A Context Query Language for Pervasive Computing Environments

Roland Reichle[1], Michael Wagner[1], Mohammad Ullah Khan[1], Kurt Geihs[1],
Massimo Valla[2], Cristina Fra[2], Nearchos Paspallis[3], George A. Papadopoulos[3]
[1] Distributed Systems Group, University of Kassel, Germany
{reichle, wagner, khan, geihs}@ vs.uni-kassel.de
[2] Telecom Italia Lab
{massimo.valla, cristina.fra}@telecomitalia.it
[3] Department of Computer Science, University of Cyprus
{nearchos, george}@cs.ucy.ac.cy

Abstract

This paper identifies requirements for querying and accessing context information in mobile and pervasive computing environments. Furthermore it studies existing query languages showing that they satisfy only a subset of these requirements or cover some of them only to a limited extent. A new context query language is presented to overcome these shortcomings, improving the state of the art in several respects: heterogeneous representations of context information, definition of complex filtering mechanisms, elaborate aggregation functions and ontology integration, all in one language.

1. Introduction

In context-aware computing, information about the execution environment is analyzed and used to adjust the behavior and properties of applications to the running context. For the development of such context-aware applications and in order to be able to reuse such a generic context management system [12], developers need methods and tools for querying and accessing relevant context information. An important aspect of this problem is the definition and provision of a suitable Context Query Language (CQL).

Several projects have dealt with the general characteristics of context information as well as the challenges of context management systems [1][3][4]. Our work is focused especially on pervasive and mobile computing environments. In such environments context management has to cope with aspects of mobility, autonomy, distribution, heterogeneity and spatio-temporal variation. Different context models and context management systems have been proposed which are tailored to a sub-set of these characteristics and challenges.

Obviously, a query language for context information has to address these characteristics. Several CQL have been proposed in the literature that provide sufficient expressive power and can satisfy most of the characteristics and challenges mentioned above. However, a detailed analysis (see Section 3) revealed, that none of them satisfies all of our requirements. For pervasive and mobile computing environments, a CQL is required to support heterogeneity of context sources (particularly with regard to different context data representations), incorporation of ontology reasoning mechanisms, complex filtering conditions and elaborate aggregation functions - all in one language. In order to overcome these limitations, we have developed a new XML-based language for context querying. It goes beyond existing CQLs as it explicitly addresses the challenges pointed in the previous paragraph.

The rest of this paper is organized as follows. Section 2 identifies the requirements for a CQL in mobile and pervasive computing environments. Section 3 gives an overview of existing approaches and discusses their shortcomings. The basic concepts of our new CQL and examples are presented in Section 4. Then, in Section 5 we evaluate our CQL and show that it meets all of the identified requirements. Finally, Section 6 concludes the paper by summarizing its main contributions.

2. Requirements

In [2], a number of characteristic aspects of context information are summarized and used to evaluate

0-7695-3113-X/08 $25.00 © 2008 IEEE
DOI 10.1109/PERCOM.2008.29

different context query languages. According to this evaluation, context information

- can be static or dynamic;
- can be in the form of continuous data streams;
- can be temporal, imprecise, erroneous, ambiguous, unavailable or incomplete [6]. Therefore it must be annotated with meta-data indicating its quality;
- often expresses location, proximity and spatio-temporal relationships;
- can represent a situation which is derived from other (more elementary) context information.

Apart from these general characteristics of context information, the nature of pervasive computing raises more challenges for context management in general. Perich *et al* [3] describe four orthogonal axes which characterise context management in pervasive systems: *autonomy, distribution, mobility* and *heterogeneity*. The first three imply that there is no centralized control of the individual context sources, and that part of the context information may reside on different devices. Also, it implies that context sources may dynamically appear and disappear. Therefore, the context queries should not address a specific context source. They should rather be formulated at a more abstract level allowing the retrieval of context information from several context sensors with a single query. Heterogeneity is also an important issue, as it can not be assumed that in a pervasive environment all context sources work on a single common data model. Hence, semantic annotations have to be incorporated to establish a common understanding of the exchanged information, and also different options for representing a certain piece of information have to be considered.

A requirements analysis that has been conducted in the MUSIC project [11] has revealed a number of additional requirements for a CQL in pervasive computing environments. These requirements are:

- The CQL has to provide support not only for accessing single context elements, but also for retrieving a whole set of context elements with a single query. For example, the mood of all persons in a room.
- The CQL has to provide support for specifying filters and conditions, in order to retrieve only context information with certain properties. Filters should be applicable to both the values of context elements and also to their associated metadata.
- Query filters and conditions should support the subscription to context information, i.e. subscription to asynchronous context change events.
- In order to combine elementary conditions to more complex ones, a set of logical operators must be defined.
- Aggregation functions should be provided, such as computing the average of a series of context values. Aggregation functions are also useful when defining subscriptions for context change events such as “average network bandwidth has dropped below 100 Kbps”.
- It should be possible to access current and past context information from a history.
- It should also be possible to formulate queries that incorporate semantic reasoning, e.g. “all persons that are in a business meeting with a specific person named Peter”.

3. Discussion of existing CQLs

An evaluation of existing CQLs in [2] argues that so far SQL-based and RDF-based query languages are the best matches for the characteristics of context management in pervasive computing environments. Of course, the way of accessing context information highly depends on the underlying context modeling approach and reflects its expressiveness and flexibility. According to that evaluation, two approaches to context modeling and context accessing stand out: the Context Modeling Language (CML) [4][5][6] and the CQP of the MoGATU framework [8].

CML is a very powerful approach providing modeling constructs for describing (fact) types of information, their classification (i.e. whether static, derived or sensed), relevant quality metadata and dependencies among different types of information. CML extends ORM and also includes a powerful language to define and reason about situations that are derived from simple facts. For accessing the context information a simple programming API is exposed to the application. The context management framework for CML [4] enables mapping its models to relational data schemes. Therefore, the corresponding CQL could also be based on SQL, which is a well established, declarative query language providing a solid basis for creating and executing a wide range of queries. In [7], it is stated that in the CML context management system, context queries are internally mapped to SQL. With SQL, most of the requirements identified above are met. However, when accessing context information spread over several tables, a number of joins may be necessary which may result in quite complex queries, in particular when performing context reasoning. Pure SQL also does not deal with semantic annotations and does not incorporate the concept of ontologies used for establishing a common understanding of the context information. Therefore, appropriate enhancements are needed. With regard to our requirements, the programming API of CML does not address the retrieval of context information with heterogeneous representations. Furthermore the inclusion of complex preprocessing functions is not supported to a sufficient extent. Another big difference to the programming API of CML is that the CQL presented in this paper provides the possibility to specify context queries in a platform and programming language independent way.

The MoGATU framework [8] incorporates the Collaborative Query Processing (CQP) protocol, which is a quite innovative approach in the area of context querying. MoGATU is a profile-driven, proactive, peer-to-peer semantic context management system for pervasive environments. Queries are defined in DAML-OIL and context is requested with the help of the DAML-OIL and DAML-S specifications. In the CQP protocol, queries can also be decomposed to sub-queries which are assigned to different context services. However, support for defining filters and operators to combine filters and aggregation functions is limited.

Another approach to context modeling and context representation was proposed by Bettini *et al* [1] as part of the CARE middleware. Shallow context information, such as devices, is modeled using simple CC/PP profiles, whereas non-shallow context data, such as the socio-cultural environment of the user, is represented using ontologies. As a consequence, this work supports a hybrid context access and reasoning approach. However, support for heterogeneous representations and easy integration of complex filtering and processing functions is also not sufficiently addressed.

In the SPICE project [9], SPARQL is used as a CQL. SPARQL [14] is a W3C standard proposal for a RDF query language whose syntax is inspired by SQL. The approach is interesting as a way of incorporating semantic concepts and ontologies into a SQL-based query language. However, pure SPARQL is found to be inadequate as a context query language. It facilitates querying concepts of an entity, but it is not intended to be used for querying complex data constructs with several levels of nesting. Appropriate support for filter operators and aggregation functions is also missing. Another obvious weakness of SPARQL is the fact that queries easily become quite long and complex, as the query must be specified on the RDF triples.

A simple XML-based context description and query language was developed in the MobiLife project [10]. This CQL provides a good set of simple relational operators and also string-based operators. There are some operators to combine simple filters to more complex ones as well. A query can be expressed with regard to a value of a parameter , the timestamp of a parameter and on associated meta-data, as for example the accuracy or the confidence of a parameter (probability of context information to be correct). There is also support for including the position of a parameter in an array of context elements, which allows the selection of a specific parameter in the array. The concept of placeholders is also supported. However, there is lack of aggregation functions and the need for ontologies and semantic reasoning is not sufficiently addressed. Furthermore, another important limitation is that the application must know beforehand the provider of the context information and then query the provider. Thus, support for specifying queries involving sub-queries for different context providers is not provided.

4. The MUSIC Context Query Language

This section describes a new CQL which is based on XML and has been designed within the scope of the MUSIC project [11]. As this CQL is strongly related to the underlying context model, we start by describing this model first.

4.1 Context Modeling Approach

For the MUSIC context modeling, a simple but highly extensible approach was chosen. The approach is based on XML and makes use of ontologies that are described in OWL. The context information is represented in terms of context elements, which provide information about *context scopes* and *context entities*. The former describe a specific domain, e.g. position, civil address, environment, user proximity, etc. The latter refers to concrete entities in the world e.g. user, room, device, etc.

As illustrated in Figure 1 the *Context Scopes*, or more precisely, all the *Concepts* in the context model, are associated with one or more *Representations* describing the structure of the context data in terms of *Parameters* (`par`: attributes of the context data and associated meta-data along with admitted types/values), structures of *Parameters* (`parS`), and arrays of (structures of) *Parameters* (`parA`). In order to establish a common understanding about the semantics of the different concepts in a heterogeneous pervasive environment, the Context Scopes, the types of Context Entities and the different Representations are all described using an ontology. Like in the Aspect-

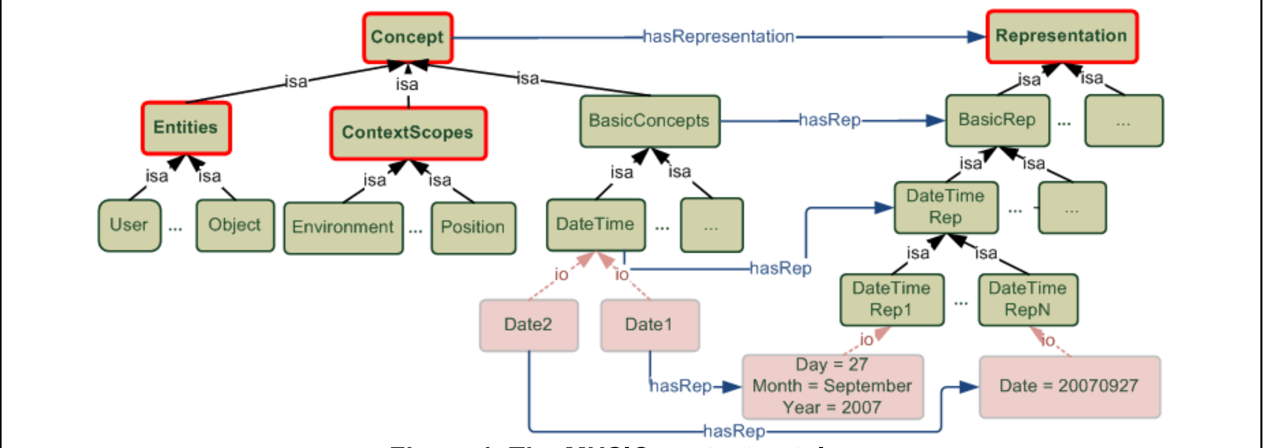

Figure 1: The MUSIC context ontology

Scale-Context (ASC) model by Strang *et al* [13], the ontology also includes so-called *Inter-Representation-Operations* (IROs), facilitating the automatic conversion of measure units, as well as more complex conversions between completely different representations (for example a position in GPS coordinates to an address in terms of street and city). For this purpose, the ontology provides the grounding to a certain method in a library or to a certain service providing the appropriate functionality. In the same way, we allow the definition of *Aggregation Functions* in the ontology, in order to enable the aggregation of (sets of) context elements to a certain value or to derive more elaborate context information. Besides, the ontology is also used to describe relationships between entities, e.g. a child has a father and a mother, or a room belongs to a building. This allows the description of semantically complex queries in a compact manner, as an ontology reasoner can be used to automatically resolve the relationships.

As ontology reasoning on mobile devices with low resources may not be feasible at run-time, we also allow characterizing context entities and context scopes simply through predefined types. The type implicitly corresponds to a certain semantic concept and to a default representation of the context information.

4.2 Context Query Language Definition

The main structure of a query is given through the enclosing XML element *ctxQuery*, the elements for the entities and scopes involved in the query, the element defining the action that has to be performed and the elements for the conditions specifying the required filtering. For example, a query may look like:

```
<ctxQuery resultName="addressOfMaryInItaly">
  <entity  ontConcept="prefix:music:username">
  user|Mary
  </entity>
  <scope
      ontConcept="prefix:music:CivilAddress"
      ontRep="prefix:music:DefaultAddressRep">
      civilAddress
  </scope>
  <action type="SELECT"/>
    <conds>
      <cond type="ONVALUE">
       <constraint par="civilAddress.country"
                   op="EQ" value="Italy"/>
      </cond></conds></action>
</ctxQuery>
```

(get address of Mary if her context shows she is in Italy)

The enclosing element, *ctxQuery*, has only one optional attribute called *resultName*. With the help of this attribute the result of the query, which may simply comprise a number but may also be a whole set of context elements or context entities, can be referred to and used in conditions of other queries.

The entities and the scopes involved in the query are described through a number of elements named *entity* and *scope*, accordingly. These elements have the attributes *ontConcept*, referring to the corresponding semantic concept described in the ontology, and *ontRep*, referring to the requested representation which is also defined in the ontology. As already mentioned above, for mobile devices with low resources this information can also be provided through some predefined types which are contained in the body of the elements. Here, it is worth noting that in the query presented above, this information is redundant. Also, the query above shows that it is also possible to set a condition directly on entities. Wildcards are supported, but the query can also be constrained to only one entity (e.g. user Mary in the example above).

The *action* element identifies through its attribute *type* an operation that the system has to perform on the context data; the action determines the structure of the query condition and the query response. Depending on the query action, the query response could be a subset of the available context data or inferred information obtained from the computation of context data. For context querying we only need the action type *SELECT*. However, in MUSIC we also provide the action types *SUBSCRIBE* and *UNSUBSCRIBE*, in order to indicate to the underlying context management system that the query is used to subscribe or unsubscribe to asynchronous notification and retrieval of context information. The optional attribute *option* is used to define a post-processing operation on the results of the query. Here, we have included some predefined operations such as *COUNT, AVERAGE, MINIMUM* and *MAXIMUM*. Other operations can also be included through a reference to the ontology which contains the corresponding grounding. Therefore, the corresponding *action* element may look as follows:

```
<action type="SELECT"
option="prefix:music:average", on="user.age"/>
```

The *conds* element encloses the conditions for the query. A query condition defines filtering criteria for the selection of context data; three types of conditions are defined:

- *ONCLOCK* is used to specify that we would like to obtain context information periodically independently of some constraints.
- *ONCHANGE* is used to indicate that we would like to be informed when there is an actual change in the context data.
- *ONVALUE* is used when the condition is related to one or more specific parameter values.

The first two types of conditions are used in queries to subscribe to context data, the last one to request for actual context data. A single condition can contain one or more constraints connected with logical operators (*AND*, *OR*) and with unlimited nesting. In this way, complex conditions can be constructed by connecting one or more simple conditions in a logical way.

For constraints, we distinguish between *ordinary constraints*, and *reasoning constraints*.

An ordinary constraint defines a constraint on a certain parameter. Therefore, the corresponding element can have one or more of the following attributes:

- *par* – the name of the parameter which the constraint refers to. Its structure follows the structure of the context data (e.g. scope.par, scope.parS.par, scope.parA[n].par, etc). This is given through the referred representation specified in the ontology and thus this attribute refers to elements in the ontology.
- *value* – the value of the parameter.
- *delta* – when the constraint refers to a continuous parameter, this attribute represents the accepted threshold.
- *op* – the operator applied to the parameter for constraint verification. The operators actually defined are arithmetic or string-based and are listed by Table 1:

Table 1: Constraint operators

GT	Greater than	NCONT	Not contains
NGT	Not greater than	STW	Starts with
LT	Lower than	NSTW	Not starts with
NLT	Not lower than	ENW	Ends with
EQ	Equals	NENW	Not ends with
NEQ	Not equals	EX	Exists
CONT	Contains	NEX	Not exists

In ordinary constraints is also possible to involve a function for the comparison value. Therefore, the *value* attribute is replaced by an element specifying the involved function and its input parameters. For example:

```
<constraint par="user.age" op="LT">
  <value>
    <function ontRef="prefix:music:average"/>
    <input>UsersInRoom.age</input>
  </value>
</constraint>
```

It is worth noting that *UsersInRoom* is the result of another query performed earlier on the context data. The underlying context query processing system is able to extract the attribute *age* from the set of users that was retrieved by the previous query and to provide it as input to the function. For the *average* function we refer to the ontology, where its semantics and grounding are specified.

A reasoning constraint is indicated through the tag *reasconstraint.* It defines a constraint on entities making use of the relationships between entities defined in the ontology, and has the listed attributes:

- *relation*: defines the relationship between the entities that is used for the constraint.
- *toEntities*: defines the set of entities with which the returned entities have the specified relation.

For example to retrieve all relatives of Mary:

```
<ctxQuery […]>
  <entity […]>user|*</entity>
  <action type="SELECT"/>
    <conds>
      <cond type="ONVALUE">
        <reasconstraint
          relation="prefix:music:isRelativeOf"
          toEntities="user|Mary"/>
      </cond>
      […]
</ctxQuery>
```

As mentioned already, queries involve an ontology reasoner which can for example automatically resolve relations between entities like *isChild* and the reverse of *hasBrother* or *hasSister* to *isRelativeOf.*

Furthermore, a query refers to the current context or the context history. This feature is modeled with a special condition indicated through the tag *timerange* which describes the effective time period. This tag has two attributes:

- *from* – identifies the initial timestamp of context data involved in the computation.
- *to* – identifies the final timestamp of context data involved in the computation.

If the time range tag is not present, it is implied that the query refers to the current context; if the timestamp in the two attributes is the same, then the query refers to a specific point (instant) in time.

5. Evaluation

In this section we show that the proposed CQL extends the state-of-the-art by fulfilling the requirements that were introduced in Section 2 and by overcoming the limitations of related works.

Many of the initially identified requirements have already been addressed explicitly in the description of the CQL in the previous section. The new CQL provides support not only for accessing single context elements, but also for retrieving a whole set of context elements with a single query. Furthermore, it offers various ways to specify filters and conditions. By using logical operators, conditions can be combined to more complex filters, theoretically in unlimited nesting. Besides, query filters and conditions are also applicable for context subscriptions. For this purpose, we have defined the actions SUBSCRIBE, UNSUBSCRIBE and the condition types ONCLOCK and ONCHANGE. The subscription mechanisms come also handy when requesting context information in terms of a continous data stream. A client subscribes for the stream using a context query, and whenever information satisfying the query is available it is notified. As the context information is represented in XML, the actual data is included in the body of an element, whereas the attributes of the element provide additional information as e.g. the position in the stream.

The proposed CQL is also capable to deal with the common characteristics of context information. With the help of the *timerange* tag clients can query for current and also past context information. Therefore, its temporal nature is considered, and dynamic context information can be handled as well. The underlying context modeling approach is not limited to certain *context scopes*. Rather, it is highly extensible as all concepts defined in the ontology can be requested. Therefore, the new CQL is inherently able to handle concepts like location, proximity and spatial relationships. As we are not constrained to certain representations, defining custom representations in the ontology is allowed, and support is provided to include arbitrary meta-data in the context. These meta-data can also be referenced in a query to construct elaborate filtering mechanisms.

One of the main advantages of our approach is also the seamless incorporation of an ontology. With the use of semantic references, reasoning on context data is supported and heterogeneity is explicitly addressed. The CML system supports powerful reasoning on context information, but ontologies have been proposed only to be included to cover some special aspects, as e.g. privacy issues. Furthermore, heterogeneity is not addressed at all. The RDF-based MoGATU system allows the semantical description of context and reasoning about this information. However, it does not support heterogeneity unlike the proposed CQL which allows the description of context entities with different representations and the definition of associated Inter-Representation-Operations. The proposed CQL also facilitates the request for context information in a specific representation. The underlying query processing system automatically performs the necessary conversions between different representations. A further shortcoming of existing approaches is the lack of appropriate sets of aggregation functions. We not only provide several predefined aggregation functions like average, sum, max or min, but we also provide support to define more elaborate aggregation functions and their groundings in the ontology.

As the proposed CQL does not refer to specific instances of context sources, it is well suited for a dynamic pervasive computing environments. Context sources may be locally deployed or remotely accessible, and context sources can appear or disappear. These characteristics - autonomy, mobility and distribution - are also explicitly supported by the corresponding context management system of MUSIC.

In conclusion, we can summarize that the proposed CQL satisfies all requirements introduced in section 2 in a single context query language. The main differences to CML system and the RDF-based MoGATU system are its integrated support for all characteristics of context information, support for context reasoning based on semantic knowledge represented using ontologies, heterogeneity and elaborate aggregation functions.

6. Conclusions and future work

This paper has presented a new CQL for mobile and pervasive computing. The requirements were derived from the analysis of related work as well as from case studies in the MUSIC project. As existing CQLs fail to satisfy all the identified requirements sufficiently, we developed a new CQL. It is shown that this CQL significantly improves the state of the art by explicitly addressing heterogeneous representations of context information, supporting the definition of complex filtering mechanisms, allowing the incorporation of elaborate aggregation functions and enabling the use of knowledge represented through an ontology.

In our future work we will further improve and finalize the structure of the CQL, and we will integrate it into the MUSIC context management middleware. As part of the MUSIC project, the new context management system and its query language will be field tested in several case study applications.

Acknowledgements

The authors of this paper would like to thank their partners in the MUSIC-IST project and acknowledge the financial support given to this research by the European Union (6th Framework Programme, contract number 35166).

References

[1] C. Bettini, D. Maggiorini, D. Riboni "Distributed Context Monitoring for the Adaptation of Continuous Services", World Wide Web Journal, Vol. 10 No. 4 Springer, 2007, pp. 503-528.

[2] P. D. Haghighi, A. Zaslavsky, S. Krishnaswamy, "An Evaluation of Query Languages for Context-Aware Computing", 17th International Conference on Database and Expert Systems Applications (DEXA), Krakow, Poland, September 2006, pp. 455-462.

[3] F. Perich, A. Joshi, T. Finin, Y. Yesha, "On Data Management in Pervasive Computing Environments", IEEE Transactions on Knowledge and Data Engineering, Vol. 16 No. 5, May, 2004, pp. 621-634

[4] K. Henricksen, J. Indulska, "A Software Engineering Framework for Context-aware Pervasive Computing", 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04), Orlando, Florida, March 2004, IEEE, pp.77-86.

[5] K. Henricksen, J. Indulska, "Developing context-aware pervasive computing applications: Models and approach", Journal of Pervasive and Mobile Computing, Vol. 2, No. 1, Elsevier, February 2006, pp. 37-64.

[6] K. Henricksen, J. Indulska, "Modelling and using imperfect context information", 1st Workshop on Context Modeling and Reasoning (CoMoRea) in conjunction with the 2nd IEEE International Conference on Pervasive Computing and Communication (PerCom'04), March 14, 2004, Orlando, Florida, IEEE Computer Society, p. 33-37.

[7] T. MacFadden, K. Henricksen, J.Indulska, "Automating Context-aware Application Development", 1st International Workshop on Advanced Context Modelling, Reasoning and Management (UbiComp 2004), Nottingham, UK, September 2004, pp. 90-95.

[8] F. Perich, A. Joshi, Y. Yesha, T. Finin, "Collaborative Joins in a Pervasive Computing Environment", The International Journal of Very Large Databases, Vol. 14, No. 2, April 2005, Springer Verlag, pp 182-196.

[9] European IST-FP6 project SPICE (Service Platform for Innovative Communication Environment), http://www.ist-spice.org/

[10] European IST-FP6 project MobiLife, http://www.ist-mobilife.org/

[11] European IST-FP6 project MUSIC (Self-adapting applications for Mobile Users In ubiquitous Computing environments), http://www.ist-music.eu

[12] N. Paspallis, A. Chimaris, G. A. Papadopoulos, "Experiences from Developing a Context Management System for an Adaptation-enabling Middleware", 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Paphos, Cyprus, June 5-8, 2007, Springer LNCS 4531, pp. 225-238

[13] T. Strang, C. Linnhoff-Popien, K. Frank, "CoOL – A Context Ontology Language to enable Contextual Interoperability", 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003), Paris, France, November 2003, Springer LNCS 2893, pp. 236-247.

[14] E. Prud'hommeaux, A. Seaborne (editors), "SPARQL Query Language for RDF", W3C Candidate Recommendation, 14 June 2007 (http://www.w3.org/TR /rdf-sparql-query/).