A Context-Aware Preference Model for Database Querying in an Ambient Intelligent Environment

Arthur H. van Bunningen, Ling Feng, and Peter M.G. Apers

Centre for Telematics and Information Technology, University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands
`{bunninge, ling, apers}@cs.utwente.nl`

Abstract. Users' preferences have traditionally been exploited in query personalization to better serve their information needs. With the emerging ubiquitous computing technologies, users will be situated in an Ambient Intelligent (AmI) environment, where users' database access will not occur at a single location in a single context as in the traditional stationary desktop computing, but rather span a multitude of contexts like office, home, hotel, plane, etc. To deliver personalized query answering in this environment, the need for context-aware query preferences arises accordingly. In this paper, we propose a knowledge-based context-aware query preference model, which can cater for both *pull* and *push* queries. We analyze requirements and challenges that AmI poses upon such a model and discuss the interpretation of the model in the domain of relational databases. We implant the model on top of a traditional DBMS to demonstrate the applicability and feasibility of our approach.

1 Introduction

With the coming anytime/anywhere ubiquitous data access paradigm in an AmI environment, context plays an important role in information delivery and dissemination. Database and recommendation systems nowadays are more and more aware of the context while serving users' information needs. In this paper, we investigate user's query preferences in an AmI environment, taking their applicable contexts into account. Our design of the context-aware preference model is influenced and guided by the following AmI philosophies:

- *Smartness requirement.* The smartness requirement in an AmI environment implies reasoning and learning capabilities that the preference model must possess, calling for an inevitable knowledge ingredient. For example, a user may input a preference like *prefer a nearby restaurant when the weather is bad.* With the model, it should be able to infer the applicability of the preference no matter whether it rains or snows, since both are bad weather.
- *Proactiveness requirement.* Following the smartness requirement, the database systems in an AmI environment should proactively deliver anytime/anywhere useful information to their users. The designed context-aware query preference model should therefore provide sufficient flexibility and adaptiveness to the two access modes, namely *pull query* where users actively query databases to pull relevant information, and *push query* where the systems push proactively possibly relevant

S. Bressan, J. Küng, and R. Wagner (Eds.): DEXA 2006, LNCS 4080, pp. 33–43, 2006.

information to users (e.g., querying the background information about a person when s/he enters a room).

- *Closure requirement.* To support preference propagation and deduction, the model should preferably possess the closure property so that the output preference can serve as the input context of some other preferences. For instance, suppose a user has two preferences: "*prefer cheerful TV programs when having a bad mood*" and "*prefer channel 5 if looking for cheerful TV programs*" As a consequence, when this user has a bad mood, *cheerful TV programs on channel 5* will be the most preferable program alternatives for him/her.
- *Scalability requirement.* Performance is highly demanded at data management level to process real-time queries raised by different users anytime/anywhere in an AmI environment. The context-aware preference model must be easily interpreted and executed in a database world to achieve scalability.
- *Traceability requirement.* The behaviors of database querying systems in an AmI environment, and thus the preference model should be traceable by the users. In other words, it should be possible for a human to conveniently enter, view, and edit context-aware preferences in a way which is close to the world model of the users. An intuitive user-friendly interface for preference declaration is therefore needed.

In the following sections, we first review some closely related work in Section 2. We present our knowledge-based context modeling approach, followed by the context-aware query preference modeling using Description Logics in Section 3. We depict a framework for implanting this model on top of a traditional DBMS, and interpret the model in a relational database in Section 4. The implementation of the model in serving pushing queries is illustrated in Section 5. We conclude the paper in Section 6.

2 Related Work

The notion of preference query was first introduced to the database field in [1]. It extended the Domain Relational Calculus to express preferences for tuples satisfying certain logical conditions. Since its introduction, extensive investigation has been conducted, and two main lines of approaches have been formed in the literature to deal with users' preferences, namely, quantitative and qualitative [2]. The qualitative approach intends to directly specify preferences between the tuples in the query answer, typically using binary preference relations. An example preference relation is "*prefer one book tuple to another if and only if their ISBNs are the same and the price of the first is lower.*" These kinds of preference relations can be embedded into relational query languages through relational operators or special preference constructors, which select from their input the set of the most preferred tuples (e.g., winnow [2], PreferenceSQL BMO [3], and skyline [4]). The quantitative approach expresses preferences using scoring functions, which associate a numeric score with every tuple of the query. Then tuple t_1 is preferred to tuple t_2 if and only if the score of t_1 is higher than the score of t_2. A framework for expressing and combining such kinds of preference functions was provided in [5]. [6] presented a more rich preference model which can associate degrees of interest (like scores) with preferences over a database schema.

Recently, situated and context-aware preferences start to receive attentions due to the fact that user preferences do not always hold in general but may depend on underlying situations [7,8]. [7] used the ER model to model different situations. Each situation has an *id* and consists of one timestamp and one location. It can also have one or more influences (e.g., a personal and a surrounding influence). Such situations are linked with uniquely identified preferences through a m:n relation. An XML-based preference repository was developed to store and manage the situated preferences. [8] used a combination of context and non-context attribute-value pairs associated with a degree of interest to define a preference. The work reported in this paper distinguishes from these two studies in the following three aspects. First, our study targets at the Ambient Intelligent environment which has high demands on smartness, reasoning, proactiveness, traceability, etc. Driven by these requirements, we propose a knowledge-based context-aware preference model, where both contexts and preferences are treated in a uniform way using Description Logics. Second, we take both pull and push queries into consideration while designing our context-aware query preference model. Third, we interpret and implant this preference model upon a traditional DBMS.

3 A Knowledge-Based Context-Aware Query Preference Model

A context-aware query preference states, among a set of alternatives, a particular like or dislike for some of these alternatives under certain contexts, like "*prefer sporting TV programs when the user is dinning*", "*prefer TV programs of the human interest genre when the user is doing some free time activity with some friend(s) around*", etc. The term *context* here refers to the situation under which a database access happens. Our context-aware query preference model tightly couples a preference with its applicable contexts, and expresses both in a uniform way.

In the following, we describe our approach of modeling context, followed by the representation of context-aware query preferences.

3.1 Context Categorization and Modeling

We view context from two perspectives: *user-centric* and *environmental* [9]. Examples of user-centric contexts are: user's background (e.g., working area, friends, etc.), behavior (activity, intention, etc.), physiological state (temperature, heart rate, etc.), and emotional state (happy, sad, fear, etc.). Environmental contexts can be physical environment (e.g., location, time, humidity, etc.), social environment (e.g., traffic jam, surrounding people, etc.), and computational environment (e.g., surrounding devices, etc.).

In the literature, there exist several possibilities to model context. Most of them are surveyed in [10,11], where it is concluded that *ontology based languages* are preferable for context modeling. Driven by the reasoning/inference requirement as described in Section 1, we exploit a variant of Description Logics (DL) to represent context for several reasons. First, DL [12] is a (decidable) fragment of first order logic, and is especially suited for knowledge representation. It forms the basis of ontological languages such as OWL, which has been used to model context in [13]. Furthermore, there exist many tools for dealing with DL knowledge bases such as reasoners and editors. Finally, extensive research has been conducted to investigate the relationship between databases and DL, and map a DL knowledge base into database schemas [14].

As known, a DL knowledge base consists of a TBox and an ABox. The TBox (i.e., the vocabulary) contains assertions about *concepts* (e.g., Child, Female) and *roles* (e.g., hasRoom, hasActivity). The ABox contains assertions about *individuals* (e.g., ROOM3061). Concepts and roles can be either *atomic* or *constructed* using concept and role constructors *intersection* ($\sqcap$), *union* ($\sqcup$), and *complement* ($\neg$) (e.g., Child $\sqcap$ Female, hasRoom $\sqcap$ hasActivity). The *top concept* ($\top$) and *bottom concept* ($\bot$) denote respectively all individuals and no individuals. A role specific operator is the *role-inverse* which defines the inverse of a certain role (e.g., *roomOf* is the inverse of *hasRoom*, denoted as $hasRoom \equiv roomOf^{-1}$). Moreover, roles can have full and existential quantification (e.g., $\forall$ hasChild.Female denotes the individuals of whose children are all female, and $\exists$ hasChild.Female denotes the individuals having a female child). A *concept expression* contains a set of concepts and/or quantified roles which are connected via concept and role constructors. The basic inference on concept expressions in DL is *subsumption* $C \sqsubseteq D$, which is to check whether the concept denoted by D (the *subsumer*) is more general than the one denoted by C (the *subsumee*).

We use DL to describe a world model (i.e., ontology) upon which our context-aware query preferences can be founded. A small ontology example is given in Figure 1, where concepts are represented in CamelCase (e.g. *OfficeActivity*) , roles in lowerCamelCase (e.g. *hasRoom*), and individuals in all capital letters (e.g. *ROOM3061*).

We express diverse contexts of query preferences via DL concept expressions. For example, the DL concept expression $\{PETER\} \sqcap (\exists\, hasActivityType.FreeTimeActivity) \sqcap (\exists\, hasFriend(\exists\, hasRoom(\exists\, roomOf.\{PETER\})))$ describes such a context that user PETER is doing some free time activity with at least one friend in the same room.

$$
\begin{array}{rlrl}
Person & \sqsubseteq Thing \sqcap \forall\, hasRoom.Room & ActivityType & \sqsubseteq Thing \\
 & \sqcap \forall\, hasActivityType.ActivityType & FreeTimeActivity & \sqsubseteq ActivityType \\
 & \sqcap \forall\, hasFriend.Person & Relaxing & \sqsubseteq FreeTimeActivity \\
 & \sqcap \forall\, hasTvInterest.Genre & Sporting & \sqsubseteq FreeTimeActivity \\
Room & \sqsubseteq Location & Location & \sqsubseteq Thing \\
TVProgram & \sqsubseteq Thing \sqcap \exists\, hasGenre.Genre & Genre & \sqsubseteq Thing \\
hasRoom & \equiv roomOf^{-1} & hasTvInterest & \equiv tvInterestOf^{-1}
\end{array}
$$

Fig. 1. A simplified ontology example using DL

3.2 Context-Aware Query Preference Modeling

Beyond contexts, DL concept expressions also offer a natural way to convey information needs. For instance, the DL concept expression $TvProgram \sqcap (\exists\, hasGenre.\{HUMAN\text{-}INTEREST\})$ can be viewed as a query which selects all *TvProgram* individuals of the *HUMAN-INTEREST* genre. Therefore, in a similar fashion as context, we describe users' preferences through DL concept expressions. Formally, we define a **context-aware query preference** as a tuple of the form $(\mathcal{C}ontext, \mathcal{P}reference)$, where

$\mathcal{C}ontext$ and $\mathcal{P}reference$ are DL concept expressions. When a preference is applicable to any context, $\mathcal{C}ontext = \top$.

As an example, user PETER's context-aware query preference "*prefer TV programs of the human interest genre while doing some free time activity*" can be specified as:

$$Context : \{PETER\} \sqcap (\exists\, hasActivityType.FreeTimeActivity)$$
$$Preference : TvProgram \sqcap (\exists\, hasGenre.\{HUMAN\text{-}INTEREST\})$$

In comparison with this preference example, where the preferred genre (*HUMAN-INTEREST*) of TV programs is a constant, sometimes, a users preference varies with the concrete context. For instance, "*prefer TV programs of the common genre interests while with at least one friend in the same room*". In this case, the preferred genres of TV programs depend on whom the user is with at that moment. In this case, we use a variable v to denote it:

$$Context : \{PETER\} \sqcap (\exists\, hasFriend.((\exists\, hasRoom.(\exists\, roomOf.\{PETER\})) \sqcap v)$$
$$Preference : TvProgram \sqcap (\exists\, hasGenre.((\exists\, tvInterestOf.\{PETER\}) \sqcap (\exists\, tvInterestOf.v)))$$

We call this kind of preferences **variable context-aware query preference**, and the former **constant context-aware query preference**.

4 Implanting the Context-Aware Query Preference Model on Top of a DBMS

Context-aware query preferences can assist two kinds database accesses. 1) In the *pull* access mode, context-aware preferences can be used for query augmentation (e.g., enforcing the query constraint *genre="HUMAN-INTEREST"* to the user's query over TV programs when s/he is doing some free time activity with some friend(s) in the same room.) 2) In the *push* access mode, context-aware preferences can be used as query triggers (e.g., retrieving the background information about a person when s/he is nearby).

4.1 The Framework

Figure 2 shows the pull-push query execution framework equipped with the context-aware query preference model. It contains six major components. 1) The *context supplier* is responsible for supplying the current query context $Context_{cur}$. Some static contexts like user's background, friends, TV programs, etc. can be obtained from *context database*; while some dynamic contexts like user's location, emotion, traffic, surrounded people, etc. can be obtained from sensors or external service providers. 2) The *preference selector* selects from the *preference repository* relevant context-aware query preferences, if necessary by reasoning. A context-aware preference ($\mathcal{C}ontext$, $\mathcal{P}reference$) is related to a database access if ($Context_{cur} \sqsubseteq \mathcal{C}ontext$) and the preference $\mathcal{P}reference$ contains concepts which can be mapped to certain relations included in the user's query q_{user} (pull query) or included in the database (push query). 3) In the pull mode, the *query adaptor* augments user's query q_{user} with the relevant preference, and optimizes it further into q'_{user}. 4) In the push mode, the *query*

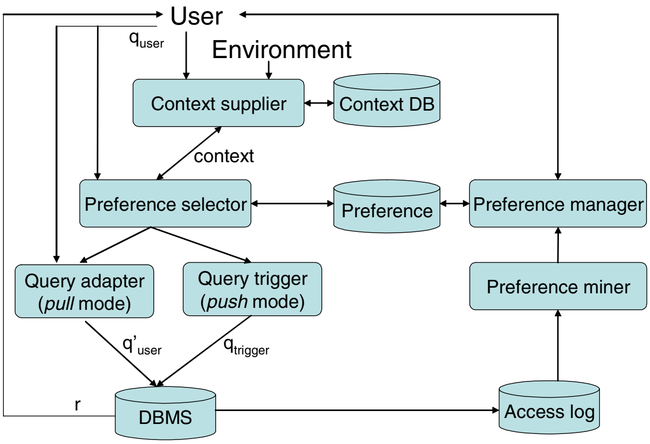


Fig. 2. The preference aware pull-push query framework

trigger proactively generates a query $q_{trigger}$ according to the preference, and sends it to the underlying DBMS for execution. 5) User-system interactions are recorded in the *access log*, from which the *preference miner* can discover users' context-aware preferences. Users can also directly input their preferences. 6) The *Preference manager* is responsible for storing, maintaining, and managing users' context-aware preferences.

4.2 Explicating the Context-Aware Query Preferences in a Database World

To integrate context-aware preferences with database queries, we need to provide a way which can explicate context-aware query preferences (including $\mathcal{C}ontext$ DL concept expression and $\mathcal{P}reference$ DL concept expression) in a database world[1].

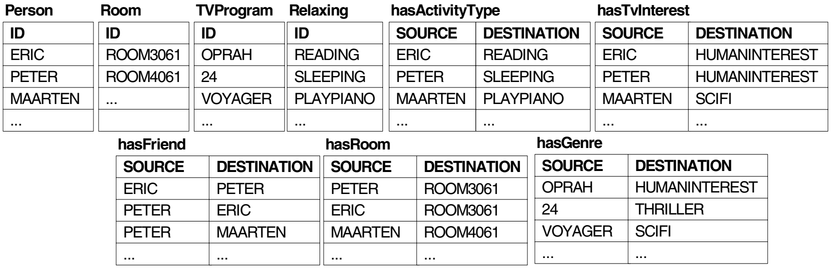
Since the basic elements of DL are concepts and roles, we propose to view each concept as a table, which uses the concept name as the table name and has one *ID* attribute. The tuples of the table correspond to all the individuals of the concept. A virtual table *TOPTABLE* contains all the individuals in the domain. Similarly, we view each role as a table, with the role name as its table name and containing two attributes *SOURCE* and *DESTINATION*. For each tuple of the table, the role relates the *SOURCE* individual with the *DESTINATION* individual. Figure 3 gives examples of this method.

We adapt the approach of [15] to express DL concept expressions using SQL queries (Table 1, where C, D, E are DL concepts, R is a DL role, and a is a DL individual).

An important remark here is that we provide a uniform tabular view towards both static and dynamic contexts, despite the later (e.g., location, surrounding people, etc.) must be acquired real-time from external sources/services like sensor networks. This is in line with the efforts of the sensornet community which has embraced declarative

---

[1] We focus on the relational data model in this study.



Person

ID
ERIC
PETER
MAARTEN
...

Room

ID
ROOM3061
ROOM4061
...

TVProgram

ID
OPRAH
24
VOYAGER
...

Relaxing

ID
READING
SLEEPING
PLAYPIANO
...

hasActivityType

SOURCE	DESTINATION
ERIC	READING
PETER	SLEEPING
MAARTEN	PLAYPIANO
...	...

hasTvInterest

SOURCE	DESTINATION
ERIC	HUMANINTEREST
PETER	HUMANINTEREST
MAARTEN	SCIFI
...	...

hasFriend

SOURCE	DESTINATION
ERIC	PETER
PETER	ERIC
PETER	MAARTEN
...	...

hasRoom

SOURCE	DESTINATION
PETER	ROOM3061
ERIC	ROOM3061
MAARTEN	ROOM4061
...	...

hasGenre

SOURCE	DESTINATION
OPRAH	HUMANINTEREST
24	THRILLER
VOYAGER	SCIFI
...	...

Fig. 3. Role and concept tables

Table 1. Mapping DL concept expressions to SQL query expressions

DL	SQL
C	SELECT ID FROM C
a	VALUES ('a')
$\top$	(SELECT ID FROM TOPTABLE)
$\bot$	NULL
$\neg D$	(SELECT ID FROM TOPTABLE) EXCEPT (SELECT ID FROM D)
$D \sqcap E$	(SELECT ID FROM D) INTERSECT (SELECT ID FROM E)
$D \sqcup E$	(SELECT ID FROM D) UNION (SELECT ID FROM E)
$\exists R.D$	SELECT R.SOURCE FROM R WHERE R.DESTINATION IN (SELECT ID FROM D)
$\forall R.D$	(SELECT ID FROM TOPTABLE) EXCEPT (SELECT R.SOURCE FROM R WHERE DESTINATION IN ((SELECT ID FROM TOPTABLE) EXCEPT (SELECT ID FROM D)))

queries as a key programming paradigm for large sets of sensors [16]. Here, we take the SQL query language as a uniform interface to the contexts. Another reason for doing this is that AmI imposes the need for storing context histories, which can be explored later for analysis purpose to achieve smartness [17].

With the mapping mechanism in Table 1, we can construct an SQL query for the DL concept expression $\mathcal{C}ontext$ in ($\mathcal{C}ontext$, $\mathcal{P}reference$). A non-empty query result implies that the current context includes/complies with $\mathcal{C}ontext$. The associated $\mathcal{P}reference$ is then activated for either query adaptation (when a user's pull query contains table(s) which is/are specified in $\mathcal{P}reference$) or query trigger (where $\mathcal{P}reference$ is translated into SQL as a proactive push query).

As an example, suppose a user raises a pull query for TV programs.

```
SELECT ID FROM TvProgram
```

The context-aware preference

$$\begin{aligned} Context\ &: \{PETER\} \sqcap (\exists\, hasFriend.(\exists\, hasRoom.(\exists\, roomOf.\{PETER\}))) \\ Preference\ &: TvProgram \sqcap (\exists\, hasGenre.\{HUMAN\text{-}INTEREST\}) \end{aligned}$$

contains a concept *TvProgram* which appears in the query. Its *Context* expression is translated straightforward into:

```
SELECT * FROM ((VALUES ('PETER'))
INTERSECT
  (SELECT hasFriend.SOURCE FROM hasFriend
   WHERE hasFriend.DESTINATION IN
      (SELECT hasRoom.SOURCE FROM hasRoom
       WHERE hasRoom.DESTINATION IN
         (SELECT roomOf.SOURCE FROM roomOf
          WHERE roomOf.DESTINATION IN (VALUES ('PETER')))
   ))) AS contexttable
```

which can then be optimized into:

```
SELECT *
FROM hasFriend, hasRoom, roomOf
WHERE hasFriend.SOURCE = 'PETER' AND
      hasFriend.DESTINATION = hasRoom.SOURCE AND
      hasRoom.DESTINATION = roomOf.SOURCE AND
      roomOf.DESTINATION = 'PETER'
```

Note that to execute the above query, some reasoning is needed based on the knowledge that *roomOf* is the inverse of *hasRoom*, and *FreeTimeActivity* embraces *Relaxing*, *Sporting*, etc.

When the query returns a non-empty result, the original pull query will be augmented with the additional constraint in the WHERE clause:

```
SELECT ID FROM TvProgram
WHERE ID IN
  (SELECT hasGenre.SOURCE FROM hasGenre WHERE hasGenre.DESTINATION IN
         (VALUES ('HUMAN INTEREST')))
```

5 Implementation

We implement the context-aware preference model by creating a plugin for Protégé [2] (a free open source ontology editor and knowledge-base framework) and apply the model to *push* queries on top of DB2 DBMS through DB2 triggers.

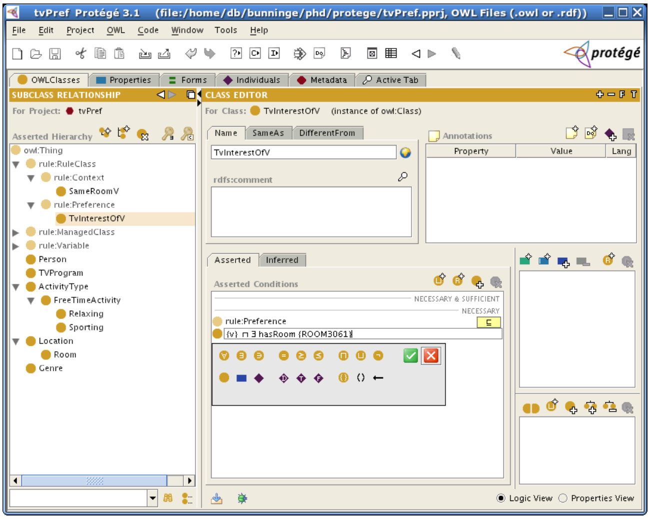
The plugin enables one to define the world model (ontology), including the context and preference notions. By combining both context and preference DL concept expressions, context-aware query preferences can then be constructed and further stored in an OWL-based knowledge base. This plugin can also facilitate the generation of the corresponding relational database schema based on the ontology and preferences. A screenshot of inputing a preference's applicable context DL expression is shown in Figure 4.

A context-aware query preference may trigger a push query proactively. For example, consider a preference "*retrieving the TV interest of the person when s/he enters room ROOM3061*"

$$\begin{aligned} \mathit{Context} &: \{v\} \sqcap \exists\, \mathit{hasRoom}.\{\mathit{ROOM}3061\} \\ \mathit{Preference} &: \exists\, \mathit{tvInterestOf}.v \end{aligned}$$

---

[2] http://protege.stanford.edu



For Class: TvInterestOfV (instance of owl:Class)

Name: TvInterestOfV

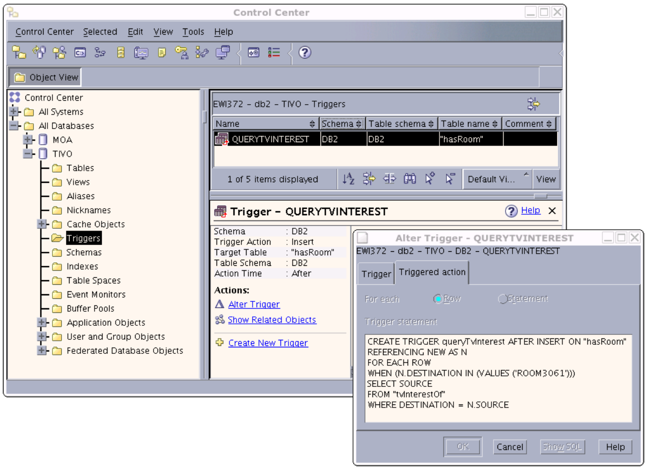
rule:Preference

{v} ⊓ ∃ hasRoom {ROOM3061}

Fig. 4. Screenshot of inputting a preference's context expression

A DB2 query trigger can be created as follows (Figure 5):

```
CREATE TRIGGER queryTvInterest AFTER INSERT ON hasRoom
REFERENCING NEW AS n FOR EACH ROW
  WHEN (n.DESTINATION IN VALUES ('ROOM3061'))
    SELECT SOURCE
    FROM   tvInterestOf
    WHERE  DESTINATION = n.SOURCE
```



EWI372 – db2 – TIVO – Triggers

Name	Schema	Table schema	Table name	Comment
QUERYTVINTEREST	DB2	DB2	"hasRoom"	

1 of 5 items displayed

Trigger – QUERYTVINTEREST

Schema : DB2
Trigger Action : Insert
Target Table : "hasRoom"
Table Schema : DB2
Action Time : After

Actions:

- Alter Trigger
- Show Related Objects
- Create New Trigger

EWI372 – db2 – TIVO – DB2 – QUERYTVINTEREST

For each ● Row ○ Statement

Trigger statement

```
CREATE TRIGGER queryTvInterest AFTER INSERT ON "hasRoom"
REFERENCING NEW AS N
FOR EACH ROW
WHEN (N.DESTINATION IN (VALUES ('ROOM3061')))
SELECT SOURCE
FROM "tvInterestOf"
WHERE DESTINATION = N.SOURCE
```

Fig. 5. The trigger in DB2

According to the example database schema in Figure 3, the query result will include *HUMAN-INTEREST*.

6 Conclusion

In this paper, we presented a context-aware query preference model for personalized information delivery and dissemination in an AmI environment. Revisiting the challenges raised by AmI, we adopted a knowledge-based approach to facilitate reasoning/inference (smartness requirement). The natural correspondence between DL concept expressions and data requests ensures the applicability of the model to both pull and push queries (proactiveness requirement). Preferences and associated applicable contexts are treated uniformly through DL concept expressions (closure requirement). Interpreting the knowledge-based preference model into a database world enables to address the scalability requirement. Through user-defined preferences, we can achieve the traceability requirement. We implanted the model on top of a DB2 DBMS and created a Protégé plugin was for defining, storing, and managing the context-aware preferences, and applying them to database queries.

Of course there remains much more to be done. Next to obvious directions, such as testing the scalability of the approach on realistic data sets and analyzing the expressive power of the model, we are currently investigating the uncertainty problem due to the imprecise context measurement and its impact on the context-aware query preference model.

References

1. Lacroix, M., Lavency, P.: Preferences; putting more knowledge into queries. In: VLDB '87. (1987) 217–225
2. Chomicki, J.: Preference formulas in relational queries. ACM Trans. Database Syst. **28**(4) (2003) 427–466
3. Kießling, W.: Foundations of preferences in database systems. In: VLDB '02. (2002) 311–322
4. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE '01. (2001) 421–430
5. Agrawal, R., Wimmers, E.: A framework for expressing and combining preferences. In: SIGMOD '00. (2000) 297–306
6. Koutrika, G., Ioannidis, Y.: Personalized queries under a generalized preference model. In: ICDE '05. (2005) 841–852
7. Holland, S., Kießling, W.: Situated preferences and preference repositories for personalized database applications. In: ER '04. (2004) 511–523
8. Stefanidis, K., Pitoura, E., Vassiliadis, P.: On supporting context-aware preferences in relational database systems. In: First International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP'2005). (2005)
9. Feng, L., Apers, P., Jonker, W.: Towards context-aware data management for ambient intelligence. In: DEXA '04. (2004) 422–431
10. Strang, T., Linnhoff–Popien, C.: A context modeling survey. In: Workshop on Advanced Context Modelling, Reasoning and Management. (2004)
11. van Bunningen, A.: Context aware querying - challenges for data management in ambient intelligence. Technical Report TR-CTIT-04-51, University of Twente, P.O. Box 217 (2004)

12. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook Cambridge University Press (2003)
13. Chen, H., Finin, T., Joshi, A.: The soupa ontology for pervasive computing. In: Ontologies for Agents: Theory and Experiences. Springer. (2005)
14. Borgida, A.: Description logics in data management. IEEE TKDE **7**(5) (1995) 671–682
15. Borgida, A., Brachman, R.: Loading data into description reasoners. In: SIGMOD '93. (1993) 217–226
16. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: VLDB '04. (2004) 588–599
17. First international workshop on exploiting context histories in smart environments (ECHISE). (2005)