Computer Science Journal of Moldova, vol.**14**, no.2(41), 2006

Query by Constraint Propagation in the Concept-Oriented Data Model

Alexandr Savinov

Abstract

The paper describes an approach to query processing in the concept-oriented data model. This approach is based on imposing constraints and specifying the result type. The constraints are then automatically propagated over the model and the result contains all related data items. The simplest constraint propagation strategy consists of two steps: propagating down to the most specific level using de- projection and propagating up to the target concept using projection. A more complex strategy described in the paper may consist of many de-projection/projection steps passing through some intermediate concepts. An advantage of the described query mechanism is that it does not need any join conditions because it uses the structure of the model for propagation. Moreover, this mechanism does not require specifying an access path using dimension names. Thus even rather complex queries can be expressed in simple and natural form because they are expressed by specifying what information is available and what related data we want to get.

1 Introduction

Let us consider the following problem domain and design of its data model. Customers order goods using orders. One order consists of several parts where each part is one product item. This can be represented by three tables `Orders`, `OrderParts` and `Products` as shown in Fig. 1. Each record from `OrderParts` references one order it belongs to and one product that has been ordered in this item (denoted by

arrows). For example, order part `<#23, #16>` references order `<#23>` and product `<#16>` (descriptive fields such as product name or order date are not shown).

One general problem that arises in the context of data modelling consists in getting *related* records given some other records and additional constraints. For example, we might want to get all orders related to one or more products or, vice versa, all products related to a selected set of orders. Notice that related records are not directly connected because there is an intermediate table `OrderParts` between tables `Orders` and `Products`. In the real world data models the structure of relationships is much more complex and normally there exist more than one intermediate table. In this case it is difficult to automatically retrieve related records because the constraints need to be unambiguously propagated over the model according to the assumed semantics of its relationships.

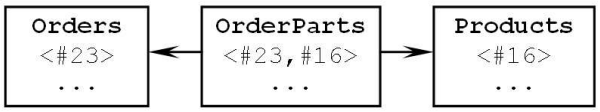

Figure 1. An example of orders and products

A wide spread approach to this problem is based on using the relational data model (RM) [4]. It consists in manually specifying all the involved tables and all their join conditions. Essentially in this case we precisely specify what kind of data we want to retrieve and how this data is related to other data in the model. Notice that all this information is provided within one query. For large data models such a method is not only tedious but also error-prone because writing long queries with complex join conditions requires high expertise. In our almost primitive example it is necessary to join two pairs of tables by choosing among many possible query options. The database cannot help us too much because it is unaware of the model semantics and what actually we want to get and hence all the peculiarities and details of the query have to be specified in an explicit form.

Another problem of such an approach is that many queries will contain the same information. For example, joining the two pairs of tables will be normally done using identical fragments of the query. These fragments will be then repeated in the same or similar form in many other queries. If in future we change a relationship between some tables then all these fragments need to be updated (the problem of code maintenance). Because of this necessity to provide a very detailed manual description for each query intended to retrieve related records, RM is not very successful for solving the problem of *logical* navigation. (Its main achievement is that it successfully solved the problem of *physical* navigation eliminating the need to know and specify physical location of records what was necessary in the preceding data models).

An alternative solution of the problem of logical navigation is provided within the concept-oriented model (CoM) [13, 15]. This model is based on *ordering* its elements, i.e., elements are positioned one under another without cycles rather than compose an arbitrary graph like in entity-relationship model (ERM) [3]. In other words, it is of primary importance if an element is above or below another element while other properties are derived from this ordered structure. In particular, if a table references some other table then it is positioned below it. For example, since table `OrderParts` references `Orders` and `Products`, we put it below the both of these tables as shown in Fig. 2. In addition, we provide names for the arrows connecting tables, which are called dimensions (all dimensions have upward direction). Thus CoM is not only aware of the structure but its direct responsibility consists in managing and using it for querying data, logical navigation and other operations with data.

Using ordered graphs it is possible to get related data by specifying an exact access path from constraints to target elements as dimension names. A direct advantage is that we avoid any join conditions which can be thought of as encoded in the model dimension structure. Such queries are much simpler than SQL queries because they do not involve repeated fragments. For example, a set of orders related to products `P` is obtained as follows:

$\texttt{P} \rightarrow \{\texttt{OrderParts} \rightarrow \texttt{product}\} \rightarrow \texttt{order}$

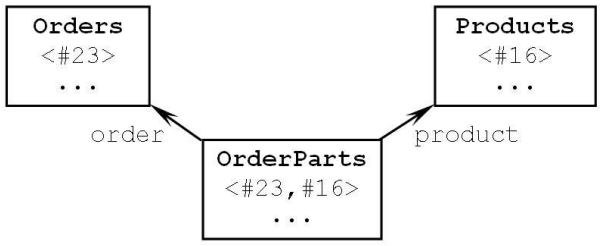

Figure 2. In CoM all elements are ordered

If there is a set of orders `O` then related products are obtained as follows:

$\texttt{O} \rightarrow \{\texttt{OrderParts} \rightarrow \texttt{order}\} \rightarrow \texttt{product}$

In both queries we de-project the initial set to `OrderParts` (denoted by dimension path in curly brackets). And then the result is projected to the target table. For more complex structures such an access path is longer and may consist of many upward (projection) and downward (de-projection) segments in the ordered table structure (zigzags). But in any case the use of dimensions effectively removes the need for explicit joins. It is the responsibility of the database to translate the access path into appropriate operations with records.

It can be noticed that a similar approach is used in other navigational data models such as the network model (NM), object-oriented model (OOM) or functional data model (FDM) [17, 7, 8]. However, the big difference is that CoM uses an *ordered* structure rather than an arbitrary graph and this fact has significant consequences. One of them is that the access path is not simply a sequence of dimensions but rather a sequence of projection and de-projection operations. If we go up in the ordered set then it is projection while if we go down then it is de-projection. These operations can also be effectively used for grouping and aggregation [16] what makes it similar to multidimensional models [1, 9, 11], OLAP [2] and formal concept analysis (FCA) [6].

Above we shortly described how related records can be retrieved in CoM by specifying an access path. However, an amazing property of this model is that in many cases the access path can be built auto-

matically using for that purpose the model structure. In this case we need to only provide a set of initial items and some target from where related records have to be retrieved. In our example this means that it is enough to specify initial products (say, by imposing some constraints on table `Products`) and then say that we want to get related orders. Or, we can specify some initial orders and then say that it is necessary to get related products. In both cases the system will be able to unambiguously build the correct access path and retrieve related records with no additional information. Such ability is again based on using order of elements. Particularly, the system knows that `OrderParts` is a common subtable for both `Orders` and `Products`. Hence it builds the access path as consisting of one de-projection and one projection. This principle can be generalized and applied to more complex data models where it is frequently possible to automatically propagate constraints and get related records expected by the user.

This problem was considered within the universal relation model (URM) where all relations are assumed to be projections of a single relation [10, 5, 12]. However, this direction did not result in an acceptable solution because some things become simpler while others become more complex. One reason is that an assumption of universal relation was shown to be incompatible with many aspects of the relational model. In contrast, CoM possesses a number of unique properties which make the solution not only possible but very natural. One of them is that CoM has canonical semantics (an analogue of universal relation) as its intrinsic feature, which allows us to define natural constraint propagation rules.

The main goal of this paper consists in describing how the mechanism of automatic constraint propagation works in CoM and how it can be used for getting related records. In Section 2 CoM is shortly described. Section 3 describes operations of projection and de-projection and how they are used for constraint propagation. Section 4 describes some more complex cases where it is not possible to find an access path for constraint propagation. Section 5 provides concluding remarks.

2 Model Structure

One of the main principles of the concept-oriented paradigm is that of *duality*, which means that any element has two sides or flavours. For example, in the concept-oriented programming (CoP), concept is a programming construct consisting of one object class *and* one reference class [14]. An object in CoP is then a combination of its fields *and* a collection of internal objects. In CoM the principle of duality can be formulated as follows: any element participates in two separate structures called *physical* and *logical* (Fig. 3). The physical structure has a hierarchical form where any element has a single parent element (dotted lines in Fig. 3). It is called physical because elements are assumed to be included by value in their parents and the element position in the parent element is thought of as its (local) identifier or reference. Thus the physical structure is responsible for object representation and access (ORA) by providing permanent references and some access procedure. For example, all product records are created within a table where they have some unique and permanent references. The table itself is included in its database.

In CoM any element, such as product, may have its internal (physical) elements however in this paper we consider only a *two-level model* which consists of one *root*, a set of *concepts* included in the root and a set of data *items* included in the concepts. (Further in the paper we will use these terms instead of their analogues like tables and records.) In Fig. 3 the physical structure is drawn along the horizontal axis and consists of one root, which includes three concepts `OrderParts`, `Orders` and `Products`, which in turn consist of some data items.

The logical structure has a form of directed acyclic graph where an element may have many parents by storing their references in its fields. In other words a reference stored in a field is interpreted as a pointer to one of the parent elements. This allows us to bring order into the logical structure by placing an attribute value above the element it characterizes. For example, order part item `<#23, #16>` references order item `<#23>` and product item `<#16>`. Hence it is positioned below both of them. Here we do not need even to know that these items represent an

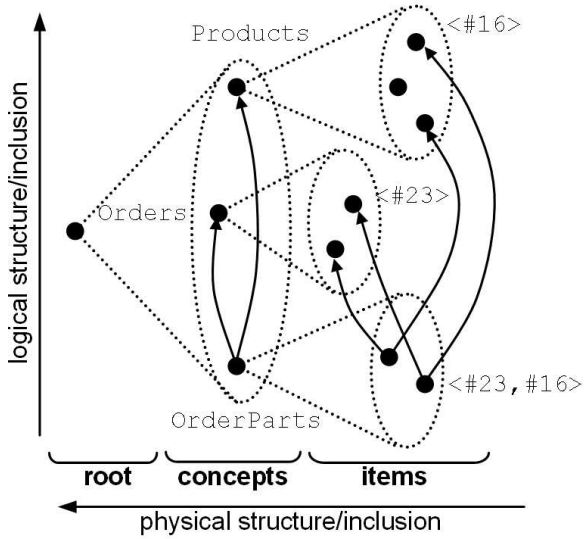

Figure 3. Physical and logical structures

order part, order and product – we simply interpret references (arrows in Fig. 3) as the logical membership relation leading from a member to its group. Such an interpretation of references (and attribute-value relation) as a membership is a very important characteristic property of CoM which allows us to consider a value as a logical set or collection for the objects it characterizes (an item is a logical collection for its subitems).

Concept S referenced by concept C is referred to as *superconcept* (and C is then referred to as *subconcept*). For example, concept `OrderParts` has two superconcepts `Orders` and `Products` positioned above their common subconcept. A named link from a subconcept to a superconcept is referred to as *dimension*. For example, concept `OrderParts` in Fig. 2 has two dimensions `order` and `product`. If a concept does not have a subconcept explicitly defined then formally introduced *bottom* concept is used for that. Analogously, if a concept does not have a superconcept then formally introduced *top* concept is used for that. Top concept is the most abstract (general) concept

and bottom concept is the most specific one. All other concepts in the model are positioned between top and bottom concepts. Direct subconcepts of top concept are referred to as *primitive* concepts.

In the logical structure any element references a number of parent elements. Theoretically an element can reference any other element provided that this structure does not have cycles. (Cycles do not allow us to define order because it is not possible to determine if an element is above or below another element from this cycle.) In particular, concepts can reference items, items can reference concepts and any item can reference any other item from the model. However, such a freedom is not desirable and CoM introduces so called *syntactic constraints*, which mean that an item can reference only items from concepts referenced by its parent concept. For example, an order part item cannot reference any other item in the model under syntactic constraints. Its domains are restricted by items from `Orders` and `Products` because they are referenced by concept `OrderParts`.

There exist two operations that can be used to get related items given some other items: projection and de-projection [16]. *Projection* is applied to items from a subconcept and returns items from some its superconcept along a dimension path. For example, `OrderParts` is a subconcept of `Products` and hence each order part item can be projected on some product. Given a subset of order parts `OP` it is possible to get all related products by projecting this subset into its superconcept `Products`:

$\texttt{OP} \rightarrow \texttt{product}$

De-projection is an opposite operation. It is applied to items from a superconcept and returns related items from some its subconcept. A path in the case is specified using dimension names which however are interpreted in the opposite direction. For example, given a set of products `P` we can get all related order parts by de-projecting them into concept `OrderParts`:

$\texttt{P} \rightarrow \{\texttt{OrderParts} \rightarrow \texttt{product}\}$

Notice that here we use *inversion operator* $\{\ \}$ in order to change the direction of dimension path. Thus projection can be viewed as moving up in the concept graph along a dimension path while de-projection is

viewed as moving down along some *inverse dimension*. A sequence of projection and de-projection operators is referred to as an *access path*.

3 Constraint Propagation

In the previous section we described a procedure for retrieving related items using the mechanism of access path. This method assumes that some initial set of items is selected and then operations of projection and de-projection are applied. This method is simpler and more natural than the existing approaches but it still requires a complete specification of the access path.

Let us now look at this process differently. The set of initial items has to be somehow specified and for this purpose some constraints are normally used. These initial constraints are imposed on items of some concept and are somehow propagated over the model changing its semantics. Finally we want to get some part of this modified (constrained) model by specifying the target concept. Due to constraint propagation this target concept will include only items which are related to the source items. An advantage of this approach is that we do not need to specify an access path because related items are retrieved as a result of constraint propagation procedure carried out automatically. For example, to get products related to some orders it is enough to specify constraints for the source orders and then the target concept – the constraint propagation path can be computed automatically.

The main question in this method is *how* initial constraints have to be propagated over the model in order to get natural and meaningful results. The simplest strategy consists of two operations:

1. Initial constraints imposed on concepts $X_1, \ldots, X_n$ are propagated down to and imposed on the most specific concept Z using de-projection.

2. The constrained semantics of concept Z is propagated up to the target concept Y using projection.

Constraints are specified using some predicate that has to be true for each selected data item:

$\bar{X}_i = \{x_i \in X_i | f_i(x_i) = \text{true}\} \subseteq X_i,\ i = 1, \ldots, n$

Here $\bar{X}_i$ is a subset of items from X_i satisfying condition f_i (bar stands for constrained).

Let us now consider how these constraints are propagated down to the subconcept Z. The idea is that if some item from X_i does not satisfy the imposed condition f_i then all its subitems from the target subconcept Z also do not satisfy this condition. In the case of many constraints the result is intersection of individual de-projections:

$\bar{Z} = \bar{Z}_1 \cap \ldots \cap \bar{Z}_n,$

$\bar{Z}_i = \bar{X}_i \to \{d_i\},\ i = 1, \ldots, n$

Here d_i is a dimension from concept Z to its superconcept X_i. (In the case of many dimensions all of them are used independently.) Equivalently, subset $\bar{Z}$ can be defined as a multidimensional de-projection [16] of items $\langle x_1, \ldots, x_n \rangle \in \bar{X}_1 \times \ldots \times \bar{X}_n$ to Z along n dimensions $d_1, \ldots, d_n$:

$\bar{Z} = \langle x_1, \ldots, x_n \rangle \to \{d_1, \ldots, d_n\}$

Thus an item from Z satisfies all the constraints $f_1, \ldots, f_n$ imposed on concepts $X_1, \ldots, X_n$ if it is projected in $\bar{X}_1, \ldots, \bar{X}_n$ along the chosen dimension paths $d_1, \ldots, d_n$:

$\bar{Z} = \{z \in Z \,|\, z \to d_i \in \bar{X}_i, i = 1, \ldots, n\} \subseteq Z$

Subset $\bar{Z}$ contains the semantics satisfying all the imposed constrains. However, we do not need it in such a detailed form. Instead, we want to get related items from concept Y. In order to decrease the level of details the data from $\bar{Z}$ has to be projected to the target concept Y:

$\bar{Y} = \bar{Z} \to m$

Here m is a dimension from Z to Y. Selected items $\bar{Y} \subseteq Y$ are related to items $\bar{X}_1, \ldots, \bar{X}_n$ chosen in $X_1, \ldots, X_n$ and are returned as a result of the constraint propagation procedure.

An example of this procedure is shown in Fig. 4. Constraints are imposed on concept `Products` by selecting a number of product items. In order to get related orders from `Orders` these products have to be propagated down to `ProductParts` using de-projection. All the selected order parts (shown in dashed line in Fig. 4) reference only the specified products. On the second step these order parts are projected

on concept `Orders`.

This procedure is analogous to inference in multi-dimensional space (Fig. 4 right). Constraints are imposed by selecting a subset of values along axis X. The available dependencies are represented as a subset of points from space Z (in circle). Intersection $\bar{Z} = \bar{X} \cap Z$ is then projected onto target axis Y and gives result $\bar{Y}$. Dependence Z can be represented using different techniques such as linear equations, differential equations, neural networks and so on. In the world of databases it is represented as a set of data items which select points from the space. In other words, a subconcept with its data items encodes a dependence between its superconcepts.

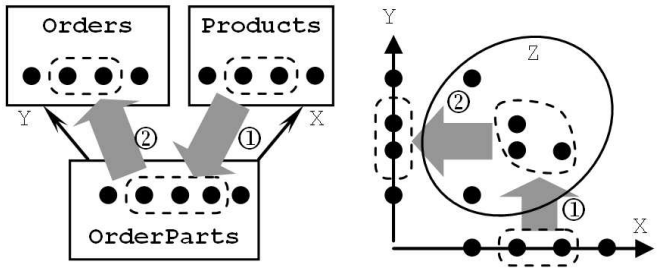

Figure 4. Constraint propagation procedure

Notice that related orders can be always obtained by manually specifying an appropriate access path. However, a property of the described procedure is that it does not require any access path and propagates constraints automatically. We need only to impose constraints and then indicate a target concept. The query for getting related orders can be written as follows:

```
SELECT * FROM Orders
WHERE Products.size > 10
```

This query will return records from table `Orders`. However its `WHERE` clause imposes additional restrictions on table `Products` by selecting only those with big size. Notice that there is no indication in this query how these two tables are connected and therefore the imposed restriction will be propagated automatically.

Fig. 5 provides an example of the described procedure. Assume that the question is what orders are related to beer and chips. This means that it is necessary to find all orders where either beer or chips are product items. The selected two products are shown in bold in Fig. 5. These two products are de-projected to concept `OrderParts` which will contain only three items. Then these three items are projected to concept `Orders`. They reference only two orders `<# 23>` and `<# 24>` which are the result of this query.

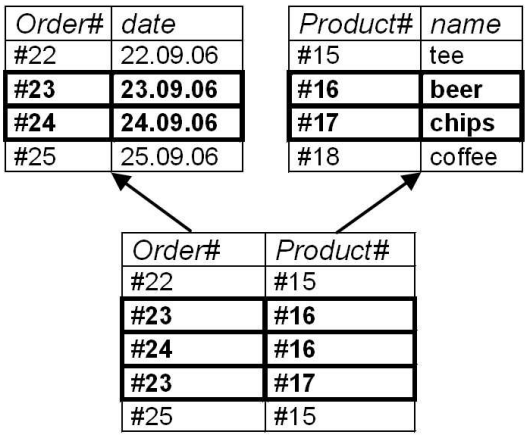

Figure 5. An example of constraints

In the case of many constraints in different parts of the concept graph they are propagated down to the most specific concept along their individual dimension paths. On the second step their intersection is propagated up to the target concept. A model in Fig. 6 consists of three already described concepts `Orders`, `Products` and `OrderParts`. However, concepts `Orders` and `Products` have their own superconcepts. In particular, each order is characterized by a customer (who made this order) and a date (when this order was made). Each customer belongs to some country from concept `Countries` and each product has a category from concept `Categories`. Let us now assume that we want to get all countries related to some product category (say,

'cars') and during some period of time (say, in 'June'). In other words, we want to learn in what countries cars were sold in June. Here again it is possible to write this query using concrete access paths for projection and de-projection. However, the method described in this paper does not need it and it is enough to simply impose our constraints and indicate what kind of result we want to get – all the rest will be done automatically.

The first constraint consists in selecting only cars. As a consequence all non-car items are effectively removed from the model. This means that all non-car products are removed and all order parts with non-car products are also removed. Thus we get some subset of all available order parts. The second constraint consists in selecting only items characterized by June as their date (concept Months). When this constraint is propagated down, all non-June items from its subconcepts are effectively removed. In particular, all non-June dates, all non-June orders and all non-June order parts do not satisfy this constraint. After that concept OrderParts will contain only items characterized by cars as its product category and by June as its date.

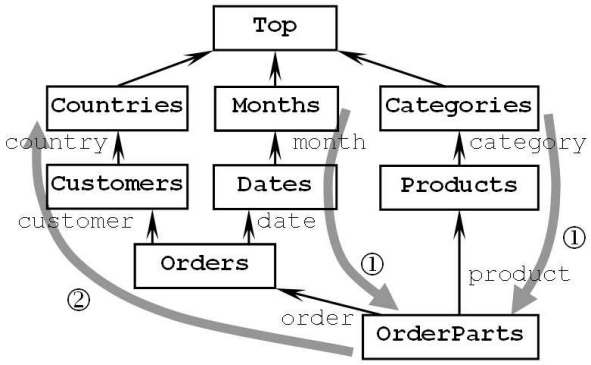

Figure 6. Many source concepts with constraints

The last step in this procedure consists in propagating the selected order parts up to the target concept Countries. The whole query can be written as follows:

```
SELECT * FROM Countries
WHERE Categories.name=='cars'
AND Months.name=='June'
```

This query selects records from one table while imposing restrictions on other tables which are not explicitly connected to the first one. In SQL such a query would need to join 8 pairs of tables.

The described procedure for automatic constraint propagation works only if an access path can be unambiguously restored. However there are models where many possible constraint propagation paths exist between source and target concepts. For example, products can be characterized by a country of origin (Fig. 7) and then concept `Countries` is a domain for two subconcepts `Customers` and `Products`. Let us assume that we want to get all related product categories for a selected country. Obviously, in this case there exist two options for constraint propagation from `Countries` to `Categories`. The first path goes through concept `OrderParts` and this propagation strategy will result in all product categories *ordered* by customers from the specified country (path ①). The second path goes through concept `Products` and it will return all categories for products *made* in the specified country (path ②).

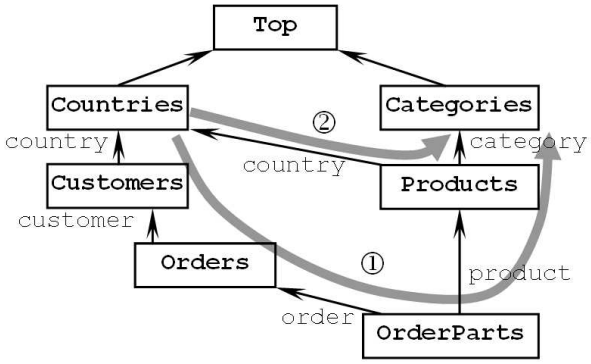

Figure 7. Multiple constraint propagation paths

In order to resolve this ambiguity and avoid the necessity to spec-

ify the complete path it is possible to use hints which help to choose one of many alternatives. One approach consists in using `'VIA'` keyword followed by concept name. This keyword means that constraint propagation path has to include the specified concept(s), i.e., the reconstructed access path has to pass through this concept(s). For example, the following query will return all categories for products made in Germany:

```
SELECT * FROM Categories
WHERE Country.name=='Germany'
VIA Products
```

However, we can change this query and get all categories for orders from Germany:

```
SELECT * FROM Categories
WHERE Country.name=='Germany'
VIA OrderParts
```

By default in the case of no additional information the imposed constraints will be propagated along all possible paths. In the above example this would produce a set of categories for products ordered *and* made in Germany.

4 Implicit Constraints

In the previous section we considered an automatic constraint propagation procedure consisting of two steps: propagating down via deprojection and propagating up via projection. In this section we consider a more complex case where this procedure does not work because the source constraints do not directly influence the target concept. Let us consider an example shown in Fig. 8. Here coaches (concept `Coaches`) train teams (concept `Teams`) while one team consists of a number of players (concept `Players`). Concepts `Trains` and `Plays` store pairs of coach-team and player-team, respectively. (It is assumed that a player may play for many teams and a coach may train many teams.) Now let us ask the following question: find players related to a selected coach. For this model this question means that we want to get all players who have ever played for a team trained by this coach.

If we select the coach and propagate this constraint down according to the 2-step procedure then we get a subset of items from concept `Trains`. However, here this procedure stops because there is no path leading to the target concept `Players`. Indeed, we can project `Trains` to its superconcept `Teams` but not to `Players` because players are not directly connected with coaches.

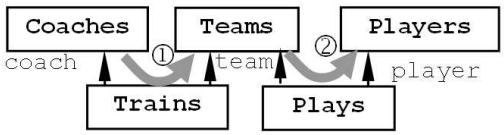

Figure 8. Multi-step inference

One simple solution to this problem consists in performing multiple de-projection/projection zigzags in concept graph. In this example, an obvious strategy is (1) to find a set of teams trained by the coach and then (2) all players from these teams. Here again we can use keyword `'VIA'` in order to specify more precisely a point in the concept graph used for constraint propagation:

```
SELECT * FROM Players
WHERE Coaches.name=='Smith'
VIA Teams
```

Another solution consists in applying the same 2-step procedure as described in the previous section but adding additional constraints which are implicitly assumed in the initial query. For example, when we want to get all players trained by some coach then it is implicitly assumed that the coach trains a team of this player. In other words, the team trained by the coach must be the same team where the player plays:

```
Trains.team == Plays.team
```

If this assumption can be explicitly formulated in the query then we can correctly propagate the initial constraints directly from coaches to players.

The concept-oriented model always has bottom concept which is a subconcept for any other concept. If it does not exist then it is added

formally. In particular, the model in Fig. 8 does not have bottom concept so we formally add it and get the model shown in Fig. 9. The semantics of bottom concept is the canonical semantics of the whole model. It is equal to the Cartesian product of all its direct superconcepts: $B = S_1 \times \ldots \times S_n$. In our example it contains all combinations of items from concepts `Trains` and `Plays`. Effectively this means that there are no dependencies between these concepts and hence we will not be able to derive any consequence in one concept given constraints in another concept.

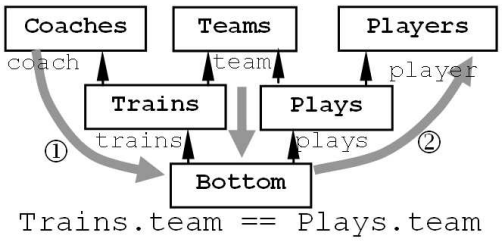

Figure 9. Inference under additional constraints

Bottom concept has a number of dimensions with domains in all its superconcepts. In our example it has 2 dimensions with domains in direct superconcepts `Trains` and `Plays` and 4 dimensions of rank 2 with domains in concepts `Coaches`, `Teams` and `Players` (so it is a 4-dimensional model). Notice however that two dimensions `Bottom.trains.team` and `Bottom.plays.team` have the same domain in concept `Teams`. We know that a coach is related to a player only if he trains the same team where this player plays. Thus we need to consider only data items from `Bottom` which satisfy this condition

$$\texttt{Bottom} \rightarrow \texttt{trains} \rightarrow \texttt{team} == \texttt{Bottom} \rightarrow \texttt{plays} \rightarrow \texttt{team}$$

It is precisely the additional constraint that has to be taken into account when carrying out inference procedure. (The first constraint is as usual a coach for which we want to get related players.) Both types of constraints (explicit and implicit) are propagated down to bottom concept and then the result is propagated up to the target concept. The whole query can be written as follows:

```
SELECT * FROM Players
WHERE Coaches.name == 'Smith'
AND Trains → team == Plays → team
```

The first line selects records from the target table. The second line imposes explicit restrictions by choosing one coach. And the third line specifies an additional implicit condition according to our understanding of the word *related* (records). In other situations this additional constraint could express some other semantics of the term related items. For example, a player might be treated as related to a coach if he has played for the team for more than some time or more than some number of games. These constraints encode dependencies in the model which are implicitly assumed when making a query.

5 Conclusions

In the paper we presented an approach to getting related data using automatic constraint propagation. This mechanism is based on specific properties of the concept-oriented data model. In particular, it assumes that elements of the model are ordered and this order is used to implement operations of projection and de-projection. This method is easy to use because in many situations it is enough to impose constraints and indicate the type of result we want to obtain. However, it makes it possible to express rather complex queries using additional hints in the case of ambiguity or additional constraints in the case of the absence of direct dependencies.

References

[1] R.Agrawal, A.Gupta, and S.Sarawagi. *Modeling multidimensional databases.* In 13th International Conference on Data Engineering (ICDE'97), 1997, pp. 232–243.

[2] A.Berson, and S.J.Smith. *Data warehousing, data mining, and OLAP.* New York, McGraw-Hill, 1997.

[3] P.Chen. *The Entity-Relationship Model. Toward a Unified View of Data.* In: ACM Transactions on Database Systems 1/1, 1976.

[4] E.F.Codd. *A relational model of data for large shared data banks.* Communications of the ACM, 13(6), 1970, pp. 377–387.

[5] R.Fagin, A.O.Mendelzon, J.D.Ullman . *A Simplified Universal Relation Assumption and Its Properties.* ACM Trans. Database Syst., 7(3), 1982, pp. 343–360.

[6] B.Ganter, and R.Wille. *Formal Concept Analysis: Mathematical Foundations.* Springer, 1999.

[7] P.M.D.Gray, P.J.H.King, and L.Kerschberg. (eds.) *Functional Approach to Intelligent Information Systems.* J. of Intelligent Information Systems, 12, 1999, pp. 107–111.

[8] P.M.D.Gray, L.Kerschberg, P.King, and A.Poulovassilis. (eds.) *The Functional Approach to Data Management: Modeling, Analyzing, and Integrating Heterogeneous Data.* Heidelberg, Germany, Springer, 2004.

[9] M.Gyssens, and L.V.S.Lakshmanan. *A foundation for multi-dimensional databases,* In VLDB'97, 1997, pp. 106–115.

[10] W.Kent. *Consequences of assuming a universal relation,* ACM Trans. Database Syst., 6(4), 1981, pp. 539–556.

[11] C.Li, and X.S.Wang. *A data model for supporting on-line analytical processing,* Proc. Conference on Information and Knowledge Management, Baltimore, MD, 1996, pp. 81–88.

[12] D.Maier, J.D.Ullman, and M.Y.Vardi. *On the foundation of the universal relation model.* ACM Trans. on Database System (TODS), 9(2), 1984, pp. 283–308.

[13] A.Savinov. *Hierarchical Multidimensional Modelling in the Concept-Oriented Data Model,* 3rd Intl. Conference on Concept

Lattices and Their Applications (CLA'05), Olomouc, Czech Republic, 2005, pp. 123–134.

[14] A.Savinov. *Concept as a Generalization of Class and Principles of the Concept-Oriented Programming,* Computer Science Journal of Moldova, 13(3), 2005, pp. 292–335.

[15] A.Savinov. *Logical Navigation in the Concept- Oriented Data Model,* Journal of Conceptual Modeling, Issue 36, 2005, August, http://www.inconcept.com/jcm.

[16] A.Savinov. *Grouping and Aggregation in the Concept-Oriented Data Model.* In 21st Annual ACM Symposium on Applied Computing (SAC'06), Dijon, France, 2006, pp. 482–486.

[17] D.W.Shipman. *The Functional Data Model and the Data Language DAPLEX.* ACM Transactions on Database Systems, 6(1), 1981, pp. 140–173.

Dr. Alexandr Savinov,
Institute of Mathematics and Informatics,
Academy of Sciences of Moldova
str. Academiei 5,
MD-2028 Chisinau, Moldova
E–mail: *savinov@conceptoriented.com*
Home page: *http : //conceptoriented.com/savinov*

Received August 24, 2006