F. A. Schreiber, L. Tanca, R. Camplani, D. Viganò

Managing and using context information within the PerLa language

(extended abstract)

Proceedings of the 19th Italian Symposium on Advanced database Systems SEBD 2011
Maratea, June 26 – 29, 2011

pp. 111 - 118

Managing and using context information within the PerLa language
(extended abstract*)

F. A. Schreiber, L. Tanca, R. Camplani and D. Viganò

Politecnico di Milano
Dipartimento di Elettronica e Informazione
{schreibe,tanca,camplani}@elet.polimi.it
diego.vigano@mail.polimi.it

Abstract. Self-adaptability in pervasive real-world applications can be achieved by adopting a context-aware middleware. In this paper, we propose a context-management extension to the PerLa language and middleware, which allows for: (i) gathering of data from the environment, (ii) feeding this data to the internal context model and, (iii) once a context is active, acting on the relevant resources of the pervasive system, according to the chosen contextual policy.

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

1 Introduction

Pervasive Systems deploy devices which are spatially distributed and possibly mobile, in order to monitor different kinds of physical phenomena for application support. 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" [5,3].
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: a *context model* [2] allows the representation of context in terms of *observable* entities, which, in their turn, 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 [11,16], which, on the 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. In literature, the context management systems can be classified considering the context

* This work has been funded by the European Commission Programme IDEAS-ERC Project 227077-SMScom.

model employed [3], the software architecture adopted [9] and the method used to query the system [8], also called *Context Query Language (CQL)* [15]. From the point of view of the architecture the choice lies between a centralised and a distributed model. Most of the recent projects adopt a distributed architecture (for example [1,7]), whose advantages lie essentially in a greater scalability of the whole system. In [6] a centralised architecture is instead adopted. By contrast, PerLa offers a distributed architecture which can be directly deployed on capable nodes (or the nearest capable node in case of nodes with reduced computational capability). Another point of analysis is represented by the CQL adopted to query the system. A first approach is represented by the use of classical method calls directly on the deployed middleware. This is partially the case of project [14], which also offers the publish/subscribe paradigm for querying the system. Other projects introduce a relational (e.g.: SQL) intermediate layer [6,10] (along with appropriate mapping mechanisms [12]) or XML-based query languages [13,15]. A final approach is based on RDF languages [1,7]. The PerLa language, instead, approaches the problem in a novel way introducing a unique CQL that allows to gather data at the preferred granularity and to perform actions on the pervasive system simultaneously. Haghighi et al. [8] list a series of characteristics that a CQL should implement: in the same paper it is shown that both RDF and SQL-based languages behave efficiently for context management, while the XML-based languages are more recommended to describe simple data structures rather than complex knowledge. Project [15] has recently improved the state of the art of this type of languages.
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 *Context Dimension Tree (CDT)* context model, introduced in Section 2. First of all the language is extended in order to introduce, create and maintain the CDT formalism into PerLa. On one hand this extension will enable the designer to declare the envisaged contexts, while on the other hand it will provide the possibility to define the actions that have to be performed upon each context activation. In parallel, the middleware architecture is also extended in order to manage the CDT structure, to verify context statuses and to physically perform the corresponding actions. To explain the concepts introduced 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 transport phase of the produced wine, as well as on some of the actors (farmers, oenologists and truck drivers) that are involved in the process. The paper is then structured as follows: in Section 2 the CDT context model is presented. Then, after a short introduction to PerLa, in Section 3 we present the extension to the language syntax used to create and manage contexts. Conclusions and future work are discussed in Section 4.

2 The *CDT* model

In this section we quickly introduce a simplified version of the formalism (i.e.: the *context model* [2]) we adopt to model the environment the pervasive system is operating into. The basic idea is to represent the context in terms of a set of

dimensions, used to capture different characteristics of the environment, and of the admissible *values* (or *concepts*) that can be assumed by the dimensions. In Figure 1 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 values.

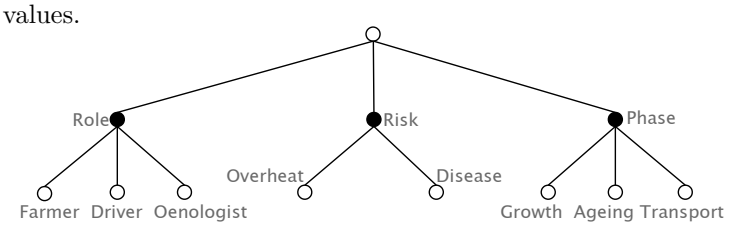

Fig. 1. The CDT schema (for the wine production process)

The `Role` dimension describes the "actors" involved: `Farmer`, `Oenologist` and `Driver`. The `Phase` dimension describes the phases of the wine production process and can assume the values `Growth`, `Ageing` and `Transport`. The third dimension is related to the risks to be kept under control, with the two values (concepts) of `Overheating`, owing to exposure of wine bottles to sunlight, and `Disease` which can affect the grapes. A *context instance* is then a conjunction of propositions, i.e. $Context \equiv \bigwedge_{i,j}(Dimension_i = Value_{i,j})$. As an example we consider the wine production process CDT and we define the *Transport_Monitoring* context. The bottled wine, in fact, must not be kept under direct sunlight for more than a certain amount of time to avoid overheat and a consequent alteration of the wine flavour: $Transport\_Monitoring \equiv (Role = Driver) \wedge (Phase = Transport) \wedge (Risk = Overheat)$. It is worth noticing that not all possible sets of concepts are valid contexts: for instance the dimension `Role` cannot assume simultaneously the `Driver` and `Farmer` values (the children concepts of a dimension are always to be instantiated in mutual exclusion). Invalid contexts are thus ruled out by appropriate constraints [2]. 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 (e.g.: temperature ranges), which are instantiated by retrieving them from the pervasive system. The PerLa[1] system, presented in the next section, allows to perform this important task effectively and efficiently.

3 Managing context through PerLa

3.1 The language

As extensively presented in [16], PerLa is a framework to configure and manage modern pervasive systems. Adopting a data-centric approach [11,4], it 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 behaviours of the devices. PerLa supports three types of queries: the *Low Level Queries (LLQs)* define the behaviour of a

[1] PerLa website: `http://perlawsn.sourceforge.net/`

(homogeneous group of) node(s), specifying the data selection criteria, the sampling frequency and the elaboration to be performed on sampled data. The *High Level Queries (HLQs)* define the high-level elaboration involving data streams coming from multiple nodes. Such queries specify operations on data streams. Finally the *Actuation Queries (AQs)* provide the mechanisms to change parameters of the devices or to send commands to actuators. A typical [16] PerLa *LLQ*, deployed on a group of sensors, and used to gather data from the field is shown below:

```
...
SELECT ID, temperature, humidity, location_x, location_y
SAMPLING EVERY 1 m
EXECUTE IF EXISTS(temperature) AND is_in_vineyard(location_x, location_y)
REFRESH EVERY 10m
```

PerLa is based on 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. However, the PerLa language, in its initial version, does not support the definition and the management of context. To achieve our goal we have extended the original language with a *Context Language (CL)*, whose syntax has been divided into two parts, called *CDT Declaration* and *Context creation*:

CDT Declaration This part allows the user to build the CDT for the specific application:

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

The *CREATE DIMENSION* clause is used to declare that a new dimension must be added to a CDT, possibly as a child of a concept node (*CHILD OF* clause). Once a dimension has been declared, it is possible to specify the values it can assume, using 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. We postpone the explanation of the *EVALUATED ON* clause to the next Subsection, where it plays a fundamental role. The CDT of Section 2 for the wine production process is specified by 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 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 have to do with other aspects of the application. This information is typically gathered from some external XML source or database. 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.

Context creation This section allows the user to declare a context from 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 translates the $Context \equiv \bigwedge_{i,j}(Dimension_i = Value_{i,j})$ statement of Section 2 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 Subsection 3.1. 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 example 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 *Transport_Monitoring* context we can use the following statements:

```
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;
```

In this example, after creating the context, a very short LLQ is issued: the *SELECT* clause specifies that both temperature and GPS location must be retrieved every two minutes (*SAMPLING EVERY* clause), while the *WHERE* clause allows to filter the sampled values. The *EXECUTE IF* finally deploys the query only on those devices located into the vineyard truck departing zone. This query features also an actuation query (AQ) introduced by the *SET PARAMETER* clause and is used to activate an alarm if the risk of overheat becomes real.

3.2 The middleware

The internal structure of the PerLa middleware as described in [16] has been modified to support the Context Language (CL). The new architecture is characterised by the introduction of a *Context Manager (CM)* 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.

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); notice that functional and non-functional data are collected in the same way. 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. If we consider again the `Overheat` dimension and supposing that the computation of the relative *EVALUATED ON* returned only the 1,3,4 IDs, a table for this dimension could be the following:

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

Table 1. Table for the *Overheat* dimension

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 1, 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()`.

Context detection The purpose of this phase is to discover if one of the declared contexts has become active or has been deactivated. Remember 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 device 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 C then the environment is exactly in the situation expressed by C, and C is considered as active.

Performing context actions 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 [16]. At this point 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.

4 Conclusions and future works

In this paper we have presented an extension to the PerLa system in order to support the definition and management of contexts, using the CDT [2] as a formalism to model the environment. Next steps of our work are focused on further system optimizations and, since more than one context could be simultaneously active, on conflict detection and resolution.

References

1. J. Euzenat, J. Pierson and F. Ramparany. *Dynamic context management for pervasive applications*, volume 23. Cambridge Journals, 2008.
2. C. Bolchini, C.A. Curino, E. Quintarelli, F.A. Schreiber, and L. Tanca. Context information for knowledge reshaping. *Intl. Journal of Web Engineering and Technology*, 5(1):88–103, February 2009.

3. C.Bolchini, C.A. Curino, E. Quintarelli, F.A. Schreiber, and L.Tanca. A data-oriented survey of context models. *ACM SIGMOD Record*, 36(4):19–26, Dec 2007.
4. D. Chu, A. Tavakoli, L. Popa, and J. Hellerstein. Enterely declarative sensor network systems. In *Proc. VLDB '06*, pages 1203–1206, 2006.
5. A.K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.
6. P. Fahy and S. Clarke. CASS - a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*, 2004.
7. T. Gu, H.K. Pung, and D.Q. Zhang. A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.*, 28(1):1–18, 2005.
8. P.D. 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.
9. J. Hong, S. Eui-ho, and K. Sung-Jin. Context-aware systems: A literature review and classification. *Expert Syst. Appl.*, 36(4):8509–8522, 2009.
10. 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.
11. S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.
12. T. McFadden, K. Henricksen, and J. Indulska. Automating context-aware application development. In *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 90–95, 2004.
13. 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.
14. 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.
15. R. Reichle, M. Wagner, M.U. Khan, K. Geihs, M. Valla, C. Fra, N. Paspallis, and G.A. 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.
16. 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.