Contextual Preference Repositories for Personalized Query Answering

Sandra de Amo, Tiago F. Mansueli, Renan G. Cattelan, Vinicius V.S. Dias, Hiran N. M. Ferreira

Universidade Federal de Uberlândia - Faculdade de Computação
deamo@ufu.br,tiagomansueli@comp.ufu.br,renan@facom.ufu.br,
viniciusvdias@comp.ufu.br,hirannonato@gmail.com

Abstract. The emerging of ubiquitous computing technologies in recent years has given rise to numerous personalized applications where user preferences and contexts are key features in providing suitable individual product recommendation as well as in personalizing query results according to user profile. Following this trend, it is noticeable the increasing interest of database researchers in developing efficient methods for incorporating context-aware preference querying facilities in database systems. In this work we develop a *Preference Repository* for storing and managing contextual preferences for each user through time. The evolution of the contexts of a specific user is handled by means of a *contextual graph* which plays an important role in the operations of storing, updating and retrieving preferences from the repository that match best to a given user context.

Categories and Subject Descriptors: H.Information Systems [**H.m. Miscellaneous**]: Databases

Keywords: preference awareness, content personalization, query answering personalization

1. INTRODUCTION

With the emerging of ubiquitous computing technologies [Abowd and Mynatt 2000; Cattelan et al. 2008; Yu and Nakamura 2010], the interest of researchers has been focusing mainly on the notion of *context-dependent preferences*. Indeed, in this new scenario users access to databases will not occur at a unique context, and consequently their expectation when querying data will vary according to a multitude of contexts including location, time and surrounding influences.

The CPrefSQL query language introduced in [de Amo and Pereira 2010] is an extension of the standard SQL providing new context-dependent preference operators allowing to rank the answers according to the user's preferences and context. In CPrefSQL, in order to get a personalized answer a user has to previously inform his preferences to the system by means of a command (CREATE PREFERENCES). Let us suppose a situation where a user u asks a query Q to the SGBD without informing his/her preferences. In this case, the CPrefSQL language would act like SQL and would not provide a personalized answer. It would be interesting to enhance the system with a *Preference Repository* which would store context-dependent preferences provided by u in the past. Then, when u asks his query Q without informing his current preferences, the system would automatically capture the user's context $\mathbf{c}$ and query the repository in order to locate the most recent preferences of user u matching the given context $\mathbf{c}$.

In this paper we focus on the problem of modeling and developing a *Context-Dependent Preference Repository* allowing easy access to user's long-term preferences. We assume that preferences are entered by the user sometime when he/she accesses the system. However, the Preference Repository Manager can be adapted to handle preferences that are automatically extracted by employing some Preference Mining algorithm [de Amo et al. 2012]. Contexts are either automatically collected from the user environment (e.g., device type, network bandwidth and calendar information) or explicitly

We thank the Brazilian Research Agencies CNPq, CAPES and FAPEMIG for supporting this work.

given by the user (e.g., time availability). Each time some set of preference rules R is informed by a previously identified user in a current context $\mathbf{c}$, this information is stored into the Preference Repository.

Operations for storing, retrieving and updating the Preference Repository are implemented in the *Preference Manager*. Besides, we show how the Preference Repository and Preference Manager can be coupled with the CPrefSQL language, constituting the main modules of the *Context Access System (CAS)* introduced in this paper. In Figure 1 the main architecture of the *Context Access System (CAS)* is illustrated. The *CAS* provides important context-aware search capabilities including personalized query-answering as well as tools allowing to adapt the format of the query answers according to the user's context. The following example aims at illustrating our proposal in a learning application. It is adapted from the application proposed in [Kärger et al. 2008][1].

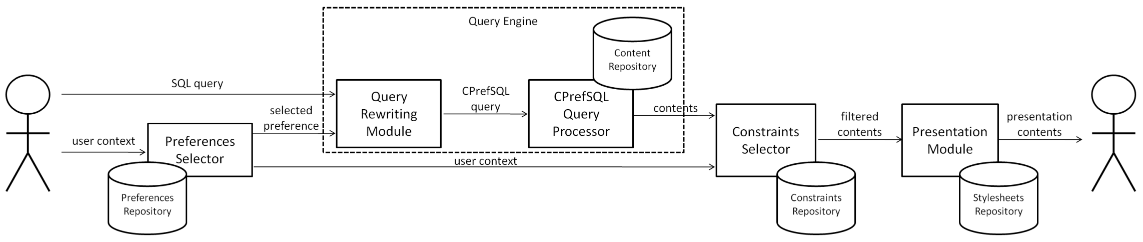


Fig. 1. *CAS* architecture and main modules

Motivating Example. Andy is a college student attending a few courses in Computer Science this term. In a particular day, Andy is worried about an upcoming test and decides to access the *CAS* during his lunch break in the university cafeteria. His test is in a couple of hours and his tablet computer is connected to the university wireless network. Andy enters his current context: he is at the university using a tablet device, with short available time and his access is due to an upcoming test. However, he does not enter any preference. The *CAS* uses Andy's context information in order to extract from the *Preference Repository* some set of preferences (he entered in some past access) whose context is most compatible with Andy's current context. In this context, Andy prefers lessons tagged as *easy* and *important*. Such tags are metadata about captured content which has been informed by the instructor or classified by students. Next, the *CAS* uses the extracted preferences in order to return a set of lessons which most fulfill Andy's expectation at the moment. Since Andy is in a public space and his tablet has no earphones, a presentation ignoring audio content will be returned by the *CAS* . The fact that Andy's tablet is not equipped with earphones is part of his context automatically captured by the system.

The main contributions of this paper are: (1) Introduction of the *Context Access System (CAS)* architecture; (2) Implementation of the Preference Repository and Preference Manager, two modules of the *CAS* and (3) Incorporation of the CPrefSQL query language [de Amo and Pereira 2010] in the *CAS*. The remaining of this paper is organized as follows. In Section 2, we discuss some related work. In Section 3 we present some necessary background. In Section 4 we present the details of the Preference Repository Model and the Preference Manager. In Section 5, we briefly describe the *CAS*. Finally, in Section 6, we conclude the paper.

2. RELATED WORK

Context-aware applications. One pioneer work in context-aware applications is [Dey 2001] where the author provides an operational definition of the notion of *context* and presents a general architec-

[1][Kärger et al. 2008] uses an extension of the RDF query language SPARQL proposed in [Siberski et al. 2006] allowing to specify soft constraints in the form of preferences. However, the problem of how contexts influence the user's preferences and how to find the best way to answer a user query even when he/she does not explicitly provide his/her preferences is not tackled in this work.

ture for building context-aware applications. In [Suryanarayana and Hjelm. 2002] the authors discuss the role of profiles within the Web architecture concerning situated (or contextualized) web services. In [Chen and Kotz. 2000] a survey on context-aware mobile computing is presented as well as a paradigm in which applications are provided with functionalities for discovering and taking advantage of contextual information (such as user location, time, nearby people and devices, user activities, etc). Several important topics are highlighted in this survey, such as *types of contexts*, *models for context information* and applications that adapt themselves to the changing context.

Preference Repositories. In [Holland and Kießling 2004] a general meta model for contexts (situations) is proposed, which can be used as a foundation for context models specification in a wide range of applications. In this approach, contextual preferences are *qualitative* (expressed by means of rules) and are stored in a XML repository. This repository can be queried in order to find the best-matching preferences to a given context. Differently from the approach presented in this paper, [Holland and Kießling 2004] does not tackle the problem of query answering with preference support. Besides, the preference model adopted in [Holland and Kießling 2004] is less expressive than ours.

3. PRELIMINAIRES

In this section we briefly present the main features of the CPrefSQL query language. The CPrefSQL language has been implemented in the core (and alternatively *on-top*) of the PostGreSQL 8.4 query processor. For more details and a more rigorous presentation see [de Amo and Pereira 2010].

3.1 The Preference Model

The preference model underlying the CPrefSQL language distinguishes *data attributes* (denoted by $A_1, A_2, ...$) from *context attributes* (denoted by $C_1, C_2, ...$). Users can specify their preferences on *data values* and their personal contexts may have influence on their respective choices. *Context values* are inherent to a situation outside the database and in general cannot be affected by user preferences. Common examples of context attributes include user location, time, temperature, bandwidth, nearby resources [Chen and Kotz. 2000]. We adopt a general framework for context specification. For a detailed presentation see [de Amo and Pereira 2010].

A *Context Schema* is a set of context attributes $C = \{C_1, ..., C_n\}$. A *context* over C is a tuple **c** $= (c_1, ..., c_n)$ where each c_i is a value for the context attribute C_i or the $*$ symbol. For instance, C $=$ {Place, Bandwith} is a context schema and **c** $=$ (home,1GB) and **c'** $=$ (*, 1Gb) are contexts over C. The latter context (**c'**) represents a situation where the user bandwidth internet connection is 1Gb. So, in this context he may prefer to download videos than text. The former context (**c**) is more *specific*, representing a situation where the user is *at home* and his internet bandwidth is 1Gb.

User preferences are expressed by *preference rules* of the form "IF (a *context* **c** and *some data condition* are given) THEN the values for data attribute X satisfying a condition S are *preferable* than the values not satisfying this condition. In this paper, we introduce the concept of *preference rules* informally through an example. For a more rigorous presentation please see [de Amo and Pereira 2010].

EXAMPLE 3.1. Let us consider the context attributes P (Place) and R (Reason) and the relation *Lessons* with the data attributes T (Title), Di(Discipline), D (Difficulty), I (Importance) and Du (Duration). The student Andy specifies the following preferences concerning his studies: when he is at the university and aims at a quick review he prefers to review the more important and easier lessons (those with importance status ≥ 4 and difficulty status ≤ 3). Moreover, since he is very busy when he is at the university, he prefers to be given lessons whose overall duration is under 60 minutes. On the other hand, if his location and interest changes his preferences change accordingly: when he is at home preparing himself for a coming test on *Databases*, he prefers to be given more detailed lessons, with at least two hours duration. The preference rules corresponding to the first and fourth statements are the following:

rule 1: `IF context = (university,quick review) THEN` (I $\geq$ 4) $\succ$ (I $<$ 4) $[T, Di]$
rule 2: `IF context = (university,quick review) THEN` (D $\leq$ 3) $\succ$ (D $>$ 3) $[T, Di]$
rule 3: `IF context = (university,quick review) THEN` (Du $<$ 60) $\succ$ (Du $\geq$ 60) $[T, Di]$
rule 4: `IF context = (home, test) AND Di=`*`Databases`* `THEN` (Du $\geq$ 120) $\succ$ (Du $<$ 120) $[T, I, D]$

The symbol $\succ$ stands for "is more preferred than". The meaning of the attributes between brackets will be clear next (see Example 3.2).

A set of preference rules induces a *preference ordering* over the objects stored in the database as shown in the following example:

EXAMPLE 3.2. Let us consider the preference rules given in Example 3.1 and the following tuples over the attributes {T, Di, D, I, Du}:

	Title (T)	Discipline (Di)	Difficulty (D)	Importance (I)	Duration (Du)
t_1	Introduction	Compilers	**5**	4	**30**
t_2	Assembler	Computer Architecture	**5**	3	**30**
t_3	ER Model	Databases	**5**	3	90

Let us suppose that currently Andy is at the university and is interested in a quick review. In this case, only rules 1, 2 and 3 can be used to compare the three lessons t_1, t_2 and t_3. Let us see why. We have that t_1 is preferred to t_2 according to rule 1, since the value for I in t_1 is ≥ 4 and in t_2 is $<$ 4. Notice that in rule 1 one has two attributes between brackets, T and Di. This indicates that the values of these attributes are completely irrelevant in the comparison. On the other hand, the values for the other attributes (Du and D) must be the same in both tuples being compared. Notice that this is the case for tuples t_1 and t_2: they coincide on Du and D respectively. This restriction on the order semantics is called the *ceteris paribus* condition [2]. In the same way, we can affirm that t_2 is preferred to t_3 according to rule 3. Then, by transitivity we can infer that t_1 is more preferred than t_3.

Consistency Issues. The preference order is obtained by taking the transitive closure of the union of the preference orders specified by each rule individually. The result may be inconsistent, that is, it can be inferred that *I prefer X better than Y and also prefer Y better than X*. So, consistency issues have to be carefully taken into account when dealing with preference rules, since symmetry of the preference ordering is not a desirable situation in a database context. A set of preference rules Γ is said to be *consistent* if for each context $\mathbf{c}$ appearing in Γ, the preference order induced by Γ_c (the rules in Γ referring to $\mathbf{c}$) is a strict partial order (irreflexive and transitive) [3]. In [de Amo and Pereira 2010] it was presented and implemented an algorithm for testing the consistency of a set of contextual preference rules. This test will be used as a subroutine in the algorithms for inserting, retrieving and updating preferences in the Preference Repository (Section 4).

3.2 The CPrefSQL Query Language

The CPrefSQL query language is an extension of the standard SQL with two preference operators: ***Select-Best*** and ***SelectK-Best***. The ***Select-Best*** operator selects from a given relation the set of the more preferred tuples according to a given set of contextual preference rules. The ***SelectK-Best*** returns the set of the first K preferred objects in the database following the preference hierarchy. The basic CPrefSQL block is presented in Figure 1. The essential difference between the SQL and the CPrefSQL query languages is the following: (1) in SQL user wishes can be either entirely fulfilled or not at all. User choices are specified as *hard constraints*, and the results delivered are exactly as requested. If there are objects satisfying these hard constraints they are all returned. Otherwise, a

[2]In the Preference Repository Model introduced in Section 4 we call the attributes between brackets the *ceteris paribus* attributes.
[3]Notice that if a preference relation satisfies both the irreflexivity and transitive properties then one cannot have X preferred to Y and vice-versa, since by transitivity this would entail that X is preferred to X, contradicting the irreflexity property.

```
SELECT <attribute-list>
FROM <tables>
WHERE <where-conditions> (% hard constraints)
ACCORDING TO PREFERENCES [K]
<list-of-preference-cprules> (% soft constraints)
GROUP BY <attribute-list>
ORDER BY <attribute-list>
```

Fig. 2. The basic CPrefSQL block

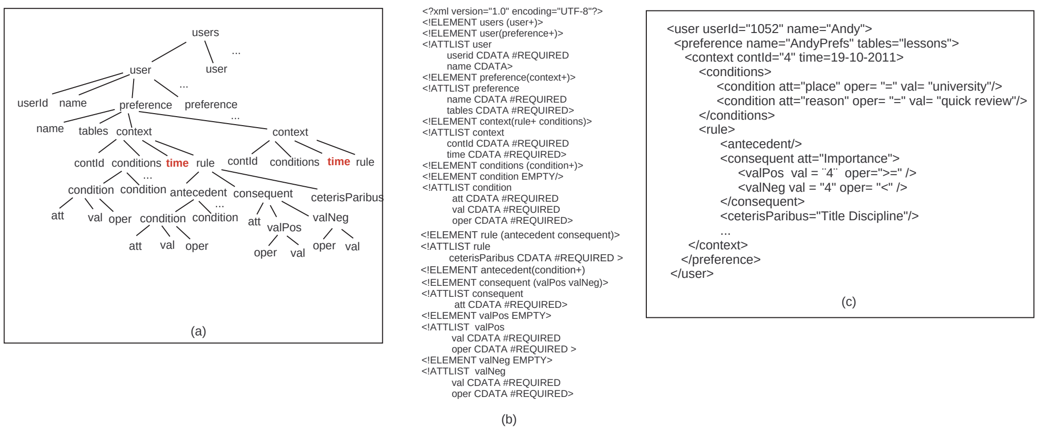
disappointing empty answer is returned; (2) CPrefSQL incorporates *soft constraints* in the form of user preferences besides the usual SQL hard constraints allowing *best possible* answers and not only *exact answers*.

4. THE CONTEXTUAL PREFERENCE REPOSITORY

The *Preference Repository* is used to store the contextual preferences of users wishing to access the main database. In order to have access to the main database, the user must identify himself and optionally inform his context and his preferences. Contextual preferences provided by the user are stored in the Preference Repository together with a timestamp informing the moment the preference was valid for the user.

4.1 The Repository Data Model

The Preference Repository is organized as a XML document. The underlying tree model of the XML document as well its DTD schema are described in Figure 3.



```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT users (user+)>
<!ELEMENT user(preference+)>
<!ATTLIST user
      userid CDATA #REQUIRED
      name CDATA>
<!ELEMENT preference(context+)>
<!ATTLIST preference
      name CDATA #REQUIRED
      tables CDATA #REQUIRED>
<!ELEMENT context(rule+ conditions)>
<!ATTLIST context
      contId CDATA #REQUIRED
      time CDATA #REQUIRED>
<!ELEMENT conditions (condition+)>
<!ELEMENT condition EMPTY/>
<!ATTLIST condition
       att CDATA #REQUIRED
       val CDATA #REQUIRED
       oper CDATA #REQUIRED>
<!ELEMENT rule (antecedent consequent)>
<!ATTLIST rule
      ceterisParibus CDATA #REQUIRED >
<!ELEMENT antecedent(condition+)
<!ELEMENT consequent (valPos valNeg)>
<!ATTLIST consequent
       att CDATA #REQUIRED>
<!ELEMENT valPos EMPTY>
<!ATTLIST  valPos
       val CDATA #REQUIRED
       oper CDATA #REQUIRED >
<!ELEMENT valNeg EMPTY>
<!ATTLIST  valNeg
       val CDATA #REQUIRED
       oper CDATA #REQUIRED>
```

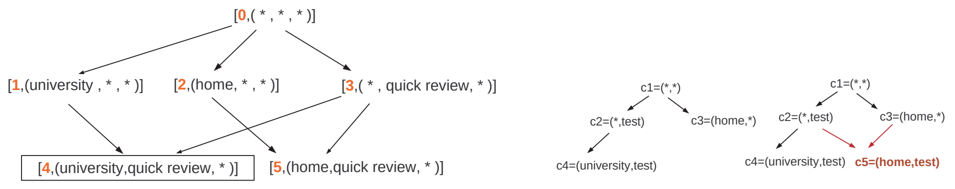
```
<user userId="1052" name="Andy">
 <preference name="AndyPrefs" tables="lessons">
   <context contId="4" time=19-10-2011>
     <conditions>
        <condition att="place" oper= "=" val= "university"/>
        <condition att="reason" oper= "=" val= "quick review"/>
     </conditions>
     <rule>
        <antecedent/>
        <consequent att="Importance">
           <valPos  val = ¨4¨  oper=">=" />
           <valNeg val = "4" oper= "<" />
        </consequent>
        <ceterisParibus="Title Discipline"/>
        ...
   </context>
  </preference>
</user>
```

Fig. 3. (a) The Repository Tree Model, (b) The DTD Schema and (c) An except from Andy's preferences

The contexts of each user are organized into a separated structure, the *Contextual Graph* (a graph for each specific user). The Contextual Graph plays an important role in the tasks related to the repository management (insertion, retrieval and update).

Definition 4.1 *Context Compatibility and Contextual Graph.* Let $\mathbf{c} = (c_1, ..., c_k)$ and **c'** $= (c'_1, ..., c'_k)$ be contexts over the context schema $\mathcal{C} = \{C_1, ..., C_k\}$. We say that **c'** is *more or equally general* than **c** (denoted by **c'** $\geq$ **c**) if and only if for all $i \in \{1, ..., k\}$ $c_i = c'_i$ or $c'_i = *$. The symbol "*" stands for "not provided" or "don't care". We say that **c** and **c'** are *compatible* if and only if (1) for each $i = 1, ..., k$ $c_i = *$ or $c'_i = *$ or $c_i = c'_i$. The *Contextual Graph* is a graph $G = (\mathcal{V}, \mathcal{A})$ satisfying: (1) each node $v \in \mathcal{V}$ is a pair v= ($ContId$,**c**), where $ContId$ is a context identifier and **c** is a context; (2) $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ and $e = ((Id1, \mathbf{c}_1), (Id2, \mathbf{c}_2)) \in \mathcal{A}$ if and only if $\mathbf{c}_1 \geq \mathbf{c}_2$.

Figure 3(c) illustrates an excerpt of the preference rules presented in Example 3.1. One supposes that Andy entered the first 3 preference rules as he accessed the system on 19th October 2011. At this moment, he was at the university and intended to make a quick review (that was his context). For lack of space, only rule 1 is presented in Figure 3(c). Figure 4(a) describes an excerpt of the contextual graph highlighting the context [4,(university,quick review,*)] entered by Andy when he accessed the system on 19 October 2011. In this example we are supposing that we have only 3 context attributes (Place, Reason, Available Time) and that Andy did not inform his time availability (represented by the "*" symbol). Notice that contexts 2 and 3 are compatible but 1 and 2 are not.



(a) A Contextual Graph (b) non-saturated (left) ; saturated (right)

Fig. 4. Contextual Graphs

Next, the operations related to the *Repository Management* are presented. In order to understand how the repository manager works, some additional definitions are needed.

Definition 4.2 *Context Saturation.* Let $\mathbf{C} = \{\mathbf{c}_1,...,\mathbf{c}_m\}$ be a set of contexts. $\mathbf{D}_i(\mathbf{C})$ denotes the set of values for the context attribute C_i appearing in the contexts of $\mathbf{C}$. $\mathbf{C}$ is said to be *saturated* if the following condition holds: for all $\mathbf{c}_i$, $\mathbf{c}_j \in \mathbf{C}$ and all contexts $\mathbf{c} \in (\mathbf{D}_1(\mathbf{C}) \cup \{*\}) \times \ldots \times (\mathbf{D}_k(\mathbf{C}) \cup \{*\})$, if $\mathbf{c}_i \geq \mathbf{c}$ and $\mathbf{c}_j \geq \mathbf{c}$ then $\mathbf{c} \in \mathbf{C}$.

EXAMPLE 4.1. Let us consider the set of context attributes $\mathcal{C} = \{P$ (Place), R (Reason)$\}$. The following tuples are contexts over $\mathcal{C}$: $\mathbf{c}_1 = (*,*)$, $\mathbf{c}_2 = (*,test)$, $\mathbf{c}_3 = (home,*)$, $\mathbf{c}_4 = (university,\ test)$. Let $\mathbf{C} = \{\mathbf{c}_1,\ \mathbf{c}_2,\ \mathbf{c}_3,\ \mathbf{c}_4\}$. $\mathbf{D}_1(\mathbf{C}) = \{home,\ university\}$ and $\mathbf{D}_2(\mathbf{C}) = \{test\}$. $\mathbf{C}$ is not saturated, since the context $\mathbf{c}_5 = (home,\ test)$ is more specific than $\mathbf{c}_2$ and $\mathbf{c}_3$, but $\mathbf{c}_5$ does not figure in $\mathbf{C}$. On the other hand, the set $\mathbf{C'} = \mathbf{C} \cup \{\mathbf{c}_5\}$ is saturated. In Figure 4(b) we show the hierarchical structure of the context sets $\mathbf{C}$ (at the left side) and $\mathbf{C'}$ (at the right side).

The Contextual Graph is always kept saturated by the Repository Manager. An important property satisfied by saturated Contextual Graphs is the following:

PROPOSITION 4.3. Let $\mathbf{c}$ be a context over $\mathcal{C}$ and G a *saturated* Contextual Graph. Let S be the set of contexts in G which are compatible with $\mathbf{c}$. If $S \neq \emptyset$ then there is a unique context $\mathbf{c'}$ in S such that $\mathbf{c'}$ is more specific than all the other contexts $\mathbf{c''}$ in S such that $\mathbf{c''} \neq \mathbf{c'}$.

The proof is straightforward and is not given for lack of space. This property will be very important in the algorithms for inserting, updating and querying the Preference Repository.

4.2 The Algorithms for Preference Management

Algorithm 1 describes the main lines of the procedure Insert($\mathbf{c}$,R) for inserting in the Preference Repository the preference rules R associated with the context $\mathbf{c}$. For lack of space we will present only the `RETRIEVE` operation, responsible for extracting preference rules from the Repository. The operations for updating the repository are similar. The `RETRIEVE` operation (see Algorithm 2) is automatically launched by the Repository Manager when the user wants to query the main database and does not explicitly inform his preferences. In this case the Repository will be searched and the *most recent preference rules (those with the most recent time value)* which best conforms to the user current context will be retrieved.

5. THE CONTEXT ACCESS SYSTEM (CAS)

In this section we describe the overall architecture of the *Contextual Access System (CAS)*, a system supporting content personalization, a highly desirable feature in ubiquitous computing. Figure 1 describes an overview of the *CAS* architecture. User context is handled by a *Preference Selector* (or *Preference Manager*), which chooses the most appropriate preference according to the context of

Algorithm 1: Insert($\mathbf{c}$,R,T)

```
Data: A set of preference rules $R$, a context $\mathbf{c}$, the Preference Repository $T$
NewCont ← $\emptyset$
$ContGraph$ ← Contextual Graph($T$) ;                /* gets the contextual graph associated to $T$ */
(N,c') ← MostSpecCompt($ContGraph$,c) ;      /* scan $ContGraph$ and gets the most specific context
compatible with c */
$R'$ ← MostRecentRules(N,c') ; /* scan $T$ and gets the most recent rules associated to context (N,c') */
if $R \cup R'$ is consistent and $\mathbf{c} \neq$ c' then          /* see remark on Section 3 about consistent rules */
    $newId$ ← CreateNewId(); InsertRep($newId$,c,$R \cup R'$,currentTime,$T$)
    InsertContGraph(newId,c,$ContGraph$); $ContGraph$ ← Saturate($ContGraph$,$NewContexts$)
    NewCont ← NewCont $\cup$ $NewContexts$ ;    /* gets the new contexts resulting from the saturation
    operation */
if $R \cup R'$ is consistent and $\mathbf{c} =$ c' then
    InsertRep($N$,c,$R \cup R'$,currentTime,$T$)
if $R \cup R'$ is inconsistent and $\mathbf{c} \neq$ c' then
    $newId$ ← CreateNewId(); InsertRep($newId$,c,$R$,currentTime,$T$)
    InsertContGraph(newId,c,$ContGraph$); $ContGraph$ ← Saturate($ContGraph$)
    NewCont ← NewCont $\cup$ $NewContexts$
if $R \cup R'$ is inconsistent and $\mathbf{c} =$ c' then
    InsertRep($N$,c,$R$,currentTime,$T$)
foreach ($N'$,c') $\in$ NewCont do  /* Update the repository by inserting the preference rules compatible
with the new contexts resulting from saturation */
    $Parent$($N'$,c'):= set of contexts in $ContGraph$ that produced ($N'$,c') by saturation
    foreach ($N_i$,$c_i$) $\in$ Parent($N'$,c') do
        $R_i$ ← MostRecentRules($N_i$,$c_i$)
    InsertRep($N'$,c',$\bigcup_i R_i$,currentTime,$T$)
```

Algorithm 2: Retrieve($\mathbf{c}$,T)

```
Input: context c, preference repository $T$
Output: set of contextual preference rules $R$
$ContGraph$ ← Contextual Graph($T$) ;                /* gets the contextual graph associated to $T$ */
(N,c') ← MostSpecCompt($ContGraph$,c) ;      /* scan $ContGraph$ and gets the most specific context
compatible with c */
$R$ ← MostRecentRules(N,c') ;  /* scan $T$ and gets the most recent rules associated to context (N,c') */
return $R$
```

access. Customized preferences are stored for each user. In the next module, *Query Rewriting Module*, an ordinary SQL query is combined with the selected preference and rewritten as a CPrefSQL query. The *CPrefSQL Query Processor* then queries the *Content Repository* (the main database) and returns all contents matching the query. The *Constraint Selector* module filters such contents according to user restrictions. The *Content Formatting Tool* finally converts the filtered contents to a presentation format (e.g., HTML). For lack of space, we only present the two first modules (the *Preference Manager* and the *Query Engine*).

5.1 The Preference Selector (Manager)

The tasks performed by the *Preference Selector* can be summarized as follows: (1) If the user explicitly inform his preferences, the *Preference Selector* queries the *Preference Repository* to locate a context matching the current user context. If such context is found, the `INSERT` operation is invoked (as detailed in the Section 4) and the current preferences with the current given context is entered in the Preference Repository. (2) If the user does not explicitly inform his preferences, the *Preference Selector* invokes the `RETRIEVE` operation in order to extract from the Preference Repository the *most recent preference rules* which best conforms to the user current context.

For instance, let us suppose that Andy access the system and does not inform his preferences at that moment. He enters a complete information about his current context $\mathbf{c}_{curr}$: Place = *university*, Reason

= *quick review*, available time = *1 hour* and device = *tablet.* The Preference Repository is searched and it turns out that the context which best conforms to Andy's current context is *(university,quick review.*,*)*. So, the Preference Selector extracts the set of preference rules associated to this context.

5.2 The Query Engine

After the *Preference Selector* returned the XML file containing the user most recent preference rules, these preferences together with a user (SQL) query Q are delivered as input to the *CPrefSQL Query Processor*. The *Query Rewriting Module* is responsible for building a CPrefSQL block Q' out of the SQL query Q and the XML file containing the preference rules.

In our running example, let us suppose Andy wants to be given the four lessons which most fulfill his preferences. The *Preference Selector* extracted from the Preference Repository an XML file *AndyPref.xml* containing the most recent Andy's preference conforming to his current context. This XML file is processed by the *Query Rewriting Module* which first transforms its contents into a list of rules (stored in a file *AndyPrefs*), as illustrated in Example 3.1. The following CPrefSQL is delivered to the *CPrefSQL Query Processor* afterwards:

```
SELECT title
FROM Lessons
ACCORDING TO PREFERENCES 4, AndyPrefs;
```

6. CONCLUSION AND FUTURE WORK

In this paper we developed a *Context-Dependent Preference Repository* allowing easy access to user's long-term preferences. We also presented the main modules of the *Context Access System (CAS)* incorporating the Preference Repository and CPrefSQL language. Our current research is focused on coupling the *CAS* system with the open source capture platform iClass ([Pimentel et al. 2007]) to provide support to C&A applications where contexts and user preferences are relevant to produce personalized multimedia artifacts.

REFERENCES

Abowd, G. D. and Mynatt, E. D. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction* 7 (1): 29–58, 2000.

Cattelan, R. G., Teixeira, C., Goularte, R., and Pimentel, M. D. G. C. Watch-and-comment as a paradigm toward ubiquitous interactive video editing. *ACM Trans. Multimedia Comput. Commun. Appl.* vol. 4, pp. 28:1–28:24, November, 2008.

Chen, G. and Kotz., D. A survey of context-aware mobile computing research. Technical report tr2000-381, Dept. of Computer Science, Dartmouth College, Hanover, New Hampshire, USA, 2000.

de Amo, S., Diallo, M., Diop, C., Giacometti, A., Li, H. D., and Soulet, A. Mining contextual preference rules for building user profiles. In *14th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), Viena*, 2012.

de Amo, S. and Pereira, F. Evaluation of conditional preference queries. *Journal of Information and Data Management (JIDM).* vol. 1(3), pp. 521–536, 2010.

Dey, A. K. Understanding and using context. *Personal and Ubiquitous Computing* vol. 5(1), pp. 4–7, 2001.

Holland, S. and Kiessling, W. Situated preferences and preference repositories for personalized database applications. In *ER.* pp. 511–523, 2004.

Kärger, P., Olmedilla, D., Abel, F., Herder, E., and Siverski, W. What do you prefer ? using preferences to enhance learning technology. *IEEE Transactions on Learning Technologies* vol. Vol. 1, (1), pp. 20–33, 2008.

Pimentel, M., Baldochi Jr., L. A., and Cattelan, R. G. Prototyping applications to document human experiences. *IEEE Pervasive Computing* vol. 6, pp. 93–100, April, 2007.

Siberski, W., Pan, J., and Thaden, U. Querying the semantic web with preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC).* pp. 612–624, 2006.

Suryanarayana, L. and Hjelm., J. Profiles for the situated web. In *Proc. 11th International World Wide Web Conference (WWW 2002).* pp. 200–209, 2002.

Yu, Z. and Nakamura, Y. Smart meeting systems: A survey of state-of-the-art and open issues. *ACM Comput. Surv.* vol. 42, pp. 8:1–8:20, March, 2010.