Association for Information Systems

AIS Electronic Library (AISeL)

ICEB 2009 Proceedings

International Conference on Electronic Business (ICEB)

Winter 12-4-2009

A Functional Model for Data Analysis and Result Visualization

Nicolas Spyratos

Ekaterina Simonenko

Tsuyoshi Sugibuchi

Follow this and additional works at: https://aisel.aisnet.org/iceb2009

This material is brought to you by the International Conference on Electronic Business (ICEB) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ICEB 2009 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

A FUNCTIONAL MODEL FOR DATA ANALYSIS AND RESULT VISUALIZATION

Nicolas Spyratos[1], Ekaterina Simonenko[2] and Tsuyoshi Sugibuchi[3]
LRI-CNRS UMR 8623, Université Paris-Sud XI
91405 Orsay Cedex, France
[1]spyratos@lri.fr; [2]simonenk@lri.fr; [3]buchi@lri.fr

Abstract

In several Web based applications (e-commerce, e-learning, digital libraries, etc.) one needs to display a dense array of information in a small amount of space (such as a screen) in a manner that communicates clearly and immediately. The information displayed is usually aggregates of results obtained through analysis of large amounts of data. We present a functional model that supports the data analysis and aggregation process, and a prototype that supports casual users in doing the following: (a) construct an analytic query visually, in an interactive manner, (b) visualize the aggregate result in a user selected mode (histogram, pie, etc.), (c) explore the query result by providing equivalent representations at different aggregation levels or for different parameter values selected by the user.

Keywords: Data Analysis, Analytic Query, Data Visualization, Visual Interaction

Introduction

In several Web based applications such as e-commerce, e-learning, digital libraries, etc. one needs to display a dense array of information in a small amount of space (such as a screen) in a manner that communicates clearly and immediately. The information displayed is usually aggregates of results obtained through analysis of large amounts of data.
For example, consider the case of a digital library, allowing a community of users to share documents in digital form. Each document resides in its owner's (local) database, and the digital library acts simply as a mediator allowing subscribers to access documents transparently. To access a desired document, a subscriber queries the library catalogue which stores each document's URI together with a description of the document that is the values of the document's attributes (language, topic, author, etc.). The catalogue can be seen as a table (see Figure 1), and a query against the catalogue is just a Boolean combination of elementary conditions of the form "$A = a$", where A is an attribute appearing in the column headings and a is a value of that attribute; for example, referring to Figure 1, the following is a query asking for documents in $French$ on $Poetry$: $(Language = French)$ and $(Topic = Poetry)$.

The digital library administration needs to perform usage analysis in order to plan the library's activities. Such analysis usually concerns the ranking of documents along several dimensions (e.g. topic, author, language or any combination thereof) according to certain indicators (e.g. number of hits). However, the results of such analysis can be very large in size, and the only way to make sense out of big volumes of data is to create summaries and to display the summarized results to the analyst in an appropriate visualization mode - ideally, one selected by the analyst himself. Moreover, the analyst should be able to perform exploratory analysis on the visualized results by changing the summarization level or by viewing different, yet equivalent representations of the results that might reveal new, interesting information. There are several offerings today by software companies that allow analyzing large volumes of data and visualizing the results [1, 3], including some open source software [2]. However all these tools are closely related to the relational model and require some knowledge of the SQL constructs related to data analysis (such as grouping sets, cube, roll up etc.). In this paper, we present an approach that supports the data analysis and aggregation process visually, at a minimum effort by the user. More precisely, we present a simple data model allowing analysts to do the following:

URI	Language	Topic	Author
1	French	drama	Dumas
2	Italian	poetry	Dante
3	English	poetry	Spenser
4	English	drama	Kipling
5	French	novel	Mérimée
6	Spanish	drama	De Vega

Figure 1: A table, representing a catalog of a digital library.

1. construct an analytic query *visually*, in an *interactive* manner;
2. *visualize* the aggregate result in a user selected mode (histogram, pie, etc.);
3. *explore* the query result by viewing equivalent representations at different aggregation levels or for different parameter values selected by the user.

We also present the basic architecture of a prototype that implements the above functionalities. A prominent feature of our prototype is that the process of creating a query, receiving results, analyzing them and exploring them in several ways is well integrated, supported by intuitive actions: the entire process is very fluent and straightforward. In the remaining of the paper, we first give an informal overview of our data model; then define the formal model and give its mapping to the relational model. Next we give the description of a prototype interface under development, as well as the visual interaction between the user and the interface. We conclude with remarks and an outline of future work.

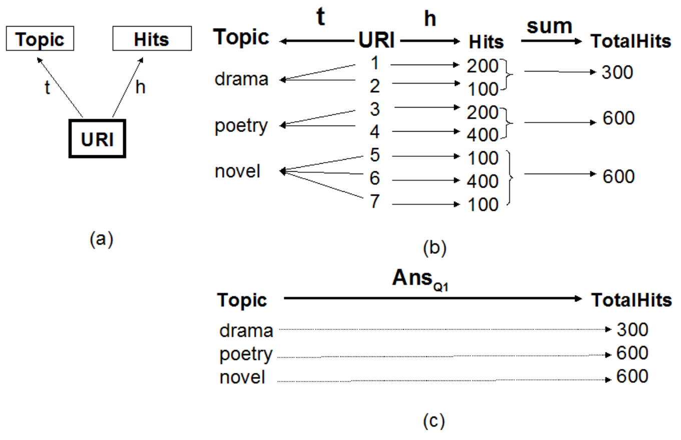

Figure 2: Example of an application represented by a graph, and a query evaluation.

Overview of the model

In this section we give an informal overview of our model using an example. Consider a digital library in which each document is identified by its URI and described by two attributes:
$Topic$, whose values are keywords describing document content (eg. drama, poetry, etc.)
$Hits$, whose values are integers representing the number of accesses to the document.
Thus each document URI in the library is associated with one keyword (its topic description) and one integer (its number of hits).
Therefore we have two functional dependencies
$t : URI \to Topic$ and $h : URI \to Hits$
(we need the labels t and h for later reference).
We can describe this application by a graph, as shown in Figure 2.(a), where the attributes URI, $Topic$ and $Hits$ are the nodes and the the functional dependencies t and h are the edges. This graph is a first, rudimentary example of what we call analytic schema (or simply "schema"). Its *origin* is the attribute URI which also happens to be a key (in this example). Roughly speaking, the origin of a schema represents the objects of interest, while all other nodes describe attributes of the objects.
In our approach, we interpret the edges
$t : URI \to Topic$ and $h : URI \to Hits$
of the schema as function signatures and their extensions as the (current) database. Figure 2.(b) shows an example of (current) database. This database represents all documents accumulated so far in the library. It contains 9 document URIs, each associated with its topic and its number of hits through the functions t and h (for simplicity, we represent URIs as integers); for example, document 3 is associated with "Poetry" as topic and with 200 as number of hits (i.e. $t(3) = Poetry$ and $h(3) = 200$).

Suppose now that we want to analyze document usage, say by finding the total number of hits by topic, that is by evaluating the answer to the following query against the current database:

Q_1: "total number of hits by topic."

In our approach, in order to answer this query we proceed in three steps as follows:

Grouping: we invert the function t, thus grouping the URIs by topic;
Measuring: in each group, we apply the function h to each URI of the group to find the corresponding number of hits;
Aggregation: in each group, we sum up the results of measuring to have the total number of hits for that group.
The final result is shown in Figure 2.(c), and it is a function from $Topic$ to a new attribute that we call $TotalHits$. This function associates each topic with the total number of hits for that topic. In other words the answer to Q_1 is the following function:

$$Ans_{Q_1} : Topic \to TotalHits, \qquad (1)$$

such that $Ans_{Q_1}(x)$ is the total number of hits, for each topic x. For example, $Ans_{Q_1}(Drama) = 300$ and $Ans_{Q_1}(Poetry) = 600$.

This pattern of grouping a set of objects by inverting a function defined on them, then measuring a property in each group by applying a second function also defined on them, and finally aggregating the measures in each group by applying an operation on the measures constitutes the basic pattern of our approach.

It should be clear from the previous example that the specification of query Q_1 requires three parameters, a function such as t for classifying the URIs by topic, a function such as h for measuring the number of hits by URI, and an operation such as "*sum*" for aggregating the measured numbers of hits. Therefore Q_1 can be specified as a triple:

$$Q_1 = \langle t, h, sum \rangle. \qquad (2)$$

Notice however that t and h have the origin of the schema as their common domain of definition, and that this condition is indispensable in order to compute the answer. Moreover, notice that the operation "*sum*" is an operation which is possible to apply over the range of h (i.e. over the integers), and that this condition is also indispensable in order to compute the answer.

In view of the previous discussion, in our approach, we define an analytic query over a schema to be a triple

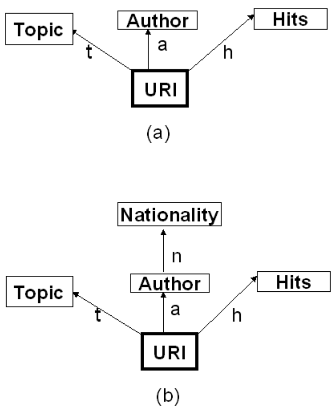

Figure 3: Adding Author and Nationality.

$Q = \langle c, m, op \rangle$, where c and m are edges of the schema having the origin as their common domain of definition, and *op* is an operation which is possible to apply over the range of m. We refer to the function c as the *classifier* or the *grouping function* of Q and to the function m as the *measure*.

It is interesting to note that, following the above definition, one can interchange the roles of c and m to obtain a different analytic query (provided of course that the operation in the resulting query is possible to apply over the range of the new measure). For example, one can interchange the roles of t and h in query Q_1 to obtain the query $Q_1' = \langle h, t, sum \rangle$. However, this query is not well formed since the operation "*sum*" cannot be applied over the range of t (i.e. over $Topic$, as topics can't be summed). By the way, if one changes the operation from "*sum*" to "*count*" then one obtains a well formed query, namely

$$Q_1'' = \langle h, t, count \rangle, \tag{3}$$

asking for the number of topics by number of hits.

Continuing with our example, suppose now that, in addition to topic description and number of hits, each document is also associated with an author. Then we obtain a new schema as shown in Figure 3.(a), where we added the edge $a : URI \rightarrow Author$. We can now formulate the following query, asking for the total number of hits per author:

$$Q_2 = \langle a, h, sum \rangle. \tag{4}$$

The answer will be a function from authors to integers: $Ans_{Q_2} : Author \rightarrow TotHits$ such that $Ans_{Q_2(x)}$ is the total number of hits, for each author x.

Now, however, it makes sense to also ask the following query:

Q_3: "total number of hits by topic-author pair".

This time the grouping of URIs will be done according to a function derived from t and a, which associates each URI with a topic-author pair. This function is called the *pairing* of t and a, it is denoted by $t \wedge a$, and it is defined as follows: $t \wedge a : URI \rightarrow Topic \times Author$ such that $(t \wedge a)(x) = \langle t(x), a(x) \rangle$.
During evaluation of Q_3, the pairing $t \wedge a$ will group together all URIs having the same topic and the same author, and the answer will associate each topic-author pair with a total number of hits. In other words, compared to Q_1, the only change is that the function t of Q_1 is now replaced by the function $t \wedge a$; other than that, the evaluation of Q_3 proceeds in exactly the same way as for Q_1. Therefore, Q_3 is specified as follows:

$$Q_3 = \langle t \wedge a, h, sum \rangle. \tag{5}$$

And its answer will be a function from topic-author pairs to integers:

$$Ans_{Q_3} : Topic \times Author \rightarrow TotHits, \tag{6}$$

such that $Ans_{Q_3}((x, y))$ is the total number of hits, for each topic-author pair (x, y).

As another example, suppose that we also add the nationality of each author. Then we obtain a new schema as shown in Figure 3.(b). We can now ask the following analytic query:

Q_4: "total number of hits by author's nationality".

This time, during evaluation, the grouping of URIs will be done according to a function derived from a and n using functional composition. The composition of a and n, denoted $n \circ a$, associates each URI with the corresponding author's nationality. During evaluation of Q_4, the function $n \circ a$ will group together all URIs having the same author nationality, and the answer will associate each author nationality with a total number of hits. In other words, compared to query Q_1, the only change is that the function t of Q_1 is now replaced by the function $n \circ a$; other than that, the evaluation of Q_4 proceeds in exactly the same way as for Q_1. Therefore Q_4 is specified as follows:

$$Q_4 = \langle n \circ a, h, sum \rangle. \tag{7}$$

And its answer will be a function from $Nationality$ to $TotHits$:

$$Ans_{Q_4} : Nationality \rightarrow TotHits, \tag{8}$$

such that $Ans_{Q_4}(x)$ is the total number of hits, for each nationality x.

As a final example, we can ask the following analytic query over the schema of Figure 3(b):

Q_5: "total number of hits by topic-nationality pair".

Again, all we have to do in order to evaluate this query is to replace the function t of Q_1 by the function $c \wedge (n \circ a)$. Therefore Q_5 is specified as follows:

$$Q_5 = \langle c \wedge (n \circ a), h, sum \rangle. \tag{9}$$

And its answer will be a function from topic-nationality pairs to integers:

$$Ans_{Q_5} : Topic \times Nationality \rightarrow TotHits, \tag{10}$$

such that $Ans_{Q_5}((x,y))$ is the total number of hits, for each topic-nationality pair (x,y).

Notice that, in all the above examples of queries (Q_1 to Q_5), we can also restrict any of the functions involved to some desirable subset of its domain of definition, to form new analytic queries. In fact, the set of all operations on functions that we shall use to derive new functions from old constitutes what we shall call the functional algebra of a schema. These operations are quite elementary: composition, pairing, restriction and projection. Yet, as we shall see, they allow us to associate each schema with a powerful language of analytic queries.

The Formal Model

As we mentioned earlier, we view the schema as a set of function signatures and a database as a set of (finite) extensions, one for each function signature. In this section, we keep with this view and we define a language in which analytic queries can be formulated over the schema and evaluated over the database. We shall illustrate the concepts introduced by the following example that we shall use as our running example for the rest of the paper.

Running example
A big catering company delivers various products to retail stores over the whole country. The following data appears on each delivery invoice:

- the invoice number
- the date of delivery
- the store identifier
- a type of product (e.g. "Coca Light")
- and the number of items delivered (from that type of product)

These data are recorded in the database of the catering company, and accumulated over long periods of time with the purpose of analyzing them in order to improve the company's delivery service. More specifically, the analyses performed are:

- by date and by month
- by store, by city and by region
- by supplier and by product category
- by combinations thereof, such as by date and store, or by month and region etc.

We assume that a supplier might supply products in two or more categories and that a product category might be supplied by two or more suppliers.
The schema concerning this application is shown in Figure 4, where O stands for "invoice number", that is O represents the set of all invoice numbers accumulated so far. We shall use this schema in order to introduce the basic concepts of the query language.

In our explanations we shall use the notation $f : X \rightarrow Y$ to denote an edge with *label* f, *source* X and *target* Y; similarly, we shall talk of the source and the target of a path.

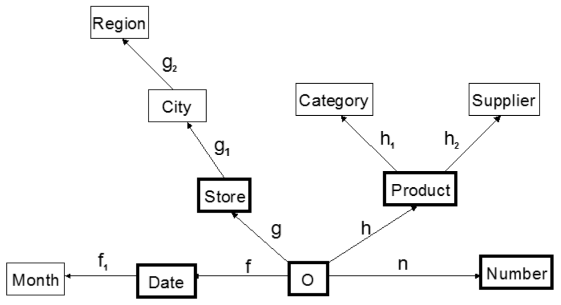

Figure 4: The functional schema S of the running example.

Functional database

Given a schema S, a *database* over S is a function δ that associates :

- each node N of S with a finite subset $\delta(N)$ of $dom(N)$, and
- each arrow $f : X \rightarrow Y$ of S with a total function $\delta(f) : \delta(X) \rightarrow \delta(Y)$.

In order to simplify notation, we shall omit the symbol δ and we shall use the expression "function $f : X \rightarrow Y$" to mean "function $\delta(f) : \delta(X) \rightarrow \delta(Y)$".
We note that the fact that all functions in the database are total imposes the following constraint:

referential constraint : for every pair of functions of the form $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ we must have $range(f) \subseteq def(g)$.

Roughly speaking, the schema is seen as a set of function signatures and the database stores their extensions.

Several remarks are in order here. First, in order to simplify the presentation, we adopt the following abuse of notation: we use an arrow label such as f to denote both the arrow f and the function $\delta(f)$ assigned to f by δ. Similarly, we use an attribute label such as X to denote both the attribute X and the finite set $\delta(X)$ assigned to X by δ. This should create no confusion, as more often than not the schema will resolve ambiguity. Second, the definition of a database requires that all functions assigned by the database δ to the arrows of S be total functions. This restriction could be relaxed, by endowing each attribute domain with a bottom element $\perp$ (meaning "undefined") and requiring that for any function $f : X \rightarrow Y$ we have (a) $f(\perp) = \perp$, that is "bottom can only map to bottom", and (b) if $x \notin def(f)$ then $f(x) = \perp$. Under these assumptions, the functions can again be considered as total functions. However, the resulting theory would be more involved and would certainly obscure some of the important points that we would like to bring forward concerning analytic queries.

The Functional Algebra

In order to combine the function extensions in the database of a schema, we need a set of operations on functions that we call the *functional algebra*. The functional algebra comprises four operations. Each operation takes as input one or two functions and returns a new function as a result:

Composition : takes as input two functions, f and g, such that $range(f) \subseteq def(g)$, and returns a function $g \circ f : def(f) \rightarrow range(g)$ defined by

$$g \circ f(x) = g(f(x)), forall x \in def(f). \quad (11)$$

Pairing : takes as input two functions, f and g, such that $def(f) = def(g)$, and returns a function $f \wedge g : def(f) \rightarrow range(f) \times range(g)$ defined by

$$f \wedge g(x) = \langle f(x), g(x) \rangle, \quad (12)$$

for all $x \in def(f)$.

Projection : it's the usual operation on the cartesian product of sets.

Restriction : takes as input a function $f : X \rightarrow Y$ and a set $E \subseteq def(f)$, and returns a function $f/_E : E \rightarrow Y$ defined by

$$f/_E(x) = f(x), \quad (13)$$

for all $x \in E$.

The following proposition states an important property of the functional algebra.

Proposition 1
For every pair of functions $f : X \rightarrow Y$ and $g : X \rightarrow Z$, we have

$$f = \pi_Y \circ (f \wedge g) \text{ and } g = \pi_Z \circ (f \wedge g).$$

Nota : $\pi_Y()$ and $\pi_Z()$ are the projection functions over $Y \times Z$, defined by: $\pi_Y(y, z) = y$ and $\pi_Z(y, z) = z$, for all pairs $(y, z) \in Y \times Z$.

Path expressions
Given a schema S, a *path expression* over S is a well formed expression whose operands are arrows from S and whose operations are those of the functional algebra. Every path expression e is associated with a source and a target, defined recursively, based on the notions of source and target of the arrows in S. For example, if $e_1 = g_2 \circ g_1$ then $source(e_1) = Store$ and $target(e_1) = Region$; similarly, if $e_2 = (g_1 \circ g) \wedge f$ then $source(e_2) = O$ and $target(e_2) = City \times Date$ (for a formal definition of a path expression see [?]).

Given a path expression e and a database δ over S, the evaluation of e with respect to δ is done in two steps, as follows:

1. replace each arrow f appearing in e by the function $\delta(f)$ that the database δ associates with f

2. perform the operations of the functional algebra as indicated in e.

Note that the result of the evaluation is always a function; therefore we have a closure property as in the case of the relational algebra.

A particular kind of path expression will be of interest, namely the one that corresponds to projection over the empty set. To understand the nature of this projection, recall that, given a Cartesian product of k sets, say $A_1 \times ... \times A_k$, there are as many projection functions as there are subsets of the set $A_1, ..., A_k$. The projection function that corresponds to the empty set is the one that we call the empty projection function, hence we denote it by $\pi_\emptyset$. Clearly, following the definition of a projection function, the function $\pi_\emptyset$ is a constant function as it associates every tuple of $A_1 \times ... \times A_k$ with the empty tuple; we shall denote the empty tuple by λ. Therefore, $\pi_\emptyset(t) = \{\lambda\}$, for all $t \in A_1 \times ... \times A_k$ (assuming there is at least one such t). In view of our previous discussion, we introduce a particular path expression, the *empty path expression*, which will always be associated with the empty projection function, in any database δ. We denote the empty path expression by ε_X, where X denotes the source of the empty path expression (its target being always interpreted as $\{\lambda\}$). The empty path expression with source O will be simply denoted by ε.

OLAP query

Given a schema S, an *OLAP query* over S is a triple

$$Q = \langle c, m, op \rangle, \quad (14)$$

where:

- c and m are path expressions over S such that $source(c) = source(m)$ and
- op is an operation among those authorized over the target of m.

For example, in the schema S of our running example (Figure 4) :

$$Q = (g \wedge (h_2 \circ h), n, sum) \quad (15)$$

is an OLAP query over S, with
- $c = g \wedge (h_2 \circ h)$, $m = n$ and $op = sum$ and we have
- $source(g \wedge (h_2 \circ h)) = source(n) = O$, and sum is an authorized operation over the target of n (which is $Number$, with $dom(Number) = Int$).

Given an OLAP query $Q = \langle c, m, op \rangle$, we call c the *classifier*, m the *measure* and op the *operation* (or the *aggregator*) of Q. Moreover, we call the target of c the classification level (or the grouping level), and the target of m the measurement level. Note that the classification level, or the measurement level, might be composed of other simpler levels, as is the case in our previous example where $target(c) = Store \times Supplier$.

Evaluation of an OLAP query

Given an OLAP query $Q = \langle c, m, op \rangle$ and a database δ over S, the answer to Q with respect to δ is a function $ans_{Q,\delta} : range(c) \rightarrow target(op)$ computed in two steps, as follows:

1. Evaluate the path expressions c and m with respect to δ.

```
{This step returns two functions that we also denote as $c$ and $m$. Let's call $X$ the common domain of definition of the functions $c$ and $m$, that is $X = source(c) = source(m)$ ; and let $Y = \{y_1, ..., y_n\}$ be the set of values of $c$, that is $Y = range(c) = \{y_1, ..., y_n\}$}

2. For each value $y \in range(c)$ do
   begin
     (a) Grouping
         compute the inverse $c^{-1}(y)$
         {let $c^{-1}(y) = \{x_1, ..., x_r\}$}
     (b) Measurement
         for each $x \in c^{-1}(y)$ do compute $m(x)$
         {this step returns a tuple
         $t(y) = \langle m(x_1), ..., m(x_r)\rangle$}
     (c) Aggregation
         apply the operation $op$ to the tuple $t(y)$
         {call the result $Res(y)$, that is $Res(y) = op(t(y))$}
     (d) Answer
         define $ans_{Q,\delta}$, $(y) = Res(y)$
   end
```

We note that the variable Res used in the evaluation algorithm (step 2.(c)) does not appear as a node of the schema; it is actually an auxiliary variable defined by the user in order to receive the results of the computation. Moreover, as δ is understood (it's always the current database), we shall drop δ from the notation of the answer, and we shall use the symbol ans_Q or $ans(Q)$.

A particular form of OLAP query is the following :

$$Q = \langle \varepsilon, m, op \rangle. \tag{16}$$

During its evaluation, step 1 of the evaluation algorithm returns a constant function with λ as its only value; step 2.(a) (grouping) returns just one group $c^{-1}(\lambda) = \{X\}$; step 2.(b) (measurement) returns the images under m of all elements of X; step 2.(c) (aggregation) applies the operation on all images to obtain a single result $Res(\lambda)$; and step 2.(d) (answer) associates every element of X to $Res(\lambda)$. Therefore, the answer $ans_{Q,\delta}$, is itself a constant function.
As an example, the query

$$Q = \langle \varepsilon, n, Sum \rangle \tag{17}$$

will return the total number of items delivered (i.e. the total number of items present in the database).

When the target of the classifier c is a Cartesian product, say $A_1 \times, ..., \times A_k$, then the representation of the answer $ans_{Q,\delta} : range(c) \rightarrow target(op)$ by cross tabulation is usually called a "data cube". Recall that two other representations of the answer are possible, by graph and by binary table, and several tools for "visualizing" and manipulating the answer are available today.

A final but important remark on the definition and evaluation of an OLAP query is in order here. As the classifier and the measure of an OLAP query $Q = \langle c, m, op \rangle$ *both* are path expressions, if we interchange them we obtain a different but valid OLAP query, possibly with a different operation *op*. For example, consider the query $Q = \langle g, n, sum \rangle$, which asks for the total number of items delivered by store. If we interchange g and n and use *count* instead of *sum* then we obtain the query $Q' = \langle n, g, count \rangle$, which asks for the number of stores by number of units delivered (i.e. given a number of items, how many stores had this number delivered to them). Note that, in the query Q', we used "*count*" as an operation, since this is the only operation allowed on the target of the measure g (which is a set of store references).

Mapping to the relational model

The schema model that we presented in the previous section can certainly be used as is to model an application in order to perform data analysis. Moreover, a schema database can be implemented using available open source DBMS technology. For example, MonetDB [?] seems to fit quite well for this purpose as its basic structure is the binary table. However, given that the vast majority of transactional databases and data warehouses today are based on the relational model, it is important to have a way of mapping the functional model to the relational model. In this section we present a method for mapping the schema model to the relational model.
Our method uses three mappings as follows :

- Mapping a schema to a relational star schema
- Mapping a path expression to a relational expression
- Mapping an OLAP query to an SQL query

Mapping a schema to a relational star schema
Given a functional schema S, there are several ways to map it into a relational schema $rel(S)$. The simplest way is to represent each arrow of S by a binary table, and define the set of all binary tables to be the schema $rel(S)$. However, the evaluation of OLAP queries will then require the frequent use of joins. A more efficient mapping is the one that produces a so called *star schema.* To simplify the presentation we shall assume that the functional schema is a tree (as in our running example). Under this assumption, here are the tables and constraints in the corresponding star schema:
Tables :

- Define a table FT containing the root of S and all its immediate successors as its attributes (these immediate successors are the *base attributes*). Call this table the *fact table.*
- For each base attribute B, if there is at least one successor of B, define a table BT containing B and all its descendants as attributes. Call this table the *B-table.*

Constraints :

1. In each of the tables defined above, any arrow connecting two of its attributes becomes a functional dependency of that table (hence it might be that some tables are not in Boyce-Codd Normal Form).

2. There is a foreign key dependency from the fact table to every other table BT: $\pi_B(FT) \subseteq \pi_B(BT)$.

Example :
The star schema for our running example is the following (underlined attributes form the key of each table) :

$FT(\underline{O}, Date, Store, Product, Num)$
$DateT(\underline{Date}, Month)$,
— with $\pi_{Date}(FT) \subseteq \pi_{Date}(DateT)$
$StoreT(\underline{Store}, City, Region)$,
— with $\pi_{Store}(FT) \subseteq \pi_{Store}(StoreT)$
$ProductT(\underline{Product}, Category, Supplier)$,
— with $\pi_{Product}(FT) \subseteq \pi_{Product}(ProductT)$

Note that the table $StoreT$ is not in Boyce-Codd Normal Form, and that the table $DateT$ is not necessary to store (as the month can be determined from the date).

Mapping a path expression to a relational expression
The following algorithm maps a path expression e over S to a relational expression $rel(e)$ over the star schema $rel(S)$:
if the target of e contains only base attributes
then $rel(e) = \pi_{target(e)}(FT)$
else $rel(e) = \pi_{target(e)}(FT \bowtie T_1 \bowtie ... \bowtie T_k)$,
where $T_1, ..., T_k$ are all the tables each of which contains at least one non base attribute appearing either in the target of e, or in the definition of a restriction.

Example:
The path expressions :

$$e_1 = g \wedge (h_2 \circ h) \tag{18}$$

and

$$e_2 = n \tag{19}$$

map to the following relational expressions :

$$rel(e_1) = \pi_{Store,Sup}(FT \bowtie ProductT) \tag{20}$$

and

$$rel(e_2) = \pi_{Num}(FT). \tag{21}$$

Mapping an OLAP query to an SQL query
To map an OLAP query $Q = \langle c, m, op \rangle$ to a relational query $rel(Q)$ it is sufficient to replace the path expressions c and m by $rel(c)$ and $rel(m)$, to obtain

$$rel(Q) = \langle rel(c), rel(m), op \rangle. \tag{22}$$

The query $rel(Q)$ is then evaluated using a "group by" instruction. This is possible if one observes that the "group by" instruction of SQL simply computes inverses of a special kind of functions, namely projections.

Example :

$$Q = \langle g \wedge (h_2 \circ h), n, sum \rangle \tag{23}$$

maps to

$$rel(Q) = \langle \pi_{Store,Sup}(FT \bowtie ProductT), \pi_{Num}(FT), sum \rangle \tag{24}$$

and $rel(Q)$ is evaluated as follows:

select $Store, Sup, sum(Num)$ as $TotNum$
from $join(FT, ProductT)$
group by $(Store, Sup)$

The previous SQL instruction computes the inverse of the projection function $\pi_{Store,Sup}$ thus creating a partition of the table $join(FT, ProductT)$ into sub-tables; then in each sub-table T, it applies the function n to each tuple of the sub-table to find the corresponding number; and finally sums up the numbers found to return the total number for each sub-table.

Result Visualization

Interactive visualization is a powerful technique to slice a data set from various viewpoints using queries and to find interesting trends from visual representations of query results. To perform interactive OLAP visualization, we need to construct many OLAP queries and dynamically map query results to appropriate visual representations. For non-professional end users these tasks are *not* so easy.

In our prototype, we follow a *template-based* approach in the user interface to improve user's experience with interactive analysis tasks. By interacting with visual components, called *visualization templates*, one can easily define both queries and visual representations of query results, simultaneously.

Moreover, our prototype also supports what we call *result exploration* that is the possibility of visualizing the query result at aggregation levels different than those specified in the query; or the possibility of creating a *parametric representation* of the result for better visualization.

Visualization template

Broadly speaking, a visualization template is an interactive component for defining a visual representation of a designated (data) *function*. Figure 6 illustrates a concept of visualization template. Visualization template T_{bar} defines a bar chart representation of a function $f_{bar} : xcoord \rightarrow length$. The function f_{bar} is a variable and we can bind it to a designated function in a data schema or to the answer of a query (which is also a function).

Our prototype implements the following user interaction and automating mechanism to define a function binding. To bind f_{bar}, our prototype requires attribute mappings from attributes in the data schema to the domain $xcoord$ and the range $length$ of f_{bar}. For this purpose, T_{bar} has visible *slots* `[xcoord]` and `[length]` on its surface to map attributes to $xcoord$ and $length$. We can specify attribute mappings just by dragging attributes from the visualized data schema and dropping them into slots. When T_{bar} accepts attribute mappings $\{A \mapsto xcoord, B \mapsto length\}$ with attributes A and B in data schema S, T_{bar} automatically binds f_{bar} to a function retrieved by the following rule.

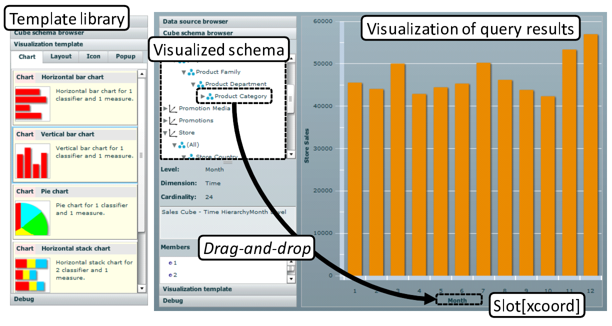

Figure 5: The prototype user interface

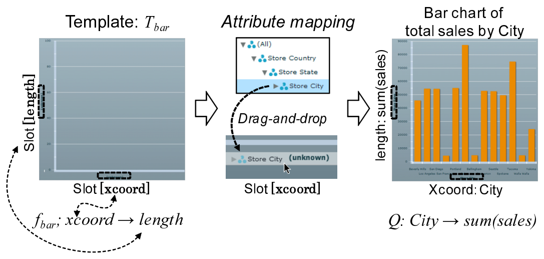

Figure 6: The concept of visual template

- If a schema function (or a derived function) $f : A \to B$ exists in S, T_{bar} binds f_{bar} to f. (If more than one function of the form $A \to B$ exist in S, the system shows a dialog for choosing one of them.)

- If no function of the form of $A \to B$ exists in S, T_{bar} binds f_{bar} to a query $Q\langle p_A, p_B, op\rangle : A \to op(B)$ such that O is the root of S, $p_A : O \to A$ and $p_B : O \to B$ are path expressions over S, and op is an operation preset by users.

Our prototype provides a visualization template library containing a set of basic chart representations (bar chart, pie chart, etc.) and layouts of multiple charts (e.g., grid layout). We can interactively define both, queries and query result representations, by just choosing templates from the library and designing attribute mappings through drag-and-drop manipulation.

Changing the aggregation level of the result

The user interaction and the automating mechanism described above allow us to quickly change aggregation levels of the result. Consider for example the query "Q: Totals by Month". Figure 5 shows the result of this query in the form of a bar chart. The user can explore this result further by asking the system to produce a visualization at levels different than those specified in the query, for example, by Date, by Product ; or by Category; or by City, and so on, just by dragging an attribute for classifier and dropping it into the slot associated with the X axis of the bar chart.

Parametric Visualization

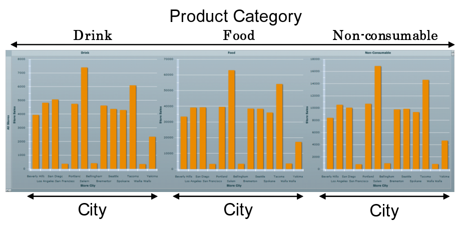

Figure 7: The example of parametric visualization

When visualizing a result involving two or more aggregation levels, it is often convenient to make a series of small plots one for every category item. For example, consider a query whose result involves two dimensions, product category and city. We can make a series of plots showing totals by cities for each product category (as shown in Figure 7). This series of plots might convey easily information difficult to grasp from a single plot showing totals by product category and city. The underlying idea here is that, as the answer to Q has the signature $Category \times City \to Totals$, we can produce an equivalent representation of the answer using the well known "curry operation": $(Category \times City \to Totals) \equiv (Category \to (City \to Totals))$, suggesting that we can use Category as a "parameter" for producing as many small plots $City \to Totals$, as there are product categories. Alternatively, one can use City as a parameter for producing as many small plots $Category \to Totals$ as there are cities. This kind of exploration is basically a repetition of a plot across a grid, where each plot has one variable which changes. In other words, it is a grid of multiple smaller plots driven around in a loop executing it once for every category item. Clearly, parametric visualization is not bound to a specific visualization mode: every basic visualization mode can be parameterized this way, whether it is scatter bar charts, heat maps, line charts, whichever.

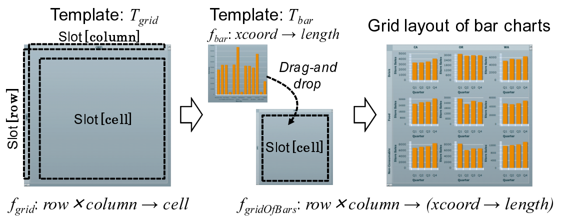
Template: T_{grid}
Slot [column]
Slot [row]
Slot [cell]
Template: T_{bar}
f_{bar}: xcoord → length
Drag-and drop
Slot [cell]
Grid layout of bar charts
f_{grid}: row × column → cell
$f_{gridOfBars}$: row × column → (xcoord → length)

Figure 8: The concept of parametric visualization

Practically, parametric visualization is performed by a combination of a chart template and a grid layout template. Figure 8 shows an outline of grid layout template T_{grid}. T_{grid} defines a grid layout of a visual representation (called *cell*) to represent the function $f_{grid} : row \times column \to cell$. The interesting feature of T_{grid} is that we can map a function instead of an attribute to the range *cell* of f_{grid}. T_{grid} has a special slot `[cell]` that accepts any kind of visualization template. By dropping a template into a slot `[cell]`, we can define a grid layout of the dropped visual representation, and map the function belonging to the dropped template to *cell*. For instance, if we drop T_{bar} with function $f_{bar} : xcoord \rightarrow length$ into `[cell]`, T_{grid} produces a grid layout of bar charts to represent function $f_{gridOfBars} : row \times column \rightarrow (xcoord \rightarrow length)$.

Parametric visualization provides yet another, important dimension of result exploration as the eye can grasp more detailed information when visualizing a set of simple plots, in addition to the information obtained from a single, higher dimensional (thus more complex) plot.

Concluding Remarks

We have seen a functional model for data analysis and an interactive interface (based on that model) that supports users in the following tasks:

- formulating an analytic query visually, in the form of a click stream;
- visualizing the query result in a user selected mode (histogram, pie, etc.);
- exploring the visualized result at different aggregation levels or for different parameter values selected by the user;

The prominent features of our model are that it is simple to grasp and easily amenable to visual interaction.
We are currently investigating the use of our model in an important application area, namely log data analysis in the context of the European project "ASSETS: Advanced Search Services and Enhanced Technological Solutions for the European Digital Library".

The basic idea is to import log data in the form of a relational table with functional dependencies, and construct a functional schema in which the base attributes are those of the table, the analysis indicators (i.e. the non-base attributes) are provided by the analyst, and the edges are generated by the functional dependencies. We note that log data analysis is a useful activity in such areas as e-learning, collaborative work or digital libraries, as it provides valuable support to knowledge discovery from past user activity. Future work will address two main issues:

1. Combining change of aggregation level and parametric visualization. For example, one can visualize totals by product, at different aggregation levels, using different visualization modes for each store.
2. The application of our model to XML data warehouses for analyzing large collections of XML documents.

References

[1] IBM Cognos 8 BI Analysis http://www.ibm.com/software /data/cognos/products/cognos-8-business intelleigence/ analysis.htm

[2] Mondrian. http://mondrian.pentaho.org/.

[3] Oracle BI Discoverer. http://www.oracle.com/technology/ products/discoverer/.

[4] P.A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. The VLDB Journal, 8(2): 101-119, Octorber 1999.

[5] N. Spyratos. A functional model for data analysis. *In Flexible Query Answering Systems, 7th International Conference, FQAS 2006, Proceedings, Milan, Italy*, June 7-10 2006, Proceedings, pages 51-64. Springer, 2006.