Context-based Exploitation of Data Warehouses

Yeow Wei Choong[2], Arnaud Giacometti[1], Dominique Laurent[3],
Patrick Marcel[1], Elsa Negre[1], Nicolas Spyratos[4]

[1]LI, Université François Rabelais de Tours, Antenne Universitaire de Blois, France
[2]HELP University College, Kuala Lumpur - Malaysia,
[3]ETIS, Université de Cergy-Pontoise, France
[4]LRI, Université Paris-Sud, France

Abstract. An OLAP analysis can be defined as an interactive session during which an user launches queries over a data warehouse. The launched queries are often interdependent, and they can be either newly defined queries or they can be existing ones that are browsed and reused. Moreover, in a collaborative environment, queries may be shared among users. This notion of OLAP analysis has never been formally defined. In this paper, we propose a clear definition of this notion, by introducing a model for sharing, browsing and reusing OLAP queries over a data warehouse.

1 Introduction

In the area of OLAP exploitation of data warehouses (what we call analysis), there is a need for organizing, reusing, sharing queries, in order to simplify and speedup the querying process [DKK05, GMN06]. Broadly speaking, it can be said that contextual information during the exploitation of data warehouses must be taken into account. We propose a model that answers these needs, by adapting the model proposed by Spyratos et al. in the context of collaborative work [TACS02, AST03, AS04].

In this model, the user defines and stores OLAP queries in what is called a *context*. In a context, the user can organize the queries so that they are easily browsed in a subsequent session. This organisation can reflect e.g., classical query containment, or an order of importance relevant to the user. In a multi-user environment, contexts can be shared among users. The set of contexts can be browsed, or queried. In addition, OLAP queries in a given context can be imported into another context to enrich the user's current analysis.

The contribution of our work includes:
- A model for OLAP query organisation, which we called *the Context Base*, that allows to easily share and reuse queries,
- The languages for defining, manipulating and browsing this context base,
- The exploitation of the structure of the context base to provide useful recommendations for facilitating user browsing.

The rest of the paper is organized as follows. Section 2 motivates through an example the need for a tool for sharing, browsing and reusing queries in a collaborative OLAP environment.

The model is first introduced informally in Section 3, and then detailed formally in Section 4 and Section 5. Section 6 presents various examples on how the model can be exploited to facilitate browsing. Finally, Section 7 concludes the paper and discusses future research directions.

2 Motivating example

In this section, we motivate through an example how the model we propose can be used to perform an OLAP analysis. This example illustrates a simple analysis performed by an user in a multi-user environment by showing the actions the user does to organize, reuse, launch, browse and share OLAP queries.

In what follows we use the terms “browse", “launch" and “share" with the following meanings:

- Browse refers to the possibility for the user to navigate among sets of OLAP queries,and among the queries within a set of queries. Browsing and navigating are interchangeable in what follows.
- Share refers to the possibility for the user to browse queries defined by others users.
- Launch refers to the possibility for the user to execute a selected query and to visualize the answer.

Example Consider two user-analysts Elsa and Yeow Wei. Let us suppose that Yeow Wei queries the datacube *Tourism in Malaysia*. Yeow Wei created a workspace to store his queries. In what follows, this workspace is called a *context*. This context is described by the text *Tourism in Malaysia*. His first query asks for the complete cube. Then he asks for tourism in Malaysia by transport and year. Then, he refines his query to specify that he is interested only in train transportation. During his analysis, to remember the different queries, Yeow Wei assigns a description to each query, and he organizes the queries so that the relation displayed on the screen reflects query containment. At any moment, he can visualize various information (that will be called *descriptors*) related to the query, like for example the query answer, the SQL code, the number of times the query has been launched etc.

As the system manages more than one user, differents sets of queries defined by different users can be exchanged between users. Elsa can thus browse Yeow Wei’s analyses and queries. Meanwhile the system can count how many times a query is launched and/or browsed, these information appearing as descriptors associated with the query.

Let us suppose that Elsa queries the set of all existing contexts to find which are the ones dealing with Malaysia, tourism, or agriculture. The system returns two contexts: *Tourism in Malaysia* (Yeow Wei’s analysis) and another context described as *Agriculture in Malaysia*. Elsa chooses to browse first the context *Tourism in Malaysia*. She selects the query described as *Transport, Year*, launches the query, visualizes the answer, and want to copy it into her context. She creates a new context containing the copy of the query and assigns her description to the query: *Tourism by Transport and Year*. By doing so, the system adds a link (called *reference*) from the query *Tourism by Transport and Year* of Elsa’s context to the query *Transport, Year* of Yeow Wei’s context. This link recalls that the query in Elsa’s context comes from Yeow Wei’s query. This reference can be used as a recommendation that the system can propose to users browsing this particular query in Elsa’s context.

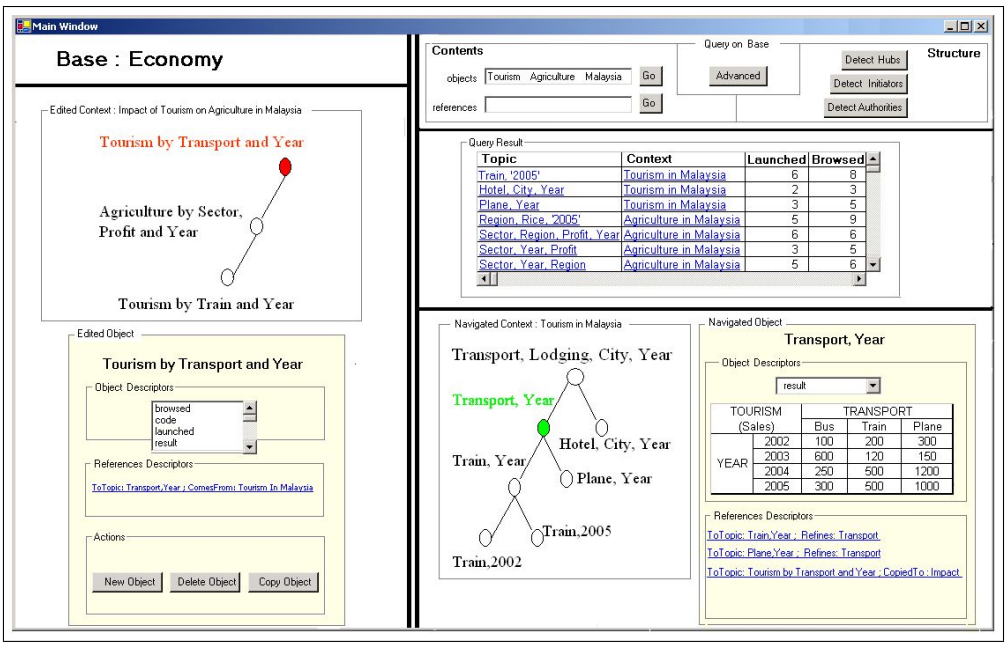

FIG. 1 – *Overview of Elsa's analysis session*

To carry on with her analysis, Elsa asks for the context *Agriculture in Malaysia*, browses the queries in this context and copies one of them. Then she goes back to the context *Tourism in Malaysia* and copies another query *Train, Year*. All the queries of the context will be described by *Impact of Tourism on Agriculture in Malaysia*, so, the context is associated with this descriptor.

Figure 1 illustrates the current state of Elsa's analysis, by describing what could be the GUI of the system implementing the operations to organize, share and browse OLAP queries. Note that the query organisation differs from one context to another: In *Tourism in Malaysia*, the context Elsa is browsing, the query organisation reflects query containment, whereas in *Impact of Tourism on Agriculture in Malaysia*, Elsa's context, query organisation reflects the order in which the user has imported the queries.

3 Intuitions

In this section, we present informally our model. This model consists of two levels:

- The data level models the data that can be edited, stored, browsed, queried,
- The system level models what is presented to the user.

Each level is associated with a language:

- The data level language is a classical data manipulation language used for querying,
- The system level language consists in operations for browsing the data level data and for editing or defining these data.

Data level Data can be of three types:
- An object, that represents a query over a data warehouse (we use the term object in order not to confuse the reader, the term query being used in what follows to denote a query over a set of objects).
- A context, that can be viewed as a set of objects, in which the user defines and stores objects. Within a context, the user organizes the objects so that they are easily browsed in a subsequent session.
- A reference, that is used to establish a link between objects.

A descriptor is a tuple $\langle attribute, value \rangle$ wich allows to describe an object or a reference. For example, an object representing a query over a data warehouse can be described by the SQL code of the query, and thus be associated with the following descriptor:
$\langle code, "Select * from Tourism where Country =' Malaysia'" \rangle$.

An object is identified by an object identifier and is described by a set of descriptors. For example, the object described by: $\langle code, "Select * from Tourism where Country =' Malaysia'" \rangle$ can also be described by:
- $\langle result, CT_1 \rangle$ where CT_1 is a cross-tab used to visualise the answer of the query,
- $\langle launched, 7 \rangle$ that indicates that the query has been launched 7 times.

A context is the structure in which objects are stored and organized. Thus a context can simply be viewed as a set of objects, which is identified by a context identifier. For example, a context can represent an analysis session of a particular user. Contexts are not directly associated with descriptors, however it can be considered that a context is described by the (union or intersection of the) descriptors of the objects it contains.

A reference is a link between two objects which are not necessarily in the same context. A reference is associated with a set of descriptors. References can be used to describe the organization of the objects in a particular context. Or they can be used to indicate similar objects that can be found in different contexts. For example, considering that objects are OLAP queries, query q_1 associated with $\langle code, "Select * from Tourism" \rangle$ and query q_2 associated with $\langle code, "Select * from Tourism$
$where\ Country =' Malaysia'" \rangle$ can be related by a reference associated with the descriptor $\langle refines, "country" \rangle$ to indicate that q_2 refines q_1 and thus is included in q_1 in the usual sense of query containment.

Note that in our model, objects and references form a graph where objects are the vertices and references are the edges.

System level This level models what is presented to the user (see Figure 1):
- The name of the context base, the instance of which is the set of all contexts, objects and references that can be browsed. This name is indicated in the top-left zone of Figure 1,
- The query over the context base, that is used to select relevant parts of the context base. This query can be keyed in using the top-right zone of Figure 1.
- The object currently browsed, as well as its context, is displayed in the bottom-right zone of Figure 1,
- The object currently edited, as well as its context, is displayed in the bottom-left zone of Figure 1.

Interacting with the system consists in changing what is displayed on the screen. It can be done by:
- Browsing the context base, that is either asking a new query over the context base, or simply viewing another object among the result of the current query on the context base,
- Editing the context base, that is defining, modifying or deleting contexts, objects or references.

The data level and the data manipulation language are presented formally in Section 4, and the system level, the navigation operations and the edition operations are presented formally in Section 5.

4 The data level

In this section, we present the data level of our model as well as the data manipulation language.

4.1 The data model

For the sake of simplicity, our data model is described by using the relational model, under the logic programming perspective [AHV95]. Let Dom be a countably infinite set of constants. A special constant $NULL$ is used to indicate the fact that no object is displayed. Moreover we assume that Dom is ordered.

4.1.1 The relations

We consider the following three relations (in what follows, o_{id}, c_{id}, att, val, o_{id_1}, $o_{id_2} \in Dom$):
- objects: Is a 3-ary relation. A fact $objects(o_{id}, att, val)$ associates the object identifier o_{id} with a descriptor which has attribute *att* and value *val*.
- contexts: Is a binary relation. A fact $contexts(c_{id}, o_{id})$ associates the object identifier o_{id} to the context identifier c_{id}.
- references: Is a 4-ary relation. A fact $references(o_{id_1}, o_{id_2}, att, val)$ associates the object identifier o_{id_1} to the object identifier o_{id_2} with a descriptor which has attribute *att* and value *val*.

Classicaly, for each relation name R, a relation instance over R is a finite set of facts over R.

4.1.2 The Context Base

Context base schema and instance The schema of a context base consists of a context base name and the set of relation names $\{contexts, objects, references\}$. A context base instance is a finite set of facts that is the union of relation instances over R, for $R \in \{$*contexts, objects, references*$\}$.

Well-formed instance of context base A context base instance I is well-formed if each object belongs to one and only one context. It means that we cannot have, e.g., $objects(o_1, att_1, val_1)$, $contexts(c_1, o_1)$ and $objects(o_1, att_2, val_2), contexts(c_2, o_1)$. Object sharing

among contexts can only be done by duplicating objects and connecting the object copies with a reference. This allows to keep trace of the duplication, an information that can be subsequently queried.

These conditions are expressed formally below:

- If $objects(o_{id}, x, y) \in I$ then there exists only one $c_{id} \in Dom$ such that $contexts(c_{id}, o_{id}) \in I$.
- If $references(o_{id_1}, o_{id_2}, x, y) \in I$ then there exists $c_{id_1}, c_{id_2} \in Dom$ such that $contexts(c_{id_1}, o_{id_1}) \in I$ and $contexts(c_{id_2}, o_{id_2}) \in I$.

4.2 The manipulation language

The language we use to describe the manipulation of the context bases is Datalog$^\neg$, under the stratified semantics [AHV95]. This is because we need to express recursion (to compute the transitive closure of the graph of objects) and we need to express the relational division (see Section 6 for examples of useful queries). This language is used to compute subsets of the context base that the user will subsequently browse.

In what follows, given a well-formed context base instance I, the semantics of a Datalog$^\neg$ program P on I is the classical stratified semantics denoted $P(I)$.

Well-formed program A program P on a well-formed context base instance I is well-formed if the following predicates belong to $P(I)$:

- $contexts\_a$: A 2-ary relation that is used to identify the contexts of I relevant for the user,
- $objects\_a$: A 3-ary relation that is used to identify the objects of I relevant for the user,
- $references\_a$: A 4-ary relation that is used to identify the references of I relevant for the user.

These predicates are used to identify the relevant part of I that the user whishes to explore. It means that, if I is a well-formed context base instance and P is a well-formed program, $\forall x, y, z \in Dom$, the following must hold:

$$objects\_a(x, y, z) \in P(I) \Rightarrow objects(x, y, z) \in I$$
$$contexts\_a(x, y) \in P(I) \Rightarrow contexts(x, y) \in I$$
$$references\_a(x, y, z, t) \in P(I) \Rightarrow references(x, y, z, t) \in I.$$

Example: We present two examples of well-formed programs:

1. The following query asks for objects topic dealing with "Tourim" or "Malaysia" but not with "Borneo":

$$\begin{array}{lcl}
objects_a(x, \text{"}topic\text{"}, z) & \leftarrow & objects(x, \text{"}topic\text{"}, z), substring^1(z, \text{"}Malaysia\text{"}), \\
 & & \neg substring(z, \text{"}Borneo\text{"}) \\
objects_a(x, \text{"}topic\text{"}, z) & \leftarrow & objects(x, \text{"}topic\text{"}, z), substring(z, \text{"}Tourism\text{"}), \\
 & & \neg substring(z, \text{"}Borneo\text{"}) \\
objects_a(x, s, t) & \leftarrow & objects(x, s, t), objects_a(x, \text{"}topic\text{"}, z) \\
contexts_a(c, x) & \leftarrow & objects_a(x, s, t), contexts(c, x) \\
references_a(x, x_1, y_1, z_1) & \leftarrow & objects_a(x, s, t), references(x, x_1, y_1, z_1)
\end{array}$$

2. The following query asks for the parts of the context base that are reachable from a particular object o_1:

$$
\begin{array}{lcl}
ans(\text{"}o_1\text{"}, y) & \leftarrow & references(\text{"}o_1\text{"}, y, \_, \_) \\
ans(x, z) & \leftarrow & ans(x, y), references(y, z, \_, \_) \\
contexts\_a(c, x) & \leftarrow & ans(\text{"}o_1\text{"}, x), contexts(c, x) \\
objects\_a(o, att, val) & \leftarrow & ans(\text{"}o_1\text{"}, o), objects(o, att, val) \\
references\_a(o, o_2, att, val) & \leftarrow & ans(\text{"}o_1\text{"}, o), ans(\text{"}o_1\text{"}, o_2), references(o, o_2, att, val)
\end{array}
$$

5 The system level

In this section, we present the system level of our model, and the language used to navigate and edit the contents of the context base.

5.1 The system model

The system is a pair $\langle B, S\rangle$ where:

- B (for $Base$) is a well-formed context base instance,
- S (for $State$) is a triple $\langle P, o_{nav}, o_{ed}\rangle$ which represents what is displayed to the user:
  - P is a query (expressed as a Datalog$^\neg$ well-formed program) over B,
  - $o_{nav} \in Dom$ is the identifier of the object browsed by the user, called the navigated object,
  - $o_{ed} \in Dom$ is the identifier of the object edited by the user, called the edited object.

Example: Consider the system which interface is depicted Figure 1. The interface is divided into 5 zones :

- Since the instance of the context base is often too large to be displayed entirely, only the name of the context base is displayed in the top-left zone.
- The bottom-left zone is the edition zone. In this zone, the edited object o_{ed} and its context are displayed.
- The top-right zone allows to query the $Base$ by defining the query P. In this zone, there are three parts: the left-hand part allows to query the contents of the base, i.e., querying the objects descriptors or the references descriptors, the right-hand part allows to query the structure with some predefined program, and the central part allows the user to directly enter a program by the means of an ad-hoc interface (not detailed).
- The central-right zone displays the result of the query on B, i.e., $P(B)$.
- The bottom-right zone is the navigation zone. In this zone, the navigated object o_{nav} and its context are displayed.

5.2 The system language

An operation on the system, to change what is displayed, can be either:

- A navigation operation, to browse the elements of the context base. Navigation operations modify only the state of the system,
- An edition operation, to edit the elements of the context base. Edition operations modify the base and may change the state of the system.

5.2.1 Navigation system operations

Navigation can be done by changing the navigated object or by changing the query over the context base. These operations allow the user to browse the contexts and their contents.

There are two different ways to change the navigated object:

• $gotoObject$ which accesses an object knowing his identifier. Consider the system depicted Figure 1. The user has queried B and has obtained $P(B)$ which object descriptors are displayed in the central-right zone. He can click on a particular object in this zone. This action is associated with the operation $gotoObject$. In the same way, the user is visualising a particular object in a context in the bottom-right zone and he can click on another object in this zone. This action is also associated with the operation $gotoObject$.

• $nextObject$ which uses the references having o_{nav} as source object. Consider the system depicted Figure 1. The user is seeing the references descriptors of the navigated object in the bottom-right zone. He can access objects referenced by this object. This action is associated with the operation $nextObject$.

In the following examples, consider the system S_1 depicted Figure 1 with $State = \langle P, 2, 15\rangle$ where P is :

$objects\_a(x, "topic", z) \leftarrow objects(x, "topic", z), substring(z, "Malaysia")$
$objects\_a(x, "topic", z) \leftarrow objects(x, "topic", z), substring(z, "Tourism")$
$objects\_a(x, "topic", z) \leftarrow objects(x, "topic", z), substring(z, "Agriculture")$
$objects\_a(x, s, t) \leftarrow objects(x, s, t), objects_a(x, "topic", z)$
$contexts\_a(c, x) \leftarrow objects\_a(x, s, t), contexts(c, x)$
$references\_a(x, x_1, y_1, z_1) \leftarrow objects\_a(x, s, t), references(x, x_1, y_1, z_1)$

In the following, we present the operations needed to navigate contexts and objects.

gotoObject For a given object that the user is seeing on the interface, this operation allows to change the navigated object, the targeted navigated object being the parameter.
Definition: Let $System = \langle Base, State\rangle$, $State = \langle P, o_{nav}, o_{ed}\rangle$, o_1 be an object identifier such that there exists att_1, val_1 such that $objects(o_1, att_1, val_1) \in P(Base)$.
$gotoObject(System, o_1) = \langle Base, State'\rangle$ with $State' = \langle P, o_1, o_{ed}\rangle$.
Example: Consider the system S_1 depicted in Figure 1 with $State = \langle P, 2, 15\rangle$.
The user decides to browse the object described by $\langle topic, "Train,\ 2005"\rangle$ in the context he is seeing, by clicking on its descriptor. This action is associated with the operation: $gotoObject(S_1, 5)$.
He obtains the new system with $State' = \langle P, 5, 15\rangle$ depicted in Figure 2.

nextObject For a given reference between the navigated object and another object that the user cannot see on the interface but the reference descriptor of which are displayed, this operation allows to change the navigated object by moving forward in the object graph.
Definition: Let $System = \langle Base, State\rangle$ and $State = \langle P, o_{nav}, o_{ed}\rangle$ such that there exists a fact $references(o_{nav}, o_1, att, val) \in Base$ and
$objects(o_1, att_1, val_1) \in P(Base)$.
$nextObject(System, att, val) = \langle Base, State'\rangle$ with $State' = \langle P, o_1, o_{ed}\rangle$.

newQuery This operation allows to issue a new query and thus change the set of contexts that the user is navigating.

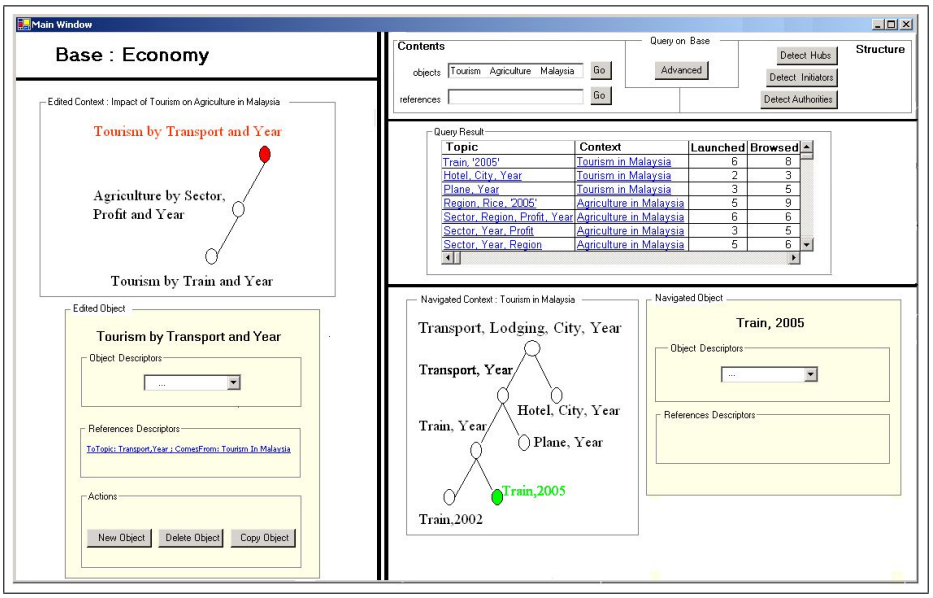

FIG. 2 – *Effect of the navigation operation gotoObject*

Definition: Let $System = \langle Base, State \rangle$, $State = \langle P, o_{nav}, o_{ed} \rangle$ and P_1 be a Datalog$^\neg$ well-formed program.
$newQuery(System, P_1) = \langle Base, State' \rangle$ with $State' = \langle P_1, NULL, o_{ed} \rangle$.
Example: Consider the system S_1 with $State = \langle P, 2, 15 \rangle$. The user decides to change P, i.e., to issue a new research on the base with:
$newQuery(S_1, P_1)$ where P_1 is:

$$
\begin{array}{lcl}
objects\_a(x, "topic", z) & \leftarrow & objects(x, "topic", z), substring(z, "France") \\
objects\_a(x, "topic", z) & \leftarrow & objects(x, "topic", z), substring(z, "Tourism") \\
objects\_a(x, s, t) & \leftarrow & objects(x, s, t), objects\_a(x, "topic", z) \\
contexts\_a(c, x) & \leftarrow & objects\_a(x, s, t), contexts(c, x) \\
references\_a(x, x_1, y_1, z_1) & \leftarrow & objects\_a(x, s, t), references(x, x_1, y_1, z_1)
\end{array}
$$

He obtains the new system with $State' = \langle q_1, NULL, 15 \rangle$ depicted in Figure 3.

5.2.2 Edition system operations

In the following, we present the basic operations needed to define and edit contexts and objects. Edition operations modify the base and may change the state of the system.
As well as the classical primitives of a DDL (Data Definition Language), this language features complex operations are defined like the $copyObject$ operation that duplicates the navigated object into the edited context.
Note that all the edition operations like delete, ... are not detailed in this section.

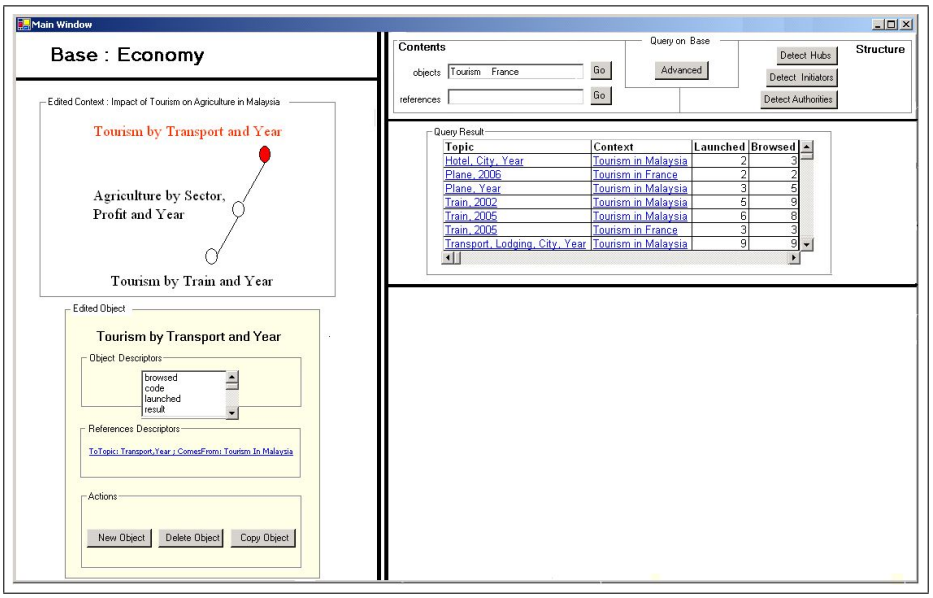

FIG. 3 – *Navigation operation example: newQuery*

createObjectInNewCxt This operation creates a new object in a new context.
Definition: Let $System = \langle Base, State\rangle$, $State = \langle P, o_{nav}, o_{ed}\rangle$, let $c_1 \in Dom$ be a context identifier that does not appear in $Base$ and let $o_1 \in Dom$ be a object identifier that does not appear in $Base$.
$createObjectInNewCxt(System, att, val) = \langle Base', State'\rangle$ where
- $Base' = Base \cup \{contexts(c_1, o_1), objects(o_1, att, val)\}$
- $State' = \langle P, o_{nav}, o_1\rangle$

createObjectInExistingCxt This operation adds an object to the context of object o_{ed}.
Definition: Let $System = \langle Base, State\rangle$, $State = \langle P, o_{nav}, o_{ed}\rangle$ and $o_1 \in Dom$ be a object identifier that does not appear in $Base$ and
$contexts(c_{ed}, o_{ed}) \in Base$.
$createObjectInExistingCxt(System, att, val) = \langle Base', State'\rangle$ where
- $Base' = Base \cup \{objects(o_1, att, val), contexts(c_{ed}, o_1)\}$
- $State' = \langle P, o_{nav}, o_1\rangle$.

copyObject This operation duplicates the navigated object o_{nav} into the edited context.
Definition: Let $System = \langle Base, State\rangle$, $State = \langle P, o_{nav}, o_{ed}\rangle$ and $o_1 \in Dom$ be a new object identifier that does not appear in $Base$.
$copyObject(System) = \langle Base', State'\rangle$ where
- $Base' = Base \cup I$ and $I = \{objects(o_1, x, y) | objects(o_{nav}, x, y) \in Base\} \cup \{references(o_1, o, s, t) | references(o_{nav}, o, s, t)\}$
- $State' = \langle P, o_{nav}, o_1\rangle$.

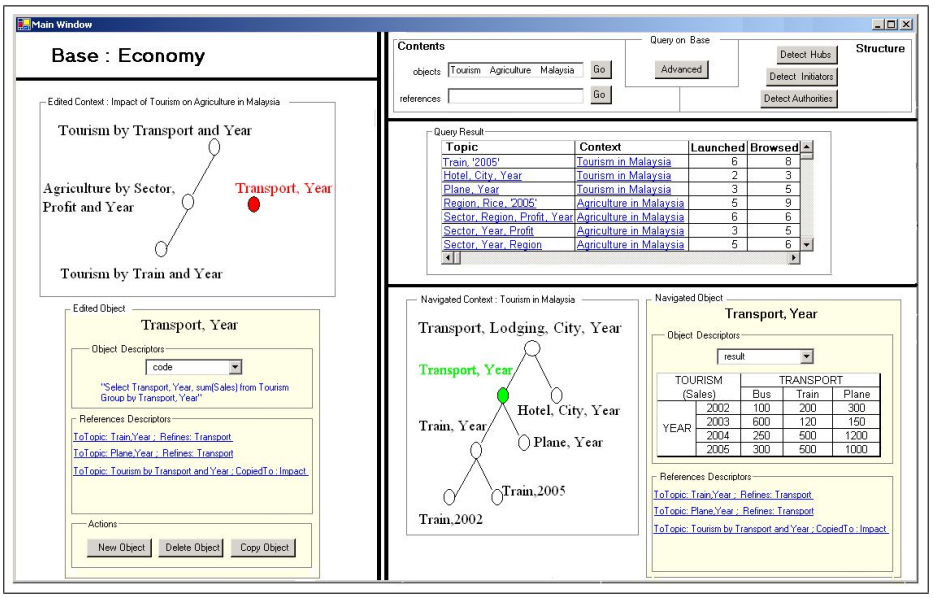

FIG. 4 – *Edition operation example: copyObject*

Example: Consider the system S_1 depicted Figure 1. The user decides to duplicate the navigated object, i.e., the object identified by 2 into the edited context (which contains o_{ed}), i.e., the context identified by 3, with $copyObject(S_1)$.
He obtains the new system depicted in Figure 4:

- $Base' = Base \cup \{objects(20, topic, "Transport,\ Year"), objects(20, code, "Select\ Transport,\ Year,\ sum(Sales)\ from\ Tourism\ Group\ by\ Transport,\ Year"), objects(20, launched, 6), objects(20, browsed, 9), objects(20, result, CT_2), contexts(3, 20), references(20, 3, intra - link, "contain"), references(20, 6, intra - link, "contain"), references(20, 15, extra - link, "copied - to")\}$
- $State' = \langle P, 2, 20 \rangle$

addDescriptorToObject This operation adds a descriptor to the edited object o_{ed}.
Definition: Let $\langle att, val \rangle$ be a descriptor, $System = \langle Base, State \rangle$ and $State = \langle P, o_{nav}, o_{ed} \rangle$.
$addDescriptorToObject(System, att, val) = \langle Base', State \rangle$ where
$Base' = Base \cup \{objects(o_{ed}, att, val)\}$.

addRefBetweenObjects This operation adds a reference between the edited object o_{ed} and the navigated object o_{nav}.
Definition: Let $System = \langle Base, State \rangle$, $State = \langle P, o_{nav}, o_{ed} \rangle$ and $\langle att, val \rangle$ be a reference descriptor.
$addRefBetweenObject(System, att, val) = \langle Base', State \rangle$
with $Base' = Base \cup \{references(o_{ed}, o_{nav}, att, val)\}$.

6 Exploiting the system

In this section, we present how the model can be used to provide relevant information for the user browsing a context base. First, we note that the semantics of the data organization in a context base is held both by descriptors and by references. The following two subsections show how this can be exploited.

6.1 Exploiting the descriptors

There are two types of descriptors: Descriptors associated with objects and descriptors associated with references. Descriptors can be added by the user, when editing an object or a reference, or by the system itself to update automatically the information collected on the data.

Descriptors associated with objects Most of the time, these descriptors are added by the user to characterize the objects. Examples of such desciptors are:

- *topic*, which value is a textual description of the query,
- *context*, which value indicates in which context the query is contained,
- *code*, which value is the SQL code of the query,
- *result*, which value is the result of the query, classicaly displayed under the form of a cross-tab, like in the bottom-right zone of Figure 1.

Examples of how to use these descriptors to query the context base have been given in the previous sections.

In some cases, descriptors are added and updated by the system to collect information related to users navigation. Example of such descriptors can be:

- *launched*, which value is a counter that indicates how many times the query has been evaluated. Each time the descriptor "result" is displayed, this counter is incremented.
- *browsed*, which value is a counter that indicates how many times the query has been browsed. Each time the objects is browsed, this counter is incremented.

Let us illustrate how these descriptors can be used. Suppose the user wants to know the queries that have been asked (i.e., launched) more than 10 times. The following program can be used to detect these queries:

$$
\begin{array}{lcl}
contexts\_a(c, o_1) & \leftarrow & objects(o_1, "launched", x), x > 10, contexts(c, o_1) \\
objects\_a(o_1, att, val) & \leftarrow & objects(o_1, "launched", x), x > 10, objects(o_1, att, val) \\
references\_a(o_1, o_2, att, val) & \leftarrow & objects(o_1, "launched", x), x > 10, \\
 & & references(o_1, o_2, att, val)
\end{array}
$$

Descriptors associated with references These descriptors are used to indicate how objects are organised within a context (the case of *intra-context* references) or are related to objects in another context (the case of *inter-context* references).

In the case of intra-context reference, if objects are queries over data warehouses, these descriptors can be for example:

- *order of importance*, which value reflects an ordering relation over the queries of a context that is interpreted as an order of importance relevant to the user.
- *query containment*, in that case the reference connects two queries such that one refines the other in the usual sense of query containment,
- *query logs*, which value reflects in which order the queries have been defined.

In the case of inter-context references, in addition to the bookmarks that a user may want to keep in order to indicate that a object in another context is of interest, descriptors can be automatically added by the system to keep information related to users editing operation. These descriptors can be for example:

- *comes from*, that indicates that the source object is a copy of another object (i.e., the target of the reference), and thus has been borrowed from another context,
- *copied to*, that indicates that the source object has been copied into another context.

6.2 Exploiting the references

In Section 3 we have noted that a context base can be described as a graph which vertices are the objects and edges are the references between objects. This is very similar to a model used to describe the World Wide Web, where web pages are viewed as vertices and links between the pages are viewed as edges. With this approach, [Kle99] noted that not only the contents of the pages but also the structure of the graph held useful information. This led him to define the notions of *hubs* and *authorities*:

- *Authorities* are pages with large in-degree, i.e., that are pointed to by a large number of hyperlinks.
- *Hubs* are pages that have links to multiple relevant authorities.

We propose to adapt these notions to our model in the following way:

- An *Authority context* is a context which is referenced by all the other contexts,
- A *Hub context* is a context which allows to acess all other contexts.

More important, hub contexts and authority contexts, as we have defined them, can be found by using a Datalog$^{\neg}$ well-formed program, with a division (hence the need for having the negation in the language):

- What are the hub contexts?

$$
\begin{array}{lcl}
link(c1, c2) & \leftarrow & contexts(c1, o1), contexts(c2, o2), \\
 & & references(o1, o2, \_, \_) \\
possiblelink(c1, c2) & \leftarrow & contexts(c1, \_), contexts(c2, \_) \\
nolink(c1, c2) & \leftarrow & possiblelink(c1, c2), \neg link(c1, c2) \\
contexts\_a(c1, x) & \leftarrow & contexts(c1, x), contexts(c2, y), \neg nolink(c1, c2) \\
objects\_a(x, a, v) & \leftarrow & contexts\_a(c1, x), objects(x, a, v) \\
references\_a(x1, x2, a, v,) & \leftarrow & contexts\_a(c1, x1), contexts_a(c2, x2), \\
 & & references(x1, x2, a, v)
\end{array}
$$

- What are the authority contexts?

$$
\begin{array}{lcl}
link(c2, c1) & \leftarrow & contexts(c1, o1), contexts(c2, o2), \\
 & & references(o2, o1, \_, \_) \\
possiblelink(c2, c1) & \leftarrow & contexts(c1, \_), contexts(c2, \_) \\
nolink(c2, c1) & \leftarrow & possiblelink(c2, c1), \neg link(c2, c1) \\
contexts\_a(c1, x) & \leftarrow & contexts(c1, x), contexts(c2, y), \neg nolink(c2, c1) \\
objects\_a(x, a, v) & \leftarrow & contexts\_a(c1, x), objects(x, a, v) \\
references\_a(x2, x1, a, v,) & \leftarrow & contexts\_a(c1, x1), contexts\_a(c2, x2), \\
 & & references(x2, x1, a, v) \\
authority(c) & \leftarrow & contexts\_a(c, \_)
\end{array}
$$

In addition to hub contexts and authority contexts, we propose the new notion of *initiator context*: An initiator context is an authority that contains a query o referencing a query o' in an authority and such that o is not referenced.

Initiators can be found with the following well-formed program:
What are the initiators contexts?

$$
\begin{array}{lcl}
initiator(c) & \leftarrow & authority(c), contexts(c,o), references(o,o',\_,\_), \\
 & & contexts(c',o'), c' \neq c, authority(c'), \\
 & & \neg references(o'',o,a,v), objects(o'',\_,\_), \\
 & & references(\_,\_,a,v) \\
objects\_a(o,x,y) & \leftarrow & initiator(c), objects(o,x,y) \\
contexts\_a(c,o) & \leftarrow & initiator(c), contexts(c,o) \\
references\_a(o,o',s,t) & \leftarrow & objects\_a(o,x,y), references(o,o',s,t)
\end{array}
$$

In the particular case whereby the descriptor of the references in the previous program is "copied-to", initiators allow to detect emergent tendencies, since they are authorities containing queries that have not been borrowed and that are copied into some other contexts.

Recommendations during browsing The term *recommendation* is borrowed from the E-commerce domain [SKR01], and consists in indicating to the user that, if he is interested in a given object, then he may also be interested by some other objects for some reasons. We illustrate how a particular type of reference descriptors can be used to propose such recommendations. The reference associated with the "copied-to" descriptor is set when a user copies an object from the navigated context (hereafter denoted the source object) to the edited context (hereafter denoted the target object).

Suppose now that, in some subsequent OLAP session, the source object is browsed by some user. The user may be interested in these objects that are referenced by the source object he is viewing. As this source object is a copy of the target object (since there is a "copied-to" reference linking the two objets), the user may also be interested in those objects that are referenced by the targeted object. The program returning such recommendations is as follows:
What are the recommendations started from o_1?

$$
\begin{array}{lcl}
ans(o_2) & \leftarrow & objects(o_2,\_,\_), references(o_1,o_2,"copied-to",y) \\
objects\_a(o_3,a,v) & \leftarrow & ans(o_2), references(o_2,o_3,z,t), objects(o_3,a,v) \\
contexts\_a(c,o_3) & \leftarrow & objects\_a(o_3,a,v), contexts(c,o_3) \\
references\_a(o_3,o_4,w,x) & \leftarrow & objects\_a(o_3,a,v), objects(o_4,\_,\_), \\
 & & references(o_3,o_4,w,x)
\end{array}
$$

7 Conclusions and Future Work

Conclusion This paper proposes a model for OLAP analysis, i.e., a model for sharing, browsing and reusing OLAP queries over a data warehouse. This model consists in two levels:

- The data level which organises the queries into a graph and a set of contexts called the context base,
- The system level which represents the interface proposed to the user for sharing, browsing and reusing OLAP queries.

We introduce two languages, i.e., a data manipulation language and a system language with navigation and edition operations, for defining, manipulating and browsing the context base. We illustrate our proposals with various examples of use, including the exploitation of the structure of the graph to provide relevant information.

Future work In the first hand, we plan to place queries as first class citizen to use the characteristics of OLAP queries and OLAP analysis.

In the second hand, we plan to extend the data manipulation language incorporating aggregation faclities in order to sophisticate our definitions of hubs, authorities, initiators, recommendations. We will also investigate new operations allowing to define new queries by combining browsed queries.

Then, we will propose an extension of the navigation language to more sophisticated forms of browsing, e.g., being able to coming back to a previously seen query or replaying an analysis.

Finally, we will implement our model for data warehouses exploitation.

Acknowledgments

This work was carried out within the context of the ICT-Asia project: EXPEDO (Exploitation of Data Warehouses).

References

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AS04] Mina Akaishi and Nicolas Spyratos. Discovering implicit relationships in a web of contexts. In *Intuitive Human Interfaces for Organizing and Accessing Intellectual Assets*, pages 175–188, 2004.

[AST03] Mina Akaishi, Nicolas Spyratos, and Yuzuru Tanaka. Contextual search in large collections of information resources. In *EJC: European-Japanese Conference on Information Modelling and Knowledge Bases*, pages 295–302, 2003.

[DKK05] Jens-Peter Dittrich, Donald Kossmann, and Alexander Kreutz. Bridging the gap between olap and sql. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1031–1042. VLDB Endowment, 2005.

[GMN06] A. Giacometti, P. Marcel, and E. Negre. OLAP: un pas vers la navigation. In *EDA : Journée francophone sur les Entrepôts de Données et l'Analyse en ligne*, 2006.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[SKR01] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.

[TACS02] M. Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. A theory of contexts in information bases. *Inf. Syst.*, 27(3):151–191, 2002.