Politecnico di Milano

Technologies for Information Systems
Prof. Letizia Tanca
Prof. Fabio A. Schreiber

2008/2009

Project Report:

A proposal for a context-aware extension of PerLa language

Student:
Marco Marino
721677

Abstract

A context model is proposed for PerLa language, aiming at implementing context-awareness capabilities in a full declarative way. An extended survey was made before proposing possible solutions, in order to find out the context model adopted by similar projects in the research environment. The dissertation continues with an analysis of what is currently used by the language to perform data tailoring on sensors, namely the PILOT JOIN function, where limitations and shortcomings of the approach are pointed out and critically pondered. A proposal of a context model is then enunciated, based on a balance between the needs of PerLa and the current state of the art in contextual modelling. Context creation and management paradigms are shown in this part, along with a conceptual context model derived from the Context-ADDICT project also developed at Politecnico di Milano.
A possible implementation of such a model is then discussed, and a suitable grammar for context generating functions and context management parameters are introduced.
Two plausible and hopefully exhaustive scenarios are shown, trying to remark all the functional properties of the proposed model.
The dissertation ends with additional features in the perspective of Plug&Play-like features and the eventuality of adding constraints to the data tailoring process and with some final words on the future improvements of the model, including automatic context generation.
The EBNF grammar of the modifications at the PerLa language is presented in Appendix A.

Table of contents

1. Introduction

Adding context-awareness to a Wireless Sensor Network is a task requiring a closer look on the definition of context itself and mainly on how it can be applied to improve the performances of the whole network.
The formal definition of context can be thought as "the set of facts or circumstances surrounding a situation or event". When dealing with such definition, some peculiar concepts are of key importance and should always be kept in top priority: i.e. are facts and circumstances static in time?
The answer to the above question is generally negative for many reasons, especially when considering the intrinsic characteristics of a WSN.
Context is in fact a mutable creature; not only with respect to time, but also for the scope we are interested in, the external factors, current trends and involved phenomena and so on.
When autonomous devices, like the ones found in Sensor Networks, are spatially distributed in order to monitor some kind of phenomenon, the capability of modelling the context can help the user to interact naturally with the devices and nevertheless to minimize the communication overhead, thus leading to a substantial saving of energy.
Things get worse – from a theoretical point of view – when a simple interface has to be created to interact with the network, like with a full declarative approach used by the PerLa language.
The deal in this case is to create and materialize a functional context management platform to be used, inside the language, by the average user, thus hiding the complexity of the underlying layers.
If the network, or more correctly the language handling it, is able to manage the context the sensors are related with, the user would be able to perform some queries which would otherwise be very difficult to implement.
The relation between context and sensor is a tight one in this dissertation, because the context here is seen from the low level point of view, in which a single physical device is tied to one or more contexts that could finally be involved in a language query.

2. State of the Art

Before analyzing the scope and the goals of designing a context model for the PerLa language, a survey was performed among similar projects already running in the scientific environment.
A brief description on how context was modelled and managed with respect to the WSN will be given, along with the main similarities and differences with PerLa.
Several projects were analyzed, each one having its own peculiar context implementation, and focusing on different aspects from energy saving to automatic context discovery.
The reference for WSN languages is without doubt TinyDB, described in [11].
This very first attempt to use a declarative language and at the same time modelling the entire network like a database system, led to many other projects in this area, some of them are related to adding context management, which is not originally present in TinyDB.
This is the case of [8], where context is modelled with respect to spatial coordinates of the sensors: it is thus possible to extract a set of coordinates and embed them into a data structure representing the context. Queries are executed later on the scope created by the context, leading to energy saving and workload optimization. Another example of a spatial use of context is found in [5], where the benefits of an accurate power management are evident.
The concept of scope is also relative to the one of context itself, this is the point expressed in [2]. The Scope as presented by Steffan et al. is intended as a data structure consisting of a unique ID, a parameter called scope-type and some membership conditions, together with an optional scope life-time parameter.
This particular implementation of context allows restricting the visibility of some areas of the network, i.e. for security reasons, and also to adopt some complex policies on the management of the sensors themselves.
Concurrency of contexts is also an important aspect, underlined also in works like [7] and [4]. Concurrent contexts introduce the notion of priority, which is not stressed in any of the analyzed projects and will be briefly overviewed in this work.
The need to physically represent the context is dealt in almost every project, an interesting example is in [4], where a context descriptor is created specifying some parameters, an example is shown in figure 1.

```
Context Descriptor {
        cid: 1
        ucid: 2
        Event {
                eid: 1
                type: temperature
                condition: ≤ 10 °C
                area: [(-50,-30), (-10,10)]
                timing: 2, (-5 sec, +5 sec)
                next: 3, [(10,-10), (30,-40)]
         }
        Action {
                id: 4
                location: [(-50,20), (-40,10)]
         }
}
```

Figure 1 (from [4])

The PROCON architecture [4] also adds a specific contextual layer to the entire network, thus allowing context information to roam across the network, which is able to handle it inside a single sensor: this step requires a certain computational ability of each sensor, but it should be pointed out that in PerLa language it is possible to model even very simple sensing devices as logical objects (e.g. RFID tags) [1].

Finally an automatic approach to context detection and/or recognition is shown in [12], where the traditional approach used in the Artificial Intelligence branch of Computer Science is applied in finding regular patterns among the readings coming from the sensors.
It is then possible to create a context structure not previously defined by an operator, even if this remains one of the most difficult tasks to implement, since the risk of creating useless contexts is still very high.
It should be pointed out that, while in [4][5][8] and [9] the project clearly defines a full declarative language approach, in the other works the language is unclear or totally absent, using instead custom representations of the tasks to be performed in the network.
The project most similar to PerLa appears to be GSN, described in [13], which uses a similar approach, but lacks of context-awareness capabilities.
A useful way to compare different works in this area is to plot them in a chart with respect to their "Intelligence" level, and to the context-awareness integration level.
The first one is about how much the software layer is able to discover and automatically handle new contexts while the former one measures how the context is integrated with the entire network, the level of its description and the overall impact it has on the user of the network.
The results are shown in figure 2.

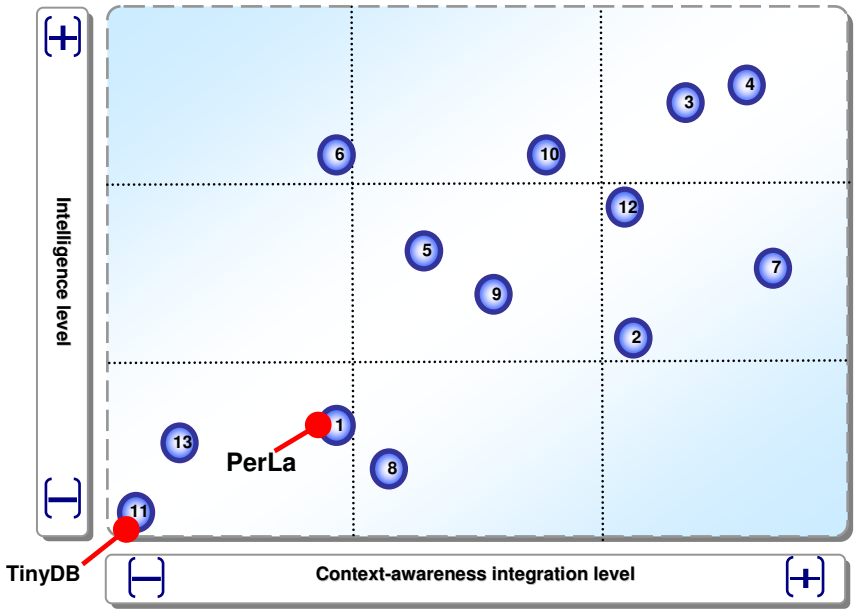

Figure 2

PerLa language at the moment only adopts a basic layer of context awareness with the PILOT JOIN operator, which will be broadly described and analyzed in the following paragraph.

3. PILOT JOIN Analysis

The PILOT JOIN function grammar definition was first introduced in [14] to perform data tailoring, useful in context-aware environments.
Its definition in [1] can be reported as "A special operation that allows dynamic changes in the set of logical objects executing a specific Low Level Query, based on the results produced by another running query." "...it forces each involved logical object to start (or stop) the query execution if the joined stream contains (or not) a record that matches the current value of logical object attributes..."
The main purpose behind the PILOT JOIN function is adding the ability to perform a first level of context aware queries, in order to save energy and computation time, which are usually scarce resources in a WSN. The language grammar definition of the function is presented in the following, where a Data Structure is mentioned along with a Boolean Condition, both of which are further explained in [14]:

<Pilot Join Clause> ➞ PILOT JOIN <Correlated Table List>

<Correlated Table List> ➞ <Correlated Table> { ',' <Correlated Table> }*

<Correlated Table> ➞ <Data Structure Name> ON <Condition>

E.g.:
PILOT JOIN BaseStationList **ON** currentBaseStation **=** baseStationList**.**baseStationID

A real life example can be helpful to understand how the function is intended to work:

Example 1

Taking some ideas from the example presented in [2], suppose we have to monitor some containers loaded in a container ship. Each container has two sensors, a light sensor on the external surface, and a temperature sensor in the inside. The two sensors are not physically connected to each other, but we want to exploit their ideal connection to the same element in order to perform a context-aware query using the PILOT JOIN operation.
Our goal is to monitor the temperature of the containers which are directly exposed to sunlight.
First of all we need to define a structure named SNAPSHOT [1][14]:

CREATE SNAPSHOT UnderTheSun (sensorID **ID**, light **FLOAT**, containerID **ID**)
WITH DURATION 1 **h AS**
LOW:
SELECT ID, light, containerID
SAMPLING EVERY 1 **h**
WHERE light **>** [threshold]
EXECUTE IF deviceType = "LightSensor"

A snapshot is a general "picture" of the logical objects satisfying a certain condition at a given time. In this particular case we are storing inside the SNAPSHOT structure all the light sensors for which the detected light level is greater than a threshold. The structure is updated every hour, and we are storing in it the unique ID of the sensor, its detected light level and the ID of the container the sensor is installed in.
The former parameter is of crucial importance and will be deeply examined later on.

Once the SNAPSHOT structure is created we have a data structure useful for data tailoring, because at a higher information level we now know which containers are exposed to direct sunlight, supposing that the threshold was accurately chosen.
The next step is to select the temperature sensors belonging to an exposed container, and in order to perform this task we use the PILOT JOIN operator as follows:

CREATE OUTPUT STREAM
Temperatures(sensorID **ID**, temp **FLOAT**, containerID **ID**) **AS**
LOW:
EVERY ONE
SELECT ID, temp, containerID
SAMPLING EVERY 1 **m**
PILOT JOIN UnderTheSun **ON** UnderTheSun.containerID = containerID
EXECUTE IF EXISTS (ALL)

In this query we are choosing the temperature sensors, in particular we are interested in their ID, the temperature value and the container ID in which the sensor is installed.
The data tailoring part of the query is expressed inside the PILOT JOIN clause, where we basically restrict the scope of the query to the sensors related to the SNAPSHOT named "UnderTheSun", by means of the condition imposed by the container ID value, which must be the same in the two structures. The PILOT JOIN function intended as a context function for PerLa has some limitations if compared to the other projects mentioned in the previous paragraph, some of them were clearly identified:

- **The binding between the two classes of sensors must be explicit.**
  As seen in other real scenarios examples [14], a direct correlation mechanism has to be created or already present between the two data structures we want to correlate with the context tailoring: for instance in the previous example the bridge entity between the two classes is the container unique ID. It will be proved in this dissertation that correlated relations like the ones usually shown in PILOT JOIN examples are only a subset of the cases manageable by the proposed context model.
  In the other examples in [14] and also in [1][15], there is always a value acting as a bridge for the join operation. Things get worse if this value is not present, because the PILOT JOIN would rather be impossible to be used or we may have the need to introduce some custom functions, thus violating the constraint of working with a full declarative language.

- **The PILOT JOIN function is not powerful enough for some kind of context-aware queries**
  Considering a slightly more complex case, we can change the specifications of our example with containers as following:

  Example 2
  Monitor the tampering sensor of the accessible containers when the ship is docked (GPS): if tampering is detected monitor nearby containers at higher frequency.
  We suppose that a tampering sensor is embedded inside the doors of the container.
  In this situation four contexts may be involved, but the last one is created only after the other ones. This query may involve a cascade of PILOT JOIN operators, with some problem with the definition of the newly created context.

- **The function doesn't allow exchange of information with external entities**
  A typical use of a WSN is to augment the sensing capabilities of a complex and extended phenomenon, which can also be influenced by elements that are alien to the network.

For instance weather conditions and forecasts may be a key-factor in managing the WSN, along with social or political emergencies that may allow the network to change some parameters independently of the values read from the sensors.

- **Time management is not possible**
  The SNAPSHOT and STREAM structures cannot help the PILOT JOIN to manage some temporal issues that may arise during query execution.
  What if in the case of example 1 we want to start monitoring the sensors from next Friday night instead of starting right now?

By combining the notions introduced in paragraph 2 with the analysis performed in this paragraph, we are able to give some answers to the modelling of context in PerLa, which will lead to a modification, or properly an improvement to the internal representation of context used by the language itself, and rethink the PILOT JOIN to be more powerful when dealing with such a model.

4. Context Model

Some key ideas have to be introduced with the context model in PerLa, these are basic concepts in the context modelling paradigm.

- **Context Reification**
  Explicit location for context information enables a reliable cross-context usage (e.g. Context node [2][3]), the context should always be materialized in a data structure.

- **Context Temporal Management**
  A context may be valid only in a given time interval, keeping it explicit allows an easier management of time. It also may be useful to let the context be valid only from a given time or date and for a maximum period.

- **Multiple Contexts per sensor**
  A sensor may join different contexts at the same time (e.g. low battery sensor and exposed to sunlight) [4]. This overlapping in context management with respect to sensor may allow contexts concurrency.

- **Context Priority**
  Given the validity condition of the context, it may be possible to model logical priority rules, so to raise the importance of some context with respect to another one.

- **Context Discovery**
  By analyzing sensor data it is also possible to discover new contexts, not previously defined by the user. This topic is still under research and will be considered at the end of this dissertation since it makes use of techniques derived from Artificial Intelligence.

The conceptual level of the context model is derived from [16][17][18][19].
The Context-ADDICT project presented many similarities in the contextual representation, which can be applied in a WSN managed by the PerLa language.
A brief analysis of the context conceptual model will be given, followed by its instantiation in a typical frame of a WSN.
The Context-ADDICT model allows to take into account the modelling of several aspects, like space and time, which can also be considered in a fashion that may be relative or absolute in the referring system, but also the context history, the subject described by the context and the user

profile [16]. Details of the architecture are deeply explained in [17][18], and only a short recall of the main data structure used for context modelling will be presented here: the Context Dimension Tree (CDT).
The overall idea behind the CDT is that perspectives data are viewed from may be many, in a sense of Ambient Dimensions arising from the point that a context is thought as a composition of several perspectives.
Each perspective, called dimension, may then be specified to have different values, each one being mutually exclusive to the other with respect to a given dimension.
For instance a dimension can express the role an entity (i.e. a logical object defining a sensor) plays in the network, an interest topic, or a geographical discrimination value.
While the Context-ADDICT platform was designed to work on semi-structured data, the approach used in this dissertation will consider the CDT model as an application for context in a WSN.
It is important to notice that the Dimension Tree is intended here as a subset of a more complex tree, since the WSN may only be a small part of a bigger decision-making system.
As an example, if we want to monitor some complex or social phenomenon, like the behaviour of people after a natural disaster, the readings from a sparse sensor network may only play a marginal role in the overall decision making process.
It could be also of great important to take into account weather forecasts, tidal predictions, and many other external factors, independently from the WSN.
A sample CDT is presented in Figure 4, where a black circle represents a dimension, and a white circle a possible value of the dimension.
It is important to notice that the dimensions are not all of the same kind: two main classes are defined, each with two sub-classes, for a total of four final classes of dimensions were identified, with respect to the Sensor Network; every dimensional value includes a list of sensors associated with it, an exception are external dimensions, which behave in a different fashion as explained (a schema is sketched in Figure 3).

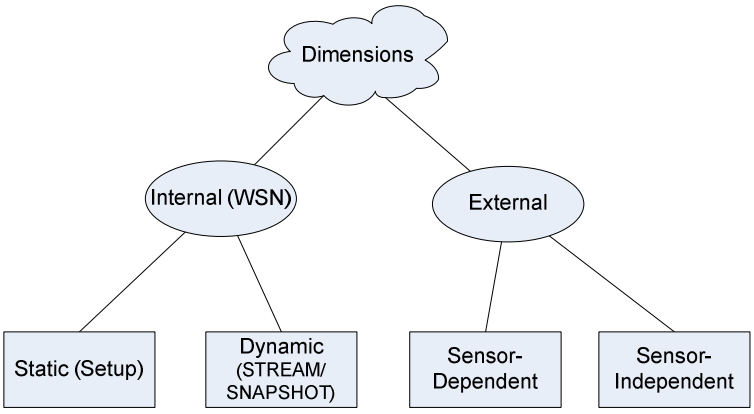

Figure 3

Internal Dimensions

Static Dimensions
In the setup phase of the network, some parameters are fixed for a sensor, as an example, if a sensor is capable of measuring only temperature, it should be inserted in a class specifying each and every single sensor which can only read temperature.
Other non trivial examples may arise: the location of a sensor may also be static, in the sense that a given sensor could only be able to work correctly at the sea-level, and so it would be impossible to be used in high mountain environment.

For political or commercial reasons it may also be possible that a sensor cannot be used to monitor some kind of phenomenon, and must only be used to monitor a particular kind of event: this exception can also be taken into account inside a Static Dimension

Dynamic Dimensions

The concept of dynamic dimension is in fact already present in PerLa, as it is another way of calling the SNAPSHOT and STREAM [1][14] data structures, but considering only the IDs of the sensors inside the structure and discarding all the other values.

For instance in the example of Figure 4, we can suppose that we are creating a SNAPSHOT of every temperature sensor which has a temperature under zero Celsius degrees, updated every X minutes, and we call this the dimension "Temperature" and put the IDs of the sensors satisfying the above condition in the "Below Zero" value. Every other temperature sensor can be included in the "Regular" value zone, or may also be left unassigned.

A remark is that with the STREAM structure instead of the SNAPSHOT one, it is possible to update the values structures in real time, if the reading of a sensor arrives suddenly without being queried.

External Dimensions

Sensor-Dependent

These dimensions are not typical of a WSN, and so it is important to notice that they may come from outside events. Even if this dimension is defined as sensor-dependent, it may still not be in the pure scope of our WSN.

Consider as an example the "Situation Type" branch of Figure 4, it may be possible that we are provided from an external source with a list of sensors, updated every hour, that are of military interest. Such a list may be configured like an external SNAPSHOT, were the criteria for which a given sensor belongs to a list is unknown at this level (WSN), but may be only accessible to a higher level.

By performing data tailoring for example on the "Civilian" value, we may be granted the right to use only those sensors which are not of military interest, with a list varying in time; this can also be thought as a security policy mechanism.

Sensor-Independent

This last dimension is the most abstract from a logical point of view; it is in fact a high level mechanism to implement a switch-like function from an external source.

Since these dimensions are not related to a particular kind of sensor, (i.e. the "Weather Forecast" branch in Figure 4), we can't explicitly add a list of sensors to them.

It may be non-sense to do such a thing, think about the list of sensors that may only be used in summer: this representation would be more suitable for a Static Dimension rather that a Sensor-Independent External Dimension.

The behaviour of an external dimension of this kind has to be thought in terms of Set Theory: when an external source enables a particular value, the value itself will act as containing the Universal Set of sensors. When the value is deactivated, the content would be the Empty Set.

Since it is always an external dimension, the user of the WSN doesn't usually have the rights to modify such values, which are treated as logical elements by the context manager in PerLa.

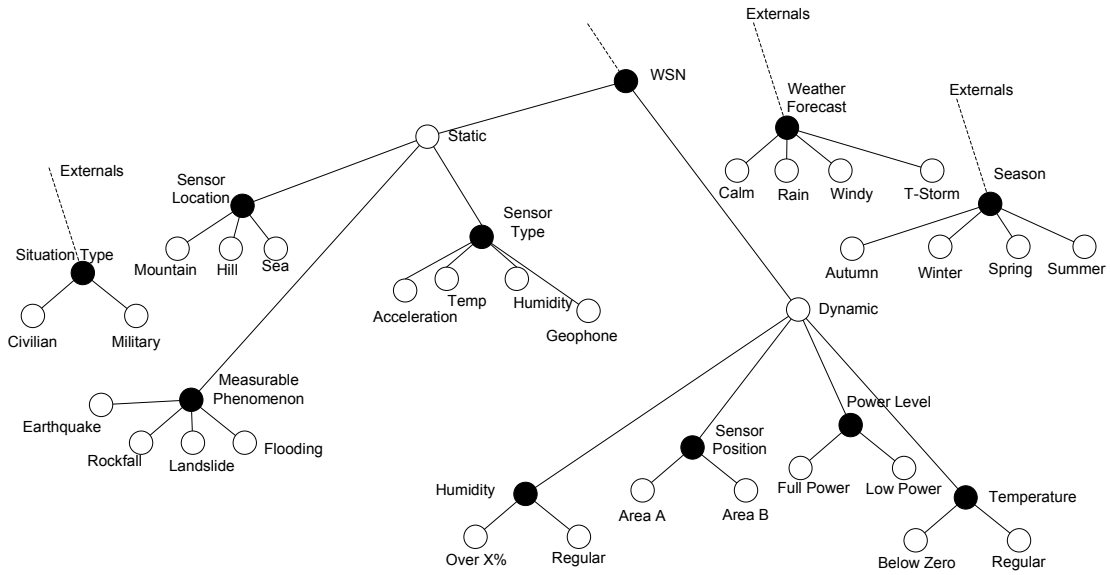

Figure 4

Summarizing what was said in this section, Figure 4 shows a sample instance of the CDT with respect to the entire network, also considering several external dimensions.

All the four kinds of dimensions as depicted in Figure 3 are shown in the picture above: Static and Dynamic dimensions are clearly identified, with some parameters which are fixed in time for the first one, like the type of sensor, the measurable phenomenon and so on, and some dynamic collections for the former kind of dimension, like area definition, temperature below zero, power level, and any other physical measure varying over time.

External dynamic dimensions are also present, in both fashions: Weather Forecast belongs to Sensor-Independent external dimensions, and the value contained in each of the possible states are to be intended with respect to set theory, i.e. if a certain value is imposed by an external entity, the content of the chosen dimension will be of the universal set of all sensors. This passage will appear clearer in the examples section.

Finally a sample Sensor-Dependent dimension is the one relative to "Situation Type", in which we receive a list of sensors from an external source, according to an unknown policy, which are of military or civilian interest. Data tailoring will be performed also on these kinds of dimensions.

5. Context Implementation

The reification process described in the previous sections ends in a data structure representation in PerLa, with a full declarative approach to create and maintain the context.
The context is intended as an intersection of dimensional values, which are previously defined with respect to the four possible ways described earlier.
In order to re-utilize the SNAPSHOT and STREAM structures, the only constraints are that the candidate table must in fact be a list in which only the ID appears, and this list should be composed on unique identifiers, thus avoiding repeating the same sensor ID twice or more.
The syntax of the context manager is described in the following:

CREATE CONTEXT <ContextName> **ON DIMENSIONS** <DimList>
[**SET** [<Enabled>] [<ActivateOn>] [<DeactivateOn> | <LifeTime>]]
[**ON ENABLE EXECUTE** '{' <Query> '}']
[**ON DISABLE EXECUTE** '{' <Query> '}']
REFRESH EVERY <Duration>

The parameters <DimList> contains the list of parameters to compute the intersection of sensors and is coded as:

<DimList> ➞ <Dimension>=<Value> { **AND** <Dimension>=<Value>}*

The <Dimension> attribute indicates the name of a possible dimension, and <Value> declares explicitly the chosen dimensional value, e.g. Sensor_Location=Mountain.
The SET clause is optional, it is important to notice that the sub-clauses of SET are important for temporal management of the context structure.
More specifically, the <Enabled> parameter may be left unspecified, in this way the context will be set with a policy specifying that if the context contains at least an element, it should become enabled, and becomes disabled if the context structure is empty. It is important to underline that the logical behaviour of the context is always binary, so the Enabled parameter may only assume True or False status; the Auto status is a shortcut to indicate that the overall status is defined by the context emptiness policy and if it is not set, the Enabled parameter has to be considered as a "Read Only" entity with respect to the sensor network, but not with respect to the human user.
The syntax is described as:

<Enabled> ➞ **ENABLED =** { **True | False | Auto** (Default)}

The <ActivateOn> may specify a timestamp the context manager uses to change its state from Enabled=False to Enabled=True or Enabled=Auto; both cases can be specified by the ALWAYS or AUTO clauses respectively.
<DeactivateOn> works in the same way, specifying a timestamp after which the context is set as Enabled=False. <LifeTime> is instead a shortcut using a temporal value that will be added to the <ActivateOn> timestamp, or to the current timestamp if the former is not specified, and works as the <DeactivateOn> clause.
The syntax of the three clauses is:

<ActivateOn> ➞ **ACTIVATE ON** <TimeStamp> [**ALWAYS | AUTO**]
<DeactivateOn> ➞ **DEACTIVATE ON** <TimeStamp>
<LifeTime> ➞ **LIFETIME** <Duration>

The next two clauses don't need any particular explanation: ON ENABLE EXECUTE and ON DISABLE EXECUTE just take care of executing a given query when the context passes from Enabled=True to False and vice-versa. These functions may be useful if we want to set some parameters or perform other kinds of queries when some context is active.
The final clause instead specifies the refresh value of the context: at the desired interval the context is updated with the sensor IDs satisfying the conditions at that moment.
Two more additional auxiliary clauses allow the user to manually enable or disable a context for whichever reason. Their meaning is intuitive as the only modified parameter is the Enabled one in the context structure:

ENABLE CONTEXT <ContextName> [**ALWAYS | AUTO**]
DISABLE CONTEXT <ContextName>

Notice that the ALWAYS and AUTO parameters have the purpose of specifying if we mean to force the context to be always enabled or if we want to let it work automatically and so decide by itself, basing on its emptiness condition.
As a brief note it may be useful to point out that forcing a context to be always active may lead to have an empty active context. This may seems awkward at a first glance, but when performing this operation we are in front of a context operating like an external entity, as discussed about external sensor-independent dimensions in the previous sections.
The internal dynamic dimensional values can be created with the usual SNAPSHOT/STREAM structure, with the constraint previously defined; but the dimension itself has to be specified, enumerating also the possible values it may contain.

CREATE DIMENSION <DimName> ' (' <Values> ') '

<Values> ➞ <Value> { ' , ' <Value> }*

The underlying software layer will later associate a snapshot structure to the dimension if the name of the snapshot is detected to be equal to a dimensional value.
Before completing this paragraph, a little modification has to be implemented in the <Condition> clause of the EXECUTE IF function as:

<Condition> ➞ …
| <ContextName> **IS ENABLED**
| <ContextName> **IS DISABLED**
…

The intuitive meaning of the above conditions is that we want to add the possibility to perform conditional comparisons even for those contexts for which we don't care about their content or which are even empty.

6. Examples

The contextual model described till this moment may seem awkward at a first glance: this section aims to introduce some examples to further clarify the meaning of the model and its theoretical power level, also considering a classical example found in [14].

Example A

The first example of the context model application is a hypothetical scenario in an area subject to random and heterogeneous phenomena, suitable for a demonstration of the capabilities of PerLa.
The north-eastern region of Sicily (Italy) is subject to intense seismic activity, which also causes severe side effects like tsunami waves after earthquakes. The presence of the tallest active volcano in Europe (mt. Etna) also requires continuous attention to eruptive phenomena, again related to the seismic activity. Nevertheless the area is also a crossing for naval traffic in the Mediterranean Sea, and it would be useful to implement a complete monitoring system comprising of several kinds of sparse sensors, thus creating a large scale WSN (despite of the distance, almost every location is in direct line-of-sight at least with the mount).
A heterogeneous set of sensors is imagined in this scenario, as depicted in Figure 5: red dots represent acceleration sensors, mounted on buoys floating in open sea, to monitor tsunamis; yellow dots stands for geophones, detecting vibrations for earthquake detection and placed inside fixed stations on land, or tied to the bottom of the sea for underwater measurements.
Green dots are temperature sensors, and blue dots represent humidity sensors; both of them could be placed on fixed stations or on floating buoys, eventually in the same floating device. Finally two areas of interest are depicted in blue (Area A) and red (Area B), surrounding respectively a high density urban agglomerate and the area historically most influenced by the volcano.

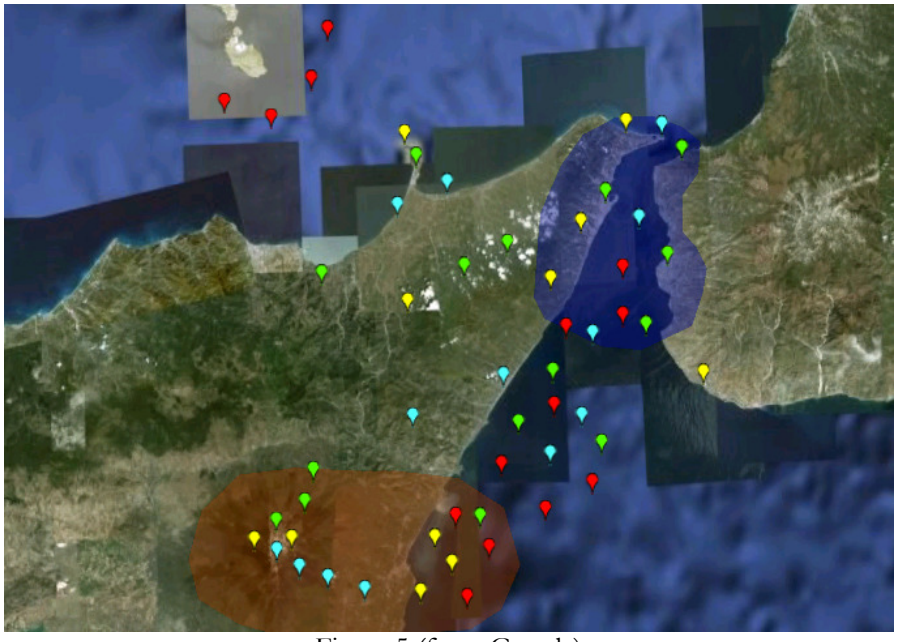

Figure 5 (from Google)

Some sample dimensions are presented, according to the schema of Figure 4. The tables are filled with the IDs of the sensors satisfying the belonging condition; T00 indicates a Temperature sensor, H00 a Humidity sensor, while A00 and G00 stands respectively for Acceleration sensors and Geophones. Only some of the dimensional value tables are presented, for the sake of simplicity.

Static Internal Dimension: Sensor Location

Mountain
T00
T01
H00
H01
T02
H02
G01
G02
G03

Hill
T03
T04
H03
H04
G04
G05

Sea
T05
T06
T07
A00
A01
A02
H05
G06
G07
G08
H06

Static Internal Dimension: Measurable Phenomenon

Earthquake
G01
G02
G05
G07
G08

Landslide
G03
G06
G04

Flooding
H02
H04
H03

Static Internal Dimension: Sensor Type

Acceleration
A00
A01
A02
A03
A04
A05

Temperat.
T00
T01
T02
T03
T04
T05
T06
T07

Geophone
G01
G02
G03
G04
G05
G06
G07
G08

Humidity
H00
H01
H02
H03
H04
H05
H06

Dynamic Internal Dimension: Sensor Position

Area A
A00
T03
T05
T06
H03
H06
G04
G05

Area B
A01
A02
G02
G03
G06
G07
H00
H01
H04
T01
T07

Dynamic Internal Dimension: Humidity

Over X%
H00
H02
H04
H06

Regular
H01
H03
H05

Sensor-Dependent External Dimension: Situation Type

Military
G01
A02
G02
G06
H00
T01

Civilian
All the other sensors

A similar architecture can provide the monitoring of a diverse and rich set of phenomena, to illustrate such a possibility, some sample contexts are created. Notice that some dynamic dimensions, like "Humidity" in the previous page, can be easily created by means of the CREATE SNAPSHOT function already present in PerLa. Static dimensions are meant to be filled at setup time or when a new sensor is added to the WSN.
The first sample context allows modelling a situation in which we are interested in monitoring seismic events in Area B; in particular since the very nature of the underwater surface, we want to monitor the phenomenon of underwater landslides.
The context can be modelled as:

CREATE CONTEXT UnderWaterLandslides
ON DIMENSIONS Sensor_Location=Sea **AND** Measurable_Phenomenon=Landslide
AND Sensor_Position=Area_B **AND** Sensor_Type=Geophone
REFRESH EVERY 1 m

The obtained output list of the context contains the following data, computed as intersection among all the dimensions involved in the creational process:

UnderWaterLandslides
G06

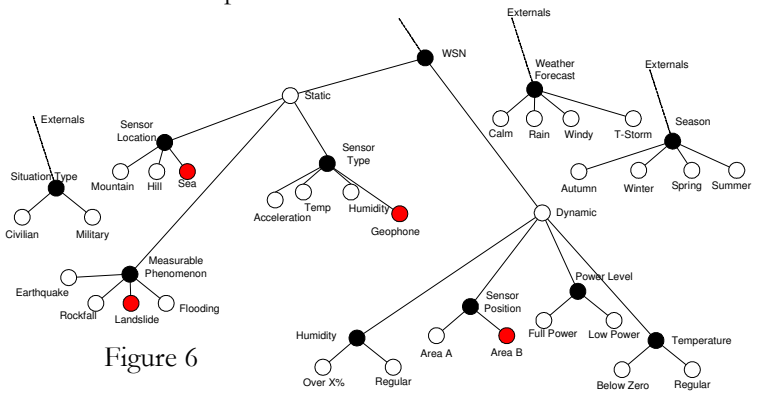

Figure 6

It is now feasible to create an output stream to read only the sensors tailored by the context, where we suppose to project the ID and the value of the geophones:

CREATE OUTPUT STREAM MonitorUWLandslides (SensorID ID, Value FLOAT)
AS LOW:
EVERY 30s
SELECT ID, Value
SAMPLING EVERY 30s
EXECUTE IF ID **IN** UnderWaterLandslides

The last EXECUTE IF line, allows to perform the query only on the tailored sensors list (i.e. just sensor G06), without having the need to query other sensors.
The second sample context can be thought as a consequence of the first one: when an underwater landslide occurs, the probability of a tsunami is very high in the area.
We should monitor the tidal status only if we are already monitoring underwater landslides, so the new context and its relative output stream should look like:

CREATE CONTEXT TsunamiAlert
ON DIMENSIONS Sensor_Location=Sea **AND** Sensor_Type=Acceleration
AND Sensor_Position=Area_B
REFRESH EVERY 1 m

CREATE OUTPUT STREAM MonitorTsunami (SensorID ID, Value FLOAT) **AS LOW:**
EVERY 30s
SELECT ID, Value
SAMPLING EVERY 30s
EXECUTE IF ID **IN** TsunamiAlert
AND UnderWaterLandslides **IS ENABLED**

TsunamiAlert
A01
A02

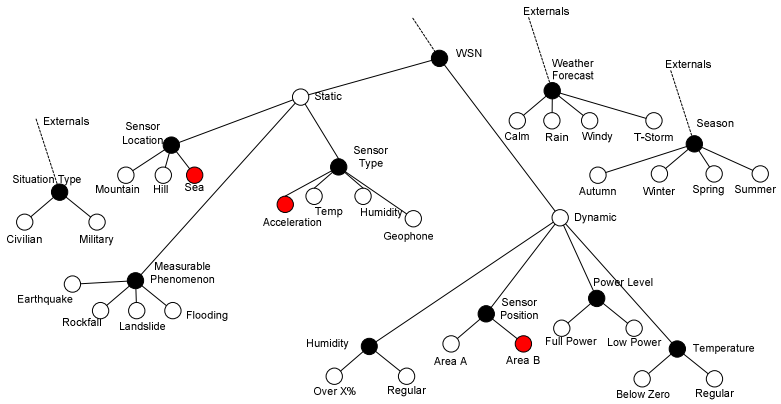

Figure 7

A useful remark is that every dimension has its own refreshing time, specified during the creation of the snapshot table, so the context can be freely refreshed without affecting the tailoring process anymore; this is one of the major differences with respect to the PILOT JOIN function.
Considering external dimensions is the task of the next context, where we want to monitor flooding phenomena in Area A, when it is winter time and the weather forecast reports thunderstorm events:

CREATE CONTEXT FloodingRisk
ON DIMENSIONS Measurable_Phenomenon=Flooding **AND** Sensor_Type=Humidity **AND** Sensor_Position=Area_A **AND** Weather_Forecast=T-Stom **AND** Season=Winter
REFRESH EVERY 1 m

FloodingRisk
-Empty- (If not in Winter and no T-Storm approaching)

Otherwise:

FloodingRisk
H03

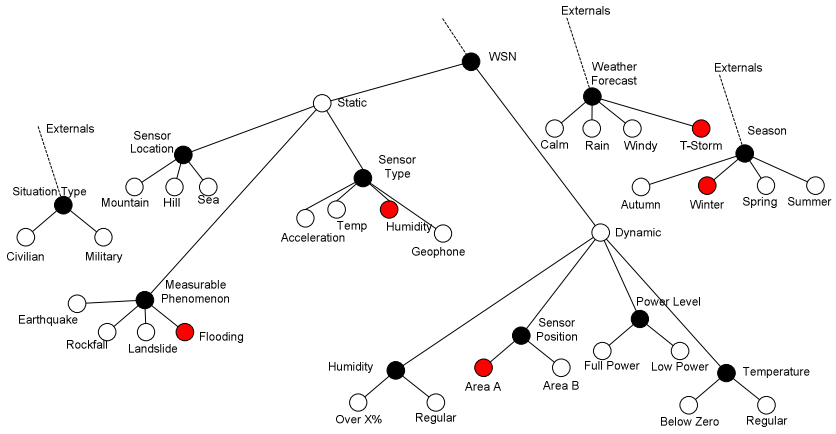

Figure 8

The created context can be used in a STREAM structure, but it is important to notice that the FloodingRisk table will be empty (from the definition of external sensor-independent dimensions) whenever there is no thunderstorm approaching or the current season is not winter. Since the ENABLED parameter was not specified, the context behaviour is automatic, in other words it will mark itself as disabled when empty, and enabled when populated by at least one element.
A similar reasoning could be performed for the "Situation Type" dimension: in this case the dimensions are treated like internal dynamic dimensions, with the only difference that we don't have to create the table with a SNAPSHOT since it will be provided by an external source; a brief example is the following query for context definition useful to detect the marine fog formation risk, a phenomenon usually occurring in spring, the additional task is avoiding to monitor the sensors dedicated to military use:

CREATE CONTEXT FogRisk
ON DIMENSIONS Humidity=Over_X% **AND** Season=Winter
AND Sensor_Location=Sea **AND** Situation_Type=Civilian
SET ACTIVATE ON 2009-03-20T04:30UTC **AUTO**
LIFETIME 15 d
REFRESH EVERY 10 m

CREATE OUTPUT STREAM MonitorFog (SensorID ID, Temp FLOAT) **AS**
LOW:
EVERY 30s
SELECT ID, Temp
SAMPLING EVERY 30s
EXECUTE IF FogRisk **IS ENABLED**

FogRisk
H06

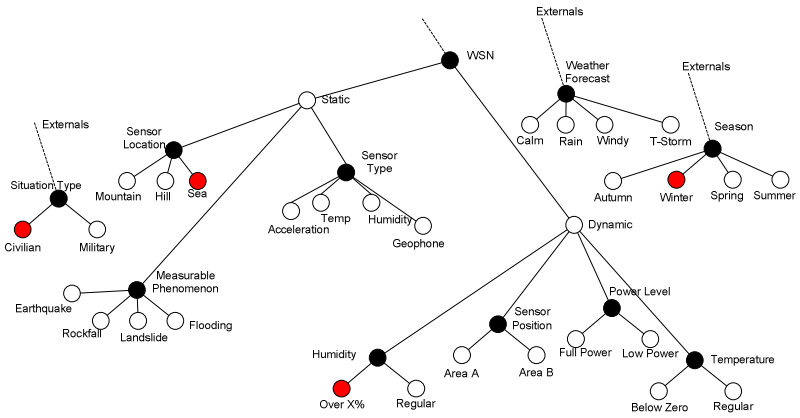

Figure 9

Note on Example A:
In 1908 an earthquake stroke what was indicated as Area A, lasting 37 seconds and destroying 90% of the buildings. Several minutes later, a submarine landslide occurred in Area B, generating a tsunami which reached Area A within minutes, killing 200.000 [20]
The last tsunami occurred in 2002 in the Aeolian Islands, north-west of Area A, with waves as high as 20 m.
About once a year a peculiar marine fog phenomenon occurs in Area A, causing frequent ship collisions and in some cases the complete stop of navigation.

Example B

The last example is a rewriting of the example found in [14] using the context model instead of the PILOT JOIN function.
The data structure used by the sensors will also be proposed for the sake of clarity, for more details on the Field Type column, see [14].
Since the binding between the two sensor typologies is explicit, this particular kind of problem can be solved by the PILOT JOIN in a lesser number of lines, but the dimension structure created with a high level SNAPSHOT is re-usable in many other kinds of similar contexts.
The moving of the data tailoring condition from the PILOT JOIN to the EXECUTE IF function also allows an overall improvement of the performance of the generated STREAM.
The example is the following:

There is a set of tanks, containing some temperature sensors. A UHF-RFID tag and a base station are mounted on each tank. The temperature sensors contained in the tanks that are in the range area of the RFID reader with ID [reader] must be sampled once every minute. The range area will be called Area A.

UHF-RFID tag			
Logical object wrapping a single UHF-RFID tag			
Field Name	**Data Type**	**Field type**	**Description**
ID	ID	ID	Logical object identifier
currentReaderID	ID	P	ID of the reader that is currently sensing the tag
linkedBaseStationID	ID	S	ID of the base station mounted over the same tank
deviceType	STRING	S	Type of device

WSN node			
Logical object wrapping a single WSN node equipped with a temperature sensor			
Field Name	**Data Type**	**Field type**	**Description**
ID	ID	ID	Logical object identifier
baseStationID	ID	NP	ID of the base station the WSN node is currently connected to
temp	FLOAT	P	Sampled temperature

Figure 10 (From [14])

CREATE SNAPSHOT TanksSnapshot (tagID ID, linkedBaseStationID ID)
WITH DURATION 1 h **AS**
LOW:
    SELECT ID, linkedBaseStationID
    SAMPLING EVERY 1 h
    WHERE currentReaderID = [reader]
    EXECUTE IF deviceType = "UHF_RFID_TAG"

CREATE DIMENSION Sensor_Position (InAreaA)

CREATE SNAPSHOT InAreaA (Identifier ID)
WITH DURATION 1 h **AS**
HIGH:
    SELECT TanksSnapshot.tagID
    FROM TanksSnapshot
UNION
    SELECT Temperatures.sensorID
    FROM TanksSnapshot (1 h), Temperatures (1 h)
WHERE TanksSnapshot.linkedBaseStationID = Temperatures.baseStationID

CREATE CONTEXT TempInAreaA **ON DIMENSIONS**
Sensor_Type=Temperature **AND** Sensor_Position=InAreaA
REFRESH EVERY 1 m

CREATE OUTPUT STREAM Detect_Temp (sensorID ID, temp FLOAT) **AS**
LOW:
    EVERY ONE
    SELECT ID, temp
    SAMPLING EVERY 1 m
    EXECUTE IF ID **IN** TempInAreaA

Note: The "Temperatures" SNAPSHOT mentioned in this example is supposed to be present as setup time, simply enumerating all the temperatures sensors installed in the entire network.
The Sensor_Type static internal dimension is also supposed to be present at setup time, with the same meaning of the previous case.
Note also that the InAreaA dimension gives us more information than the PILOT JOIN counterpart: we now know every kind of sensor which is in Area A, disregarding of its type.

7. Additional Features

In [16] some constraints for context tailoring were defined. The constraints were essentially created to avoid inconsistencies in contexts generation, and discretional policies to implement restrictions. Such constraints can also be reported in the context model presented here: PerLa should communicate with the upper layer managing constraints when a context creation function is called, and generate an error if the context is detected as forbidden or useless.
For example, going back to Example A, a plausible constraint is that we can't perform data tailoring on the "Military" value of the Situation Type dimension and (at the same time) on the "Geophone" value of the Sensor Type one. The overall result is that we are forbidden to use geophones dedicated to military use.
Such constraints can also be thought in terms of auto configuration of the new devices added to the network. PerLa development is going toward the implementation of a Plug&Play-like feature for the addition of a new sensor device, essentially based on XML configuration files.
An idea is to include some high-level contextual constraints in such XML file, that will travel along the PerLa context layer to the upper layer of constraints management, whenever a new sensor type is added.
A newly added class of sensor may in fact embed a policy including some constraints (e.g. the sensor can't be used in a specific situation). The presence of the constraints management layer, already implemented in the Context-ADDICT project, allows PerLa to consider the whole problem as an external one, requiring only negative or positive reply from the upper layer, which knows the conceptual context model used by PerLa at any time.

8. Future Work

The context-model presented here is meant as a proposal to enhance the capabilities of PerLa over the simple data tailoring capabilities offered by the PILOT JOIN operation. More features could be added in the future, the most important being:

- **Operations on dimensions and/or contexts**
    - It may be useful to introduce some logical operators when managing dimensions or contexts. In this work only the AND operator is considered to tailor dimensional values, but some other operators may be added, like OR, XOR, IF-THEN, UNION etc. This addition would require the inclusion of a software management layer dedicated to exploiting propositional logic properties with respect to dimensions and contexts, but would also allow a more complete management of inter-context priorities, which can be modelled using well known logical structures.

- **Automatic context detection and/or recognition**
    - Detecting an unforeseen context is an ambitious goal that requires the application of Artificial Intelligence techniques and a robust implementation of statistical techniques. For instance again in Example A, a statistical function to detect outliers may estimate that some temperature sensors, all placed in a mountain area of the Area B are reading values ten times greater than the annual average (e.g. 300°C instead of 30°C). An automatic context generation function may include those sensors in a new auto-generated context. A human operator will then see a list of such outliers, thus knowing their position: this could be an excellent way to detect an unforeseeable lava leak, not rare in the region.

9. Appendix A: EBNF Grammar

Formal grammar definition of the context model is reported here:

CREATE CONTEXT <ContextName> **ON DIMENSIONS** <DimList>
[**SET** [<Enabled>] [<ActivateOn>] [<DeactivateOn> | <LifeTime>]]
[**ON ENABLE EXECUTE** '{' <Query> '}']
[**ON DISABLE EXECUTE** '{' <Query> '}']
REFRESH EVERY <Duration>

<DimList> → <Dimension>=<Value> { **AND** <Dimension>=<Value>}*

<Enabled> → **ENABLED =** { **True | False | Auto** (Default)}

<ActivateOn> → **ACTIVATE ON** <TimeStamp> [**ALWAYS | AUTO**]

<DeactivateOn> → **DEACTIVATE ON** <TimeStamp>

<LifeTime> → **LIFETIME** <Duration>

ENABLE CONTEXT <ContextName> [**ALWAYS | AUTO**]

DISABLE CONTEXT <ContextName>

CREATE DIMENSION <DimName> ' (' <Values> ') '

<Values> → <Value> { ' , ' <Value> }*

<Condition> → …
| <ContextName> **IS ENABLED**
| <ContextName> **IS DISABLED**
…

DROP CONTEXT <ContextName>

10. Bibliography

[1] F.A. Schreiber, R. Camplani, M. Fortunato, M. Marelli; *A Middleware Architecture for Data Management and Integration in Pervasive Heterogeneous Systems;* Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milano, Italy; IEEE PerCom 2009

[2] Jan Steffan Ludger Fiege Mariano Cilia Alejandro Buchmann; *Scoping in Wireless Sensor Networks*; Department of Computer Science, Darmstadt University of Technology; ACM International Conference Proceeding Series; Vol. 77 Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, Toronto, Ontario, Canada Pages: 167 - 171

[3] Chong, Krishnaswamy, Loke; *A Context-Aware Approach to Conserving Energy in Wireless Sensor Networks*; School of Computer Science and Software Engineering Monash University,Melbourne, Australia; PERCOMW 2005; Pages: 401 - 405

[4] Sungjin Ahn and Daeyoung Kim; *Proactive Context-Aware Sensor Networks;* Real-time and Embedded Systems Laboratory, Information and Communications University (ICU); European Workshop on Wireless Sensor Networks, Feb. 13-15, 2006, Switzerland.

[5] Kyungseo Park, Byoungyong Lee, Ramez Elmasri; E*nergy Efficient Spatial Query Processing in Wireless Sensor Networks;* Computer Science and Engineering, University of Texas at Arlington; Advanced Information Networking and Applications Workshops, 2007, AINAW '07. Publication Date: 21-23 May 2007

[6] Eiman Elnahrawy and Badri Nath; *Context-Aware Sensors;* Department of Computer Science, Rutgers University; Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 2920/2004

[7] Amirhosein Taherkordi, Romain Rouvoy, Quan Le-Trung, and Frank Eliassen; *A Self-Adaptive Context Processing Framework for Wireless Sensor Networks;* University of Oslo, Department of Informatics; Middleware Conference, Proceedings of the 3rd international workshop on Middleware for sensor networks table of contents, Leuven, Belgium, Pages 7-12 , 2008

[8] Paolino Di Felice, Massimo Ianni, Luigi Pomante; *A spatial extension of TinyDB for wireless sensor networks;* DlEI DEWS - Universita dell'Aquila, Italy; Computers and Communications, 2008. ISCC 2008. IEEE; 6-9 July 2008

[9] Henri Dubois-Ferriere; *Efficient and Practical Query Scoping in Sensor Networks;* School of Computer and Communication Sciences, EPFL Lausanne, Switzerland - Department of Computer Science UCLA, Los Angeles, CA; Center for Embedded Network Sensing. Technical Reports. Paper 28. 2004.

[10] Markus C. Huebscher, Julie A. McCann; *Adaptive middleware for contextaware applications in smarthomes;* DSE Group, Department of Computing Imperial College London; ACM International Conference Proceeding Series; Vol. 77 , Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, Toronto, Ontario, Canada, Pages: 111 – 116, 2004

[11] SAMUEL R. MADDEN Massachusetts Institute of Technology
MICHAEL J. FRANKLIN and JOSEPH M. HELLERSTEIN UC Berkeley
WEI HONG Intel Research, Berkeley; *TinyDB: An Acquisitional Query Processing System for Sensor Networks; ACM Transactions on Database Systems (TODS)* Volume 30 , Issue 1 (March 2005)

[12] Chia-Hsing HOU, Hung-Chang Hsiao, Chung-Ta King, Chun-Nan Lu; *Context Discovery in Sensor Networks;* Computer Science, National Tsing-Wua University, Hsinchu, Taiwan; Information Technology: Research and Education, 2005. ITRE 2005. 27-30 June 2005; Pages: 2- 6

[13] Ali Salehi, Karl Aberer; *GSN, Quick and Simple Sensor Network Deployment;* Ecole Polytechnique Federale de Lausalle, Switzerland; European conference on Wireless Sensor Networks (EWSN), Delft, 29-31 Jan 07

[14] M. Fortunato, M. Marelli; *Design of a declarative language for pervasive systems;* Politecnico di Milano Master thesis 2006/2007, Prof. F.A. Schreiber, Ing. R. Camplani

[15] F.A. Schreiber, R. Camplani, M. Fortunato, M. Marelli; *Full declarative SQL-like high level language to query pervasive systems hiding the complexity of handling different technologies*; Politecnico di Milano Dipartimento di Elettronica e Informazione, Milano, Italy; EuroSSC 2008

[16] C. Bolchini, C.A. Curino, E. Quintarelli, F.A. Schreiber, L. Tanca; *Context Information for Knowledge Reshaping;* Politecnico di Milano Dipartimento di Elettronica e Informazione, Milano, Italy; TBP on Int. Journal of Web Engineering and Technology, 2009

[17] C. Bolchini, C.A. Curino, E. Quintarelli, R. Rossato, F.A. Schreiber, L. Tanca; *And what can context do for data?;*, Politecnico di Milano Dipartimento di Elettronica e Informazione, Milano, Italy, TBP in Communications of ACM, Sep. 2009

[18] F.A. Schreiber*; Automatic Generation of Sensor Queries in a WSN for Environmental Monitoring;* Politecnico di Milano Dipartimento di Elettronica e Informazione, Milano, Italy; Proceedings of the 4th International ISCRAM Conference, Delft, The Netherlands, May 2007

[19] C. Bolchini, G. Orsi, E. Quintarelli, F.A. Schreiber, L. Tanca; *Progettazione dei dati con l'uso del contesto;* Politecnico di Milano Dipartimento di Elettronica e Informazione, Milano, Italy; Mondo Digitale n.3 – settembre 2008

[20] http://en.wikipedia.org/wiki/1908_Messina_earthquake