based on such approach and adopts a hierarchical service-oriented architecture [10]. The reason of choosing a distributed hierarchical form for the architecture is principally based on two points: natural hierarchy of geographic locations and distribution of query evaluation charge at different levels in the architecture. In fact, sensors are distributed in physical environments and their measures are strongly coupled with their location. Environments can be hierarchically divided into sub-regions which can be controlled relatively independent one from the other. This leads more modular systems, while still keeping the control on sub-hierarchies.

Moreover, our service-oriented approach provides an efficient way for the management of the dynamicity of sensor systems. SStreaMWare supports pervasive sensor data management services that may arrive, leave, or be modified at the different levels of the architecture at any time during the application lifetime. Service dynamicity is automatically handled by lookup and binding services at every level of the architecture.

The number of levels in the hierarchical architecture depends on the scale targeted by the system. However there are 4 main elements in the architecture:

Sensors. Sensors are distributed in the environment. They form the lowest level in the hierarchy. In general, each sensor forms a leaf in the hierarchical tree. Nevertheless, a sensor network behind a proxy can be considered as a leaf as well. Sensors may be of different types from different manufacturers. They detect events or they send periodically their measures. SStreaMWare does not control the sensors directly. It rather accesses their data through their proprietary software.

Proxies and adapters. Sensors are of different manufacturers that propose their own specific software which we call *proxy*. In general, the proxy is accessible by an API which provides specific methods to query and manipulate its sensors. A proxy may be ”SStreaMWare-aware”, thus provide sensor access methods and data formats known by SStreaMWare. Otherwise, an *adapter* should be conceived in order to integrate the sensor specific proxy to our generic middleware. In fact, one of the main objectives of SStreaMWare is to be able to reuse sensors already deployed in the environment by providing a common way of access to sensors. Adapters should provide the translation of method calls and data format between SStreaMWare and sensor specific proxies.

Gateways. A gateway is a software platform that hosts different types of proxies and/or adapters. Gateways are responsible for the sensor management in a particular region, i.e., there is one gateway per sub-region of an environment (one gateway per room, per building, per section, etc.). Queries concerning the sensors of a sub-region are evaluated at the gateway of that region. Queries concerning several sub-regions are evaluated on higher levels (e.g., control sites) where partial results are integrated. Gateways are assumed to be hosted on relatively powerful devices such as PCs, industrial or domestic gateways.

Control sites. A control site constitutes the highest level of the hierarchical architecture. It is responsible of an entire environment which is possibly composed of several sub-regions. If we consider, for instance, an industrial site as the environment to monitor, a control site accepts queries on all the sensors located in the site. Sub-queries are then evaluated at the gateways responsible of the sub-regions, e.g., warehouses, production units, distribution units. See Figure 3 for a deployment example.

Larger environments can be managed by several control sites. For instance, while one control site manages a local industrial site, another control site can manage several control sites at a regional or national scale. Query evaluation is distributed among the levels of the architecture presented above. It is provided by various services which will be presented in section V-A. Queries are formulated according to a common global schema. Next section presents the data and the query model of SStreaMWare.

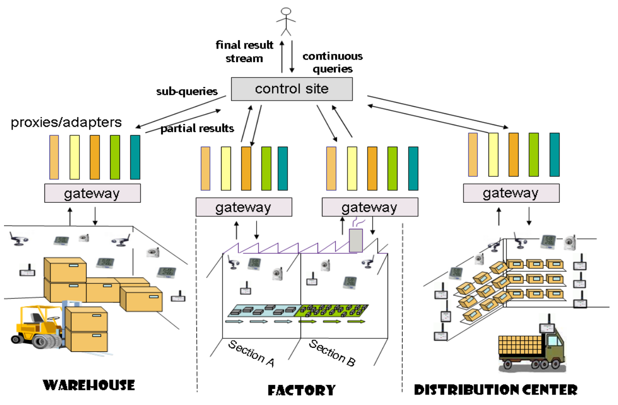


Fig. 3. Deployment example of the hierarchical architecture

IV. Data Representation and Query Evaluation

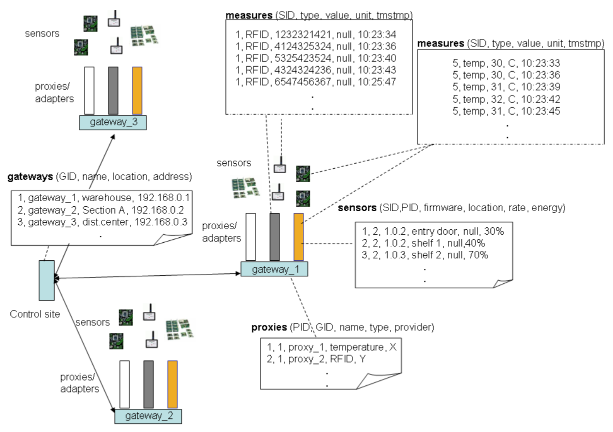
There have been various propositions for sensor data representation and querying. The most popular way is to represent sensor data as a (possibly infinite) sequence of tuples ordered by their timestamp attribute. SQL-like queries are then evaluated continuously on data streams. SStreaMWare is based on the data and query model *SStreaM* [29]. Section IV-A and IV-B present respectively the SStreaM approach for sensor data representation and continuous query evaluation.

A. A generic schema for sensor data representation

Existing proposals define data schemas strongly related to the type of considered sensors. SStreaM proposes a schema template which aims to be generic in order to represent data of various sensor types. The main idea is to distinguish the different types of information included in sensor data. We have identified three of them : measures, timestamps and properties.

Measures are sensed values representing different physical quantities. They are grouped under one attribute. The semantic interpretation of its value is made at the application level. For instance, a measure can represent a temperature value, GPS coordinates or a RFID tag number depending on the sensor. Measures are timely varying values and they are the main interest of continuous queries; e.g., alert me when a sensor measures a temperature less than 0 °C.

Timestamps represent time instants at which measures are made. They determine the order of tuples in a data stream. They are also important for the queries concerning temporal (e.g., sliding) data windows; e.g., computing, **every minute**,



measures (SID, type, value, unit, tmstmp)
1, RFID, 1232321421, null, 10:23:34
1, RFID, 4124325324, null, 10:23:36
1, RFID, 5325423524, null, 10:23:40
1, RFID, 4324324236, null, 10:23:43
1, RFID, 6547456367, null, 10:25:47

measures (SID, type, value, unit, tmstmp)
5, temp, 30, C, 10:23:33
5, temp, 30, C, 10:23:36
5, temp, 31, C, 10:23:39
5, temp, 32, C, 10:23:42
5, temp, 31, C, 10:23:45

gateways (GID, name, location, address)
1, gateway_1, warehouse, 192.168.0.1
2, gateway_2, Section A, 192.168.0.2
3, gateway_3, dist.center, 192.168.0.3

sensors (SID,PID, firmware, location, rate, energy)
1, 2, 1.0.2, entry door, null, 30%
2, 2, 1.0.2, shelf 1, null,40%
3, 2, 1.0.3, shelf 2, null, 70%

proxies (PID, GID, name, type, provider)
1, 1, proxy_1, temperature, X
2, 1, proxy_2, RFID, Y

Fig. 4. Data distribution in the architecture

the average of measures made **since last 10 minutes**.

Properties form a sort of ”meta-information” of sensors. Typical properties of a sensor are its identifier, location, type, measuring unit, sampling rate, etc. They are used to identify sensors whose measures are necessary for a continuous query. Mostly, queries contain one part to be evaluated on sensor properties, e.g., send me every **5 sec.** the **temperature** of the **Room A**).

Let us consider the following sample schema:
$sensor\_measures(gateway.location, proxy.type, sensor.id, sensor.location, sensor.rate, sensor.energy, measure.type, measure.unit, measure.value, measure.tmstmp)$

The first 8 attributes of this schema form the properties of sensors followed by the measure value and its associated timestamp. Typically such a schema may include more sensor properties such as names, descriptions of sensors or their proxies, configuration parameters, firmware versions, etc. Sensor properties are mostly stored in persistent relations that are distributed in the system at different levels such as control sites, gateways and proxies (see Figure 4). Some properties may also be stored at sensors (e.g., energy level, location for mobile sensors). Measures are not materialized; they are rather generated continuously, thus creating a data stream. They are time-stamped (mostly) by sensors. This schema represents thus a view over the persistently stored sensor properties and non-materialized sensor measures and their timestamps.

Queries are formulated according to a global database schema such as the one given above and are continuously evaluated on data streams composed of tuples conforming to the given schema.

B. Continuous query evaluation

For sensor data querying, the most popular approach is to use a SQL-like relational query language [18], [6], [3], [7], [16]. SStreaM as well is based on the relational model and allows declarative queries on sensor data streams. SStreaM distinguishes two parts in continuous queries:

- The **one-time part** defines the scope of the query; i.e., it identifies the sensors whose measures are concerned by the query. This part is evaluated on the **properties** of sensors.
- The **continuous part** is evaluated on the data stream containing the **measures** of the sensors that are identified by the one-time part of the query.

For instance, let us consider the industrial site example given above and a query such as:

Every 5 seconds, retrieve the temperature of each section of the industrial site; and group by the location, the images captured by the camera of each section where the temperature value is greater than 30 °C.

The one time part of this query concerns: at the control site, finding all the gateways of the site; at each gateway, finding the temperature sensor and camera proxies; and finally at each proxy, finding the sensors measuring in Celsius every 5 seconds[1]. In fact, the one-time part of the query concerns a sort of localization phase of data sources. Concerned gateways, proxies and sensors are localized thanks to a catalogue.

Once the one-time part is evaluated, i.e., concerned sensors are localized, data flowing from these sensors is initiated. Data transit from the sensors until the users or the applications, passing by different levels of the architecture where the continuous part of the query is evaluated: at the temperature proxies, selection operators on temperature readings; at the camera proxies, no specific operations, just transmitting images; at the gateways, Cartesian products between images and temperature readings; and finally at the control site, grouping operations. Query execution continues until the user stops the query or until the query lifetime expires, if this latter is defined in the query. As a consequence, the query results are also continuous, thus form a data stream. Next section gives more details about continuous query evaluation.

In SStreaMWare, access to sensor data and evaluation of queries are provided in terms of services. The service oriented approach facilitates the integration of services managing different kind of sensors in one organization or between several organizations. In addition, its loosely coupled approach leads to flexible, modular and dynamic systems, which are important properties for sensor systems. Details on the architectural services of SStreaMWare are given in the next section.

V. Implementation details

This section details the implementation of SStreaMWare architectural services and other principal structures for query evaluation.

A. Service Implementations

SStreaMWare adopts a service oriented approach, which is implemented above the OSGi Service Platform [30]. The OSGi standard provides modular software deployment and a set of facilities for their management such as lifecycle

[1]Note that, normally this part of the query necessitates to configure sensors to sample every 5 seconds and configure the proxy to deliver measures in Celsius. We consider this operation should be done before the execution of the continuous query. If this operation is not permitted by the proxy, only sensors and proxies conforming to the requirements of the query will be selected for the query evaluation

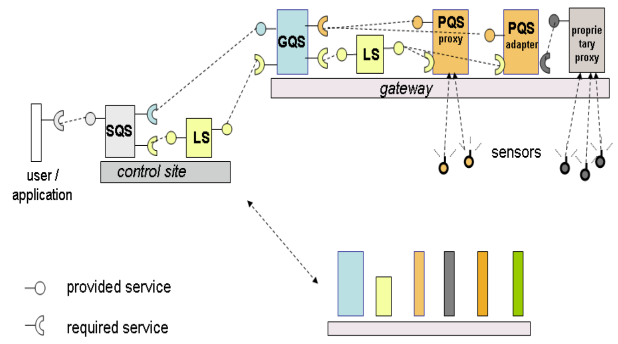


Fig. 5. Services in the architecture

management, automatic resource dependency resolution and event management. It additionally defines a service-oriented collaboration model between the modules, which are named bundles. The OSGi service layer includes the basic features for service oriented programming: a service registry with service event management.

SStreaMWare uses an extended model, namely Extended Service Binder [31], that augments the OSGi service collaboration model to a set of remote OSGi platforms. The overall architecture benefits from the OSGi dynamic service collaboration model and from the communication transparency offered by the extension. The latter makes remote calls undistinguishable from direct local calls in the developer perspective. Local calls remain direct calls in the given programming language.

SStreaMWare architectural entities consists of three querying services and one service discovery and binding service which are implemented as OSGi services deployed on distinct distributed platforms:

- Sensor Query Service (SQS),
- Gateway Query Service (GQS),
- Proxy Query Service (PQS),
- and Lookup Service (LS).

Figure 5 illustrates the service oriented architecture and the interaction between different services. Following sections give details of the implementation of these services.

1) Sensor Query Service: is located at a control site, and is used by users and/or applications. Its responsibility towards query evaluation concerns:

- accepting queries addressed to all the sensors of the environment managed by the control site ;
- decomposing queries and sending them to concerned gateways (which are localized thanks to the Lookup Service) ;
- possibly contributing to the query evaluation ;
- returning results as data streams.

A control site can be an OSGi gateway or another entity which can host the SQS (e.g., in a form of a web service in a web server). In our case, it is implemented as a remote service accessible by an IP address and a port number and hosted by an OSGi gateway. Used remote communication protocol is Java RMI (Remote Method Invocation). SQS is registered in a RMI registry which allows the clients to discover and invoke it.

SQS exposes several distant methods:

- the method `QueryRegistration receiveQuery (String query, Receiver client)`,
  - which takes the reference of the client and a query as input
  - and returns a `QueryRegistration` created for this query. This registration contains information on the query (e.g., its identifier) and on the client of the query (e.g., its reference).
- the method `stopQuery(int QId)`, which stops the execution of the query with the given identifier,
- and methods to get various information on gateways, proxies and sensors present in the system; e.g., `getGatewaysInfo`, `getSensorsInfo`. This information is maintained thanks to the Lookup Service.

2) Gateway Query Service: is charged to evaluate sub-queries concerning a given sub-region. At each gateway, there is one GQS providing a query service for the sensors managed by the proxy services of the gateway. The SQS sends the sub-queries to the relevant GQSs. The GQS re-decomposes the sub-queries in order to send them to concerned proxies/adapters which are localized thanks to the LS instance running on the gateway. After the sub-query evaluation, the results are then sent back by the GQS to the SQS.

Gateways of the architecture are implemented by OSGi platforms. A Gateway Query Service is an OSGi service that exposes locally and remotely:

- the method `Sender evaluateQuery(QueryPlan qp, Receiver receiver).`
  - which takes as inputs, a sub-query execution plan (conceived by the SQS) and the reference of the `Receiver` which will receive the query results,
  - and returns as output, the reference of the `Sender` which will send the query results.
- and similarly to SQS, various other methods such as to stop a query and to acquire information on proxies and their sensors.

3) Proxy Query Service: is a query service which is related to one specific type of sensor. This service is provided by proxies if they are ”SStreaMWare-aware”, or by adapters otherwise. The implementation of this service is specific to sensors. Under a generic interface, PQS thus hides the heterogeneity of sensor software. PQSs are also catalogued by the LS, which enables GQSs to localize and use them. Typically, GQSs discover PQSs with lookup queries containing predicates on the *type* of the proxies; e.g., proxy of temperature sensors, proxy of a certain provider. PQS also implements the method `Sender evaluateQuery(QueryPlan qp, Receiver receiver)` and provides methods which return information on the sensors it manages.

4) Automatic Service Lookup and Binding Service: The Lookup Service (LS) is a service discovery and binding service. This service manages a service registry that contains information about different services present in the system. The registry may be centralized on the control site or may

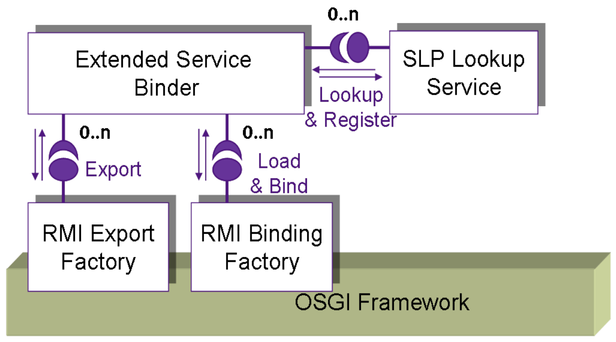


Fig. 6. Architecture of the Extended Service Binder

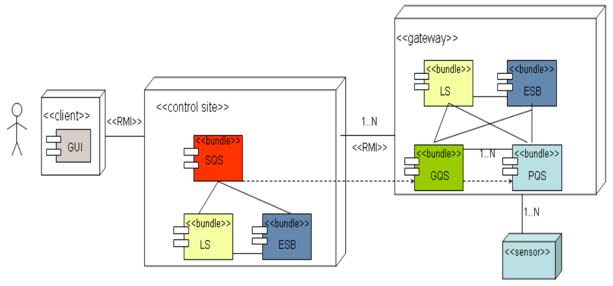


Fig. 7. Deployment of the services

be distributed in the system. One instance of LS runs on the control site and on each gateway. LS allows local and remote discovery of services in a transparent way. For this, each query service is registered with its description that allows other services to discover them by using lookup queries.

SStreaMWare aims to support dynamically evolving sensor systems. Therefore, late binding of software entities at runtime and binding automation were prior objectives of the project. That is partly why the OSGi platform was chosen as a basis for our development. The OSGi standard not only enables a modular development and deployment of SStreaMWare architectural entities, it also provides a basic environment for service oriented programming. Late binding of local services is permitted by the platform.

To meet the need for binding automation, the *Service Binder* [32] solution has been chosen. This project, which has given birth to the OSGi Declarative Services specification [33], provides the designers with a new component model above the OSGi deployment model. This functional component type separates the application logic from the service dependency management. The developers have only to write application service requirements and provide it in an XML document. The service runtime component is responsible for dynamic service discovery and binding of the available required services at runtime.

Since our sensor services are distributed in the environment, a distribution support was needed for service discovery and binding. For this purpose, we have used the *Extended Service Binder* [31]. Relying on the export-binding pattern defined in ODP [34], this solution renders the Service Binder mechanisms distributed. When the XML component configuration file indicates that a provided service is wished to be exported outside the OSGi local registry, the Extended Service Binder registers a reference on a remote service registry; e.g., an SLP Directory Agent [35] on local networks or an ad hoc directory at the Internet scale. When a component service requirement is not resolved locally and is indicated that it can be resolved remotely, the binder looks for a remote reference at the known service directories. The remote service reference is parsed and the service proxy retrieved on the platform thanks to the Java RMI technology. The internal architecture of the Extended Service Binder is depicted in Figure 6.

This extension is transparent for the developer that has only to indicate whether the service accepts remote bindings. Service references are searched against a service interface name and/or some additional service properties. Local bindings are kept as direct programming language calls while remote bindings transparently serialize and de-serialize calls between objects running on distinct OSGi platforms. Similar proposals, which address service distribution transparency on a set of OSGi platforms, have been made (e.g., [36]).

SStreaMWare leverages both active and passive service discovery modes:

- *active discovery*: for instance, when the SQS starts, it queries the Lookup Service for the references of available GQSs distributed in the environment. Similarly, the GQSs use the Lookup Service to find locally available PQSs.
- *passive discovery*: for instance, when a GQS starts running on a platform, it automatically registers a representative service to the Lookup Service. The SQS is then notified by this service registration and is automatically linked with the GQS. Similarly, new PQSs are automatically binded with existing GQSs.

5) Service Deployment: Services presented above are exported as OSGi ”bundles” encapsulating the service functionality. The main developed bundles are:

- **SensorQueryService** deployed on the control site
- **GatewayQueryService** deployed on every gateway
- **XXXProxyQueryService** providing sensor specific querying service (e.g., TemperatureProxyQueryService, RFIDProxyQueryService, GPSProxyQueryService) deployed at concerned gateways.
- **Common** is library bundle. It provides common useful classes (e.g., operators, queries, queues, tuples, interfaces, abstract classes) for query execution and is deployed on the control site and the gateways.

In addition, other useful bundles are: **LookupService** (LS), which exposes a service discovery service; **ExtendedService-Binder** (ESB), which provides the automatic service binding service. The figure 7 illustrates the deployment of bundles and their interactions.

B. Query Evaluation

Queries are received by SQS and they are then evaluated at different levels of the architecture by the concerned services. Each service has a local `QueryManager` (see section V-B.1) that is responsible of query evaluation at the site where the service is started. Queries are composed of various operators that are presented in section V-B.2. Queries are evaluated on data streams and windows. Their structure is presented in section V-B.3.

1) Query Managers: Each service (i.e. SQS, GQS, PQS) incorporates a Query Manager (QM) which is responsible for:

- analysing query plans, decomposing them if necessary and sending the sub-queries to relevant services at other sites;
- creating necessary structures (operators, queues, queries) for the execution of the continuous part of the query which will be executed at its site.
- managing the lifecycle of queries:
  - assigning identifiers to queries
  - starting query execution
  - maintaining information on active queries (e.g. identifier, periodicity, sender, receiver)
  - stopping query execution; either when its lifetime expires or when it is explicitly demanded by the user

Each query service, after receiving a query, sends it to its local Query Manager which then creates the necessary operators and data queues interconnecting them. Query managers at the proxy level starts the data flow from the sensors and queries are then executed continuously in an autonomous way. The following section introduces the structure of queries.

2) Queries: Queries are composed of several sub-queries – which are also queries – executing at different levels of the architecture (control site, gateway, proxy). Basically, a query is a tree of operators interconnected by queues. Operators are created from an execution plan by the Query Manager of each query service (SQS, GQS, PQS). A query execution plan is a tree of operator descriptions. Descriptions contain necessary information to create operators; e.g., type, predicates for selection operator, window descriptions for windowed operators. An `OperatorFactory` is conceived to create and instantiate operators from operator descriptions.

Each operator has a certain number of input queues and one output queue which represent, respectively, input streams and the output stream. Among other attributes, each operator has a *type*; e.g., selection, projection, windowed join, and a *timer* for periodic queries. Each operator also has specific attributes, such as a predicate expression for selection operators or an attribute list for projection operations. Each operator implements an abstract method `receive(Tuple T)`, which defines the behaviour of the operator at tuple reception (e.g. filter, project). Binary operators have another method in addition, `receiveFromRight(Tuple T)`, in order to deal with two input streams.

Each query has one root operator whose output queue forms the result stream of the query. The output of one query may form the input for another query. Query outputs and inputs between different sites are connected (possibly in a remote way) by means of two special operators: `Receiver` and `Sender`. The structure of a query is illustrated in the figure 8.

Another special kind of operator is *windowed operator* which receives windows as inputs and produce streams. There are unary and binary windowed operators. For instance, a windowed average operator, at an instant t, takes a data window as input and produces a tuple including the average of the values of a given attribute. Continuous execution of this operator on timely-varying data windows (e.g. sliding windows) produces therefore a data stream. Unary windowed operators implements a method `receiveWindow(Window W)` which defines the behaviour of the operator. Similarly, binary windowed operators have the additional method, `receiveWindowFromRight(Window W)`.

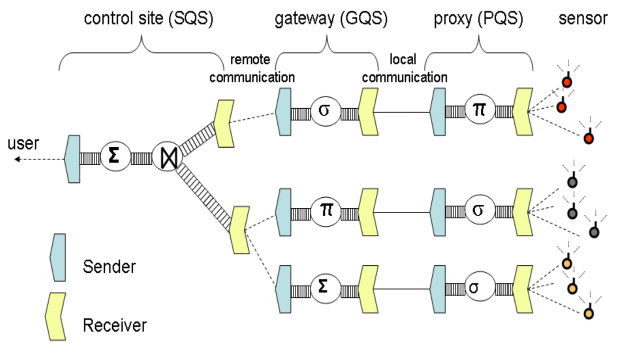


Fig. 8. Structure of queries

3) Queues: A Queue represents a data stream which is basically a sequence of tuples. A tuple is a list of (attribute, value) pairs. It is represented by a `Tuple` class that provides various methods, for instance, to retrieve all attribute values, to retrieve value of a given attribute, to concatenate two tuples or to add/remove attributes. Queues provide two main functions: `enqueue(Tuple T)` and `dequeue()`. Operators use these methods asynchronously in a FIFO manner. Each queue is associated to an operator from which it receives tuples (`feedingOperator`) and an operator to which it provides tuples (`feedingOperator`). Depending on the nature of queries (e.g., periodic or aperiodic), either the feeding operator pulls the tuples from its input queue or the queue pushes the tuple to its feeding operator.

Queues provide more utility methods to access certain tuples of the queue (e.g., tuple at a given position, tuples older than a given time instant). These methods are used in particular by *window creators*. A window is subset of a queue. A window creator is associated to a windowed operator and to an input stream. It creates windows from the input stream and sends them to the windowed operator according to *window descriptions*. Basically, a window description concerns maintaining two pointers to the input stream. Data between these two pointers are used to produce the windows. Window descriptions contain information about the position of these pointers and their behaviour in time, e.g., periodically sliding. There are two types of windows: temporal and position based windows. In the former, data included in the window is determined by the timestamps of the tuples, e.g., window of measures made since 4 minutes. In the latter they are determined by the index of the tuples in the stream, e.g., window of last 10 measures. More details on the query and data model can be found in [29].

VI. Experimentation

SStreaMWare is used in the context of a French government funded project with industrial and academic contributors [27]. The main objective of the project was to conceive a software infrastructure that can dynamically host and supervise technical and business services in electric distribution domain. Sensors on electric power materials are used to measure quantities related to the quality of transmitted electricity. Various kinds of materials are used. Specific proxy services are conceived for each kind of material. These services export methods for acquiring sensor measures (e.g., intensity, tension) and sensor properties (e.g., name, provider, location). They also provide a service of subscription to the data of the sensor that they control. For instance business services use these services to integrate the business logic in the system, e.g., break the circuit in case of overload.

One of the objectives of the project was to provide a more generic declarative way for querying data of sets of heterogeneous materials. The idea is to abstract data acquisition and processing part from the business services and let them focus only on the business logic. SStreaMWare provides a homogeneous view over heterogeneous sensor data and offers a query evaluation service on this data. Business services then use SStreaMWare's query service to implement their logic, e.g., monitoring service, alarm service.

For this purpose, we have developed adapters, i.e., Proxy Query Services, for the materials of the project. Adapters are charged to transform the SStreaMWare's queries into methods comprehensible by the specific proxies and translate data from the sensors to SStreaMWare's tuple structure. Adapter services have two ways for data acquisition: *pull*, by calling the adequate methods on the specific proxy; *push*, by subscribing itself to the publish-subscribe service provided by the proxy.

Taking advantage of the service oriented-approach, SStreaMWare could easily be integrated into the already existing infrastructure of the project.

Utility of SStreaMWare in other contexts is also demonstrated in [28]. More sensors such as cameras, temperature sensors and GPS devices are added to the SStreaMWare's set of supported sensor types. Adapters for each of the sensor type are developed. Thanks to the service oriented approach, these services are plugged into the SStreaMWare on the fly, and are ready to be used immediately. One feature currently being developed for SStreaMWare is to deal with the dynamicity caused by arrival, departure, modification of sensors and services. Extended Service Binder provides capture of this kind of update events – local and remote ones. These updates are then taken into account immediately by currently executing continuous queries in the system in an appropriate way. For instance, "do not modify a sensor property if it is in use by a query" or "if a new temperature service arrives, include its data in the results".

A graphical user interface has been developed for query formulation. The interface first localizes a SStreaMWare control site and then may visualize all the desired entities (gateways, proxies, sensors) managed by the control site, as well as information on them (location, type, name, etc.). The user then can choose the sensors to query and construct a query plan. Several operator types are supported: selections; projections; aggregation operators on data windows such as average, minimum, maximum, sum, count; join operations on data streams and windows. Textual SQL-like query formulation for simple queries is also supported. Once query plan is constructed, query can start to be evaluated continuously on data streaming from the chosen sensors.

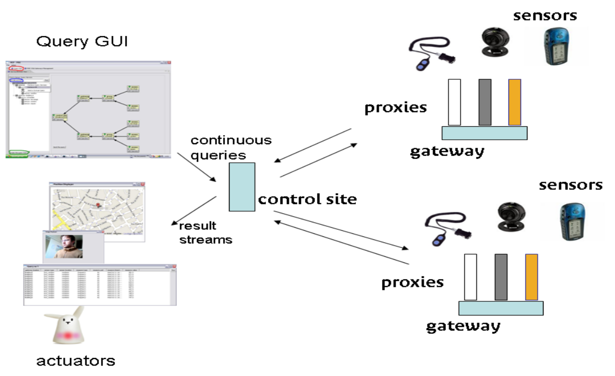


Fig. 9. Demonstration example

Query results are visualized in a table-like window in which produced result tuples are appended. The result window also provides some control such as stopping the query or handling the result in a specific way, e.g., for GPS data, use the results to localize in a map; for temperature measures, visualize the evolution of the measures in a graph or send them to an actuator; for camera data, visualize the images on an adapted component. Concurrent queries are equally supported, each one being visualized in a separate window. See Figure 9 for an illustration of the demonstration.

VII. Conclusion

This paper presented SStreaMWare, a service oriented middleware for heterogeneous sensor data management. SStreaMWare follows a hybrid approach which acts as a distributed sensor data stream management system. Some similar proposals, as well as their positioning with respect to SStreaMWare, are presented in the paper.

We have particularly outlined the importance of sensor distribution and heterogeneity management. In fact, the increasing number of sensor-based applications emphasizes the need for integrating data from distributed sensors of various types and of different manufacturers. Service oriented architectures, thanks to their loosely-coupled approach, facilitate this integration. Besides, they are well adapted to the dynamic nature of sensor systems. As an implementation of this approach, adapter services for different kinds of sensors – electrical sensors, cameras, temperature sensors, GPS devices and presence detectors – are already developed and integrated in SStreaMWare, which provides a homogeneous database view over these heterogeneous data sources.

One of the perspectives of this work is to integrate a complete device management support to SStreaMWare while providing system reliability and data coherency. Device management operations consist of read and write operations on sensor properties, configuration operations and software management. These operations should coexist with continuous queries. Having a database point of view, we consider these different operations as ”transactions”, which should assure some properties such as ACID properties. Some preliminary work on this subject can be found in [25].

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey.” *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] P. Bonnet, J. Gehrke, and P. Seshadri, “Towards sensor database systems,” in *MDM '01: Proceedings of the 2nd Int. Conference on Mobile Data Management*. London, UK: Springer-Verlag, 2001, pp. 3–14.

[3] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TinyDB: an acquisitional query processing system for sensor networks,” *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[4] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, “Aurora: a new model and architecture for data stream management.” *VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[5] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom, “STREAM: The stanford stream data manager.” *IEEE Data Eng. Bull.*, vol. 26, no. 1, pp. 19–26, 2003.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, “TelegraphCQ: Continuous dataflow processing for an uncertain world.” in *CIDR*, 2003.

[7] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong, “Design Considerations for High Fan-In Systems: The HiFi Approach,” in *CIDR*, 2005, p. 290.

[8] L. Gürgen, “Scalable management of heterogeneous sensor data (in french),” *PhD Thesis. Grenoble Institute of Technology (INPG)*, 2007.

[9] G. Wiederhold, “Mediators in the architecture of future information systems.” *IEEE Computer*, vol. 25, no. 3, pp. 38–49, 1992.

[10] L. Gürgen, C. Labbé, V. Olive, and C. Roncancio, “A scalable architecture for heterogeneous sensor management.” in *DEXA Workshops*, 2005, pp. 1108–1112.

[11] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, “Directed diffusion for wireless sensor networking,” *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 2–16, 2003.

[12] S. Madden and M. J. Franklin, “Fjording the stream: An architecture for queries over streaming sensor data,” in *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2002, p. 555.

[13] D. Abadi, W. Lindner, S. Madden, and J. Schuler, “An integration framework for sensor networks and data stream management systems.” in *VLDB*, 2004, pp. 1361–1364.

[14] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, “IrisNet: An architecture for a worldwide sensor web,” *IEEE Pervasive Computing*, vol. 02, no. 4, pp. 22–33, 2003.

[15] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, “Hourglass: An infrastructure for connecting sensor networks and applications,” Harvard University, Tech. Rep. TR-21-04, 2004.

[16] K. Aberer, M. Hauswirth, and A. Salehi, “The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks,” Ecole Polytechnique Fdrale de Lausanne (EPFL), Tech. Rep. LSIR-REPORT-2006-006, 2006.

[17] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TAG: A tiny aggregation service for ad-hoc sensor networks.” in *OSDI*, 2002.

[18] A. Arasu, S. Babu, and J. Widom, “The CQL continuous query language: Semantic foundations and query execution,” Stanford University, Tech. Rep. 2003-67, 2003.

[19] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” in *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. NY, USA: ACM Press, 2002, pp. 1–16.

[20] L. Gürgen, “State of the art of sensor data management (in french),” ACSE Poject, WP2-Architecture. Orange Labs, France Telecom Group, Tech. Rep., 2008.

[21] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, “The Design of the Borealis Stream Processing Engine,” in *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[22] N. Tatbul and S. Zdonik, “Dealing with overload in distributed stream processing systems,” in *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*. Washington, DC, USA: IEEE Computer Society, 2006, p. 24.

[23] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan, “Cache-and-query for wide area sensor databases,” in *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 2003, pp. 503–514.

[24] S. Rizvi, S. R. Jeffery, S. Krishnamurthy, M. J. Franklin, N. Burkhart, A. Edakkunni, and L. Liang, “Events on the edge,” in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. NY, USA: ACM Press, 2005, pp. 885–887.

[25] L. Gürgen, C. Roncancio, C. Labbé, and V. Olive, “Transactional issues in sensor data management,” in *3rd Int. Workshop On Data Management for Sensor Networks in conjunction with VLDB*, 2006, pp. 27–32.

[26] K. Aberer, M. Hauswirth, and A. Salehi, “A middleware for fast and flexible sensor network deployment,” in *VLDB'2006: Proceedings of the 32nd international conference on very large data bases*. VLDB Endowment, 2006, pp. 1199–1202.

[27] F. Baude, A. Bottaro, J.-M. Brun, A. Chazalet, A. Constancin, D. Donsez, L. Gürgen, P. Lalanda, V. Legrand, V. Lestideau, S. Marié, C. Marin, A. Moreau, and V. Olive, “Extension de passerelles OSGi pour les domaines de la distribution électrique: Modèles et outils,” in *OSGI Workshop in conjonction with UBIMOB 06, 3rd French-speaking conference on Mobility and Ubiquity computing*, 2006.

[28] L. Gürgen, C. Roncancio, C. Labbé, V. Olive, and D. Donsez, “SStreaMWare : un intergiciel de gestion de flux de donnes de capteurs hétérogènes (demonstration paper).” in *23th French-speaking conference on Advanced Databases (BDA'07)*, October 2007.

[29] L. Gürgen, C. Labbé, C. Roncancio, and V. Olive, “SStreaM: A model for representing sensor data and sensor queries,” in *Int. Conf. on Intelligent Systems And Computing: Theory And Applications (ISYC)*, July 2006.

[30] OSGi Alliance, ”OSGi Service Platform Core Specification Release 4”, October 2005. http://www.osgi.org/.

[31] A. Bottaro, A. Gérodolle, and P. Lalanda, “Pervasive spontaneous composition,” in *First IEEE International Workshop on Service Integration in Pervasive Environments*, Lyon, France, June 2006.

[32] H. Cervantes and R. S. Hall, “Autonomous adaptation to dynamic availability using a service-oriented component model,” in *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 614–623.

[33] OSGi Alliance, ”OSGi Service Platform Service Compendium Release 4”, October 2005. http://www.osgi.org/.

[34] ITU-T & ISO/IEC, ”ODP Reference Model: Overview, Foundations, Architecture”, Recommendations X.901, X902, X903 & International Standards 10746-1, 10746-2, 10746-3, 1995.

[35] E. Guttman, “Service location protocol: Automatic discovery of ip network services,” *IEEE Internet Computing*, vol. 3, no. 4, pp. 71–80, 1999.

[36] J. S. Rellermeyer, G. Alonso, and T. Roscoe, “R-osgi: Distributed applications through software modularization.” in *Middleware*, ser. Lecture Notes in Computer Science, R. Cerqueira and R. H. Campbell, Eds., vol. 4834. Springer, 2007, pp. 1–20.