EXECUTIVE SUMMARY

PERLA – PERvasive LAnguage

DECLARATIVE LANGUAGE AND MIDDLEWARE FOR PERVASIVE SYSTEMS

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

The system described in this brochure is basically composed of three components:

• **THE NODES** are heterogeneous devices equipped with sensors, able to collect data and to send them in the network managed by the middleware. Nodes can either be very simple devices (such as *RFID tags* or *WSN* nodes such as *MOTES*) or more complex devices (such as palms, portable computers or ad hoc boards).

• **THE MIDDLEWARE** is a stack of software layers (called *LOGICAL OBJECT*) providing a high level abstraction of each node. The middleware also implements a set of functionalities to allow communications among logical objects and to manage devices that enter and leave the system (following a "*Plug and Play*" behaviour).

• **PERLA (PERvasive LAnguage)** is an *SQL* like language designed to allow the querying of a pervasive system as it were a database. The goal of the language is to enable the final user to collect data in a fast and easy way, without dealing with low level programming issues.

THE LANGUAGE

The language allows the user to interact with logical objects wrapping (and masking) physical devices. It is worth to notice that a logical object can abstract both a single node and a set of devices (*e.g.* a whole *WSN*).

The language currently supports three kinds of queries:

• **LOW LEVEL QUERIES.** These queries are executed on a single logical object and define how and when the sampling should be performed, how sampled data should be locally processed and which results have to be produced. A clause is provided by the syntax of these queries to specify the conditions defining the set of logical objects the query will be instantiated and executed on. As an example, consider the following low level query:

```
CREATE STREAM Results (id ID, temp INTEGER) AS
LOW:
     EVERY 1 hour
     SELECT id, AVG(temp, 1 hour)
     SAMPLING EVERY 1 min
     EXECUTE IF powerLevel > 0,90
```

This query will be executed over all the nodes reporting a power level greater than 90%. Each involved node will collect a temperature sample every minute and it will produce an output record every 60 minutes, to retrieve the average of the last 60 temperature samples. The final result of this query is then a stream of records, that are generated once for hour and logical object.

• **HIGH LEVEL QUERIES.** These queries are very similar to the normal streaming databases ones and they allow to manage different streams produced by low level queries. To better clarify, consider again the previous example in which a temporal average is computed over single node samplings: a high level query could instead perform a spatial average over results produced by the previous query.

• **ACTUATION QUERIES.** These low level queries are not intended to collect data from a logical object, but rather to send some commands to the logical object they are executed on. For example, suppose that a logical object is wrapping an "ad hoc" board that performs signal filtering to generate sampled data. In this case, an actuation query could be used to change the filter parameters.

THE MIDDLEWARE

The goal of the middleware is to provide an abstraction for each device in terms of logical objects and to support the execution of ***PERLA*** queries. During the design of the middleware, we tried to make the definition and the addition of new devices easier. We also tried to minimize the amount of low level code the user has to write to make the new device recognizable by the middleware.

The middleware is composed of the following layers:

• **LANGUAGE PARSER.** It receives textual queries as input, verifies their syntax and transforms them in a suitable format for distribution and execution.

• **LOGICAL OBJECT REGISTRY.** It is the component that maintains the list of the logical objects currently registered in the system. It is also used to find the set of logical objects that will be involved in the execution of a certain low level query.

• **HIGH LEVEL QUERY EXECUTOR.** Basically, it is a streaming database.

• **LOW LEVEL SOFTWARE.** This software layer adapts the different physical devices to the middleware. It also provides the abstraction of logical objects. We decided to implement the whole software, from the logical objects layer up, using *JAVA* technology. We also assumed that each logical object (that we can now consider as a *JAVA* remote object) is reachable via *TCP/IP*. If the physical device is connected to a different network (*e.g.* a *CAN-BUS* channel), the correspondent logical object will be instantiated on the nearest device equipped with both a *JAVA Virtual Machine* and a *TCP/IP* connection to the network managed by the middleware. This middleware layer implements the communication protocol between the logical object and the physical device.

• **LOW LEVEL QUERY EXECUTOR.** This is a *JAVA* component, contained in each logical object, whose goal is to receive and execute a parsed low level query.

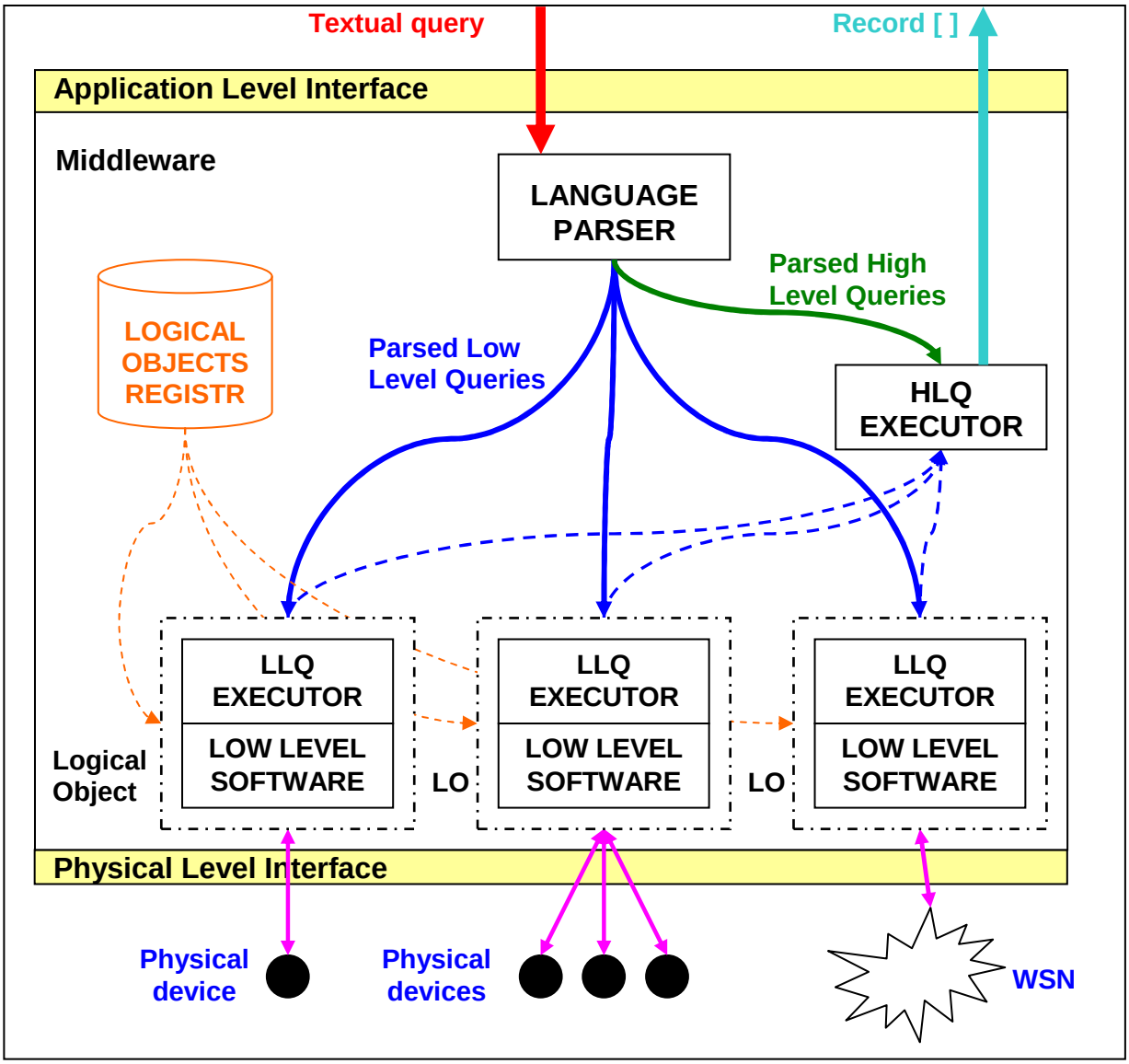

Figure 1: ***PERLA*** middleware

Figure 1 shows all the components described above and highlights the interactions among them. Basically, the middleware exposes two interfaces to the final user:

- **APPLICATION LEVEL INTERFACE:** it provides methods to submit textual ***PERLA*** queries and to receive the results of executed queries. A precise definition of the interaction model between the user and the middleware has not been established yet; probably, two socket channels will be provided: the first one will be used to receive queries, while the second one will return produced records.

- **PHYSICAL LEVEL INTERFACE:** the goal of this interface is to allow the user to integrate a new device into the system. It is basically a *C* library, that should be extended and recompiled whenever a new device should be integrated in the system. We tried to minimize the low level programming effort by the user to integrate a new technology in the middleware. To reach this goal we defined the structure of a XML file that should be generated for each new device. This file contains a full description of the device in terms of:

- Available sensors;

- Measures that can be sampled;

- Communication protocol between the device and its logical object (*e.g.* structure of exchanged packets).

Figure 2 has the role of highlighting the interaction between the physical node and the middleware.

- **"A"** is the physical device equipped with sensors, where the user *C* code is executed.

- **"B"** is the embedded board or PC placed nearest to ***"A"*** and having an environment where the logical object can be hosted (*i.e.* ***"B"*** must execute a *JAVA Virtual Machine* and must be connected to the middleware via *TCP/IP*).

- **"C"** is the server where the logical objects registry is executed.

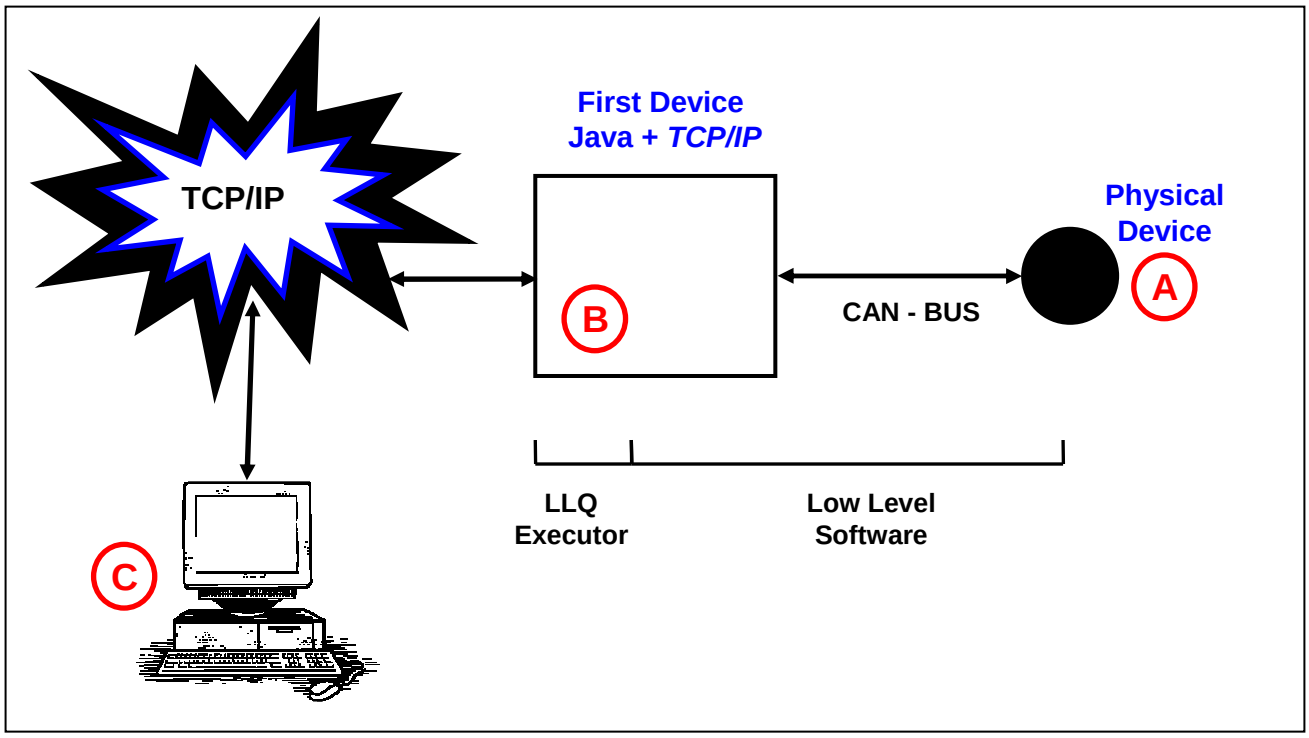

Figure 2: Interaction between physical nodes and ***PERLA*** middleware

Communications between ***"B"*** and ***"C"*** are completely managed by the middleware, while communications between ***"A"*** and ***"B"*** (that can be performed over a non *TCP/IP* channel, such as a *CAN-BUS* or serial channel) is only indirectly managed by the middleware. In fact, the XML file presented above defines the parameters and the messages of the communication, that is performed executing some code contained in the *C* library.

It is worth to notice that the scenario presented in *Figure 2* is the most general one, but ***"B"*** and ***"C"*** components can be hosted on the same physical device when it is powerful enough.

CONCLUSIONS

In the following the main features of the middleware are reported:

- ***PERLA*** supports alarms and signals management;
- ***PERLA*** is partially distributed;
- ***PERLA*** supports heterogeneous devices at run time;
- ***PERLA*** supports the addition of new devices and the removal of existing ones in a *"Plug and Play"* fashion;
- ***PERLA*** provides support for local computation of sampled data (*built-in SQL* functions/aggregations and *user-defined* functions/aggregations);
- ***PERLA*** provides support for *built-in SQL* data types and provides hooks to define new *user-defined* data types.

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.