POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Computer Science and Engineering

Distributed Data Collection in a Context Aware Pervasive System

Relatore: Prof. F.A.Schreiber

Tesi di Laurea Magistrale di

FRANCESCO MARIA FILIPAZZI

Matricola: 816424

Abstract

Con l'imminente avvento delle reti di telecomunicazione di quinta e sesta generazione, il ruolo dei Sistemi Pervasivi, delle reti di sensori wireless e concetti come quello di Internet of Things saranno centrali nello sviluppo di nuove tecnologie. La presente tesi spiega come, in un Sistema Pervasivo, i dati possano essere raccolti da sensori fisici e comuni canali di diffusione delle informazioni, per essere poi inseriti in database di ultima generazione e trattati ad alto livello all'interno di un Contesto. La progettazione parte dal linguaggio PerLa e dal middleware ad esso dedicato, che è stato potenziato, prevedendo la possibilità di implementare una struttura distribuita per la raccolta dei dati. La trattazione giunge allo sviluppo di un prototipo che, ispirandosi al disastro di Rigopiano, ha lo scopo di gestire una situazione emergenziale in caso di valanghe collegate a scosse sismiche.

Abstract

With the imminent advent of fifth and sixth generation telecommunication networks, the role of Pervasive Systems, wireless sensor networks and concepts such as the Internet of Things will be crucial in the development of new technologies. This thesis explains how, in a Pervasive System, data can be collected by physical sensors and common information distribution channels, to be then entered into the latest generation databases and processed at a high level within a Context. The design starts from the PerLa language and from the middleware dedicated to it, which has been enhanced, providing the possibility of implementing a distributed structure for data collection. The discussion leads to the development of a prototype which, inspired on the Rigopiano disaster, has the purpose of managing an emergency situation in the event of avalanches connected to seismic earthquakes.

Contents

List of Figures

Listings

Chapter 1

Introduction

In 1991 Mark Weiser [1] envisioned "pervasive systems" for the first time. He described ubiquitous computers that disappear into the background, not visible to users, that assist people in everyday life. "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."
A pervasive system is composed of hundreds of heterogeneous and independent devices, usually connected with a network. Wireless sensor networks are infrastructures composed by wirelessly connected devices, called nodes, used to detect, measure and monitor physical phenomena and collect data about it. PerLa, Pervasive Language, is a software infrastructure for data management and integration in Pervasive Information Systems. Its development started in 2006 with a thesis entitled "A Declarative Language for Pervasive Systems" [5], written by Marco Fortunato and Marco Marelli, master students of Politecnico di Milano. In 2014 Guido Rota in his master thesis [6] described and developed an high-level abstraction layer, that can be used to collect information from a Pervasive System, i.e., a heterogeneous network composed of sensing and actuating devices with different characteristics. The software was conceived to work on a central server to which devices send data collected during their work. In 2014 Julia Malovic described a plug-in to store data arrived from sensors in a NoSql-Cassandra database. In 2015 Guido Rota developed the Low Level Query Language, to allow final users

to write sql-like queries and collect data from devices in a transparent way.
We can define a distributed database "as a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users" [13].
Oracle defines a distributed database as "a set of databases in a distributed system that can appear to applications as a single data source" [14]. Data are stored in multiple computers usually distributed over a network, to store large amount of data and ensure fault tolerance, because even if a system fails, the integrity of the database is maintained.
The DBMS market offers many solutions to manage distributed databases. Oracle developed distributed SQL queries and during last years many No-Sql engines have been developed supporting DDBMS. PerLa needs to support data storage in some distributed databases, because this feature is actually required for industrial and research use.
The aim of this master thesis is to show two recently developed important system improvements:
-a plug-in to connect PerLa Middleware to a MySql database and the completion of the support of distributed databases using Cassandra;
-the distribution of Middleware and, consequently, of calculation and data aggregation among nodes connected to the network.
A simple prototype application will be developed at the end of the work. It will be composed of the distributed Middleware, the query system and the database storage plugin.

Chapter 2

Contex-Aware Pervasive Computing and PerLa

In the coming years with the advent of technologies such as 5g and 6g [7], systems will become increasingly "pervasive" and "ubiquitous". However, technological evolution has led to the production of billions of devices constantly connected to the global network, which collect and send data to the servers of many organizations. The treatment and conservation of the overwhelming amount of information that is collected is a daily challenge. In this scenario, the pervasive computing paradigm, together with the concept of Internet of Things, is increasingly important, but must necessarily be associated with efficient data management. In fact, data must be processed, stored, studied and used to improve people's lives and to better develop the technology itself.

"The scope of data management in pervasive systems is to give the users an instantaneous and complete access to any information at any time and anywhere" [2].

A pervasive system is therefore composed of sensors, actuators and RFID identification systems. In addition, an advanced data collection system must be provided, in our case a middleware that acts as an intermediary between the physical layer and the layers responsible for information management, such as database or context management.

2.1 Wireless Sensor Networks

From the 70s to today, information technology has evolved [3] in the area of local and geographic networks, in the field of personal computers, distributed systems, wireless technologies and sensors. All these technologies make up the elements of pervasive systems. In particular, an extremely important development is that of Wireless Sensor Networks (WSN).

2.1.1 Sensors and Actuators

Leaving aside the network infrastructure, the main components of a WSN are sensors.These are electronic devices that collect data from the surrounding environment. The sensors read a physical quantity and transform it into an electrical signal, which is then read and stored or used to act accordingly to environmental changes.
The sensors can have various sizes, but currently very small devices have been developed, capable of disappearing from view. A Wireless Sensor Network can therefore be a network of invisible sensors, realizing the paradigm proposed by Weiser in 1991. Sensors can measure many physical quantities, such as temperature, pressure, brightness, speed and acceleration.
Actuators are components that produce changes in the system, bringing it from a starting state to an arrival state. The actuators can be, for example, electric, electromagnetic or hydraulic and can be of various kinds, such as motors or diodes. If, for example, in a specific context, an incorrect state (danger) is detected, actuators can be activated to bring the system back to a correct state (safety).

2.1.2 Rfid

Since the 1940s there has been research to develop electronic systems to automatically identify objects. This technology is called RFID (Radio Frequency IDentification) and in recent years, due to the significant reduction in costs, it had a great diffusion.
This technology can be divided into two groups: active tags and passive tags.

Since active tags require the use of batteries for power, they take up some space and their use is not economically convenient. Passive tags, on the other hand, are composed only of an antenna and a chip in semiconductor material.
A passive tag reader therefore needs to supply energy to the tag to communicate with it, in order to collect the information saved in it. This technology therefore makes it possible to produce extremely miniaturized devices that carry information.
RFID technology today has a lot of potential, because tags can be created with information inside that goes beyond the simple identification of an object. For example, information about the owner or destination of a shipped item can be saved.
Another very interesting use is that of RFID that incorporates sensing devices. [8].

2.2 Pervasive Systems and IoT

The concepts of Pervasive Computing and Internet of Things (IoT) should not be confused, even if they are closely linked. Speaking of IoT, a concept formulated for the first time in 1999, we indicate the set of all objects, of various kinds, connected to the global network. It is not just about personal computers and smartphones, but about vehicles, appliances, smartwatches, medical equipment and many other types. These objects are geolocalizable, can transmit and receive information, interact with people and with each other. The IoT Paradigm tends to create an environment made up of interconnected objects. These are dynamic environments, which change size quickly, in which the elements connect and disconnect without central control.
In an IoT system, basically everything has an electronic identity and the real world is mapped as a network of devices.

2.3 Modern Pervasive systems

2.3.1 Scalability and Adaptability

Pervasive systems are dynamic in nature. The amount of connected devices can change very quickly and, especially in cases where their number increases significantly, the infrastructure must be able to work without discontinuity. The scalability of a software system is the ability to adapt quickly and, possibly, transparently its operation in the event of a change in size and a significant increase in the amount of data collected.
This is a fascinating challenge, taking into account the fact that in a pervasive system communication does not take place unidirectionally because each device receives and sends data. In addition, each device communicates with many other devices, thus creating "many to many" relationships.

2.4 Context Aware Pervasive Systems

The concept of "context" has been defined over the years in various ways. A very important definition in literature is that of Dey: "Context is any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects ".
To define the context we also need three computational points of view.

- Physical: it is the set of environmental conditions of a system, such as temperature, heat and humidity.
- User: people's position, identity, role within the environment.
- Computing environment: devices present in the system, input and output interfaces, network.

The context therefore serves to provide the essential information to identify the state of a system, so that you can choose what is really useful for your

purposes.
Another important concept is that of "context awareness". A system capable of attending its behavior based on the change of the surrounding context, is a "context aware system".
This is another very important feature in modern Pervasive Systems.

2.5 PerLa

2.5.1 Why the PerLa project?

As already mentioned, in a pervasive system a software structure is needed to collect and analyze data. PerLa, the Pervasive Language, was created to have a sensor system that can be interrogated as if it were a database, regardless of the hardware differences of the various sensor models. For this reason PerLa is an SQL-Like language.
In order to overcome the technical differences between the sensors, it was decided to abstract them and make them "present" to the system through a descriptor, written in the form of structured data.
Therefore PerLa middleware was born, which deals with the organization of communication with sensors through an abstraction called the Functionality Proxy Component and the descriptors in XML format.

2.5.2 Context PerLa

Originally PerLa was conceived only for collecting data from a pervasive system but, because of the great potential of the developed middleware, more functionalities have been added. Context PerLa is an expansion of PerLa [4] that:

- defines the environment with a suitable model;
- creates a context on the defined model;
- acquires the sensor readings and external inputs which define a specific context;

- activates or deactivates a context at run-time depending on the actual values of the context variables;
- performs the context-aware actions on the system.

2.5.3 Some scenarios of Pervasive Systems

- In 2012, a research team from the Politecnico di Milano devised a process for monitoring the production and transportation of wine. Through the use of the PerLa language, of technologies such as RFID and GPS receivers, a pervasive system was defined that followed the entire wine-making process. [9]
- In [2] a museum has been modeled as a Context Aware Pervasive system. Figure 2.1 shows the Context Schema modeled as a Context Dimension Tree (CDT) of the studied system.
- [10] Describes a pervasive system that aims to monitor a large-scale agricultural environment in Australia. For example, the use of sensors positioned on cattle is proposed.
- [11] explains the possibility of integrating Wireless Sensor Networks and drones for disaster management support acts.

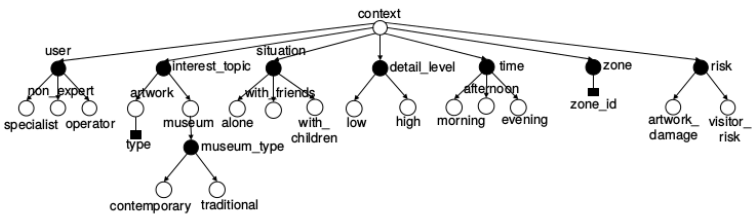

Figure 2.1: The Context Dimension Tree context model for a museum

Chapter 3

Starting situation and feasible improvements

The PerLa middleware has been developed around a set of APIs [6], that are interfaces towards the final users. They allow everyone to add plug-ins and software units to PerLa. This modular design underlies the possibility to connect the systems to many types of databases provided they are supported by Java Libraries.

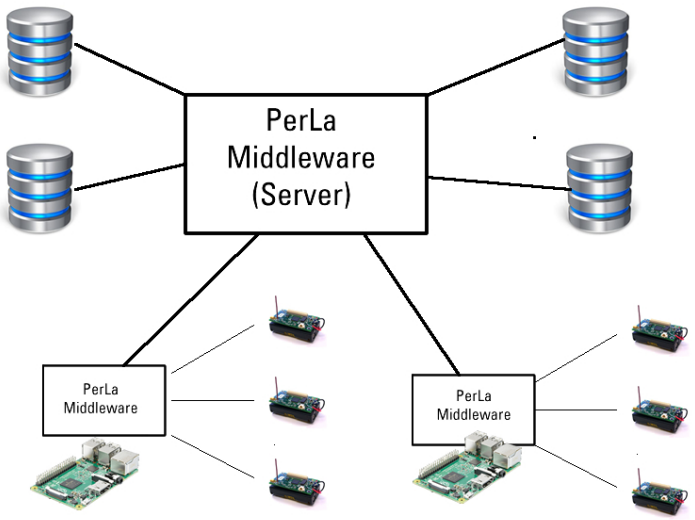

Figure 3.1: Distributed PerLa

3.1 PerLa Middleware

3.1.1 The FPC

The Middleware is based on the Functionality Proxy Component (FPC), that is a data interface. FPCs provide a high-level abstraction of a Pervasive System, through a single consistent Advanced Programming Interface (API). Every instance of the PerLa Middleware hosts multiple FPCs, one for each sensing node.

FPC is a self contained proxy object that abstracts the physical and other

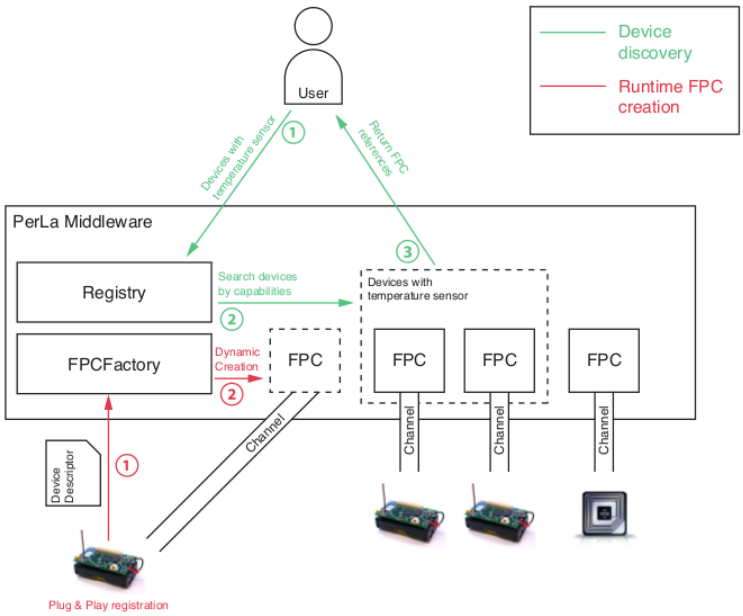

Figure 3.2: PerLa Middleware

information sources by embedding all the logic required to communicate with a single remote device. The most prominent trait of the FPC is its interface, an API that allows PerLa users to interact with the sensing network through a compact set of hardware-agnostic communication primitives reminiscent of classic Java getter and setter methods. Use of this interface neither requires knowledge of the sensing network, nor of the device that will ultimately perform the requested operation. This components allow to expand

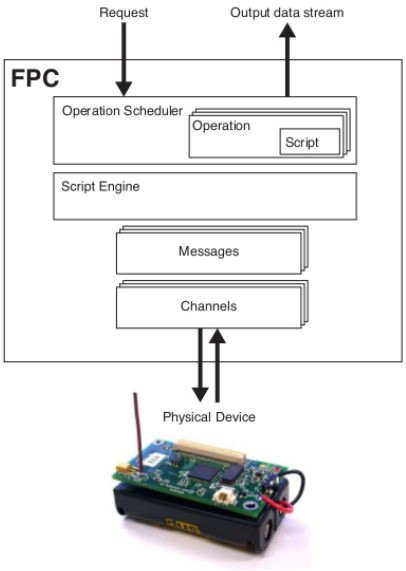

Figure 3.3: FPC internal structure

the structure of PerLa and make it compatible with any source of structured data. Using a set of high level functions, users can manage a heterogeneous Pervasive System composed of many types of sensor and http services. An FPC can communicate with data sources through Channels and can provide data coming from them to a destination decided by the user. In our case, the destinations are two DBMS: Apache Cassandra and MySql.
Task and TaskHandler are the components that allow plugins development to expand PerLa features.

Task

The Task is an object for controlling the execution of an asynchronous FPC operation. A Task can be employed to stop the ongoing operation or to query its current state.

```
1
2 public List<Attribute> getAttributes();
3
4 public boolean isRunning();
5
```

```
6 public void stop();
```

Listing 3.1: FPC Task

TaskHandler

The TaskHandler is a general handler interface for collecting the results of an asynchronous FPC Task.
When a sample is ready, the *data()* method is invoked, to use sampled data for the current purpose.

```
1
2 public interface TaskHandler {
3
4    public void data(Task task, Sample sample);
5
6 }
```

Listing 3.2: FPC TaskHandler

3.1.2 The Channel

The Channel is an interface for performing I/O operations. It represents the principal abstraction used by the middleware to communicate with hardware devices and external software services. The Channel is not tied to a specific technology so a wide variety of data management tasks can be instantiated as a Channel. The current Middleware architecture encourages the creation of several highly specialized Channels, which are usually developed around third-party communication libraries.
For example, to develop the TinyOsChannel, the Java TinyOs libraries and the PerLa Channel Interface have been used.

```
1
2 public class AbstractChannel extends Channel;
3
4 public class TinyosServerListener
5       implements
6    net.tinyos.message.MessageListener;
```

```
7
8  public class TinyosChannel extends AbstractChannel;
9
10 public TinyosChannel(int id, TinyosChannel tosl);
```

Listing 3.3: The TinyOs Channel Java constructor

3.1.3 The Device Descriptor

To create a new FPC a Device Descriptor is required. The Descriptor is a XML file that contains the details of all the features that characterize the node in terms of data structures, attributes, protocols of communication, computational capabilities, and behavioral patterns.
The XML file is parsed and the FPC is created following its description by the FPC Factory, a software component that guarantees a Plug&Play addition mechanism.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2   <device type="test" xmlns="http://perla.dei.org/device">
3
4  <attribute>
5  <!-- Attribute declarations -->
6  </attribute>
7  <channel>
8
9  <!-- Channel declarations -->
10 </channel>
11 <message>
12
13 <!-- Message declarations -->
14 </message>
15 <request>
16
17 <!-- IORequest declarations -->
18 </request>
19 <operation>
20
21 <!-- Operation and Scripts -->
22 </operation>
```

```
</device>
```

Listing 3.4: The Device Descriptor Template

For example, in 3.5 an attribute section is shown. It describes a sensor that measures temperature, pressure and humidity.

```
<attributes>
   <attribute id="temp" type="float" permission="read-only"/>
   <attribute id="pressure" type="float" permission="read-only"/>
   <attribute id="humidity" type="float" permission="read-only"/>
</attributes>
```

Listing 3.5: Device Descriptor Attributes

The Descriptor must declare the channel to communicate with the described sensor. For example in 3.6 a sensor communicating with TinyOS is described.

```
<channels>
   <http:channel id="tinyos"/>
</channels>
```

Listing 3.6: Device Descriptor Channel

3.1.4 Device Description with SensorML

PerLa interacts with sensors and generic data sources described by XML schemas. A good XML syntax allows to create a good abstraction and work with a wide range of devices. The just described syntax has been developed for the particular scope of PerLa Middleware.
Sensor ML [12] is a standard description model developed by Open Geospatial Consortium, an international standard organization, composed by 481 governmental, commercial and non profit organizations. Sensor ML could be an important opportunity for the PerLa Project, because it is a complete standard, with a lot of features and has been projected for a large range of uses and could allow the interaction with systems already based on OGC standards.
The University of Alabama has initiated a number of projects to use this

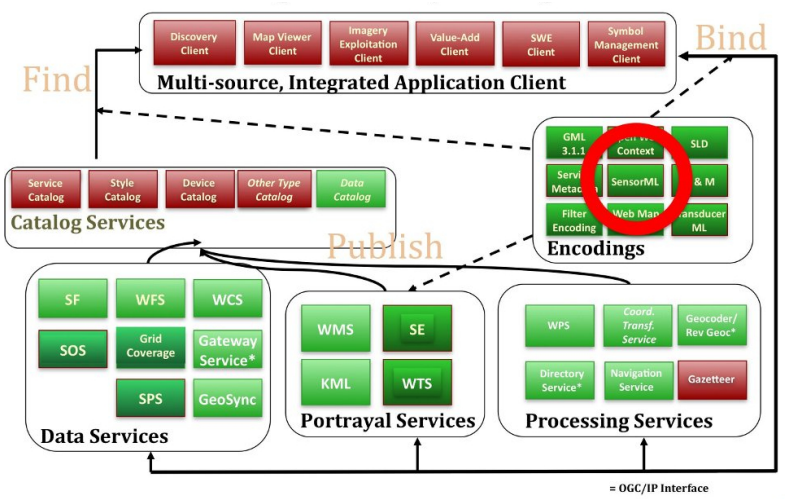

Figure 3.4: The Open Geospatial Consortium standard

language, including an editor and a Java library application, JAXB, which is useful for creating Java classes from XML data structures. SensorML is inserted and included in a wide system of standards for analysis, description, cataloging and archiving created by the OGC and other bodies.
Using Sensor ML, PerLa could be internationalized and could have an industrial scope, because the OGC consortium works with companies as Airbus, Lockheed Martin or as Oracle and Google.

3.2 The levels

A PerLa application is composed of three levels:

- Low Level or data collecting level, that collects data from physical sources. Example of Low Level modules are the TinyOs and the Http modules;

- Middleware that receives data coming from the Low level and processes them using the FPC abstraction. The PerLa middleware is provided with some structures to interact with the other two levels. The Channel sends information coming from the data sources to the FPCs and the FPCs, using the *data()* method, can interact with the Upper Level.

- High Level that makes data available to the final user and allow to store them or to place them in a context. Example of High Level modules are the database modules or the Query language.

3.3 Relational database integration into PerLa

The first part of this thesis describes the project and development of a plug-in that connects a PerLa system to a MySql database. However No Sql databases have been conceived to overcome relational databases. During the last few years, these technologies achieved success, because they can be easily used in real-time applications, they can be distributed and they can manage big data. PerLa must be able to connect to a NoSql database system, to be used for advanced projects. So an example shows that the plug-in can be used to connect PerLa to a NoSql database as Cassandra starting from the plugin developed for MySql connection.

3.4 Data management distribution

The Middleware provides a single server that receives all data coming from channels. Channels can be connected to http services or physical devices [6]. Furthermore, all queries are sent from the server to devices, because all FPCs are registered on the server. This structure can be useful in the presence of a few channels, but can become a bottleneck if a system is composed of hundreds of heterogeneous devices. The second part of this thesis focuses on the definition of a software distribution infrastructure that can be used to divide high level computation and query activity into smaller, independent units of work to be executed on the individual nodes of the sensing network.

Chapter 4

Connecting the PerLa Middleware to a Database

4.1 Relational Databases and MySql

In 1970 Edgard Codd spoke about the need for the users to manage data banks without knowing how the data is organized in the machine [21] and proposed the "relational data model", applying principles of relations to data systems.
In the following years the relational model became the most important data model and has been used all over the world for storage and processing data. SQL, Structured Query Language, is a programming language designed to manage relational databases and it is the primary database language in the world. SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. The Sql support is a critical feature in PerLa development, because the project has the purpose to be available for the largest possible number of users.
MySql is a DBMS that uses SQL to store and manage data.

4.1.1 Why Mysql?

MySql is an open-source relational database management system. In 2013 it was the world's second widely used RDBMS. It provides the capacity to handle embedded applications that use a few MBs, to massive databases holding terabytes of information. It is developed by Oracle Corporation that supports users with a lot of guides and a rich documentation.

4.2 NoSql and Apache Cassandra

4.2.1 NoSql

Someone considers NoSql databases as an evolution of Relational databases, because they don't use the traditional tabular model, substituted by a key value model. Most of the NoSql structures are designed similarly to an object oriented model, to avoid expensive mapping of code to relational structures. These technologies are used by the most important software enterprises in the world, as Google or Amazon, and have a lot of advantages as high horizontal scalability or lower cost for setting up a node with respect to Sql. They are useful for distributed systems, for big data management and cloud computing.
They allow to manage complex data, because usually these databases don't handle the simple tabular data. Another advantage of NoSql management systems is the possibility to manage huge numbers of users, accessing systems concurrently and constantly.This is a very important feature for an application as distributed PerLa.

NoSql categories

There are four categories of NoSql databases [15]:

- Key-values Stores. Used for applications that have frequent small reads and writes along with simple data models. Values can be simple, as integers or booleans, or they may be structured data types, as lists.

- Column Family Stores. In a relational database, data access is by row. A Column DBMS changes the focus from the row to the column. This model improves performance when large data aggregations are needed, but reading operations are fewer than writing operations. Usually "columnar databases" can manage large amounts of data that require high availability.

- Document Databases. Probably the most popular of the NoSql databases, document databases are used to store varying attributes. In relational model, different data is stored in different tables. In the document-oriented model, data that is frequently queried together is stored in the same document. Each document is identified by a key. So, in relational model each table has uniform records, while in document-oriented model each record has its own characteristics.

- Graph Databases. Systems that can be represented as networks of connected entities, can be well supported by graph databases.

4.2.2 Why Apache Cassandra

Starting from early years of twenty first century, a lot of NoSql systems have been developed to meet specific market demand. Apache Cassandra is one of the most important available projects. We decided to use it to connect the PerLa Middleware to a NoSql database, because it can be used for academic, industrial and personal projects. For example, the Atlas project at Cern is based on a Cassandra database. A wide Wireless Sensor Network can produce hundreds of heterogeneous megabytes of data per year, so Cassandra has a lot of interesting features that could be used for an advanced application of PerLa.
Cassandra can be seen as a hybrid between a column-oriented and a key-value system. So it supports large aggregations, but supports a lot of contemporary writing operations too. Google demonstrated that a Cassandra database, mounted on an adequate hardware structure, can support one million writes per second [16] These features are fundamental for the PerLa

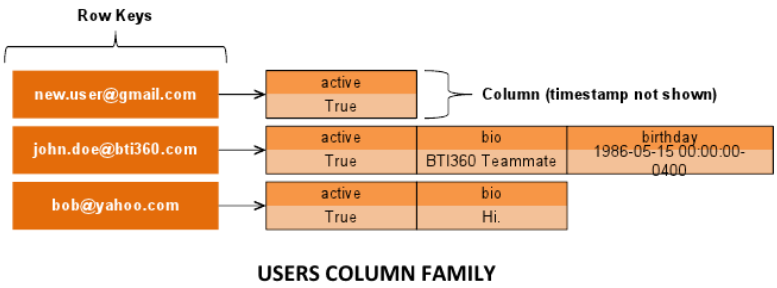

Figure 4.1: Structure of a Cassandra Schema

system. Cassandra is used to build fault tolerant systems because data is automatically replicated to multiple nodes and can be replicated across different data-centers. So we can prevent loss of data even if an entire data center fails. This NoSql system supports massive insertion data rates and a database can be developed following the evolution of the system.

4.2.3 Structure of a Cassandra Database

Cassandra is essentially a hybrid between a key-value storage and a database. Instead of tables, "column families" are used. Column families contain rows and columns. The basic object is the row, uniquely identified by a key. Each row has multiple columns, each of which has a name, value, and a timestamp. Unlike a table in an RDBMS, different rows in the same column family do not share the same set of columns, and a column may be added to one or multiple rows at any time. This structure is named "sparse column storage engine", while traditional relational model is named "static column storage engine". In a relational system each row must reserve memory space for every column of the schema. In this model, space is used only if a column is not empty.

4.3 Database connection development

To connect PerLa to a database, we need to create a software layer between the final user and the FPC. This layer is the Wrapper class, that is connected

to the classes Handler and TaskHandler. These classes can be adapted to be used for a connection between PerLa and a database. Now we are going to explain the structure of these classes and to make a comparison between the code used for connecting to Cassandra and the code used for connecting to MySql.

4.3.1 Class: DatabaseTask

DatabaseTask implements the Task class in the Middleware, that controls the synchronous and asynchronous operations of the system. DatabaseTask receives the name of table to be used as a parameter. In this way the user can find out the table name which the system has created to save the records that he has requested.To control the operation, the methods of a Task must be used. So the class receives a Task as parameter too. Each Task corresponds to a table.

```
1 public class DatabaseTask implements Task {
2
3    private String tableName;
4    private Task task;
5
6    public DatabaseTask(Task task, String tableName){
7       this.task = task;
8       this.tableName = tableName;
9    }
```

Listing 4.1: Database Task

4.3.2 Class: DatabaseWrapper

This class is the layer between the FPC and the user. It receives data coming from an FPC and saves it into the chosen database. Each FPC corresponds to a schema. The global variables are the FPC and the schema name that will be created and used to save data coming from it.

```
1 protected FPC FPC;
2    protected String schemaName;
3
```

```
4     public DatabaseWrapper(FPC FPC) {
5         this.FPC = FPC;
6         schemaName = "FpcSchema" + FPC.getId();
7     }
```

Listing 4.2: Database Wrapper

This class is provided with three methods, which wrap the corresponding FPC methods, that are useful to exemplify how the system works, because they are related to the most important features of the system. In particular a user can call two "get" methods and one "async" method. The methods *get()* and *periodicGet()* receive the Collection of FPC attributes as parameter. These attributes will be mapped as table columns. The methods also receive a Task Handler that notifies data coming from the FPC to the component that required them. These methods call the *get()* method provided by the FPC, that is a part of the Middleware.

```
1 public Task get(Collection<Attribute> attributes,
2     TaskHandler handler,String tableName)
```

Listing 4.3: Get Method

The *periodicGet()* is similar to the first method, but it receives a long type attribute. This is a sample period, that is used to manage periodical sampling.

```
1 public Task periodicGet(Collection<Attribute> attributes,
2     long periodMs,TaskHandler handler, String tableName)
```

Listing 4.4: Get Method with sample period

The *async()* methods is used to manage the asynchronous calling. It uses the *async()* method of FPC.

```
1 public Task async(Collection<Attribute> attributes,
2     long periodMs,TaskHandler handler, String tableName)
```

Listing 4.5: Async Method with sample period

4.3.3 Class: DatabaseHandler

DatabaseHandler is a nested class of DatabaseWrapper. **This class is the focal point for the database connections. It depends on the chosen DBMS and must be changed accordingly**. In wrapper class, DatabaseHandler is initialized using a TaskHandler and the FPC attributes that will be used to build the table.

```
private DatabaseHandler(TaskHandler handler,
      Collection<Attribute> attributes) {
      connect(hostAddr, port);
      createSchema();
      this.setAttributes(attributes);
      this.setHandler(handler);
      }
```

Listing 4.6: Database Handler

The *data()* method is invoked to store the new record into the database.

```
public void data(Task task, Sample record) {

   saveRecord(record);
   handler.data(task, record);
```

Listing 4.7: Data method

Connect() method uses java.sql.* libraries methods to connect to a MySql server and com.datastax.* libraries methods to connect to a Cassandra Server. Then *createSchema()* is used to build a string, in the chosen database language, to create a schema.
MySql Version:

```
private void createSchema() {
   String schemaCreationQuery="CREATE DATABASE IF NOT EXISTS "
    + schemaName;
   cmd.executeUpdate(schemaCreationQuery);
   this.hostAddr= "jdbc:mysql://localhost/"+schemaName;
   connect(this.hostAddr);
      }
```

Listing 4.8: Creation of a MySql database by Java Code

Cassandra version:

```
private void createSchema() {
   session.execute("CREATE KEYSPACE IF NOT EXISTS "
         + schemaName
    + " WITH replication "
   + "= {'class':'SimpleStrategy',
   'replication_factor':3};");
      }
```

Listing 4.9: Creation of a Cassandra Schema by Java Code

As it can be seen, the code differs only if a method connects directly to the DBMS. This example shows that PerLa can simply connect to many types of databases, with a limited interfacing effort.

Chapter 5

Message Exchange between nodes

In order to develop a distributed query system, to allow the communication between the central server and the nodes, a messaging protocol is required. To achieve this goal a Message Service can be used. The message service can be developed using a Message Oriented Middleware (MOM) [19].

5.1 Message Oriented Middleware

A MOM is an infrastructure for sending and receiving messages between distributed systems. MOMs are widely used in systems composed by heterogeneous components. They create a communication layer between the operating system and the application, so the application developers don't have to know details of network interfaces. A system based on distributed PerLa will be composed by servers, single-board computers, mobile phones and other devices, therefore a MOM is needed.
A distributed system must work even if a node is not connected to the network and problems related to intermittent connectivity had to be solved. A MOM provides an asynchronous [18] communication model to allow many clients to send data at the same time. It uses queues for temporary storage of messages to allow message exchange even if a component is temporarily not available.

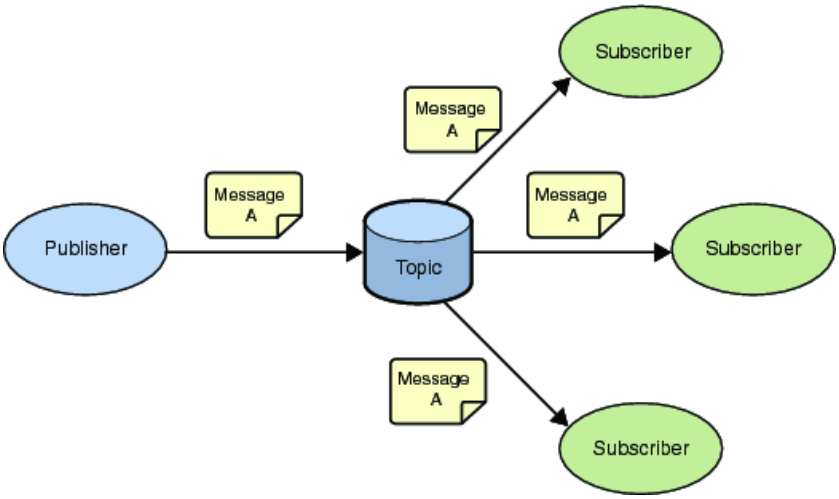

Figure 5.1: The Publish and Subscribe design pattern

APIs are usually provided by MOMs, to allow the development of specific messaging services.

5.1.1 The Publish and Subscribe design pattern

The publish and subscribe [22] messaging pattern is a design architecture in which the message sender, called publisher, doesn't send data to specific receivers, called subscribers; it just publishes messages into a queue, which are automatically delivered to clients which have declared their interest. Publisher and subscriber are decoupled, because they don't need to be connected at the same time, they don't need to know all the addresses of the other participants and they don't need to be synchronized. This pattern provides a dynamic network topology because a component can subscribe or create a new queue anytime.

5.2 The Java Message Service API

The Java Message Service API is a Java Message Oriented Middleware that provides the Publish and Subscribe design pattern. It is a part of the Java Enterprise Edition platform. Using JMS can be advantageous for the de-

velopment of distributed PerLa, because we expect that it will be used in wide area wireless sensor networks. Furthermore, thanks to the http-channel module, we can expect that some components of PerLa will be connected to the central server using the tcp/ip protocol from a long distance.
Due to asynchronous messaging implemented by JMS, it is not necessary that all the nodes of PerLa be up for the the whole application to work. The JMS server stores the messages on behalf of the receivers when they are down and then sends them once they are up.
The following JMS components are the classical components of a MOM.

- *JMS client*: an application or a process that sends or receives a message;
- *JMS message*: an object that contains data; it is a serializable Java Class;
- *JMS publisher*: a client that sends a JMS message;
- *JMS subscriber*: a client that receives a JMS message;
- *JMS queue*: a queue that contains messages that are waiting to be read; each message can be processed only once;
- *JMS topic*: a structure that allows messages delivering to multiple destinations.

5.3 PerLa and JMS

5.3.1 Mirror FPC

When a FPC is registered on a distributed Middleware of Perla, a Mirror FPC is registered in the FPC Registry of the central PerLa Middleware and is seen as an ordinary FPC. This Mirror FPC has the same attributes of the just registered FPC, so the query language is allowed to execute queries on the whole set of FPCs. Each Mirror FPC can distribute the queries to the corresponding FPC.

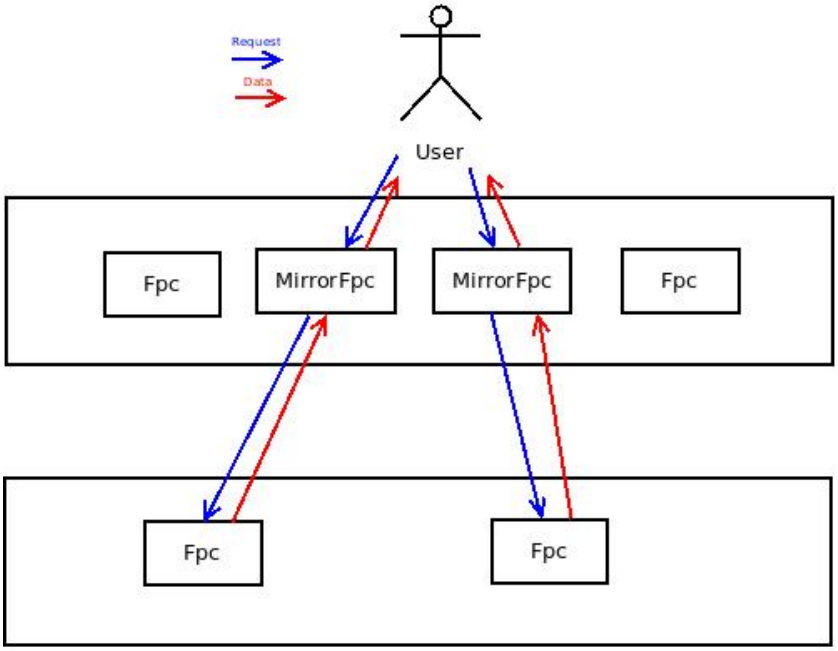

Figure 5.2: MirrorFpc

5.3.2 Types of message

-The "Added FPC Message": to create the Mirror FPCs, the server must receive the attributes of the FPCs connected to the nodes so each node must send a message for each of its own FPCs. When the server receives a "Added FPC Message" it initializes a new Mirror FPC and adds it to the registry.

```
1
2 public class AddFpcMessage implements Serializable{
3
4    private String nodeId;
5    private int fpcId;
6    private Collection<Attribute> attributes;
7
8  }
```

Listing 5.1: JMS class to add a FPC

-The "Get Message": the user's requests must be routed to the specific nodes. So the message service must be provided with a class that specifies details of requests to the node.

```
public class GetMessage implements Serializable{
   private String nodeId;
   private int fpcId;
   private boolean async;
   private boolean strict;
   private List<Attribute> attributes;
   private long periodMs;
   private String queue;


   }
```

Listing 5.2: JMS class for sending data request from the server to the nodes

If the request corresponds to the FPC's *async()* method, the attribute "async" is set to true by the constructor.
If the request is a periodic sampling operation, the attribute periodMs is initialized by the constructor, otherwise it is set to null.

-The "Data Message":when a node collects sampled data, it creates a "Data Message" containing it.

```
public class DataMessage implements Serializable{

   private Sample sample;

   public DataMessage (Sample sample){
      this.sample = sample;
   }
```

Listing 5.3: JMS class for sending collected data from the nodes to the server

5.3.3 An example

An application of the message service developed using JMS is shown below and in figure 5.3. The system is composed by:

- a metereological service;

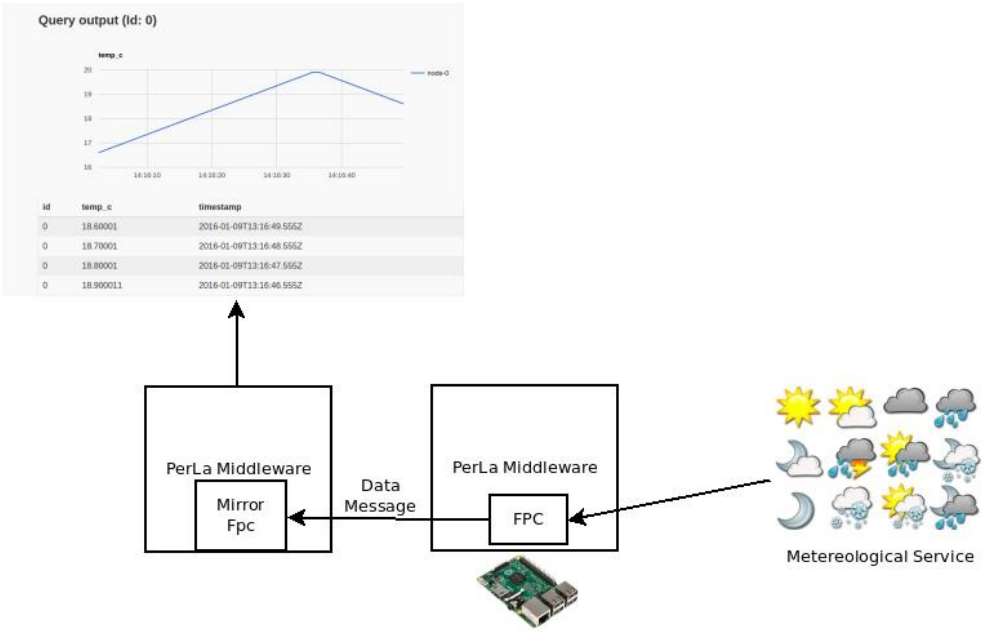

Figure 5.3: PerLa and Jms. An example

- a PerLa Middleware that runs over a Raspberry PI2, a single-board computer, connected to the network;
- a PerLa Middleware that runs over a server.

When the system starts, each PerLa Middleware launches a Message Consumer, that receives messages sent to the dedicated queue. Each Middleware has one or more dedicated queues.
The metereological service is described by an Xml descriptor. The supervised attributes are:

```
<attributes>
<attribute id="city" type="string" />
<attribute id="temp_k" type="float"/>
<attribute id="temp_c" type="float"/>
<attribute id="temp_f" type="float"/>
<attribute id="pressure" type="float"/>
<attribute id="humidity" type="float"/>
<attribute id="wind_speed" type="float"/>
<attribute id="wind_deg" type="float"/>
```

```
</attributes>
```

Listing 5.4: FPC: XML description

When the user sends a request to the MirrorFpc, the system initializes a GetMessage and sends it to the FPC located on the Raspberry.

```
GetMessage reqMess = new GetMessage(atts, strict, false,
        periodMs, nodeId, queue, this.fpcId);

serverMsgSender.sendGetMessage(reqMess);
```

Listing 5.5: Request sending

The FPC receives the request and starts a classical data collection. When data arrives from the weather service the TaskHandler calls the *data()* method, that sends a data message to the server.

```
public void data(Task task, Sample sample) {

    DataMessage dataMessage = new DataMessage(sample);
    aggregatorMsgSender.sendDataMessage(dataMessage);

    }
```

Listing 5.6: Sending Data

The Server Consumer collects data coming from the FPC and activates the *data()* method of the MirrorFpc, that sends it to the chosen application. In this case data is used to draw a graph.

Chapter 6

The PerLa Query Language

The PerLa Query System is an infrastructure that allows the user to submit a query to the Middleware and to receive responses, data and samplings from the devices involved in the query. It provides data aggregation clauses too. Required mathematical operations, as sums or mean, are carried out by the system. Queries are written in the PerLa Query Language [5], a declarative sql-like language. The basic idea is to abstract a sensing network as a table, whose columns correspond to a specific attribute. In this way, users don't have to know specific structures of nodes, so the system is "transparent". Information can be sampled from a group of sensors, from a memory of an endpoint device or extracted from a web service. PerLa Query is used to define the behaviour of a single or of a group of devices.

6.1 Query Structure

Each query is built following a fixed schema [6]:

- Data management: introduced by the SELECT clause, defines which data elements (Attributes) are to be collected from the Pervasive System. Then it indicates computations and aggregations that must be performed on the extracted information. Operations can be sum (SUM), counting (COUNT), maximum (MAX) , minimum (MIN) and average (AVG).

- Sampling: introduced by the SAMPLING clause, is used to specify how and when the data attributes requested by the SELECT statement are to be extracted from the nodes. Sampling can be time-based (ex. SAMPLING EVERY 1 MINUTES) or event based (a.g.: SAMPLING ON EVENT lastReaderChanged).

- Conditional Execution: introduced by the EXECUTE IF clause, contains a boolean expression that must be satisfied by sensing devices in order to be considered as a data source. It can be complemented by a REFRESH clause, to specify when the execution condition is re-evaluated to update the list of nodes involved in the query.

- Termination: introduced by TERMINATE AFTER clause. Can be used to stop a query after a specific time frame (TERMINATE AFTER 1 DAY) or after a number of selection (TERMINATE AFTER 10 SELECTIONS).

6.2 Query Example

```
1 CREATE OUTPUT STREAM Table (Temperature FLOAT) AS:
2 EVERY 5 MINUTES
3 SELECT MAX (TEMP, 10 MINUTES)
4 SAMPLING
5   EVERY 1 MINUTES
6 EXECUTE IF EXISTS (temp )AND EXISTS (room) AND room=3
```

Listing 6.1: A PerLa Query

This query initiates a temperature sampling operation on all temperature sensors located in room number 3.

- The SAMPLING clause specifies that devices send a FLOAT every minute. This sample is collected in the Local Buffer;

- The EVERY 5 MINUTES clause specifies that the SELECT statement is activated every five minutes, to create a new record;

- The MAX aggregation expression specifies that the query contains the maximum temperature collected in the previous 10 minutes.

6.3 Query Distribution

6.3.1 Intermediate Nodes

An intermediate node is usually a single-board computer equipped with a hardware configuration that allows network connection. A complete PerLa system runs on each Intermediate node.
In order to distribute calculation and data collection, query distribution is required. This means that intermediate-nodes must have enough computing power to compute some mathematical operations and data aggregation. Queries must be parsed to recognise which part of them can be executed on nodes and how many devices are involved for each node. The communication is Middleware-to-Middleware, because each intermediate-node hosts PerLa Middleware. This structure allows to build a tree with a lot of levels, because each node can be server and client at the same time.

6.3.2 The Registry

The Registry [6] is a simple in-memory database that stores FPC objects so it is a complete directory of all data sources connected to the PerLa Middleware. It is primarily employed for the discovery of sensing devices registered in a running PerLa instance, and its services are extensively exploited by the Query Executor component for the management of EXECUTE IF statements because, by means of the Registry, users can examine all nodes of the network, and select those that best suit their current computational needs. The Registry is the key-component to exploit Query Distribution.

6.3.3 Query distribution

The decision about participation of a node to a query has been one of the most important issues in PerLa design since the starting phases. Currently,

a PerLa query is parsed and distributed to FPCs that can satisfy it. In particular, the clause EXECUTE IF EXISTS (temp)AND EXISTS (room) AND room=3 is parsed and assigned to FPCs that have "temp" and "room" among their attributes and the "room" set with "3" value. The clause SELECT MAX (TEMP, 10 MINUTES) selects the maximum value of attribute "temp" sampled in last 10 minutes. These operations are pretty simple, because they work on a single data flow and all nodes send their data to a single server that hosts all the FPCs. In the following table we can see a possible set of records.

ID	Temp	Timestamp
34	21.0	12/09/2015@12:00
35	21.2	12/09/2015@12:00
36	21.1	12/09/2015@12:00
34	21.1	12/09/2015@12:01
35	21.0	12/09/2015@12:01
36	20.9	12/09/2015@12:01
34	21.2	12/09/2015@12:02
35	21.2	12/09/2015@12:02
36	21.1	12/09/2015@12:02

6.4 Issues in distributed PerLa

The first problem in developing a distributed version of PerLa is to retrieve devices and services involved in a query. An interrogation must be addressed only to the intermediate nodes linked to devices that can satisfy the requirements. Figure 6.2 shows a system composed by a central server, three intermediate nodes and a device directly connected to the server. All nodes and devices are in room 3. We can speculate about system behaviour studying the following query:

```
1 CREATE OUTPUT STREAM Table (Temperature FLOAT) AS:
2 EVERY 5 MINUTES
3 SELECT MAX (TEMP, 10 MINUTES)
4 SAMPLING
5   EVERY 1 MINUTES
```

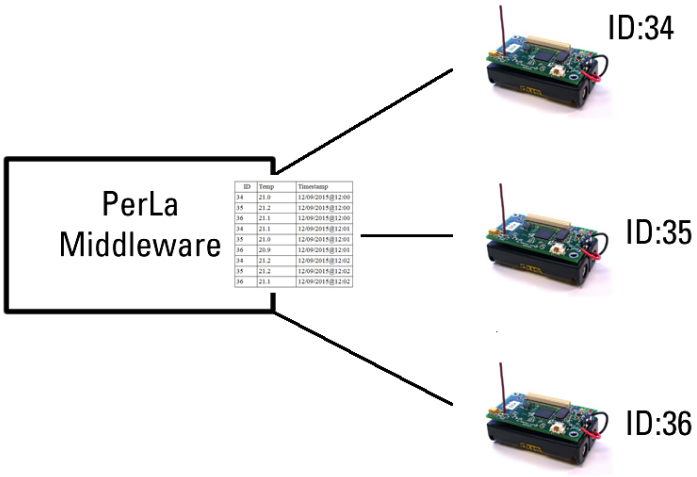

Figure 6.1: Data flow in non-distributed PerLa

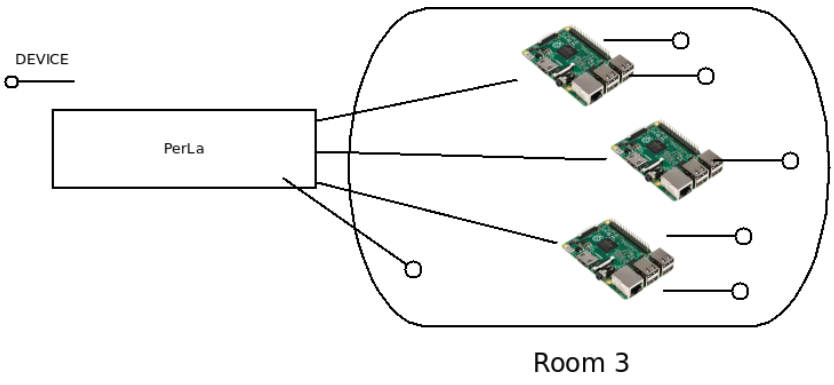

Figure 6.2: Room3

```
6 EXECUTE IF EXISTS (temp )AND EXISTS (room) AND room=3
```

6.4.1 Distributed data collection in a pervasive system

A sensing network managed by PerLa is abstracted as a large table in a streaming database, so we can apply to Distributed PerLa the principles used to manage the built-in functions of a distributed database [17]. Every

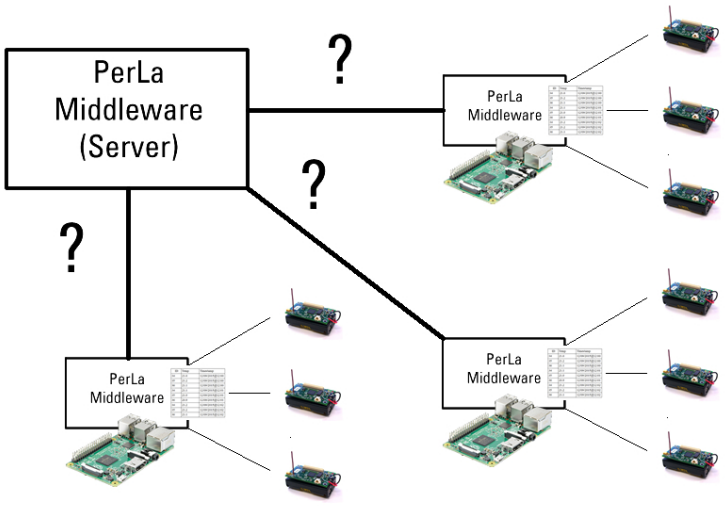

Figure 6.3: Data flow in distributed PerLa

distributed query works in two phases:

- Obtain required data from the system. Each intermediate node collects data, executes the query and sends the server the result of the execution. From the point of view of the server, the result of a distributed query is a number of fragments satisfying the qualification part of the query.

- Apply to the obtained data the required operators. Built-in operators in PerLa are the same commonly found in a Distributed Information System. They are the following:

    1. max-of(X): it computes the maximum element of X. In the PerLa query language the relative clause is SELECT MAX().

2. min-of(X): it computes the minimum element of X. In the PerLa query language the relative clause is SELECT MIN().
3. sum-of(X): it computes the sum of the element of X. In the PerLa query language the relative clause is SELECT SUM().
4. count-of(X): it computes the cardinality of X. In the PerLa query language the relative clause is SELECT COUNT().
5. average-of(X): it computes the mean of the element of X. In the PerLa query language the relative clause is SELECT AVG().

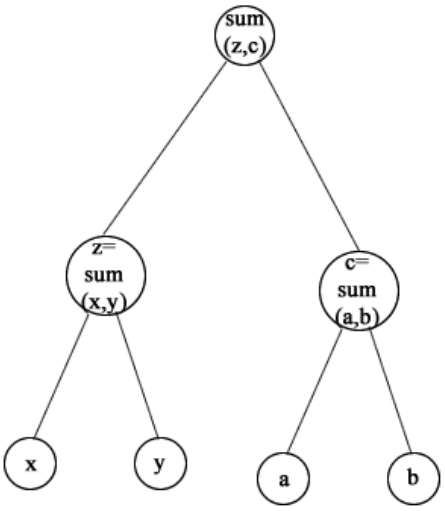

Figure 6.4: Decomposition of "sum" function

Max-of, min-of, sum-of and count of are Homogeneously decomposable functions [17] so they can be simply decomposed. Just as an example, Sum-of can be decomposed as follows:

$$sumof(X \cup Y) = sumof(sumof(X), sumof(Y)$$

The function average-of is a Non-Homogeneoysly Decomposable Function [17] so to decompose it a computing effort is required. When the server

receives data, it has to calculate the weighted average of the partial averages. The node must send the partial average and the weight to allow the calculation.

$$WeightedAverage = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}$$

$$averageof = (X \cup Y) = \frac{(sumof(averageof(X)sumof(X), averageof(Y)sumof(Y))}{sumof(countof(X), countof(Y))}$$

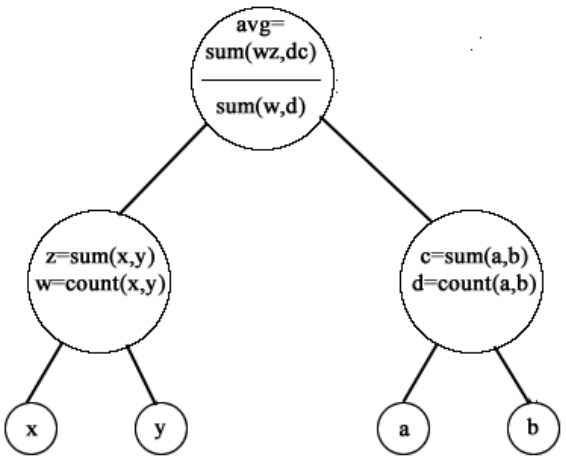

Figure 6.5: Decomposition of "average" function

6.5 Query distribution development

In the previous section we explained the query distribution from a logical point of view. To implement the query distribution in PerLa a new component has been introduced. The software schema is shown in figure 6.6.

1. The Query Parser contains the PerLa Language definition. The Language component receives a PerLa Query and creates a Statement, that is a structure that contains

-the attributes requested by the query;

-the data aggregations (max, min, avg ecc) requested by the query.

2. The Statement is parsed by the Query Parser, that reads the FPC Registry and obtains the list of FPCs involved in the query.

3. The Query Parser matches the FPC list with the Aggregator Registry. This registry contains the list of the aggregators. Each aggregator is coupled with a JMS queue. A message directed to an aggregator must be sent to its queue.

4. If a significative number of the FPCs hosted by an aggregator are involved in the query, the Message Service sends the queue to the aggregator.

5. If a minor part of the FPCs hosted by an aggregator is involved, the communication structure is shown in the chapter 5.

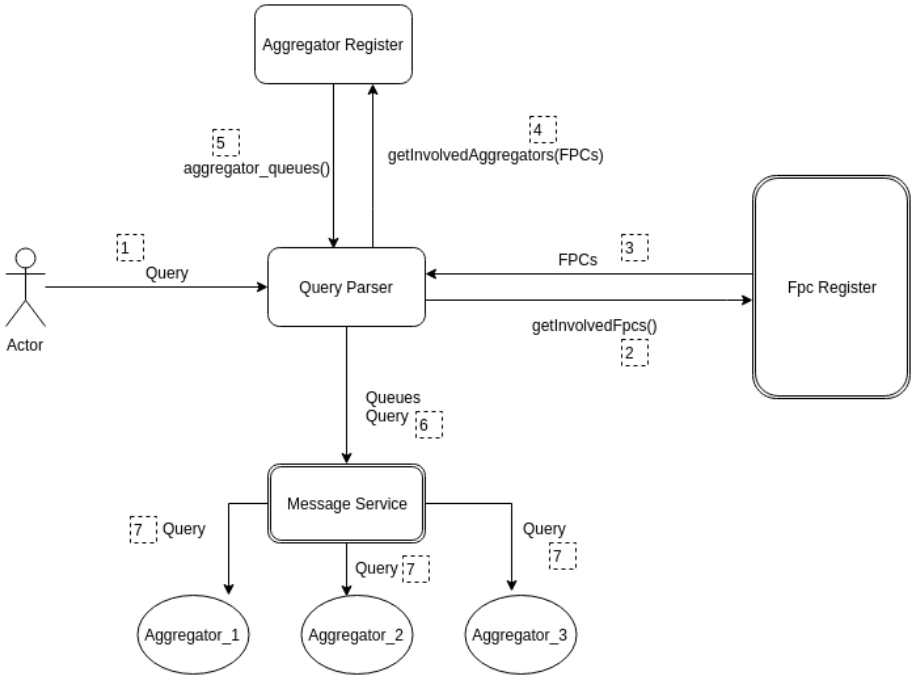

Figure 6.6: Query Distribution

6.6 Data collection

To collect data arriving from the nodes, we must consider the properties of the network and we must know some information about the aggregators that received the query. Just as an example, to calculate a weighted average, we must know how many FPCs hosted by an aggregator are involved in the query. (figure 6.5)
To collect data in the right way, the "Data collector" component has been introduced. Furthermore, the aggregators send data using the Query Data Message class, that is the implementation of a JMS message. In the figure 6.7 the flow of operations is shown.

1. The aggregators send the server queue (implemented using JMS) the messages that contain the data requested by the query and all the information useful to data aggregation;
2. The message service sends the information to the Data collector, that aggregates data;
3. Data can be used by another PerLa component. They can be stored in a database using the PerLa database component or used in a context oriented application [4].

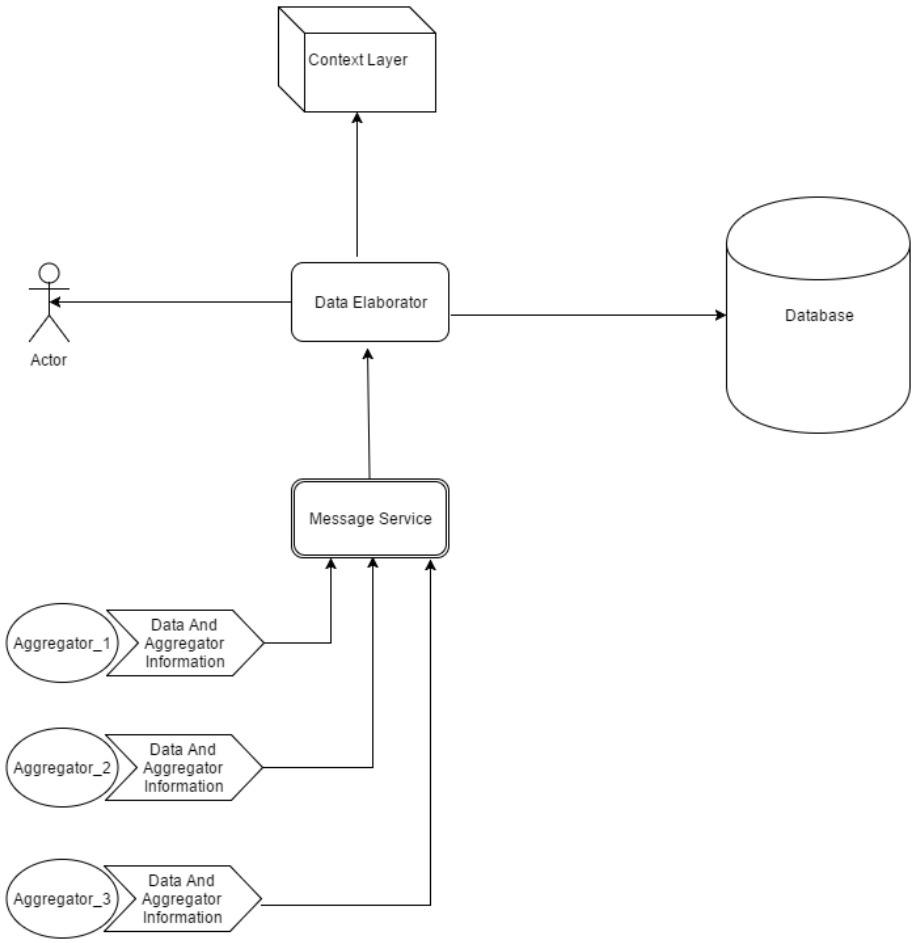

Figure 6.7: Data Collection

Chapter 7

The Prototype

In the first days of 2017, a hotel has been hit by an avalance in Abruzzo, an Italian region, causing the death of 29 people. Tecnhical findings stated that the incident was triggered by a series of earthquakes in central Italy, in conjunction with the raise of the atmospheric temperature that melted the snow. Is there a way to avoid these type of tragedies?

The scope of this project is the development of a monitoring system that could help to prevent such events in future. It relies on three architectural components:

- Hardware: sensors and boards (STM32 Nucleo and Raspberry PI2);
- Software: the PerLa System;
- Data sources: the INGV Rss Feeds that publishes information about last earthquakes.

7.1 ST MicroElectronics Sensors

ST MicroElectronics developed the STM32 Nucleo boards. These high affordable boards allow anyone to try out new ideas and to quickly create prototypes with any STM32 MCU. The most important feature developed by ST Micro is the very easy way of use of the board, that can be programmed with a C-Like language.

Our prototype is composed of:

- A Nucleo 64. The basic board provided of an ARM Cortex CPU;
- A Nucleo Expansion Board provided of inertial, barometer, humidity and temperature module;
- A Wi-Fi module.

7.2 Rss Feed

Rss, Rich Site Summary, is a XML standard used to publish asynchronous updated information from web sites and data sources. Usually RSS is used by newspapers but it can be used for all types of frequently updated information. The INGV (Istituto nazionale di geofisica e vulcanologia) research group uses a RSS feed to publish information about earthquakes that occur in Italy. Our project is based on "open source" Rss Feeds that could be replaced by "ad hoc" sources because PerLa can work with all types of structured data in an etherogenous environment.

7.3 The Context

Figure 7.1 shows the context, modeled as a Context Dimension Tree (CDT) used in the prototype; black circles represent the context dimensions, white circles represent their possible values, square nodes represent dimension attributes.

7.4 The Prototype

The prototype that we are showing is based upon two queries written in PerLa.

The first query collects temperature, humidity and position of all the ST electronics sensors connected to the system.

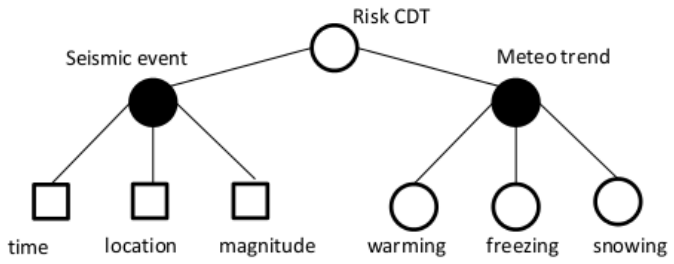

Figure 7.1: Context Dimension Tree

```
CREATE OUTPUT STREAM st_el(temperature,humidity,axeX,axeY,axeZ)

SAMPLING EVERY 5 MINUTES

EXECUTE IF EXISTS temperature,humidity,axeX,axeY,axeZ''
```

Listing 7.1: Query Prototype 1

The second one queries the RSS feed and collects the earthquakes data made available by the INGV institute

```
CREATE OUTPUT STREAM feed()

SAMPLING EVERY 5 MINUTES
```

Listing 7.2: Query Prototype 2

The collected information is saved in a MySql database.
We built a prototype composed by a WiFi network composed by:

- Two Raspberry Pi2;
- One server;
- Two St Electronics Systems, as previously described.

The PerLa Middleware is installed on the server and on the Raspberry boards. When query 1 is started, the system identifies the nodes that can provide temperature, humidity and position, and trasmits the instructions to each

involved node. The ST sensors send data to the Raspberry boards, that are intermediate PerLa node. The boards then send the data to the PerLa central node.

Since the user is not involved in the communication with the sensors, the system can be further expanded with the same functionality.

7.5 Development of the prototype

7.5.1 Description

The System receives data from two types of sources:

- RSS feed (Ingv and weather forecast);
- Physical devices (St Microelectronic Nucleo Boards).

RSS Feeds are used for:

- Data storage and statistical analysis. For example, ground vibrations in a specific geographic point can be correlated with seismic activities occurred in an other geographic zone.
- Context activation. For example, if the weather forecast expects a temperature increase during seismic activities, the system can forecast a snowslide activating the context "Avalanche".

```
CREATE CONTEXT avalanche_danger
ACTIVE IF seism_event_mag      3 AND meteo_trend = warming OR
meteo_trend = snowing
ON ENABLE:
EVERY 5min SELECT max_temp, max_vibration
SET alarm
SAMPLING EVERY 1min
ON DISABLE:
EVERY 60min SELECT temp, vibration
STOP alarm
SAMPLING EVERY 15min
EXECUTE IF EXIST termometer OR accelerometer OR alarm
```

```
13 REFRESH EVERY 30MIN
```

Listing 7.3: Avalanche Context Creation

Physical devices are used for monitoring the real situation of a critical zone and collect environmental metrics to react in real time to a dangerous situation. Crossing environmental data and RSS feeds will help to design a strong prevention system.

To accomplish the project, two FPC structures have been developed.

7.5.2 Rss FPC

The following program listing shows the RSS feed structure provided by the INGV. RSS feed are asynchronous sources of data and they can be managed by PerLa.

```
1 <entry>
2 <id>
3    smi:webservices.ingv.it/fdsnws/event/1/query?eventId=14955981
4    </id>
5    <title>
6       <![CDATA[ 2017-04-24 07:41:51 UTC -
7          Magnitude(ML) 2.1 - L'Aquila ]]>
8    </title>
9    <updated>
10      2017-04-24T08:09:37+00:00
11   </updated>
12   <link rel="alternate" type="text/html"
13         href="http://cnt.rm.ingv.it/event/14955981"/>
14   <summary>
15      <![CDATA[ A magnitude ML 2.1 earthquake occured
16            in the region: L'Aquila on 2017-04-24 07:41:51 UTC ]]>
17   </summary>
18   <dc:date xmlns:dc="http://purl.org/dc/elements/1.1/">
19      2017-04-24T07:41:51+00:00
20   </dc:date>
21   <geo:Point xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#
     ">
22      <geo:lat>
23         42.4583
```

```
24        </geo:lat>
25        <geo:long>
26            13.3083
27        </geo:long>
28    </geo:Point>
29    <georss:point>
30        42.4583 13.3083
31    </georss:point>
32
33 </entry>
```

Listing 7.4: Invg RSS

To read RSS feeds, a lot of Java libraries are provided. Java Rome is one of the most robust implementation and ease to use library. In order to manage the feeds, a PerLa FPC must be defined.

The following listing shows the basic structure of the RssFPC class:

```
1   public    class RssFpc implements FPC {
2
3   private static String rssFeedUrl;
4   private String magnitude;
5   private String latitude;
6   private String longitude;
7   private String toponym;
8
9   //The constructor
10
11  public RssFpc(String rssFeedUrl){
12
13    this.rssFeedUrl = rssFeedUrl;
14
15    //SyndFeedInput and SyndFeed are the classes
16    //provided by Java Rome
17
18    SyndFeedInput input = new SyndFeedInput();
19    SyndFeed feed = input.build(new XmlReader(rssFeedUrl));
20
21 }
```

Listing 7.5: RssFpc

7.5.3 Socket FPC

The ST WiFi Nucleo Board can use sockets to provide TCP connections. In order to communicate with the devices we have implemented the Socket FPC class, that uses the socket infrastructure provided by the Java standard library. A Socket FPC is the abstraction of a physical device, thus an attribute list is needed to create it. When the system receives an XML descriptor from a board, it initializes a SocketFPC class.

```
1  private    ArrayList<Attribute> AttributeList = new ArrayList<
     Attribute>();
2
3  public SocketFpc(String XMLDescriptor){
4
5  AttributeList = createAttributeList(XMLDescriptor);
6
7  }
```

Listing 7.6: Socket FPC

7.5.4 Board Programming

The core of the board action is the following code listing. During the "while" cycle, the board retrieves data from the sensor modules and sends it to the PerLa system.

```
1  while  (1) {
2    printf("\r\n");
3
4    /* The board requests temperature and humidity */
5    temp_sensor1->GetTemperature(&value1);
6    humidity_sensor->GetHumidity(&value2);
7
8    /* The wifi module sends two array to the socket
9   connection */
10   printf("HTS221: [temp] %7s C ,    [hum] %s%%\r\n",
11            printDouble(buffer1, value1),
12            printDouble(buffer2, value2));
13
```

```
14      printf("---\r\n");
15
16      /* The board request the accelerometer data */
17      accelerometer->Get_X_Axes(axes);
18
19       /* The wifi module sends the phisycal data to
20    the socket connection */
21      printf("LSM6DS0 [acc/mg]:       %6ld, %6ld, %6ld\r\n",
22              axes[0], axes[1], axes[2]);
23
24      /* The requests are sent every 5 seconds */
25      wait(5);
26    }
```

Listing 7.7: Board Programming

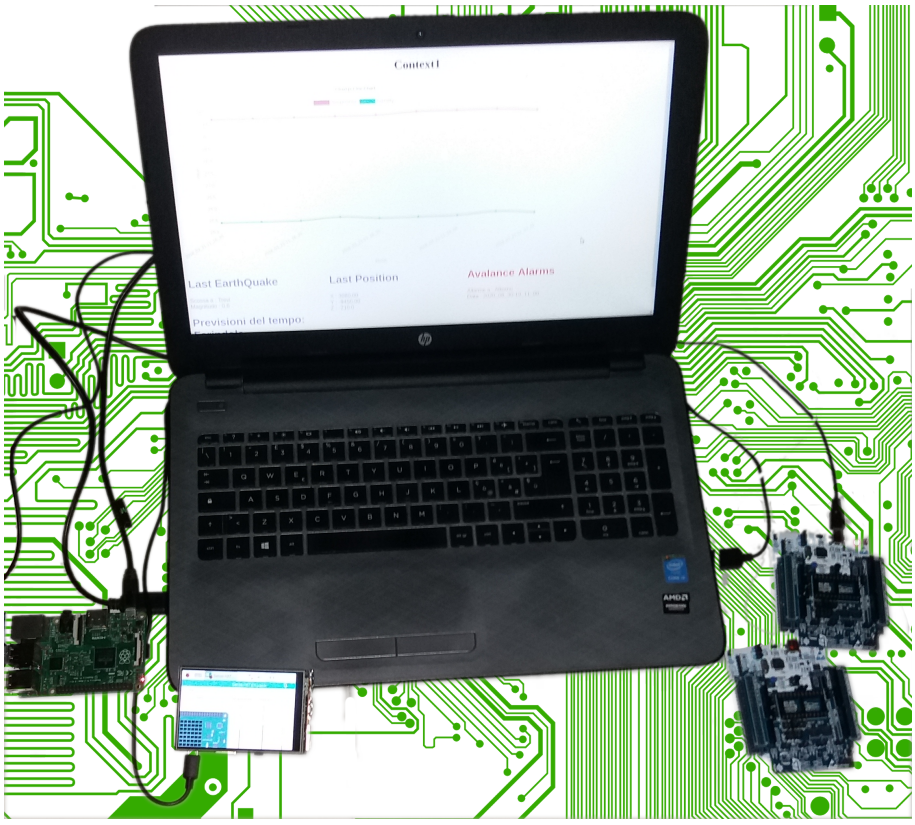

Figure 7.2: The Prototype

Chapter 8

Conclusion

In this thesis we presented the possibility of using PerLa in a real scenario, showing the possibility of integrating PerLa Middleware with a series of products of different nature.
In particular we have shown the possibility of integration with relational databases, such as MySql, in order to make it compatible with as many past and future uses as possible. We have shown also the possibility of integrating non-relational NoSql databases, such as Cassandra, in order to make PerLa usable with the most modern systems and to use the great possibilities provided by them. We have also expanded the system, making it distributed, through the use of a message service that exploits the publish and subscribe paradigm.
Finally, we applied these developments to the creation of a prototype, which uses electronic products available on the market. In particular, through the use of wireless networks, Raspberry PI cards, sensors produced by ST Micro-Electronics and public RSS feeds, we have integrated data flows of different nature to create an avalanche monitoring and prevention system, inserted in a management context of emergencies.
The most stimulating moment of the work was the successful attempt to develop new modules, starting from the work of Guido Rota, adapting the system to a practical scenario.
The developed system is complete and further expandable, usable not only in

restricted scenarios, but also in geographically wide scenarios, thanks to the possibility of distributing the middleware on nodes consisting of low energy consumption minicomputers.
In this way we have created a system that can be used in advanced pervasive systems and within the Internet of Things.

Appendix A

Comparison between PerLa and other systems

Wireless Sensor Networks (WSNs) are networks of wirelessly connected devices called nodes, that are able to measure or detect physical properties (such as temperature, pressure, humidity...) from their surrounding environment. The current usage of the term WSN includes also devices with actuation capabilities.

A WSN can be a variegated software and hardware environment, so a framework or a middleware, to provide an easy way of access to the functionalities of all components of the network is needed.

There are two characteristics that a system should have in order to be considered as pervasive, according to the definition given by Mark Weiser: physical integration, that is the ability of hardware and software to hide themselves into everyday life objects, and spontaneous interoperability, that is the devices ability to communicate whenever required by the context.

To manage pervasive systems, the idea of a high level language to hide the user the whole complexity of the underlying system has been conceived and a lot of project attempted to realise it.

A.1 Data Stream Management Systems and "WSN as a database"

"A data stream is a sequence of digitally encoded coherent signals (packets of data or data packets) used to transmit or receive information that is in the process of being transmitted". Today data streams are used in wireless sensor networks and Internet of Things. The increasing diffusion of these infrastructures, made the creation of Data Stream Management Systems model necessary. The aim of a DSMS is to process and store unbounded data streams. An important issue in applying the DSMS model is the creation of a persistent storage to save all relevant data. To achieve this objective, the concept of data stream query language has been developed, creating the concept of "WSN as a database". It means that a sensor network can be queried as a database. The result of a query is the sampling stream coming from the devices.

A.1.1 TinyDB

TinyDB [25] is one of the first projects that tried to provide a generic abstraction suitable for the development of sensing applications based on Pervasive Systems. A sensor network managed by TinyDB is presented to the final users as a streaming database controlled through SQL-like queries. Limits: TinyDB requires all nodes in the network running TinyOS, so the variety of devices which can be managed is limited. Furthermore, if the user is concerned in high speed databases and high performances, TinyDB is the wrong choice. PerLa exceeds TinyDB, because the middleware plugin system allows to manage nodes running TinyOS in a transparent way and in a variegate environment, together with nodes that run other OS. PerLa can also interact with high performance databases and can reach high performances.

TinyREST

TinyREST provides a RESTful data access interface to each node available in the pervasive system. An HTTP GET request is used to sample a phys-

ical phenomena and a PUT operation is used to command actuators or set variables on remote devices. TinyREST requires all nodes running TinyOS, so it is vulnerable to the same critiques made to TinyDB.

A.1.2 Global Sensor Network

Global Sensor Network's [26] central concept is the virtual sensor abstraction which enables the user to declaratively specify XML-based deployment descriptors in combination with the possibility to integrate sensor network data through plain SQL queries over local and remote sensor data sources. GSN is a Java Middleware based on the concept of Virtual Sensor, a high-level component (similar to the PerLa FPC) aimed at providing a common interface that corresponds to a physical sensor and allows to interact with it. Virtual Sensors are created by means of a declarative XML descriptor. Limits. GSN doesn't provide a system for the automatic creation of virtual sensors. PerLa exceeds GSN because when a device is connected to the middleware, the FPC creation is automatic and the user doesn't care about it.

A.1.3 Contiki

Contiki [27] is an advanced operating system for the Internet of Things. It is designed for memory-limited and low power consuming devices. Contiki uses TCP/IP and a limited number of wireless standards. PerLa exceeds Contiki because it is designed not only for limited devices, but it has been conceived for all devices that can collect and send data using all types of communication protocol.

8.2 Tesla and PerLa comparison

8.2.1 T-Rex and Tesla

Distributed applications systems require huge amounts of memory. It is appropriate to imagine that some of these systems could be used also with

processing data as a stream of data, and not necessarily storing them. As an example, a system can operate by observing the happening of primitive events, combining them to create composite events and finally sending the notification to the components in charge of reacting to them. Complex Event Processing (CEP) middlewares, are responsible for managing events got from sources. CEPs operate interpreting sets of event definition rules, that describe how complex events are defined from simple ones.
T-rex is a Cep middleware which works with a language named Tesla [28]. Tesla allows to express all the constructs that are needed by a CEP system, as content-based event filtering, customizable event selection and consumption policies, Event sequences with timing constraints, negations (time-based and interval-based), parameters and aggregates. In Tesla it is assumed that events occur instantaneously at some points in time, after happening, they are encoded by sources into event notifications. Each event notification has a type and a timestamp.

8.2.2 Tesla language vs PerLa language

Tesla and PerLa must be compared holding the following preconditions:

- For basic requests, PerLa and Tesla are interchangeable;
- Tesla doesn't support static data;
- Tesla doesn't support the current time sampling;
- PerLa requires the construction of at least two queries for each request. A query must be written in PerLa language and the second query must be written in the database language relative to the DBMS in which data is stored.

Example 1. A similar behaviour

Environment: car rental company.
Request: find cars that have been used without rental contract.
Condition: Speed>0 when (Current Date)>(Last Rental Release Date)

PerLa query: selects the speed of all cars that are moving. The query samples on the specific event "Speed>0".

```
CREATE OUTPUT STREAM Theft_Table (int idCar, date Released, date
    CurrentDate) AS:

EVERY 5 MINUTES

SELECT Speed

SAMPLING

ON EVENT Speed>0

EXECUTE IF EXISTS Speed AND IF EXISTS Released AND IF EXISTS
    CurrentDate AND IF (CurrentDate>Released)
```

Listing 8.1: PerLa Application 1

High level (database) query: selects all the cars in Theft Table.

```
SELECT idCar FROM Theft_Table
Tesla query:
define Theft (ID : String )
from VehicleData (idCar = $id and speed > 0) and
lastRelease (idCar = $id ) within 10day from VehicleData and
not Taken (idCar = $id ) between Release and VehicleData
where ID = VehicleData.idCar
```

Listing 8.2: PerLa Application 2

Comment: in this case, the query complexity is quite similar, but Tesla requires to put a value in the clause "within". In this case the value "10 day" makes data sampled before 10 days unavailable.

Example 2. A request that can't be satisfied by Tesla

Environment: car rental company. Test if the estimated travel time to release point is slightly below than remaining time from reservation expira-

tion.
We need to know:

- travel time to release point
- reservation expiration time.

Sampling attribute: current data.
Comment: Tesla can't satisfy this request, because it doesn't provide the current time. On the contrary, PerLa can achieve the objective.

Example 3. Another request that can't be satisfied by Tesla

Environment: car rental company. Say if the vehicle had stopped in a place where it couldn't (car in a no-parking zone, for example). **We need to know**:

- the vehicle's position
- the roads network

We also need a table that contains the list of "legal" positions.
Comment: Tesla can't satisfy this request, because it doesn't allow static data. PerLa, on the contrary, allows static data.
Conclusion: It appears that PerLa is more powerful than Tesla [29] in many applications. However, it should be noted that PerLa needs two queries in each execution. This means that PerLa requires more programming effort, while Tesla has a more immediate use.

Bibliography

[1] Mark Weiser, *The Computer for the 21st Century*, Scientific American (265:94-104), 1991.

[2] F.A. Schreiber, L. Tanca et al., *Data Management in Pervasive Systems*, Springer, 2015

[3] P. N. Mahalle, P. S. Dhotre., *Context-Aware Pervasive System and Applications*, Springer, 2020

[4] R. Sabatino,*Context Management in a Pervasive System*, Master Thesis Politecnico di Milano, 2015.

[5] M.Fortunato, M.Marelli, *Design of a declarative language for pervasive systems*, Master Thesis Politecnico di Milano, 2006/2007.

[6] G.Rota,*Design and development of an asynchronous data access middleware for Pervasive Networks: the case of PerLa*, Master Thesis Politecnico di Milano, 2013.

[7] Y.Zhao, G. Yu, H. Xu, *6G Mobile Communication Network: Vision, Challenges and Key Technologies*, 10.1360/N112019-00033, 2019.

[8] R. Want, *An Introduction to RFID Technology*, Pervasive Computing (IEEE) january, 2006.

[9] F.A.Schreiber, R.Camplani, G.Rota, *Extracting Data from WSNs: A Data-Oriented Approach*, 2012

[10] T.Wark, P.Corke et al. *Transforming Agriculture through Pervasive Wireless Sensor Networks*, Pervasive Computing (IEEE) aprile-june, 2007.

[11] M. Erdelj, E.Natalizio et al., *Help from the Sky:Leveraging UAVs for Disaster Management*, Pervasive Computing (IEEE) january-march, 2017.

[12] Sensor ML Documentation, http://sensorml.com/

[13] M. Tamer Ozsu, Patrick Valduriez, *Principles of Distributed Database Systems*, Springer, 2010.

[14] Oracle Documentation, *Database Administrator's Guide*, Part V, https://docs.oracle.com/cd/E1188201/server.112/e25494.pdf, 2015

[15] Dan Sullivan, *NoSQL for Mere Mortals*, Pearson-Addison-Wesley Professional, April 2015.

[16] Google Cloud official Blog. *Cassandra Hits One Million Writes Per Second on Google Compute Engine*,https://cloudplatform.googleblog.com/2014/03/cassandra-hits-one-million-writes-per-second-on-google-compute-engine.html, 2014

[17] F.Crivellari, F.Dalla Libera, S.Frasson and F.A.Schreiber, *Computation of statistical functions in distributed information systems*, Information Systems, Vol.8, No 4 (303-308), 1982.

[18] Kenneth P. Birman, Thomas A. Joseph, *Exploiting Virtual Synchrony in Distributed Systems*, SOSP '87 Proceedings of the eleventh ACM Symposium on Operating systems principles (123-138), 1987.

[19] G. Banavar, T. Chandra, R. Strom, D. Sturman *A Case for Message Oriented Middleware*, Proceedings of the 13th International Symposium on Distributed Computing (1-18), 1999.

[20] Edward Curry, *"Message-Oriented Middleware"* Chapter in Middleware for Communications (1-28), John Wiley and Sons, 2004.

[21] Edgard Codd, *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, Volume 13 Issue 6 (377-387), 1970.

[22] P.Eugster, P.Felber, R.Guerraoui, A.Kerrmarrec, *The Many Faces of Publish/Subscribe* ACM Computing Surveys (CSUR) Volume 35 Issue 2 (114-131), 2003.

[23] C.Cappiello, F.A. Schreiber, *Experiments and analysis of quality-and energy-aware data aggregation approaches in WSNs* 10th International Workshop on Quality in Databases QDB 2012 (Co-located with VLDB 2012) (1-8), 2012.

[24] F.A. Schreiber, R. Camplani, M. Fortunato, M. Marelli, G. Rota, *PerLa: A Language and Middleware Architecture for Data Management and Integration in Pervasive Information Systems*, IEEE Transactions on Software Engineering http://doi.ieeecomputersociety.org/10.1109/TSE.2011.25, 2011.

[25] S.R.Madden, M.J. Franklin, J.M.Hellerstein, W.Hong, *TinyDB: an acquisitional query processing system for sensor networks*, ACM Transactions on Database SystemsMarch, https://doi.org/10.1145/1061318.1061322, 2005

[26] J. Eberle, J.P. Calbimonte, S. Sarni, K. Aberer, *Global Sensor Networks: an evolving middleware for sensor data stream processing* Presentation at the Global Fair and Workshop on Long-Term Observatories of Mountain Social-Ecological Systems, University of Nevada at Reno, USA, July (16-19), 2014.

[27] A. Dunkels, B. Gronvall, T. Voigt, *Contiki a lightweight and flexible operating system for tiny networked sensors*, Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (455–462), 2004.

[28] G. Cugola, A. Margara, *TESLA: a Formally Defined Event Specification Language*, In Proceedings of 4th ACM International Conference On Dis-

tributed Event-Based Systems (DEBS 2010). Cambridge, United Kingdom July (12-15), 2010

[29] S. Crotti, *Comparison and Evaluation of Some Stream Processing Language*, Relazione di Progetto di Ingegneria informatica