Towards autonomic pervasive systems: the PerLa context language

Fabio A. Schreiber
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5
20133 Milano
schreibe@elet.polimi.it

Letizia Tanca
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5
20133 Milano
tanca@elet.polimi.it

Romolo Camplani
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5
20133 Milano
camplani@elet.polimi.it

Diego Viganó
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5
20133 Milano
diego.vigano@mail.polimi.it

ABSTRACT

The property of context-awareness, inherent to a Pervasive System, requires a clear definition of context and of how the context parameter values must be extracted from the real world. Since often the same variables are common to the operational system and to the context it operates into, the usage of the same language to manage both the application and the context can lead to substantial savings in application development time and costs. In this paper we propose a context-management extension to the PerLa language and middleware that allows for declarative gathering of context data from the environment, feeding this data to the internal context model and, once a context is active, acting on the relevant resources of the pervasive system, according to the chosen contextual policy.

Categories and Subject Descriptors

H.2 [**Database Management**]: Languages—*Query Languages*; H.3.4 [**Systems And Software**]; C.2 [**Computer Communication Networks**]: Distributed Systems

General Terms

Languages

Keywords

Pervasive system, context-awareness, hybrid intelligence, context management.

NetDB'11, June 12, 2011, Athens, Greece.

1. INTRODUCTION

Context-awareness is a property inherent to an autonomic Pervasive System and requires a clear definition of what context is and how the context parameter values can be extracted from the real world. Context can in fact be thought of as "any information that can be used to characterise the situation of an entity" [10]. Pervasive Systems deploy devices which are spatially distributed and possibly mobile, in order to monitor different kinds of physical phenomena for application support; however, some of these same data can also be used to detect the situation of the entities present in the environment, that is, to characterise the context and to provide the actual values for the context parameters. Indeed, while context variables could be considered just as all the other system variables (*flat* view), since many years they have been acknowledged a special role in determining the system behaviour [6]. Only if the network, or more correctly the language handling it, is able to manage the context the sensors are related with, the relationship between context and data can be correctly mastered. The literature presents different approaches to context management in pervasive systems that will be presented in Section 2.

Differently from most of existing approaches, we separate the Operational Pervasive System and its Context Management System in two different layers, while embedding in the same language the functionalities of both: context is analysed in terms of *observable* entities, which have some *symbolic* representation within the system and some of which correspond to *numerical* values gathered from the environment sensors. Gathering context data from the environment requires a simple interface, possibly based on a declarative approach [16, 21], which, on one side, interacts with the network of highly heterogeneous physical devices and, on the other, is correctly interfaced with the internal, symbolic representation of context, based on a context model. A context-aware database design methodology, a set of techniques and tools for context-aware data querying have been developed within the Context-ADDICT project [5]. According to this approach, the internal *symbolic observables* are modelled as

context dimensions and organised as a *Context Dimension Tree* [6], or CDT for short. Some of these values (like the user's role or current interest topic) are gathered from previous interaction with the user, while some others can be directly "read" as numerical values from the environment. Examples of these values are the time of day or the GPS coordinates and we call them *numerical observables*. Numerical observables are gathered from the environment by means of the declarative sensor query language PerLa[1] [21, 22]. The fact that PerLa is declarative allows a very easy integration between the activity of querying numeric observables and that of querying other observables like the role of the system user or the current step of the workflow process. In order to explain the concepts exposed in this paper we shall use a running example where a wine production process is to be monitored. The attention will be focused on the growth and delivery phase of the produced wine, as well as on some of the actors (farmer, oenologist and truck drivers) that are involved in the process. The paper is then structured as follows: in Section 2, we discuss the current state of the art in managing context-aware pervasive systems; in Section 3 the CDT context model is presented. Then, after a short introduction to the PerLa project, we present in Section 4 the extension to the language syntax used to create and manage contexts. Sections 4.1 and 4.2 highlight how the combined actions of the CDT model and the PerLa language can be used to introduce context-awareness into pervasive systems. Section 5 deals with the evaluation of our proposal. Finally, conclusions and future work are discussed in Section 6.

2. RELATED WORK

Dealing with context-aware systems always requires to separate different dimensions of analysis that concur in the realisation of the final system:

- the context model employed [7];
- the software implemented to realise the system [14];
- the method used to query the system [13] (sometimes called *Context Query Language (CQL)* [20])

Literature contains a vast number of context models classifications, for example [7, 4] and partially [3, 5]. In relation to the software dimension, most of the implemented solutions follow the distributed paradigm: exemplifying projects of this kind are [1, 12]; on the contrary, a centralised architecture is employed by project [11]. The last point of analysis is represented by the CQL adopted to query the system. A first approach is to extend existing languages with an appropriate set of statements in order to support context-awareness [2]. Another approach is represented by the use of classical method calls directly on a dedicated middleware. This is partially the case of project [19], which also offers the publish/subscribe paradigm for querying the system. Taking into consideration the abstraction of the pervasive system as a database [16], others projects introduce a relational (e.g.: SQL) intermediate layer [11, 15] (along with appropriate mapping mechanisms [17]) or XML-based query languages [18, 20]. RDF-based languages, such as [1, 12], have more recently delineated a fourth possible choice in the language to be adopted.

[1]http://perlawsn.sourceforge.net

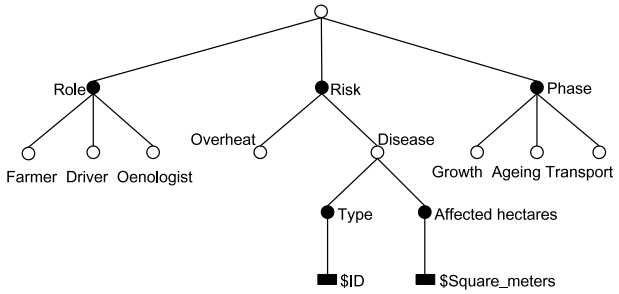

Figure 1: The CDT schema (for the wine production process)

3. THE CDT MODEL

A key element in the design of context-aware systems is the representation and management of the context itself. In this section we briefly sketch the formalism (i.e.: the *context model* [6, 5]) we adopt to model the environment where the pervasive system is operating. The choice of the context is a burden of the system designer who singles out the context *dimensions* and *concepts*: the former are used to capture different characteristics of the environment, while the latter are used to represent the admissible *values* that can be assumed by the dimensions. Both dimensions and concepts can be semantically enriched using *attributes*, that are parameters whose values are provided at run-time. Moreover, the "tree" term suggests that the designer can model the environment using the preferred granularity, nesting more than one level of dimensions with the unique restriction that every dimension can only have concept children and vice versa. This requirement is clearly exposed in Figure 1, where we present the *context schema* in the form of a *Context Dimension Tree (CDT)* in its graphical notation for the wine monitoring example introduced in Section 1. In particular we use black nodes for the dimensions and white nodes for the concepts, while attributes are represented using squared nodes.

We firstly describe the "actors" involved; for this purpose we use the `Role` dimension whose possible concept nodes can be `Farmer`, `Oenologist` and `Driver`. The second dimension, which allows us to describe the various phases of the wine production process, is the `Phase` dimension that can assume the concepts `Growth`, `Ageing` and `Transport`. The third dimension is related to the risks to be kept under control, with the two facets (concepts) of `Overheat` owing to exposure of wine bottles to sunlight, and of `Disease` which can affect the grapes. In the latter case, we are interested in the disease `Type` and in the amount of affected surface (`Affected hectares`) sub-dimensions; however the domains of the values for these dimensions are too large to be explicitly enumerated by means of concept nodes, so we can use the attributes `$ID` and `$Square_meters` whose values will be provided at run-time. Once the environment has been modelled using the CDT it is possible to formalise the context notion. In order to denote that a dimension has assumed the value of one of its possible children we use the following formalism $Dimension = Value$, where *Dimension* represents a CDT dimension and $Value$ can be either a concept node or an attribute node according to our notation. It is now possible to formalise a *context instance* as a conjunction of propositions $Context \equiv \bigwedge_{i,j}(Dimension_i = Value_{i,j})$. As an example

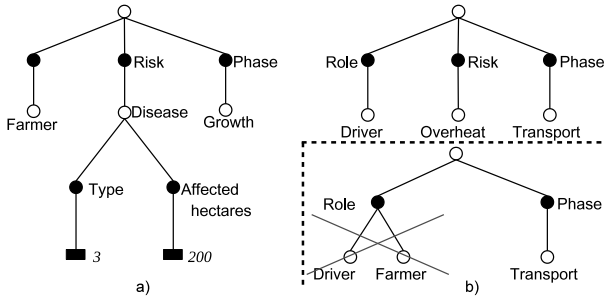

Figure 2: Contexts declaration: graphical notation

we consider the wine production process CDT and two possible contexts. The *Growth_Monitoring* context is used in the preliminary phase of the production process: temperature and humidity (see Section 4.1) are critical attributes that must be kept under constant control by the farmer when the grapes are affected by certain diseases. Successively the *Transport_Monitoring* context is employed during the transport phase. The bottled wine must not be kept under direct sunlight for more than a certain amount of time in order to avoid overheat and a consequent alteration of the wine flavour. Formally:

$$Growth\_Monitoring \equiv (Role = Farmer)$$
$$\wedge(Phase = Growth) \wedge (Disease.Type = 3)$$
$$\wedge(Disease.AffectedHectares = 200)$$

$$Transport\_Monitoring \equiv (Role = Driver)$$
$$\wedge(Phase = Transport) \wedge (Risk = Overheat)$$

Figure 2 shows the graphical notation that can be adopted to visually represent the defined context. From the same figure it clearly emerges that both context instances are composed by a subset of the initial CDT schema elements shown in Figure 2a). It is worth noticing that not all possible subsets are valid contexts. This is the case of Figure 2b) where the dimension `Role` assumes the values `Driver` and `Farmer` simultaneously (the children concepts of one dimension are always to be instantiated in mutual exclusion).

Once the context schema has been defined in terms of *symbolic* observables (e.g. the `Overheat` risk), it is possible to analyse how these can be mapped to *numeric* observables, which are instantiated by retrieving all the necessary information from the pervasive system. The PerLa system, which is presented in the next section, allows to perform this important task in an efficient way.

4. MANAGING CONTEXT USING PERLA

As extensively presented in [21, 22], PerLa is a framework to configure and manage modern pervasive systems and, in particular, wireless sensors networks. PerLa adopts the database paradigm of the pervasive system: such approach, widely adopted in the literature [16, 9], is data-centric and relies on a query *language* using an SQL-like metaphor.

PerLa queries allow to retrieve data from the pervasive system, to prescribe how the gathered data have to be processed and stored and to specify the behaviour of the devices. In particular PerLa supports three types of queries:

```
/*High level query*/
CREATE OUTPUT STREAM Monitoring
  (temperature FLOAT, humidity FLOAT,
  location_X FLOAT, location_y FLOAT)
AS LOW:
/*Low level query*/
EVERY ONE
SELECT temperature, humidity,
  location_x, location_y
SAMPLING EVERY 1 m
  EXECUTE IF EXISTS (temperature) AND
    isInVineyard(location_x, location_y)
  REFRESH EVERY 10m
```

Table 1: A PerLa query example

1. *Low Level Queries (LLQs)* define the behaviour of every single or of a homogeneous group of nodes; in particular LLQs specify the data selection criteria, the sampling frequency and the elaboration to be performed on sampled data.

2. *High Level Queries (HLQs)* define the high level elaboration involving data streams coming from multiple nodes. Such queries are equivalent to SQL operations on data streams.

3. *Actuation Queries (AQs)* provide the mechanisms to change parameters of the devices or to send commands to actuators.

A first example of a typical PerLa query for gathering data from the field is shown in Table 1 at the top: most of the keywords are self-explaining and we will focus in details on the grammar in Section 4.2 (for the complete EBNF definition refer to the final system report[2]). This brief example highlights how the three different types of queries concur in the composition of a final PerLa query. Furthermore, the other fundamental PerLa component is a *middleware* whose architecture exposes two main interfaces: a high-level interface, which allows query injection and a low-level interface that provides plug&play mechanisms to handle devices and energy savings [8]. However, the PerLa language, in its initial version, does not support the definition and the management of context. In this paper we present an extension of the PerLa language and middleware that overcomes this limitation by introducing a query-based context management system relying on the CDT context model. Our work has two main objectives: first of all the language is to be extended in order to introduce, create and maintain the CDT formalism into PerLa. On one hand this extension shall enable the designer to declare the desired contexts on the CDT, while on the other hand it shall provide the possibility to define the actions that have to be performed upon context activation. In parallel, the middleware architecture is to be extended in order to manage the CDT structure, to verify context statuses and to physically perform the corresponding actions. Section 4.1 focuses on the language ex-

[2]F. A. Schreiber, R. Camplani, M. Fortunato, M. Marelli, and G. Rota - Design of PerLa, a declarative language and a middleware architecture for pervasive systems - ARTDECO R.A.11b, DEI 2010.9 in http://perlawsn.sourceforge.net

tension, called **Context Language (CL)**. The middleware enhancements, consisting in a **Context Manager (CM)** entity, will be discussed in Section 4.2.

4.1 The Context Language

The syntax of the CL has been divided into two parts, called *CDT Declaration* and *Context creation*:

4.1.1 CDT Declaration

This part allows the user to specify the CDT, i. e. all the application-relevant dimensions and values they can assume. The syntax is the following:

```
CREATE DIMENSION <Dimension_Name>
  [CHILD OF <Parent_Node>]
  [CREATE ATTRIBUTE $<Attribute_Name>] |
  {CREATE CONCEPT <Concept_Name>
        WHEN <Condition>
    [CREATE ATTRIBUTE $<Attribute_Name>]*
    [EVALUATED ON <Low_Level_Query>]}*
```

The *CREATE DIMENSION* clause is used to declare that a new dimension must be added to a CDT. If no father is specified the dimension is meant to be a child of the tree root. Otherwise the dimension can be considered as a child of the node specified within the *CHILD OF* clause. Once a dimension has been declared, it is possible to specify the values that the dimension can assume, using the *CREATE ATTRIBUTE* or the *CREATE CONCEPT/WHEN* pair. For each pair the designer must specify the name and the condition for assuming the specified values by means of *numeric* observables that can be measured from the environment. When, instead, the design requires the presence of attributes the *CREATE ATTRIBUTE* clause must be used, using the $ sign as a prefix before the name of the attribute, meaning that its value will be supplied by the application at runtime. We postpone the explanation of the *EVALUATED ON* clause to Section 4.2, where it plays a fundamental role. If we consider the CDT of Section 3 for the wine production process, we can thus use the following set of statements:

```
CREATE DIMENSION Role
    CREATE CONCEPT Farmer
          WHEN get_user_role()='farmer'
    CREATE CONCEPT Oenologist
          WHEN get_user_role()='oenologist'
    CREATE CONCEPT Driver
          WHEN get_user_role()='driver'
CREATE DIMENSION Risk
    CREATE CONCEPT Disease
          WHEN get_interest_topic()='disease'
    CREATE CONCEPT Overheat
          WHEN temperature > 30
           AND brightness>0.75;
CREATE DIMENSION Type
    CHILD OF Disease
    CREATE ATTRIBUTE $id
CREATE DIMENSION Affected_hectares
    CHILD OF Disease
    CREATE ATTRIBUTE $square_meters
CREATE DIMENSION Phase
    CREATE CONCEPT Growth
          WHEN get_phase()='growth'
    CREATE CONCEPT Ageing
          WHEN get_phase()='ageing'
    CREATE CONCEPT Transport
          WHEN get_phase()='transport'
```

This CDT is declared using an important feature of the PerLa language: the `get_user_role()`, `get_phase()` and `get_interest_topic()` functions are employed to retrieve context information that cannot be deduced from sensors readings, but has to do with other aspects of the application. This information is typically gathered from some external XML source or a database. Although through a simple example, this clearly highlights how PerLa supports the passage from *symbolic* to *numeric* observable: the `Overheat` *symbolic* value is in fact defined in terms of the `Temperature` and `Brightness` physical quantities (thus *numeric* observables) that can be sampled from the environment using very simple queries.

4.1.2 Context creation

This section allows the user to declare a context on a defined CDT and control its activation:

```
CREATE CONTEXT <Context_Name>
ACTIVE IF <Dimension>=<Value>
          [AND <Dimension>=<Value>]*
ON ENABLE <PerLa_Query>
ON DISABLE <PerLa_Query>      /*one-shot only*/
REFRESH EVERY <Period>
```

The *CREATE CONTEXT* statement is used to create a context instance in PerLa and allows to associate a unique name to it. The *ACTIVE IF* statement is used to translate the $Context \equiv \bigwedge_{i,j}(Dimension_i = Value_{i,j})$ statement of Section 3 into PerLa. This statement is fundamental for the middleware in order to decide if a context is active or not. The actions that must be performed in both these situations must be specified using the *ON ENABLE* clause and are expressed using any type of query exposed in Section 4. The *ON DISABLE* clause can be coupled only with "one-shot" queries, that is, queries that are executed only once upon deactivation of a context, and thus do not create conflicts with the queries enabled by the next active contexts. The middleware will also perform the necessary controls according to the condition specified in the *REFRESH* clause that completes the syntax.
In the next examples we show how context management statements and queries/actuation commands on the target system are uniformly mixed in order to achieve a context-aware behaviour. For the *Growth_Monitoring* context we can thus use the listing 1:

Listing 1: Growth monitoring example

```
CREATE CONTEXT Growth_Monitoring
ACTIVE IF phase = 'growth' AND role='farmer'
        AND Disease.Type=3
        AND Disease.Affected_Hectares = 200
ON ENABLE:
SELECT humidity,temperature
WHERE humidity > 0 AND temperature > 0
SAMPLING EVERY 6 h
EXECUTE IF EXISTS humidity,temperature
   AND location='vineyard'
ON DISABLE:
   DROP Growth_Monitoring;
REFRESH EVERY 1 d;
```

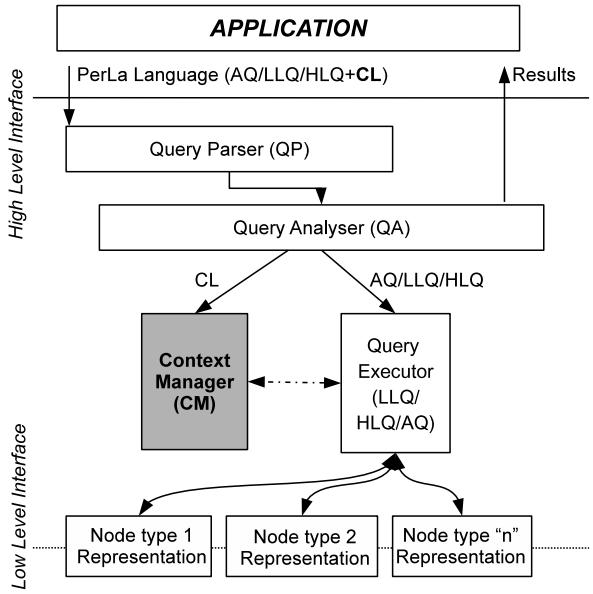

Figure 3: Middleware architecture with the CM component

In this example, after creating the context, a very short LLQ is issued: the *SELECT* clause specifies that the humidity and temperature values must be retrieved four times a day (*SAMPLING EVERY* clause), while the *WHERE* clause allows to filter the sampled values (in this case we discard false readings). The *EXECUTE IF* finally deploys the query only on those devices that host both the humidity and temperature sensors and that are located into the vineyard where the farmers are working. The *Transport_Monitoring* context can be defined as:

Listing 2: Transport monitoring example

```
CREATE CONTEXT Transport_Monitoring
ACTIVE IF phase = 'transport'
    AND role='driver'
    AND Risk='overheat'
ON ENABLE:
 SELECT temperature, gps_latitude,
     gps_longitude
 WHERE temperature > 30
SAMPLING EVERY 120 s
   EXECUTE IF location = 'truck_departing_zone'
   SET PARAMETER 'alarm' = TRUE;
ON DISABLE:
   DROP Transport_Monitoring;
   SET PARAMETER 'alarm' = FALSE;
REFRESH EVERY 24 h;
```

This example contains a similar LLQ, but employs an actuation query (AQ), introduced by the *SET PARAMETER* clause. This AQ is used to activate an alarm in order to signal that the trucks are currently under the risk of overheating. The LLQ, except for the attributes to be sampled, is very similar to the one adopted for the previous context.

4.2 The Context Manager

In this section we introduce the enhancements to the internal structure of the middleware, in order to support the Context Language. The new architecture is characterised by the introduction of a **Context Manager (CM)** as exposed in Figure 3. The CM is in charge of: 1) creating and maintaining the CDT; 2) detecting which contexts are active or not in a precise moment; 3) performing the correct actions expressed by the user according to context statuses. In the following we analyse these steps.

ID	temperature	brightness
1	28	0.60
3	31	0.71
4	33	0.80

Table 2: Table for the *Overheat* dimension

4.2.1 Creation of the CDT

During this phase all the necessary *numeric* observables (declared using the *CREATE CONCEPT/WHEN* clauses) are retrieved, and the *EVALUATED ON* clause becomes important. In fact, as long as this clause is unemployed, the CM executes a series of independent LLQs in order to retrieve the necessary information from the pervasive system. The designer could be interested in modifying this default behaviour, especially when the environment changes rapidly and the same observable is employed in different concepts (leading thus to some inconsistencies using different LLQs). This clause is useful also to introduce some optimisations (e.g.: discarding some unwanted devices). For example, on the `Overheat` dimension:

```
CREATE CONCEPT Overheat WHEN temperature > 30
                AND brightness > 0.75;
EVALUATED ON:
SELECT temperature, brightness
EXECUTE IF location='truck_departing_zone'
            AND battery>0.7
```

In this example the observables `temperature` and `brightness` are sampled simultaneously using one single query (instead of two independent LLQs). Moreover the query is executed only on those devices that are located in the truck departing zone and whose battery power is enough to operate efficiently (*EXECUTE IF* clause). Once all the results are available (independently of the presence of the *EVALUATED ON* clause) the system can create a series of tables (one for each dimension with concepts nodes) that contain a column for every attribute expressed in the *CREATE CONCEPT/WHEN* clauses. The table reports also the IDs of the devices that were taken into account during the retrieval phase. Every table entry then represents the actual value (sampled from the environment) and the device that physically produced it. Table 2 shows the results returned by the computation of the*EVALUATED ON* clause for the `Overheat` dimension, where the 1 3 and 4 sensors are involved.

Once all the necessary information has been gathered it is possible to evaluate every condition expressed by the *WHEN* clauses used during the CDT declaration. In particular, simply looking up every table, the CM assigns to a CDT concept node the ID(s) of those devices whose sampled values satisfy the condition expressed by the *WHEN* clause of the concept definition. When this phase is concluded the system knows which devices are in the situation described by the concepts of every dimension of the CDT. For example,

referring to the `Overheat` in Table 2, the CM can deduce that only sensor number 4 is detecting the risk of overheat since both *temperature>30* and *brightness>0.75* conditions are true simultaneously, while this is not the case of sensors number 3 and 1. However it might happen that a *WHEN* clause be not satisfied by any of the sampled attributes: in this case no ID can be associated to the corresponding CDT concept. Analogously an ID can appear in more than one concept (as long as they are not children of the same dimension): this is the case of modern "intelligent" sensors that can sample more than one attribute simultaneously. With similar computations the CM also selects the concepts that correspond to the results calculated by the static functions, such as `get_user_role()` or `get_phase()`.

4.2.2 Context detection

The purpose of this phase is to discover if one of the declared contexts has become active or has been deactivated. It must be remembered that a context is active if the dimensions that define it assume the values specified by the $Context \equiv \bigwedge_{i,j}(Dimension_i = Value_{i,j})$ statement. Considering also the results of the static functions, the system recognises as active all the contexts whose CDT concepts contain a not-empty devices list. In fact, from the CM point of view, if one ID has been associated with a concept it means that, for at least one device, a CDT dimension is currently assuming that value. If this situation is true for every $< Dimension >=< Value >$ used to define a context then the environment is exactly in the situation expressed by the context, and can be considered as active.

4.2.3 Context-aware behaviour

Once a context has been recognised as active, the CM simply injects the query specified by the *ON ENABLE* clause into the middleware dedicated components [21]. In the example shown in listing 1, the context activation "triggers" the node sampling (i.e., of humidity and temperature sensors). The execution flow equals the one of any other query that is manually injected into the system, and is thus completely controlled and managed by the middleware dedicated components.

5. FINAL REMARKS

As mentioned in Section 2, context management consists of three different aspects (i.e: *Context model*, *CQL*, *middleware*). In the following, we focus on evaluating the Middleware and the CQL adopted, while detailed comparisons between context models can be found in [7, 5, 3].

Differently from the other projects presented in Section 2, the PerLa language approaches the problem in a novel way introducing a *unique CQL* that allows to gather data context at the preferred granularity and to perform actions on the pervasive system simultaneously by relying on the middleware. Our proposed solution has two main advantages: (i) by adopting a database metaphor, which decouples the middleware from the high level applications (which are going to exploit the pervasive system), it is possible to use the network independently for detecting context and for retrieving application data; (ii) the granularity of the language allows to define precisely context changes, as well as dictate any changes on the desired behaviour of the system.

We can evaluate quantitatively the system scalability in terms of nodes: as far as the sensors readings are concerned,

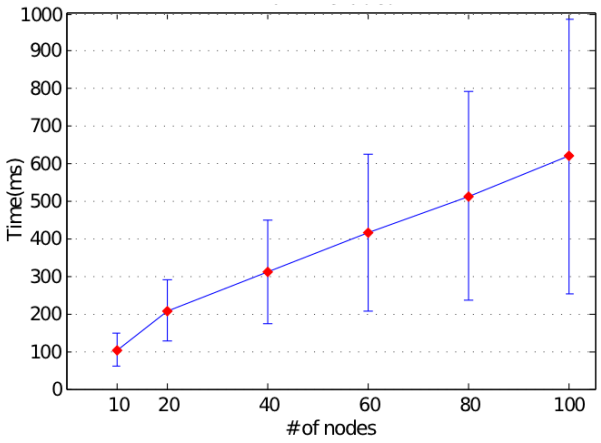

Figure 4: PerLa scaling capabilities, from [22]

a series of stress tests were performed in [22], mainly based on the execution time w.r.t. the number of nodes totally deployed. The results exposed in Figure 4 confirm that PerLa scales linearly with the number of nodes deployed. The introduction of context management in PerLa does not change the scalability feature: in fact, the detection of an active context requires (i) to retrieve sensors readings and (ii) to look up every table associated with the CDT dimensions. However, looking up every table requires a quantity of time that is linear with the number of CDT dimensions which define a valid context, and to each CDT dimension is normally associated one (or very few) node(s). In its totality the whole process (readings and context detection) thus scales linearly with the devices number.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented an extension to the PerLa system in order to support the definition and management of contexts, using the Context Dimension Tree as a formalism to model the environment. This effort has yielded a *Context Language*, as well as a set of enhancements to the middleware internal structure, including the *Context Manager* which operates on the contexts defined using the new language and performs user defined actions accordingly. The next steps of our work contemplate: (i) context conflict management: the system is, in fact, unable to autonomously select between contexts if more than one is currently active; (ii) context schema evolution and reconciliation among different applications within the same domain. The purpose of our future works is to investigate how to implement such features in PerLa and to prevent and detect context conflicts.

7. ACKNOWLEDGEMENTS

This work has been funded by the European Commission Programme IDEAS-ERC Project 227077-SMScom. We also gratefully acknowledge the assistance of Ing. Guido Rota in developing the PerLa middleware.

8. REFERENCES

[1] J. Euzenat, J. Pierson and F. Ramparany. *Dynamic context management for pervasive applications*, volume 23. Cambridge Journals, 2008.

[2] M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, and M. Perscheid. A comparison of context-oriented programming languages. In *International Workshop on Context-Oriented Programming*, COP '09, pages 6:1–6:6, New York, NY, USA, 2009. ACM.

[3] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, June 2007.

[4] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6:161–180, April 2010.

[5] C. Bolchini, C. Curino, E. Quintarelli, and F. Schreiber. Context information for knowledge reshaping. *International Journal of Web Engineering and Technology*, 5(1):88–103, 2009.

[6] C. Bolchini, C. A. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. A. Schreiber, and L. Tanca. And what can context do for data? *Commun. ACM*, 52:136–140, November 2009.

[7] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. A data-oriented survey of context models. *SIGMOD Rec.*, 36:19–26, December 2007.

[8] C. Cappiello and F. Schreiber. Quality- and energy-aware data compression by aggregation in wsn data streams. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1 –6, march 2009.

[9] D. Chu, A. Tavakoli, L. Popa and J. Hellerstein. Enterely declarative sensor network systems. In *Proc. VLDB '06*, pages 1203–1206, 2006.

[10] A. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.

[11] P. Fahy and S. Clarke. CASS - a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*, 2004.

[12] T. Gu, H. Pung, and D. Zhang. A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.*, 28(1):1–18, 2005.

[13] P. Haghighi, A. Zaslavsky, and S. Krishnaswamy. An evaluation of query languages for context-aware computing. In *Database and Expert Systems Applications, 2006. DEXA '06. 17th International Conference on*, pages 455 –462, 2006.

[14] J. Hong, S. E., and K. S. Context-aware systems: A literature review and classification. *Expert Syst. Appl.*, 36(4):8509–8522, 2009.

[15] G. Judd and P. Steenkiste. Providing contextual information to pervasive computing applications. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 133 – 142, 23-26 2003.

[16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[17] T. McFadden, K. Henricksen, and J. Indulska. Automating context-aware application development. In *In: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 90–95, 2004.

[18] D. Nicklas, M. Grossmann, J. Minguez, and M. Wieland. Adding high-level reasoning to efficient low-level context management: A hybrid approach. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pages 447–452.

[19] H. Peizhao, J. Indulska, and R. Robinson. An autonomic context management system for pervasive computing. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pages 213 –223, mar. 2008.

[20] R. Reichle, M. Wagner, M. Khan, K. Geihs, M. Valla, C. Fra, N. Paspallis, and G. Papadopoulos. A context query language for pervasive computing environments. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pages 434 –440, 2008.

[21] F. A. Schreiber, R. Camplani, M. Fortunato, M. Marelli, and F. Pacifici. Perla: A data language for pervasive systems. *Pervasive Computing and Communications, IEEE International Conference on*, pages 282–287, 2008.

[22] F. A. Schreiber, R. Camplani, M. Fortunato, M. Marelli, and G. Rota. Perla: A language and middleware architecture for data management and integration in pervasive information systems. *IEEE Transactions on Software Engineering*, (PrePrints), 2011.