An XML-Based Model for Supporting Context-Aware Query and Cache Management

Essam Mansour and Hagen Höpfner

International University in Germany
School of Information Technology
Campus 3, D-76646 Bruchsal, Germany
essam.mansour@ieee.org, hoepfner@acm.org

Abstract. Database systems (DBSs) can play an essential role in facilitating the query and cache management in context-aware mobile information systems (CAMIS). Two of the fundamental aspects of such management are update notifications and context-aware query processing. Unfortunately, DBSs does not provide a built-in update notification function and are not aware of the context of their usage. This paper presents an XML model called XREAL (XML-based Relational Algebra) that assists DBSs in extending their capabilities to support context-aware queries and cache management for mobile environments.

1 Introduction

Database systems (DBSs) can play an essential role in facilitating the advanced data management required to modern information systems, such as context-aware mobile information systems (CAMIS). Usually, this advanced data management is provided by adding middle-wares over DBSs. Update notifications [8] and context-aware query processing [9] are part of the fundamental management aspects in CAMIS. Unfortunately, DBSs does not provide a built-in update notification function and are not aware of the context of their usage.

The main focus of this paper is to provide a model, which could be directly integrated into existing DBSs. One of the main requirements for this model is to be realized within DBSs in a way, which assists in extending DBSs capabilities to support cache management and the processing of context-aware queries as built-in DBS functions. Such extension is to reduce the code-complexity and increase the performance of CAMIS due to avoiding the middle-wares.

This paper presents a model called XREAL (XML-based Relational Algebra) that supports context-aware queries and cache management in CAMIS. The XREAL model provides XML representation for the contextual information of the mobile clients, queries issued by these clients and manipulation operations. This XML representation is to be stored as XML documents in modern DBSs that provide XML management support, such as DB2 and Oracle.

The rest of this paper is organized as follows. Section 2 highlights related work. Section 3 outlines the context-aware services and cache management. Sections 4, 5 and 6 present the three sub-models of XREAL. Section 7 presents our DBS-based implementation for XREAL. Section 8 outlines evaluation results. Section 9 concludes the paper.

A.P. Sexton (Ed.): BNCOD 2009, LNCS 5588, pp. 107–119, 2009.

2 Related Work

In caching techniques, query results are stored and indexed with the corresponding query [6]. So, one can analyze whether new queries can be answered completely or partially from the cache [10]. Semantic caches tend to use semantic correlations between different cached items as a cache replacement criteria [1]. However, updating the base table is ignored in these works. Our proposed model supports the cache maintenance as a DBS built-in function.

Different research efforts, such as [4,7], investigated into the topic of XML algebra. Our proposed model, XREAL, is distinguished by providing an XML representation for relational algebra queries. XREAL is to extend relational algebra to support context-aware operators.

3 Query and Cache Management

This section discusses the management requirements for supporting advanced context aware services. These services support mobile service providers to be more receptive to mobile users needs. The management requirements are classified mainly into two categories query and cache requirements.

3.1 Advanced Context Aware Services

In our proposed context aware services, we assume a mobile service provider (*MSP*) is to prepare for their customers contextual information document. We utilized the classification of contexts in the ubiquitous computing environment proposed by Korkea-Aho in [5]. This document divides the contextual information into several contexts, *physical*, *environmental*, *informational*, *personal*, *social*, *application*, and system. For the *physical* context, *MSP* detects the location of a mobile user (*MS*) and the corresponding time of such location. The *environmental* context includes information related to the current location of *MS* or locations of interest, such as user's home and work location. Examples for information related to this context are a traffic jam, parking spots, and weather.

A mobile user might ask *MSP* to consider interesting business quotes in the user's *informational* context. For example, the user might need to know the newspaper's name, date, section, and page number combined with the quote value. The *personal* context records the user's personal plans, such as plans for working days, and hobbies, such as drinking coffee or visiting new food shops. The *social* context includes information about social issues, such as the user's kids, neighbors and social activities. The *application* context includes information concerning used applications, such as email accounts of the user in order to provide notification by received emails. Finally, the *system* context records information concerning systems used by the user, such as heating system and water supply. The users expects a very high level of security and privacy for their contextual information. The mobile users are supposed to issue ad-hoc or pre-register queries. These queries might be continuous or non-continuous queries.

A	B
$q : \{\pi\|\pi^a\}([\sigma]([\rho](R)))$ $q : \{\pi\|\pi^a\}([\sigma](\rho(q)))$ $q : \{\pi\|\pi^a\}([\sigma](cp))$ $cp : \{[\rho](R)\|\rho(q)\} \times \{[\rho](S)\|\rho(q)\|cp\}$ $q : \{\pi\|\pi^a\}(q\{\cup\|-\|\cap\}q)$	$QS \leftarrow_{\pi} {}_{ShopName,tele,PID}(\sigma_{status='NEW'}(shop))$ $QL \leftarrow_{\pi} {}_{street,ID}(\sigma_{postal\_code=76646}(location))$ $Q \leftarrow_{\pi} {}_{ShopName,tele,street}(\sigma_{PID=ID}(QS \times QL))$

Fig. 1. A) the relational algebra recursive structure; B) A relational algebra of *NCQ*

3.2 Query Representation

In mobile information systems, applications generate queries and send them to the server. Therefore, there is no need to support descriptive query languages, such as SQL. Queries are to be represented in a useful way for storage and retrieval. The relational algebra representation [2] is an efficient way to represent queries over data stored in relational database. However, one can always translate SQL-queries into such expressions.

The query notation used in this paper is the notation of the relational algebra operators [2], such as selection (σ), θ-join ($\bowtie_\theta$), and projection (π). The θ-join is represented in our work as Cartesian product ($\times$) and selection (σ). It is possible to optimize the transformation into query trees in order to improve the support for query indexing by reducing the number of alternatives.

In general and formally, a database query q can have the recursive structure shown in Figure 1.A. A relational algebra query can be interpreted as a tree. However, there are several semantically equivalent relational algebra trees for each query. We, for the purpose of detecting update relevance [8], push selection and join operations inside a join using the algebraic properties for query optimization [2]. Then, we convert the θ-join to Cartesian product ($\times$) and selection (σ) to be in the form shown in Figure 1.A.

The query *NCQ* that retrieves the attributes (ShopName, Tele and *street*) of shops, whose status is *NEW* and postal_code is the postal_code of the user's current position. Assume that the user was in the area, whose postal_code is 76646, when the user requested the result of the query *NCQ*. The tables *shop* and *location* are joined using the attributes *PID* and *ID*. The query Q shown in Figure 1.B is an example of a relational algebra query for the query *NCQ*. The query Q is represented using the recursive structure shown in Figure 1.A.

3.3 Cache Management

Managing redundant data is a must for mobile information systems. However, redundancies might also lead to inconsistencies. Replication techniques use synchronization approaches but tend to strongly restrict the language used for defining replicas. Caches, on the other hand, are created by storing queried data implicitly. In [3,8] we discussed how to calculate the relevancy of server side updates for data cached on mobile devices and how to invalidate outdated caches. Therefore, the server must keep track of the queries posted by the mobile clients. Hence, the DBS must maintain a query index that indexes the caches and the clients. Each server update has to be checked against this index. All clients that hold updated data must be at least informed by such update.

4 Formalizing Contextual Information

This section presents the overall structure of the XREAL *mobile client* sub-model and examples for the sub-model.

4.1 The Structure

The *mobile client* sub-model consists of an identification attribute, called *MCID*, and a sequence of elements (*physical context, environmental context, informational context, personal context, social context, application context,* and *system context*). Figure 2.A shows the XML schema of *mobile client* at an abstract level.

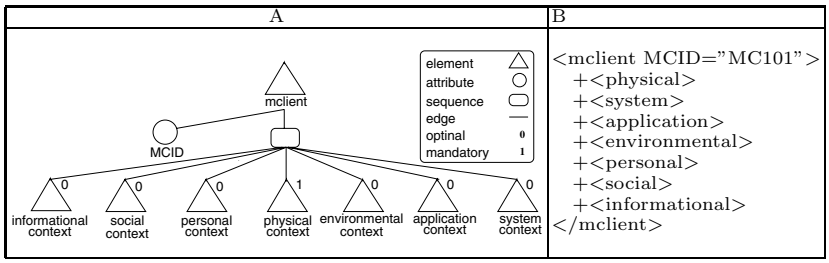

Fig. 2. A) the XREAL query model; B) the XREAL contextual information document

Any *mobile client* is assigned a *MCID* number, to be recognized by the system. *Physical context* provides information related to location and time. The location is a position, elevation, and direction. The position could be represented using a geographical coordinates and/or relative coordinates, such as a street, area and city. The time represents time zone, which could be inferred from the location information. The time zone determines the absolute time, day, week, month, quarter, and year. *Physical context* might help to infer information at a generic level related to *environmental context*, such as weather and light. Other methods are needed to determine an accurate environmental information.

Informational context formalizes information of interest to the mobile client, such as currency rates, stoke quotes and sports scores. *Personal context* specifies information such as health, mood, biographical information, habit and activities. *Social context* formalizes information concerning group activity and social relationships. *Application context* models information, such as email received and websites visited. The *system context* represents information related to systems used by the client and specs of her mobile, such as processor, and memory.

The user of a mobile client might provide personal and social information to be recorded as contextual information related to her mobile client. It is assumed that the minimum level of information is the information of *physical context*. So, the *physical context* element is a mandatory element. However, the other elements are optional. Furthermore, it is assumed that there is a repository of contextual information related to the environment, in which mobile clients are moving, such as parking spots or food shops.

4.2 Examples

Figure 2.B shows the contextual information document specified using XREAL that is generated by a *MSP* for one of its customers as discussed in Section 3. The contextual information document is assigned *MC101* as an ID. The XML language is very flexible in representing variety of contextual information. Figure 3 depicts part of the *physical* and *informational* contexts. Figure 3.A shows a representation for information of the relative position, and Figure 3.B illustrates a representation for business information as a part of informational context.

A	B
<relative_position> <country>Germany</country> <city>Bruchsal</city> <area>south</city> <street>Durlacher<street> <postal_code>76646</postal_code> </relative_position>	<quote> +<value> +<newspaper> +<section> +<date> +<description> </quote>

Fig. 3. A) part of the physical context, B) part of the informational context

5 Formalizing Queries

This section presents the XREAL sub-model for formalizing a relational algebra query based on the recursive structure discussed in Section 3.

5.1 Fundamental Elements

The XREAL model formalizes a relational algebra query as a *query* element that consists of two attributes, *QID* and *MCID*, and a sequence of elements, *relations*, *projection* and *join*. Figure 4.A shows the XML schema of XREAL *query*. The *QID* attribute represents a query identification. The *MCID* attribute represents the identification number of a mobile client that issued the query. A query might access only one relation. Therefore, a *query* element contains at least a *relations* element and *projection* element, and might has a *join* element. The *query* sub-model provides a formalization for queries represented as discussed in Section 3.

The *relations* element is composed of a sequence of at least one *relation* element. The *relation* element consists of an identification attribute, called *RID*, and a sequence of elements, *name, rename, selections* and *rprojection*. The *name* element represents the relation name. The *rename* element denotes the temporally name used to refer to the relation in the query. The *selection* element is composed of a sequence of a *spredicate* element of type *predicateUDT*. The *rprojection* element consists of a sequence of at least one *attribute* element of type *attributeUDT*. The *predicateUDT* type is a complex type that is able to represent simple predicate or composite predicate. The *projection* element is similar to the *rprojection* element, but *projection* represents the original projected attributes used in the query. The *join* element specifies the join predicates used to join together the relations (sub-queries).

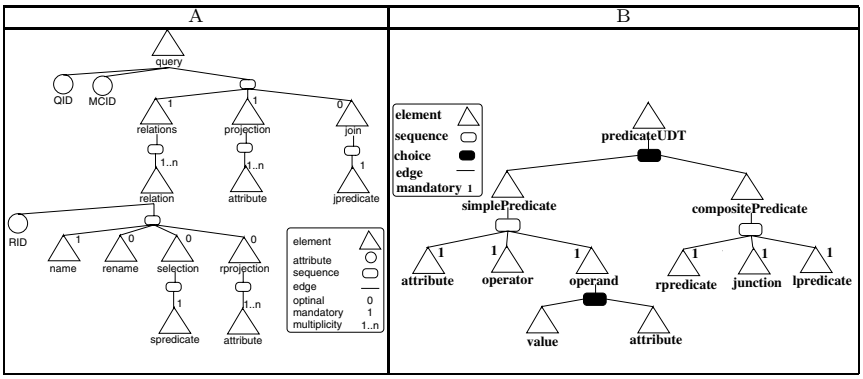

Fig. 4. A) the XML Schema of the query model; B) the predicateUDT Schema

The *query* sub-model is able to represent a query, such as the query Q shown in Figure 1.B. Th query Q consists of two sub-queries QS and QL. The specification of the query Q is to contain the elements *relations*, *projection* and *join*. Each sub-query is to be represented as a *relation* element that has its own *selection* and *rprojection* element. The *projection* element represents the original projection of the query. The *join* element represents the join predicate that joins QS and QL.

5.2 User Defined Data Types

The *query* sub-model has two main user defined data types (UDT), *predicateUDT* and *attributeUDT*. The *predicateUDT* type is a complex type composed of one of the elements *simplePredicate* or *compositePredicate*, as depicted in Figure 4.B. The *simplePredicate* element consists of a sequence of elements, *attribute, operator* and *operand*. The *attribute* element is of type *attributeUDT*. The *operator* element is of type *logicalOperatorUDT*, which is a simple type that restricts the token datatype to the values (*eq, neq, lt, lteq, gt,* and *gteq*). Respectively, they refer to equal, not equal, less than, less than or equal, greater than, and greater than or equal. The *operand* element is composed of one of the elements *value* or *attribute*. The *value* element is to be used with selection predicates. The *attribute* element is to be used with join predicates.

The *compositePredicate* element consists of a sequence of elements, *rpredicate, junction* and *lpredicate*. The *rpredicate* and *lpredicate* elements are of type *predicateUDT*. Consequentially, the *rpredicate* and *lpredicate* elements might consist of simple or composite predicate. The *junction* element is of type *junctionUDT*, which is a simple type that restricts the token datatype to the values (*and* and *or*). The *attributeUDT* type is a complex type composed of an attribute, called *ofRelation*, and a sequence of elements, *name* and *rename*. The *ofRelation* attribute represents a relation ID, to which the attribute belongs. The *name* element denotes the name of the attribute. The *rename* element represents the new name assigned to the attribute in the query.

```
<query QID="QID1" MCID="MC101">
  <relations>
    +<relation RID="RID01">
    +<relation RID="RID02">
  </relations>
  +<projection>
  <join>
    <jpredicate>
      <simplePredicate>
        <attribute ofRelation="RID01">
          <name>PID</name>
        </attribute>
<operator>eq</operator>
<operand>
  <attribute
   ofRelation="RID02">
    <name>ID</name>
  </attribute>
</operand>
</simplePredicate>
</jpredicate>
</join>
</query>
```

Fig. 5. The specification of *NCQ*

```
<relation RID="RID01">
  <name>shop</name>
  <selection>
    <spredicate>
      <simplePredicate>
        <attribute>
          <name>status</name>
        </attribute>
        <operator>eq</operator>
        <operand>
          <value>'NEW'</value>
        </operand>
      </simplePredicate>
    </spredicate>
  </selection>
<projection>
  <attribute>
    <name>ShopName</name>
  </attribute>
  <attribute>
    <name>tele</name>
  </attribute>
  <attribute>
    <name>PID</name>
  </attribute>
</projection>
</relation>
```

Fig. 6. The specification of the relation RID01

5.3 An Example

Figure 5 illustrates an overview of the XREAL specification for the query *NCQ*, whose relational algebra expression is shown in Figure 1.B. This specification consists of a *query* element. The query ID is *QID1* and is issued by a mobile client, whose ID is *MC101*. There are two sub-queries over the relations (*shop* and *location*), which are joined together using one join predicate.

Figure 6 illustrates the XREAL specification of the sub-query *QS*, which queries the relation (*shops*). The ID of the relation is *RID01*. This specification consists of a *relation* element, whose name is *shop*. The selection predicate associated with *shop* checks that the shop's status is equal to *NEW*. There is also a projection operation that picks the attributes (*ShopName,tele* and *PID*).

6 Formalizing Manipulation Operations

The following sub-sections present the structure of *moperation* and examples.

6.1 The Structure

A manipulation operation might be an insert, delete or update operation. Figure 7 shows the XML schema of the *moperation* component, which might consists of one *IStatement, DStatement*, or *UStatement*. The *IStatement* element

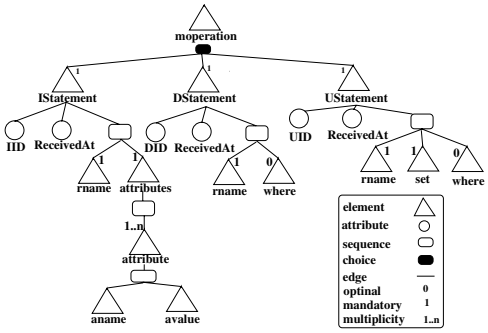

Fig. 7. The XML Schema of the manipulation operations

```
<IStatement IID="I3001"
receivedAt="2008-09-12T11:34:27">
  <rname>shop</rname>
  <attributes>
    <attribute>
      <aname>SID</aname>
      <avalue>9905</avalue>
    </attribute>
    <attribute>
      <aname>
      SHOPNAME</aname>
      <avalue>MEMO</avalue>
    </attribute>
    <attribute>
      <aname>PID¡/aname>
      <avalue>102¡/avalue>
    </attribute>
```

```
<attribute>
  <aname>TELE</aname>
  <avalue>111333888</avalue>
</attribute>
<attribute>
  <aname>RATE</aname>
    <avalue>7</avalue>
</attribute>
<attribute>
  <aname>
  STATUS</aname>
  <avalue>NEW</avalue>
</attribute>
</attributes>
</IStatement>
```

Fig. 8. The specification of ***MO1***

consists of attributes, *IID* and *ReceiveAt*, and a sequence of elements, *rname* and *attributes*. The *rname* element represents the name of the manipulated relation. The *attributes* element represents the attributes of the inserted tuple and the corresponding value for each attribute.

The *DStatement* element consists of attributes, *DID* and *ReceiveAt*, and a sequence of elements, *rname* and *where*. The *where* element is of type *predicateUDT*. The *UStatement* element consists of attributes, *UID* and *ReceiveAt*, and a sequence of elements, *rname, set* and *where*. The *where* element is of type *predicateUDT*. The *set* element is of type *simplePredicate* and restricted to use an equal operator only. Assume the update statement modifies only one attribute.

6.2 Examples

It is assumed that the server is to execute several manipulation operations over the *shop* table. The first operation (MO1) inserts a new shop, whose id, name, tele, rate and status are 9905, MEMO, 111333888, 7 and NEW, respectively. This shop is located in Karlsruhe, whose position ID is 102. The second operation (MO2) deletes a shop tuple, whose id is 9903.

```
<DStatement DID="D5001"
receivedAt="2008-09-12T11:34:27">
<rname>shop</rname>
<where>
  <spredicate>
    <simplePredicate>
      <attribute>
        <name>SID</name>
      </attribute>
      <operator>eq</operator>
      <operand>
        <value>9903</value>
      </operand>
    </simplePredicate>
  </spredicate>
</where>
</DStatement>
```

Fig. 9. The specification of ***MO2***

```
<UStatement UID="U7001"
  receivedAt="2008-09-12T11:34:27">
  <rname>shop</rname>
  <set>
    <spredicate>
      <simplePredicate>
        <attribute>
          <name>RATE</name>
        </attribute>
        <operator>eq</operator>
        <operand>
          <value>7</value>
        </operand>
      </simplePredicate>
    </spredicate>
  </set>
<where>
  <spredicate>
    <simplePredicate>
      <attribute>
        <name>PID</name>
      </attribute>
      <operator>eq</operator>
      <operand>
        <value>103</value>
      </operand>
    </simplePredicate>
  </spredicate>
</where>
</UStatement>
```

Fig. 10. The specification of ***MO3***

The third operation (MO3) updates the rate of the shop tuples, which is located at 103, to seven. Figure 8 illustrates the XREAL specification for the insert operation. *IStatement* of the insert operation consists of attributes, *IID* whose value is *I3001* and *receivedAt* that determines the receipt time. There are six elements of type *attribute* that specify the name and value of an attribute, such as *SID* and *9905* for the first attribute of the insert statement. Figure 9 illustrates the XREAL specification for the delete operation. *DStatement* of the delete operation consists of attributes, *DID* whose value is *D5001* and *receivedAt* that determines the receipt time.

There is a *where* element under *DStatement* that formalizes the where clause of the delete statement, which is *SID = 9903*. Figure 10 illustrates the XREAL specification for the update operation . *UStatement* of the update operation consists of attributes, *UID* whose value is *U7001* and *receivedAt* that determines the receipt time. The *set* element formalizes the set clause of the update statement, which is *RATE = 7*. The *where* element of *UStatement* formalizes the where clause of the update operation, which is *PID = 103*.

7 Realizing the XReAl Model within DBSs

Modern DBSs support XML data management by providing an XML data type with storage and retrieval support. The XREAL specifications are represented as well-formed XML documents that could be stored in an attribute of XML type. This XML document could be validated against an XML Schema.

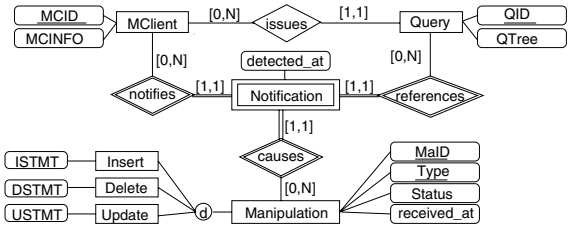

Fig. 11. The ER diagram of the XReAl Repository

7.1 The XReAl Repository Structure

The XReAl repository is based on a relational database schema, in which XML type is supported to store well-formed and validated XML documents. Figure 11 depicts the database schema of the XReAl repository. The schema consists of four fundamental relations, *mclient, query*, *manipulation*, and *notification*.

The relations, *mclient* and *query*, consist of a primary key attribute (*MCID* and *QID*) and an attribute of XML type (*MCINFO* and *QTree*). Each manipulation operation has an identification number and is classified into three types, *insert, delete*, and *update*. The attributes *MaID* and *Type* store the identification number and the type of the manipulation operation. Both attributes represent the primary key of the relation. Manipulation operations are classified also into two status new (N) or tested (T) operations. The *Status* attribute represents the status of an operation. The time at which the operation is received is to be stored into the *received_at* attribute. The *ISTMT, DSTMT* and *USTMT* attributes are of XML type and store XML documents representing XReAl specification for *insert, delete* or *update* operations respectively. The content of the attributes of XML type is to be validated by the XML schema of the XReAl model.

7.2 Repository Maintenance

The XReAl specifications are to be maintained (modified or queried) as any other XML documents using XQuery scripts. Modern DBSs provide means for maintaining the XML documents. In particular, DB2, which was adopted in this research, supports both the XQuery language and the XQuery update facilities provided by W3C. Moreover SQL/XML language is also supported by DB2. Such language provides support for querying the relational data of the application and the XReAl specifications. That assists in providing a unified management environment within the DBS for *CAMIS*.

8 Evaluation

We have utilized DB2 Express-C 9.5 and the Sun Java 1.6 language to implement XReAl, and built-in functions within DB2 for update notification and the context-aware query processing. This section outlines the fundamental ideas of these functions, and shows our empirical results of the update notification.

8.1 Update Notification

Based on the XREAL model, we have developed DBS-built-in method [8] that detects the relevance of manipulation operations over multi-set semantics of the relational data model. The modified data is specified by a manipulation operation, which is formalized using XREAL. Also, the cached data is specified by a query, which is formalized using XREAL.

The main idea of detecting the relevancy of an operation is to check the intersection between the specifications of the operation and query. A non-empty intersection means that there is cached data affected by the operation. Consequentially, the operation is a relevant update. For more details concerning our method for update notification based on XREAL, the reader is referred to [8].

As shown in Figure 11, a manipulation operation might cause notification(s) to be sent to mobile clients issuing queries, whose cached result intersects with data affected by the manipulation operation. The *notification* relation shown in Figure 11, consists of the attributes, *MCID, QID*, (*MaID, Type*) and *detected_at* that represents the time at which the notification is detected. The tuples of the *notification* relation are to be inserted as a result of testing the intersections between cached and modified data.

8.2 Context-Aware Query Processing

The XREAL specifications of the contextual information is the base for processing any context-aware query. Our main idea is to represent the context-aware semantics using relational algebra operations. The specifications of the contextual information and a query is used to generate an instance of this query according to the current context(s) of the user, who issued this query. This instance is generated by replacing relatives attributes with its corresponding values from the context of the user.

The query *NCQ* shown in Figure 1.B is an example for such process. Figures 5 and 6 show part of the specification of the instance query. Finally, a SQL query is generated from such instance and executed using the DBS, which at the same time manages the relational data of the application, in this case shops database. Our context-aware query processor is in-progress.

We are implementing the processor as built-in DBS function supported with Java-stored procedures. Currently, we are supporting context-aware queries based on location specified using relative position, such as postal_code. More advanced context-aware functions, such as *close to* and *towards*, are to be supported.

8.3 Experimental Results

Our experiments were done on a standard PC running Ubuntu 8.04 Linux (Intel(R) Core(TM)2 Duo CPU @ 2.20 GHz with 2 GB of RAM). Figure 12 illustrates the time consumption for registering queries on the server and for checking the relevance of insert, update and delete operations. As shown in Figure 12, our method is scalable to the number of queries registered in the systems. Moreover, the maximum required time for checking the relevancy of a manipulation operation to 16,384 related queries is approximately 50 seconds.

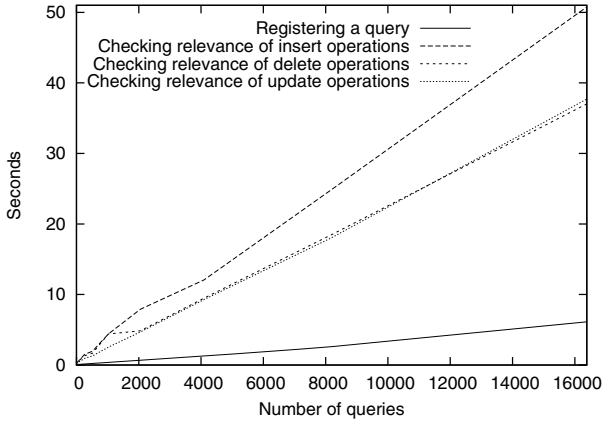

Fig. 12. Evaluation of time consumption

9 Conclusions and Outlook

This paper has presented an XML model called XREAL (XML-based Relational Algebra) that assists DBSs in extending their capabilities to support context-aware queries and cache management for mobile environments. XREAL models the contextual information related to mobile clients, queries issued by these clients and manipulation operations, such as insert, delete and update.

The main advantages of XREAL are inherited from the use of XML, such as: 1) flexibility in exchange and sharing the XREAL specification, 2) high compatibility in representing relational algebra query trees, and 3) seamless integration of XREAL management into DBS. The third point leads to performance improvement due to avoiding several middle-wares introduced to support the advanced management of CAMIS, such as update notifications and context-aware query processing.

The presented work is part of a continuous research project aiming at developing a framework for advanced query and cache management in CAMIS based on DBSs. The development of our proposed context-aware query processor is in-progress. There is a need to extend relational algebra to represent context-aware functions, such as *close to*, *around*, *towards*, and *approaching*.

References

1. Dar, S., Franklin, M.J., Jónsson, B., Srivastava, D., Tan, M.: Semantic Data Caching and Replacement. In: Proc. of 22nd International Conference on Very Large Data Bases (VLDB 1996), September 1996, pp. 330–341. Morgan Kaufmann, San Francisco (1996)
2. Elmasri, R., Shamkant, B.N.: Fundamentals of Database Systems. Addison Wesley, Reading (2007)

3. Höpfner, H.: Relevanz von Änderungen für Datenbestände mobiler Clients. VDM Verlag Dr. Müller, Saarbrücken (2007) (in German)
4. Jagadish, H.V., Lakshmanan, L.V.S., Srivastava, D., Thompson, K.: TAX: A Tree Algebra for XML. In: Ghelli, G., Grahne, G. (eds.) DBPL 2001. LNCS, vol. 2397, pp. 149–164. Springer, Heidelberg (2002)
5. Korkea-Aho, M.: Context-aware applications survey. Technical report, Department of Computer Science, Helsinki University of Technology (2000)
6. Lee, K.C.K., Leong, H.V., Si, A.: Semantic query caching in a mobile environment. ACM SIGMOBILE Mobile Computing and Communications Review 3(2), 28–36 (1999)
7. Magnani, M., Montesi, D.: XML and Relational Data: Towards a Common Model and Algebra. In: IDEAS 2005: Proceedings of the 9th International Database Engineering & Application Symposium, pp. 96–101. IEEE Computer Society Press, Washington (2005)
8. Mansour, E., Höpfner, H.: An Approach for Detecting Relevant Updates to Cached Data Using XML and Active Databases. In: 12th International Conference on Extending Database Technology (EDBT 2009) (2009)
9. Mansour, E., Höpfner, H.: Towards an XML-Based Query and Contextual Information Model in Context-Aware Mobile Information Systems. In: The MDM Workshop ROSOC-M (2009)
10. Ren, Q., Dunham, M.H., Kumar, V.: Semantic Caching and Query Processing. IEEE Trans. on Knowl. and Data Eng. 15(1), 192–210 (2003)