EXECUTIVE SUMMARY

PERLA – PERvasive LAnguage

INTRODUCTION TO LANGUAGE FEATURES

SCHREIBER F.A., CAMPLANI R., FORTUNATO M., MARELLI M.
Dipartimento di Elettronica e Informazione - Politecnico di Milano

There are many real word applications that are continuously monitored using a large number of heterogeneous sensing devices. Different kinds of sensors, both in terms of technology and functionality, are often simultaneously used within the same application.

In the following, we refer to the automation of a large wine production farm, from the vineyard to the table, as a motivating example. This is one of the case studies in the ART DECO project, a large project funded by the Italian University Ministry.

The process has to be completely monitored, starting from the growing in the vineyard to the transport and the bottles maintenance in the wine cellars. A specific monitoring system is required for each of the previous phases: Wireless Sensor Networks can be the best technology to sense useful vineyard parameters, like temperature and humidity; bottles identification and tracing can be performed applying them an RFID tag; finally, GPS devices allow to monitor the current location of the trucks involved in the wine transport process.

The integration of data collected using the various above mentioned technologies is certainly an interesting challenge, but the different interfaces provided to control and query each involved kind of device make this goal very hard to achieve.

The project presented in this brochure aims at analysing the described problem and suggests a possible solution: the main idea is the definition of a full declarative high level language that allows to query a pervasive system, hiding the difficulties related to the need of handling different technologies.

The expression “pervasive system” refers to a large heterogeneous network composed of many devices, belonging to different technologies, like wireless sensors networks (*WSN*), *RFID* systems, *GPS* and other kinds of sensors.

We aim at providing a database like abstraction of the whole network in order to hide the high complexity of low level programming and allowing users to retrieve data from the system in a fast and easy way.

In the following we list the main aspects that have been considered during the design phase of PERLA language:

- *Data representation and abstraction*
- *Physical devices management*
- *Functional language features*
- *Non Functional language features*

Analyzing the previous requirements and taking into account the large heterogeneity of the considered devices, we decided to split the language into two levels:

- *A* **LOW LEVEL LANGUAGE***, that allows to precisely define the behaviour of a single device. The main role of a low level statement is to define the sampling operations, but also to allow the application of some SQL operators (grouping, aggregation, filtering) on sampled data.*
- *A* **HIGH LEVEL LANGUAGE***, that allows to define data manipulation over the streams coming from low level queries.*

Both languages have an SQL like syntax, but the low level language semantics is quite different with respect to the standard SQL one: in fact, specific clauses have been introduced to manage sampling operations. Moreover, the language has been designed to make easier the generation of aggregated data starting from sampled data. On the contrary, the high level language semantics is very similar to that of streaming databases.

The definition of the language semantics is based on the concept of logical object. This is an abstraction of physical devices that allows application objects to interact with the hardware, through the definition of a suitable interface. Each logical object wraps a single or a homogeneous aggregate of physical devices.

At the application layer, a query analyzer handles user submitted queries and retrieves the list of logical objects composing the system, using a specific component

(**REGISTRY**). This information is then used to select the logical objects that will take part in the query.

The described architecture allows to abstract the whole pervasive system as a collection of logical objects. In this way, each request submitted to a node can be thought as a request to the corresponding logical object, through the exposed interface. The precise definition of this interface allows to describe the language semantics as a set of interactions between the query executor and the logical objects. As an example, from the language point of view, a sampling request to a node having a temperature sensor on board is abstracted as an attribute reading from the node interface. Similarly, when an *RFID* tag is sensed by a certain *RFID* reader, the language detects an event fired by the logical object wrapping the tag.

The interface a logical object should expose to be involved in the query execution process should include the following functionalities:

- **RETRIEVING ATTRIBUTES.** Attributes can be used to retrieve both the state of the node and the sampled data. An attribute can be the abstraction of a sampled value, of a cached one or a constant. Attributes can be classified in three categories:
  - **STATIC ATTRIBUTES:** represent constant values describing the characteristics of the node (e.g.: type of the node, maximum sampling rate, etc). These values don't change during the logical object lifecycle.
  - **PROBING DYNAMIC ATTRIBUTES:** when an object tries to retrieve this kind of fields, the logical object must refer to the physical device to produce the value before returning it. Probing attributes can be mapped on a real sensor (e.g.: temperature, pressure, etc) or on a memory area (e.g.: information stored in a device RAM, ROM, flash, etc). Note that dynamic attributes also include policies whose value must be sampled whenever required (e.g.: battery level).
  - **NON PROBING DYNAMIC ATTRIBUTES:** when an object tries to retrieve this kind of fields, the logical object returns a local cached value without dealing with the physical device. The cached value is periodically updated by the logical object.
- **FIRING NOTIFICATION EVENTS.** Some operations defined in the declarative language are activated after an event is risen. Thus, the logical object

interface must be able to fire some events coming from physical devices (e.g when an *RFID* reader senses an *RFID* tag).

- GETTING THE LIST OF SUPPORTED ATTRIBUTES AND EVENTS. A function should be provided to allow the query analyzer to discover if a logical object supports a certain attribute. That function should also provide the type of a given attribute, both in terms of data type (integer, string, etc) and attribute type (static, probing or not probing).

A user-defined query can be expressed as a graph, whose nodes define both data structures and low and high level statements composing the query. The data structures can be stream tables and snapshot tables: streams are unbounded lists of records produced by queries, while snapshots are sets of records produced in a given period and stored in dedicated buffers. This second kind of data structure has been introduced to support *pilot join*, that is a special operation that allows to start the sampling of some nodes, depending on data sampled by other nodes.

To provide a first idea of the language, we now report a quite complex query related to the wine transport scenario presented above. The goal of this query is to retrieve temperature samplings from sensors placed in the truck that is currently closest to a given point P. Figure 2 shows the graph corresponding to this query.

A

```
CREATE STREAM TanksPositions (gpsID ID, linkedBaseStationID ID, distanceFromP FLOAT) AS
LOW:
  EVERY ONE
  SELECT ID, linkedBaseStationID, dist_from_P(locationX, locationY)
  SAMPLING EVERY 1 h
  EXECUTE IF deviceType = "GPS"
```

B

```
CREATE SNAPSHOT NearestTank (gpsID ID, linkedBaseStationID ID)
WITH DURATION 1 h AS
HIGH:
  SELECT TanksPositions.gpsID, TanksPositions.linkedBaseStationID
  FROM TanksPositions (1 h)
  WHERE TanksPositions.distanceFromP = MIN(TanksPositions.distanceFromP)
```

C

```
CREATE OUTPUT STREAM Temperatures (sensorID ID, temp FLOAT) AS
LOW:
  EVERY ONE
  SELECT ID, temp
  SAMPLING EVERY 1 m
  PILOT JOIN NearestTank ON NearestTank.linkedBaseStationID = baseStationID
```

Figure1: Example of query

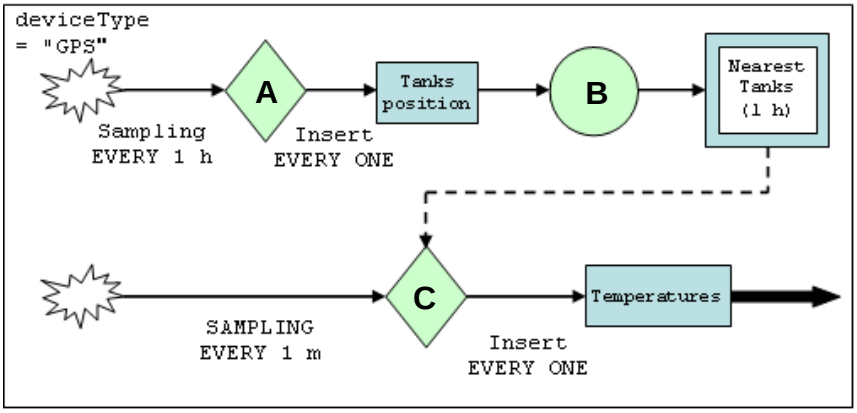

Figure 2: Query graph correspondent to Figure 1 query

To conclude, we summarize the main results of our work:

- *Definition of a fully declarative language that allows to uniformly querying a pervasive system composed of very heterogeneous devices (e.g. WSNs and RFIDs).*
- *Introduction of an event based semantics that allows to query inherently event based devices (such as RFIDs) using a SQL like syntax.*
- *Introduction of the pilot join operation, which doesn't seem to be supported in similar existing projects.*

BIBLIOGRAPHY

- Fabio A. Schreiber, Romolo Camplani, Marco Fortunato, Marco Marelli, Filippo Pacifici: "*PERLA: a Data Language for Pervasive Systems*", in Proceedings of Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008). Honk Kong, pp. 282-287, 2008.

- Schreiber F.A., Camplani R., Fortunato M., Marelli M.: "*Design of a Declarative Data Language for Pervasive Systems*" - Art Deco R. A. 11.1b, January 2008.