Pushing context-awareness down to the core: more flexibility for the PerLa language

Fabio A. Schreiber, Letizia Tanca, Romolo Camplani, D. Viganò

Politecnico di Milano, Italy

PersDB 2012

6th International Workshop on

Personalized Access, Profile Management, and Context Awareness in Databases

August 31, 2012 - Istanbul, Turkey

In conjunction with VLDB 2012

pp. 1 – 6

http://persdb2012.cs.umn.edu/papers/4.Schreiber-PersDB12.pdf

Pushing context-awareness down to the core: more flexibility for the PerLa language

Fabio A. Schreiber
Politecnico di Milano, Italy
fabio.schreiber@polimi.it

L. Tanca
Politecnico di Milano, Italy
letizia.tanca@polimi.it

R. Camplani
Politecnico di Milano, Italy
romolo.camplani@polimi.it

D. Viganò
Politecnico di Milano, Italy
diego.vigano@mail.polimi.it

ABSTRACT

Information technology is increasingly pervading our environment, making real Mark Weiser's vision of a "disappearing technology". The work described in this paper focuses on using context to enable pervasive system personalization, allowing context-aware sensor-data tailoring. Since sensor networks, besides data collection, are also able to produce active behaviours, the tailoring capabilities are also extended to these, thus applying context-awareness to generic system operations. Moreover, because the number of possible context can grow rapidly with the complexity of the application, the design phase is also supported by the possibility to speed-up and modularize the definition of the data and operations associated with each specific context, producing a support tool that eases the job of the designers of modern context-aware pervasive systems.

1. INTRODUCTION

Information technology is increasingly pervading the surrounding environment, making real the vision of a "disappearing technology" expressed by Mark Weiser in his work [14]. Wireless Sensor Networks (WSN) are a clear example of this phenomenon, since they allow humans to interact with the environment, feeding the user with large set of heterogeneous data, which, considered together, could potentially generate confusion instead of widening the knowledge about the environment and its potential benefits. Filtering this plethora of data according to user's needs and, more importantly, her/his current situation is then the key for effective and efficient generation of knowledge. The concept of *context*, describing the "situation of every entity in the environment" [6], becomes a first-class citizen since it allows to *tailor* the available data for the user, reducing the "information noise". Furthermore, this perspective opens the way for a *proactive* interaction with the environment, enabling the possibility of enacting, for each particular user under a particular situation, the actions to be performed on/by the WSN.

Merging modern pervasive system frameworks with the concept of context is thus a fundamental cornerstone that must achieved. In our previous work [12] we started focusing on this goal, using the sensors of the network to characterize and discover possible context, and defining suitable actions accordingly. As presented in Section 3, we adopted the *Context Dimension Tree (CDT)* model [2] to represent the possible contexts, using the *PerLa* framework [13] to manage the pervasive system.

The contribution of the work described in this paper, originating from our previous promising results and also from previous applications of the data tailoring concept to more traditional databases [2, 4], focuses instead on using context to enable the sensor-data tailoring capabilities previously discussed. Since sensor networks, besides data collection, are also able to enforce active system behaviours, the tailoring capabilities are also extended to these, thus applying context-awareness to generic system operations. Moreover, as shown in detail in Section 4, since the number of possible context can grow rapidly with the complexity of the application, the design phase is also supported by the possibility to speed-up and modularize the definition of the data and operations associated with each specific context, producing a support tool that eases the work of designing modern context-aware pervasive systems.

Throughout this paper, we consider, as an example, a wine producer, where a set of networked sensors is employed to monitor the different phases of wine production. The wine lifecycle can be divided into the *Growth*, *Ageing* and *Transport* phases. Each phase is characterized by different risks that must be monitored respectively by the *farmer*, the *oenologist* and the truck *driver* who delivers the bottled wine. During growth, grapes can be afflicted, with different degrees of importance, by various types of *diseases*. Moreover, the bottled wine can't be put under direct sunlight because of the *overheating* risk.

In Section 2 projects addressing similar issues are briefly introduced; in Section 3, we introduce the CDT model and its representation in PerLa; in Section 4 the automatic composition of the contextual block is described. In Section 5 we present as case study an application to monitor the wine production; finally, we discuss the conclusions in Section 6.

2. RELATED WORK

A very comprehensive analysis of other projects active in this research field can be found in [8]. From these works a shared approach to context-aware management emerges, in which context is mainly analyzed at the *application* level, after having retrieved all the necessary information from the sensors possibly using a dedicated language [11]. This is, for example, the case of *CIS* [9] where contextual data is stored in a central database later queried using SQL. The project *CASS* [7] adopts a similar centralised database approach. *iQueue* (with the *iQL* language [5]) allows instead to compose contextual information according to data specification requirements and provides a library that can be used to build context-aware applications. With our previous work we argued that a middleware-based approach is feasible, with the overall computational complexity growing linearly with the number of deployed sensors. Literature also contains some detailed surveys both on the different models adopted to represent context and on pervasive management system frameworks available nowadays. As an example, the work presented in [1] gives a historic overview on various context models, while [3] presents a framework used to compare different formalism (our CDT model is also taken into account in this analysis). As far as pervasive management system frameworks are concerned, a detailed comparison can finally be found in [13].

3. PERLA AND THE CDT

This section briefly introduces both the context model we employed to model contexts as well as the framework we adopted to manage pervasive systems and which allows us to specify the desired actions.

3.1 PerLa

As extensively presented in [13] PerLa is a framework to configure and manage modern pervasive systems and, in particular, wireless sensor networks. PerLa adopts the database metaphor of the pervasive system: such approach, already adopted in the literature [10], is data-centric and relies on a SQL-like query language. 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 behaviors of the devices. Perla currently support three types of queries: *Low Level Queries (LLQ)*, which define the behavior of every single or of a homogeneous group of nodes, and specify the data selection criteria, the sampling frequency and the computation to be performed on sampled data; *High Level Queries (HLQ)*, which define the high level elaboration involving data streams coming from multiple nodes, and are equivalent to SQL operations on data streams and *Actuation Queries (AQ)*, which provide the mechanisms to change parameters of the devices or to send commands to actuators. The other fundamental component of PerLa 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 seamlessly add new devices and support energy saving.

3.2 Context Dimension Tree (CDT)

According to the CDT model [4] the set of possible contexts of the environment can be modeled as a labeled tree composed of *dimensions* and *concepts* nodes. The former are used to capture the 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 constraint imposes that the node colors alternate while descending the tree, as clearly shown in Figure 1a, where the visual representation of CDT of our running example is exposed, containing the fundamental aspects of context in the vineyard scenario. For example the dimension `Role` captures the actors involved in the wine production while the `Phase` dimension, together with the `Risk` dimension, model the various phases and the possible risks that could compromise the final produced wine.

In order to denote that a dimension has assumed a certain value we use the $< Dimension = Value >$ notation, called a **context element**. A context C can then be formalized as the conjunction of one or more context elements: $C \equiv \bigwedge_i < Dimension_i = Value_{i_j} >$. As an example, on the CDT of Figure 1a, a possible context could be the one graphically represented in Figure 1b as a subtree of the one of 1a. It is worth noticing that not all possible subtrees are valid contexts. This is the case of Figure 1c where the dimension Role assumes the values `Driver` and `Farmer` simultaneously (the children values of one dimension are always to be instantiated in mutual exclusion). The designer can specify further constraints forbidding some context elements to be used in the same context definition. These constraints, called *useless context constraints* are depicted using a line that links the mutually exclusive values. In our example, the truck `Driver` will never be involved in any of the activities which are pertinent of the grapes `Growth` phase.

The introduction of the CDT context model and the PerLa framework make now possible to describe how their combined action can be used to achieve a context-aware pervasive systems management.

3.3 Embedding context into PerLa

As discussed in the introduction, "pushing" the knowledge of context from the application down to the middleware level was the main contribution of previous work [12]. To achieve our goals we designed and implemented:

- an extension of the existing PerLa language syntax, called **Context Language (CL)**, in order to declare, inside PerLa, the CDT, the contexts as well as the actions to be performed accordingly;
- the **Context Manager (CM)**, able to maintain and manage the declared CDT, detect active contexts and performs the desired actions accordingly;

The syntax of the CL has been divided into two parts, called *CDT Declaration* and *Context creation*, both presented in details in our previous work [12].

CDT Declaration. This part allows the user to specify the CDT, i.e. all the application-relevant dimensions and values

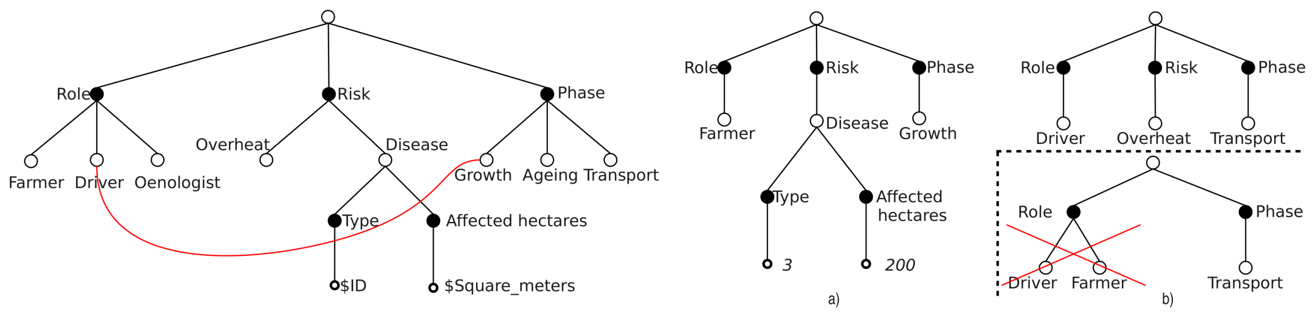

Figure 1: CDT

they can assume. As an example of its usage we report the syntax to define completely the `Risk` dimension of our running example as well as the definition of the `Driver-Growth` useless context constraint.

```
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 Role
 ...
 CREATE CONCEPT Driver
 EXCLUDES Phase.Growth
```

A set of *CREATE DIMENSION/CONCEPT* statements allows to declare the dimensions as well as their concepts nodes. The sibling of an internal dimension can be specified adopting the *CHILD OF* clause, otherwise the dimension is meant to be a child of the tree root. When creating a concept of a dimension, 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* clause). 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. The *EXCLUDES* clause is employed to express useless contexts constraints.

Context declaration. This part of the syntax allows the designer to declare a context on a defined CDT and control its activation by defining a **contextual block**, which is composed by four fundamental parts, called **components**:

- **ACTIVATION component**: allows the designer to declare a context, using a *CREATE CONTEXT* clause and associating a name to it. The *ACTIVE IF* statement is used to translate the $Context \equiv \bigwedge_{i,j}(Dimension_j = Value_i)$ statement into PerLa.

- **ENABLE component**: introduced by an *ON ENABLE* clause, allows to express the actions that must be performed when a context is recognized as active;

- **DISABLE component**: introduced by an *ON DISABLE* clause is the counterpart of the previous one, allowing to chose the actions to be performed when the declared context stops being active;

- **REFRESH component**: instructs the middleware on how often the necessary controls must be performed.

In the following listing we report a block declaration for the 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 (Growth Monitoring):
 SELECT humidity, temperature
 WHERE humidity > 0 AND temperature>0
 SAMPLING EVERY 6 h
 EXECUTE IF device_location = 'wineyard'
ON DISABLE:
 DROP Growth Monitoring
REFRESH EVERY 1d
```

4. AUTOMATIC COMPOSITION

In the previous section we have shown how a growing tree depth of the CDT allows the designer to capture the aspects of the environment with different granularities, since more dimensions (and thus concepts) allow to express more possible contexts. In particular the total number of contexts grows exponentially with the number of concept nodes, and, even for middle-sized CDTs, the task of declaring all the contexts using the aforementioned syntax becomes rapidly unfeasible for the designer. As an example from a CDT with 5 dimensions with 3 concepts nodes each, even with many constraints, more than 500 different meaningful contexts can be generated, charging the designer with the hard task of declaring i) every single context and ii) a set of actions for each one of them. A more engineered approach is the main objective of the second phase of our work. The next section illustrates the possibility of relieving the designer from this arduous task, enabling the middleware to automatically build the contextual block starting from the contextual block components, called *partials*.

Partial components. The precise syntax of the PerLa language allows to separate the block components into one or more *partials*, as shown in Figure 2.

This division is particularly meaningful for the ENABLE and DISABLE components, while the only block that can't

```
ON ENABLE ...:
 SELECT pressure,humidity
 WHERE pressure>0
      AND humidity>0
 SAMPLING EVERY 2m
 EXECUTE IF device_id>3
```

```
ON ENABLE ...:
 SELECT pressure
 WHERE pressure>0
 SAMPLING EVERY 2m
```

```
ON ENABLE ...:
 SELECT humidity
 WHERE humidity>0
 EXECUTE IF device_id>3
```

Figure 2: Partial (ENABLE) components definition

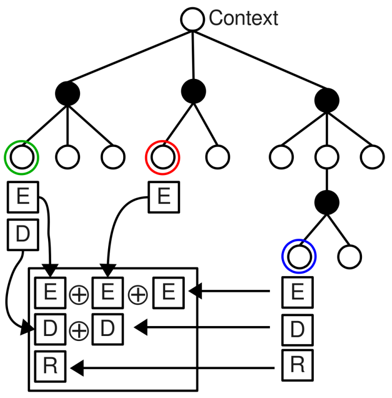

Figure 3: Contextual block composition $\oplus_D$

be divided is the ACTIVATION block since it deals with the definition of context itself. The only requirements for the partials is that they must be syntactically and semantically correct.

Automatic composition. With the introduction of partials it is now possible to describe the main concepts behind automatic composition of the contextual block. The main idea, already adopted in [4] for the tailoring of data, is illustrated in Figure 3. The designer must only associate one or more *partials* with each context element of the CDT. When the system has to compose a contextual block, it starts from the partials associated with the context elements which are part of the context and combines them by means of a generic operator, represented here by the symbol $\oplus$. This operator can be implemented in different ways, two of which are proposed in the following.

The association of the partials with the CDT context elements can be performed using the following syntax, which enriches the CDT declaration section of the CL.

```
CREATE DIMENSION <Dimension_Name>
 [CHILD OF <Parent_Node>]
 [CREATE ATTRIBUTE $<Attribute_Name>] |
 {CREATE CONCEPT <Concept_Name> WHEN <Condition>
  [WITH ENABLE COMPONENT: <PerLa_Query>]
  [WITH DISABLE COMPONENT: <PerLa_Query>]
  [WITH REFRESH COMPONENT: <Period>]
  [CREATE ATTRIBUTE $<Attribute_Name>]*
  [EVALUATED ON <Low_Level_Query>]}*
```

Listing 1: Altered CL syntax to support partial components

The *WITH ENABLE COMPONENT* clause may contain any query expressed using PerLa. The same holds for the *WITH DISABLE COMPONENT* clause. The last clause (*WITH REFRESH COMPONENT*) allows to specify the time period (always using PerLa's syntax) to be used. Finally the composition can be carried out both at design and at run-time (the main differences will be analyzed in the following paragraphs). In order to indicate the adopted strategy we make a difference between the two versions of the $\oplus$ operator: the design-time version $\oplus_D$ and the run-time one $\oplus_R$.

Design-time composition $\oplus_D$. When the association phase is complete and before the system is put into an operational state, it is possible to combinatorially generate all the possible contexts that are defined by the CDT and that are not forbidden by the constraints. For each possible context the relative contextual block is then automatically generated composing the partials associated in the previous phase. In the following the algorithm of composition is shown in pseudo-code.

This algorithm, as its first step, retrieves all the relative context elements (i.e.: the couples ($Dimension_i = Value_j$) for all possible context. This phase is represented by the **getContextElements()** function, which is executed for all possible contexts in $\mathcal{C}$. This function returns the set (represented as an array in the algorithm) containing the context elements ($Dimension_i = Value_j$) that are associated with the context passed as argument. With the context elements "at hand", the CM exploits three functions[1] in order to retrieve the partial components associated with every context element retrieved at the previous step. The **getEnableComponents()**, **getDisableComponents()** and **getRefreshComponents()** are in charge to accomplish this task. When all these inputs have been retrieved a **composeBlock()** function is invoked. This function firstly creates an empty contextual block. All the retrieved partial components are attached (*attach()* function) to the empty block in the right position (using the *dot* notation to indicate the access to a precise component of a contextual block). As far as the REFRESH component is concerned, the **composeBlock()** function computes (and attaches) the lowest refresh value among the ones contained in the R[ ] set. It seems reasonable, in fact, to say that the context whose state must be controlled with a higher frequency (smallest temporal values) is the most critical one and its refresh value is to be chosen during composition. Except for the discussed REFRESH component, the ENABLE and DISABLE components are formed by multiple clauses expressed using PerLa syntax: the simple append, one after other, of all the clauses contained in this components is not enough. Thanks to the precise internal structure of PerLa queries discussed in [13], an **optimizeBlock()** function is in charge, acting on the composed ENABLE and DISABLE components, of placing every single PerLa clause in the right order and position according the HLQ, LLQ and AQ syntax. This task is particularly important for LLQs, since, as shown in [13], these queries are composed by at most four different sections: every statement of every component must then be placed in the correct order. Further optimizations can be achieved, es-

[1]The algorithm reports only one function, being the three functions operatively identical.

```
Input  : The $\mathcal{C}$ set of all possible contexts
Output: BS set with the composed contextual blocks
BS=$\emptyset$;
for (context $c_i \in \mathcal{C}$) do
    /*Context elements retrieval*/ ;
    CE[ ] $\leftarrow$ getContextElements ($c_i$);
    /*Components retrieval*/ ;
    E[ ] $\leftarrow$ getEnableComponents (CE[ ]);
    D[ ] $\leftarrow$ getDisableComponents (CE[ ]);
    R[ ] $\leftarrow$ getRefreshComponents (CE[ ]);
    B$_i$ $\leftarrow$ composeBlock (E[ ],D[ ],R[ ]);
    optimizeBlock (B$_i$);
    if (parseBlock (B$_i$)=='OK') then
        BS = BS $\cup$\{B$_i$\};
    end
    else
        return WARNING('Parse Error')
    end
end
return BS;

Procedure composeBlock(E[ ],D[ ],R[ ]) ;
B = $\emptyset$;
for (enable comp. $e \in$ E[ ], disable comp. $d \in$ D[ ] ) do
    attach (B.E, $e$);
    attach (B.D, $d$);
end
B.R = $min$(R[ ])
return B;

/*Identical for Disable and Refresh*/ ;
Procedure getEnableComponents(CE[ ]) ;
EB = $\emptyset$;
i = 0;
for ($ce \in CE$) do
    if ($Context\_Enable\_Rel(ce) \neq \emptyset$ ) then
        EB[i] = $Context\_Enable\_Rel(ce)$;
        i++;
    end
end
return EB;
```

pecially when different components contain the same clause with different parameters that can be merged together in a single clause. In the following Section an example is given of such optimization on the *SELECT* clause. The last step of the algorithm instructs the CM to inject the composed contextual block into the middleware QueryParser component using the *parseBlock()* function. The QueryParser is, in fact, able to verify the syntactic and semantic validation of the composed block and to raise a warning message in case some inconsistencies are detected.

Run-time composition $\oplus_R$. In the run-time approach, the association of the partials with the context elements of the CDT is the same as above. However, in this case the composition of a contextual block is carried out at run-time only when its relative context is recognized as active by the middleware. Advantages and disadvantages are illustrated below.

Differences between the $\oplus_D$ *and* $\oplus_R$ *approaches.* The two proposed approaches show a trade-off that enables us to advise their use in different situations. At design-time, in fact, the designer has total control over each composed block before its behavior is enacted; as a consequence, he or she can still modify the composed blocks in case the requirements ask for particular attention. On the other side, many contextual blocks will possibly be generated and controlled by the designer even if their actual activation happens very seldom, because this more static vision of the whole system considers all the contexts at the same level of plausibility. At run-time the system behaves with a more autonomic fashion, but the designer cannot modify the composed contextual blocks. In addition, performance must be kept under control: more than one context can, in fact, be active simultaneously, and also the context switches may be very frequent. The frequent on-line composition of several context elements, maybe involving complex partials, could then potentially slow down the whole system performances.

5. RUNNING EXAMPLE

In the following example we suppose that the designer has associated to the `Growth` concept of the `Phase` dimension the following *partials*, using the syntax of Section 4:

```
CREATE CONCEPT Growth WHEN date IS BETWEEN '01-05-2012'
     AND '30-09-2012'
 WITH ENABLE COMPONENT:
  SELECT ph_value,humidity
  WHERE sensor_group = 'north_sensors'
 WITH REFRESH COMPONENT: 5s
```

The designer declares the `Growth` concepts using the *WHEN* clause. Since grapes are typically planted and grow starting from half summer until the first days of September, the *numeric* observable `date` is used to characterize the `Growth` *symbolic* observable. Using the *WITH ENABLE COMPONENT* syntax the designer associates a query aiming at sampling the pH value of the terrain as well as the humidity of the air. Finally this query keeps only the sampled records coming from the north sector of the vineyard (*WHERE* clause). As last, the *WITH REFRESH COMPONENT* sets a refresh period of 5 seconds. Acting in the same way, the designer assigns to the `Overheat` concept the following *partials*:

```
CREATE CONCEPT Overheat WHEN temperature > 30
WITH ENABLE COMPONENT
 SELECT MAX(temperature)
 SAMPLING EVERY 20s
 SET PARAMETER 'alarm' = true;
WITH DISABLE COMPONENT:
 SET PARAMETER 'alarm' = false;
WITH REFRESH COMPONENT: 1s
```

The syntax is very similar to the previous associations. The *ENABLE COMPONENT* features an *Actuation Query (AQ)* which sets an alarm in case of overheat. The same alarm is turned off again in the *AQ* specified within the *WITH DISABLE COMPONENT* block. Before discussing the composition of these two assignments into the final contextual block, it is worth underlining that the single associated queries are syntactically and semantically meaningful and that the designer is not obliged to associate to a context element every possible type of *partial.* As an example, the first association lacks of the partial disable type, since the *WITH DISABLE COMPONENT* syntax is missing. We focus now on the contextual block that is composed when the vineyard is in the growth phase and there is an overheat risk. In other words we are interested in the following context:

$$\begin{aligned} Monitoring\_Context \equiv & (Phase = \text{`}Growth\text{`}) \wedge \\ & (Risk = \text{`}Overheat\text{`}) \end{aligned}$$

The final contextual block will be composed starting from the associations described before and according to the strategy chosen (i.e.: combinatorially at design-time, or when the Monitoring context is recognized as active at run time). The final block will then be the following:

```
CREATE CONTEXT (Monitoring_Context)
ACTIVE IF Phase = 'Growth' AND Risk = 'Overheat'

ON ENABLE (Monitoring_Context):
  SELECT MAX(temperature),ph_value,humidity
  WHERE sensor_group = 'north_sensors'
  SAMPLING EVERY 20s

  SET PARAMETER 'alarm' = true;

ON DISABLE (Monitoring_Context):
 SET PARAMETER 'alarm' = false;

REFRESH EVERY 1s
```

The example features one simple optimization that is carried out during composition. The Context Manager is, in fact, able to detect that two different *SELECT* clauses (coming from two different associations) must be composed together. Instead of simply appending one clause after the other, their correspondent arguments are merged together into a single *SELECT* clause. Finally it is worth noticing that the refresh period is correctly chosen as the $\mathtt{min}\{\mathtt{1s}, \mathtt{5s}\}$.

6. CONCLUSIONS

In this paper we described some advances on previous work presented in [12] for providing a context-aware support for pervasive system users.

In particular, we propose an automatic definition and composition mechanism of context-aware sub-query blocks aiming to cope with the combinatorial explosion of context that can be generated from a simple CDT model.

Given the growing spread of anytime, anyplace connectivity for users, with RFID tags, wireless sensors and embedded devices made transparent to the application and to the user by sophisticated middleware, we believe pervasive systems to be among the main challenges for personalization.

Future improvements will contemplate: a) a formal definition of the composition operator constraints; b) how to evaluate the consistence of a generated context-aware query; c) a finer control of context concurrency and switching; d) as well as its full-fledged application to wide-range projects.

Acknowledgments

This work was partially funded by the European Commission, Programme IDEAS ERC, Project 227977-SMSCom and Industria 2015, Programma n° MI01_00091 SENSORI.

7. REFERENCES

[1] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161 – 180, 2010. Context Modelling, Reasoning and Management.

[2] C. Bolchini, C. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. A. Schreiber, and L. Tanca. And what can context do for data? *Commun. ACM*, 52(11):136–140, 2009.

[3] 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.

[4] C. Bolchini, E. Quintarelli, and L. Tanca. Carve: Context-aware automatic view definition over relational databases. *Information Systems*, Accepted manuscript (unedited version available online: 12-MAY-2012).

[5] N. H. Cohen, A. Purakayastha, L. Wong, and D. L. Yeh. iqueue: A pervasive data composition framework. In *Proceedings of the Third International Conference on Mobile Data Management*, MDM '02, pages 146–, Washington, DC, USA, 2002. IEEE Computer Society.

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

[7] P. Fahy and S. Clarke. Cass: Middleware for mobile, context-aware applications. In *Workshop on Context Awareness at MobiSys 2004*, June 2004.

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

[9] G. Judd and P. Steenkiste. Providing contextual information to pervasive computing applications. In *Proc. of IEEE International Conf. on Pervasive Computing and Communications (PerCom'03), Fort Worth, Texas, USA*, pages 133–142, 2003.

[10] 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.

[11] R. Reichle, M. Wagner, M. Khan, K. Geihs, J. Lorenzo, M. Valla, C. Fra, N. Paspallis, and G. Papadopoulos. A comprehensive context modeling framework for pervasive computing systems. In *Distributed applications and interoperable systems*, pages 281–295. Springer, 2008.

[12] F. Schreiber, L. Tanca, R. Camplani, and D. Viganó. Towards autonomic pervasive systems: the PerLa context language. *Electronic Proc. 6th NetDB*, pages 1–7, 2011.

[13] 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 Trans. Software Eng.*, 38(2):478–496, 2012.

[14] M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3:3–11, July 1999.