INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians–
Universität
München

Grouping Constructs for Semistructured Data

François Bry, Dan Olteanu, Sebastian Schaffert

Tehnical Report, Computer Science Institute, Munich, Germany
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-2001-7, May 2001

Towards Grouping Constructs for Semistructured Data

François Bry, Dan Olteanu, Sebastian Schaffert[*]
Institute for Computer Science, University of Munich
Oettingenstrasse 67, D-80538 München, Germany
`http://www.pms.informatik.uni-muenchen.de`

Abstract

Markup languages for semistructured data like XML are of growing importance as means for data exchange and storage. In this paper we propose an enhancement for the semistructured data model that allows to express more semantics. A data model is proposed and the implications on pattern matching are investigated.

1. Introduction

Languages for semistructured data (SSD) like XML have by now gained widespread acceptance as a data exchange format. Also growing is the importance of the SSD data model for database management. Query languages like XQuery [21] and its predecessors QUILT [7] and XQL [13] are visible signs of this development.

In this paper, we suggest an enhancement called *grouping constructs* to the SSD data model. This enhancement allows to establish explicit semantic relationships between data items in semistructured databases. Usually these relationships are given either implicitly through the meaning of element names or are implemented in the application software processing the data. A declarative localization language (e.g. XPath [18]) that not only copes with grouping constructs but also uses them for a more efficient localization and thus querying is further described.

This paper is organized as follows: Section 2 shows on an introductory example the deficiencies of ordinary SSD data models and how they can be overcome with grouping constructs. Section 3 introduces the grouping facets which make up the grouping constructs. Section 4 gives an overview of a formalism for grouping constructs. Sections 5 and 6 will deal with matching and its results in a grouping context. Finally, current investigations on the subjects, plans and visions for the future as well as related work are briefly presented.

[*]Contact: `schaffer@informatik.uni-muenchen.de`

2. Motivation

Consider the following example of a curriculum at the University of Munich:

In the first 4 terms, some courses are optional while others are required. Thus there are **and** and **or** connections between courses.

Terms	Courses		
	Comp. Sc.	Mathematics	Projects
1	CS I	Algebra I **and** Analysis I	
2	CS II **and** Hardware Basics	Algebra II	
3	CS III	Graph Theory **and** App. Analysis	Programming **or** Systems **or** Hardware **or** Logics
4	CS IV **and** Advanced Algorithms	Stochastics **or** Numerical Mathematics	

While this table reminds of a standard database item like e.g. a (non-first normal form) relation, the **and** and **or** connections show a very different semantics. Obviously, a data model is needed with which grouping constructs like the **and** and **or** connectives can be expressed and used during localization and data retrieval in general. Note that object data not meta-data are grouped. In the following we will see that other grouping constructs are also desirable.

Let us first consider a similar example at a more abstract level:

Imagine that you want to store data the semantics of which is a **and** $(b$ **or** $c)$. In a relational database model, this would be achieved by transforming this to its conjunctive normal form $(a$ **and** $b)$ **or** $(a$ **and** $c)$ and then storing each of the conjuncts (a, b) and (a, c) in a relation R (informally, the **and** is between the columns, or attributes, while the **or** is between the rows, or tuples, of the table). This would result in the relation $R = \{(a, b), (a, c)\}$.

This representation has two drawbacks:

- *redundancy*: The information about a is stored several times. This is inefficient in both, space and computation, and error prone for updates.
- *information loss*: The fact that b and c are **and** connected to the common item a might be important from a semantics viewpoint. This information has to be recomputed (i.e. the conversion to the conjunctive normal form has to be reversed).

With the SSD data model things are even worse: While the relational model at least has the connections **and** and **or** built-in (i.e. the **and** connections between the columns and the **or** connections between the rows), in the SSD data model it is not possible to express such information in an application independent manner.

As semi-structured data, the previous course of studies example could be expressed as follows (an XML syntax is used here to ease the understanding for the reader; any other SSD syntax would be possible).

```
<course_of_studies>
  ...
  <term>
    <number>4</number>
    <computer_sciences>
      <course>CS IV</course>
      <course>
        Advanced Algorithms
      </course>
    </computer_sciences>
    <mathematics>
      <course>Stochastic</course>
      <course>
        Numerical Mathematics
      </course>
    </mathematics>
    <seminars>
      <course>
        Programming Course
      </course>
      <course>
        System Course
      </course>
      ...
    </seminars>
  </term>
</course_of_studies>
```

In this excerpt, some information is missing: It is not expressed which courses are optional and which are required. A common solution would be to provide this information in an application dependent query interface.

However, this approach would not be portable since every application would have its own data format. This would be unfortunate because it is the idea of SSD to be application *in*dependent. Note that such semantic groupings occur frequently in data exchange (e.g. in e-commerce catalogs, bioinformatics databases [9],etc.).

Our proposal is to add general constructs to the SSD data model so as to allow the grouping of elements according to certain properties ("grouping facets"), thus trying to overcome the above mentioned deficiencies of the relational and the standard semi-structured model.

With grouping facets the introductory example can be represented as follows. Again, the XML syntax has been retained. Also note that grouping can be expressed through other constructs than through elements.

```
<course_of_studies>
  ...
  <term>
    <number>4</number>
    <computer_sciences>
      <AND>
        <course>CS IV</course>
        <course>
          Advanced Algorithms
        </course>
      </AND>
    </computer_sciences>
    <mathematics>
      <OR>
        <course>Stochastic</course>
        <course>
          Numerical Mathematics
        </course>
      </OR>
    </mathematics>
    <seminars>
      <OR>
        <course>
          Programming Course
        </course>
        <course>
          System Course
        </course>
        ...
      </OR>
    </seminars>
  </term>
</course_of_studies>
```

In the next section, grouping is investigated systematically and other grouping facets than **and** and **or** are suggested.

After that, a matching technique for localization in a context with grouping constructs is discussed.

3. Grouping Facets

Since our extension *groups* data items and adds additional information to the already-existing structure, it is called **grouping constructs**. The individual kind of grouping is called **grouping facets**. The following grouping facets are suggested:

- *connector*: for grouping items with the connectors AND, OR and XOR (the connector facet has one of the properties “AND”, “OR” and “XOR”).
- *order*: for specifying whether items are ordered or not (properties “ordered”, “unordered”)
- *repetition*: for specifying whether items of the same type may be repeated or not (properties “repetition allowed” and “repetition not allowed”).
- *selection*: for allowing a query to select/match a certain number of the items (property “n to m”)
- *exclusion*: for excluding certain items (property “excluded”)
- *depth*: for allowing a pattern to span several levels in a matched tree (property “n to m”).[1]

From the facets presented above, the first 4 are sibling relationships and the last 2 are parent-child relationships.

Grouping facets can be of importance in three areas: database modeling, query patterns/schemas and answers to a query.

Not all of the mentioned grouping facets fit equally well to databases and to patterns/schemas. While e.g. the connector facet may be of relevance in both databases and patterns/schemas, the exclusion facet makes sense for patterns/schemas only.

In this paper we deliberately impose the following restrictions on grouping facets:

- only one grouping facet can be specified for a group of nodes
- the specified grouping facet always applies to all immediate children
- the data model is currently limited to trees[2]

The rationale for these restrictions is the focus on the novel issue. An extension is possible (and desirable) in the future.

[1]Allowing infinity as value for m allows to express the classical quantifiers “*”, “+” and “?” as “n to m” facets

[2]Extensions to DAGs and forests do not pose principal problems

Ontologies([11],[5],[15]) have constructs similar to these grouping constructs, however they use them primarily for structuring meta-data.

Also schemas for XML (XML-Data [16], DDML [17], XML-Schema [20] etc.), have constructs similar to some of the above mentioned grouping constructs, but not all of them. Note that these constructs are only used for grouping in a schema, not in the data. Using grouping constructs in queries and answers is not considered in these specifications.

XML-Schema grouping constructs are less expressive than the present proposal. With XML-Schema one can represent only the *connector, order, repetition* and *exclusion* facets. The **AND connector** and **ordering** can be represented in XML-Schema as a *sequence* of children declaration for the content model of the parent. The **XOR connector** reminds of the *choice* group element or of the *enumeration* facet from XML-Schema. The **OR connector** cannot be completely modeled using XML-Schema constructs (part of it can be achieved uing a combination between the *choice* group element and the *minOccurs* and *maxOccurs* facets). The **unordering** can be represented in XML-Schema by the *all* group element. The **repetition** and **exclusion** facets are covered by the *minOccurs* and *maxOccurs* facets.

4. Data Model: Trees with Grouping Facets

In this section, a data model of data trees with grouping facets is introduced.

Let $\mathbf{Nodes}$ denote a set of nodes, $\mathbf{Edges} \subseteq \mathbf{Nodes} \times \mathbf{Nodes}$ a set of edges and $\mathbf{Labels}$ a set of node labels. Furthermore, let $\mathbf{Power}(X)$ denote the set of all (finite) repetition free lists with elements from X.

4.1. Data Trees (DTs)

The semistructured data model considered is based upon node-labeled trees, hence (slightly) different from other approaches such as UnQL [2] and OEM [4] or ACeDB [6].

A tree T = ($\mathbf{Nodes}, \mathbf{Edges}$) is a rooted DAG (directed acyclic graph), where every node $n \in \mathbf{Nodes}$ there is a unique path from the root **root** to n.

Definition 4.1 (elementary data tree)
*An **elementary data tree** DT, with set of nodes* $\mathbf{Nodes}$*, set of edges* $\mathbf{Edges}$ *and root **root**, is represented by the tuple (*$\mathbf{Nodes}$*, name, children, **root**), where:*

- $name$: $\mathbf{Nodes} \rightarrow \mathbf{Labels}$ *is a function mapping each node to its label*
- $children$: $\mathbf{Nodes} \rightarrow \mathbf{Lists}(\mathbf{Nodes})$ *is a function mapping each node* $n \in \mathbf{Nodes}$ *to its children (thus children are ordered)*

Our model considers by default an elementary data tree as unordered tree. The ordering can be explicitly stated using the *order* facet.

Trees with ordered children will be written as $A(B_1, \ldots, B_n)$, meaning a tree with root node A and subtrees $B_1, \ldots, B_n$ in the given order. Trees with unordered children will be written $A\{B_1, \ldots, B_n\}$ denoting the same tree, but with the subtrees $B_1, \ldots, B_n$ in any order.

Elementary data trees are extended to **enriched data trees** by adding grouping facets as follows:

Definition 4.2 (data tree with grouping facets)
Given a set $\mathbf{G}$ *of grouping facets as defined in Section 3, a data tree with grouping facets is defined as a tuple (*$\mathbf{Nodes}$*,name,children,****root****,grouping), where:*

- *(*$\mathbf{Nodes}$*, name, children,* ***root****) is an elementary data tree*
- $grouping : \mathbf{Nodes} \rightarrow \mathbf{Power}(\mathbf{G})$ *is a function mapping each node to a set of corresponding grouping facets.*

Note that Definition 4.2 allows grouping facets for all the children of a node, following an assumption made at the end of Section 3.

The meaning of a data tree with grouping can be expressed as a set of elementary data trees. For example, the data tree with OR-grouping expresses the set of elementary data trees consisting of all combinations between children (see Definition 4.3 below). By contrast, a data tree with AND-grouping expresses the set of elementary data trees represented only by the elementary data tree with all children. A possible comparison between data trees with grouping facets is addressed in Section 6.

4.2. Semantics of DTs with grouping facets

The semantics of a data tree with grouping facets is defined in terms of elementary data trees (without grouping). Thus, data trees with grouping facets can be seen as “factorization” of several elementary data trees.

Definition 4.3 (Interpretation of grouping facets)
Let DT be a data tree with grouping facets. A given node $N \in \mathbf{Nodes}(DT)$ *with a grouping facet* $\mathcal{G} \in grouping(N)$ *and children* $T_1, \ldots, T_n$ *is interpreted as its correspondent forest of data trees* $\mathbf{I}(N_\mathcal{G})$ *with root node* N *and without* $\mathcal{G}$ *as defined in table 1.*

$\mathbf{I}$ *applied recursively to all nodes from the data tree* DT *beginning with the root node generates a forest of elementary data trees. This forest is called the* interpretation *of* DT*, written* $\mathbf{I}(DT)$*.*

enriched subtree $N_\mathcal{G}$	interpreted as
$\mathbf{I}(N())$	{ N() }
$\mathbf{I}(N(T_1, \ldots, T_n))$	$\{N(T'_1, \ldots, T'_n) \mid T'_i \in \mathbf{I}(T_i), 1 \leq i \leq n\}$
$\mathbf{I}(N\{\})$	$\mathbf{I}(N())$
$\mathbf{I}(N\{T_1, \ldots, T_n\})$	$\bigcup\{\mathbf{I}(N(T_{\pi(1)}, \ldots, T_{\pi(n)})) \mid \pi\, permutation\, of\ \{1, \ldots, n\}\}$
$\mathbf{I}(N_\epsilon())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_\epsilon(T_1, \ldots, T_n))$	$\mathbf{I}(N\{T_1, \ldots, T_n\})$
$\mathbf{I}(N_{AND}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{AND}(T_1, \ldots, T_n))$	$\mathbf{I}(N\{T_1, \ldots, T_n\})$
$\mathbf{I}(N_{OR}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{OR}(T_1, \ldots, T_n))$	$\bigcup\{\mathbf{I}(N\{P_1, \ldots, P_k\}) \mid \{P_1, \ldots, P_k\} \subseteq \{T_1, \ldots, T_n\}, 1 \leq k \leq n\}$
$\mathbf{I}(N_{ord.}())$	$\mathbf{I}(N())$
$\mathbf{I}(N_{ord.}(T_1, \ldots, T_n))$	$\mathbf{I}(N(T_1, \ldots, T_n))$
$\mathbf{I}(N_{unord.}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{unord.}(T_1, \ldots, T_n))$	$\mathbf{I}(N\{T_1, \ldots, T_n\})$
$\mathbf{I}(N_{repeat}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{repeat}(T_1, \ldots, T_n))$	$\bigcup\{\mathbf{I}(N\{T'_1 \circ \ldots \circ T'_n\}) \mid T'_i = (T_i, \ldots, T_i),\ \lvert T'_i \rvert = k_i, 1 \leq i \leq n, k_i \geq 0\}$
$\mathbf{I}(N_{i\ to\ j}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{i\ to\ j}(T_1, \ldots, T_n))$	$\bigcup\{\mathbf{I}(N\{P_1, \ldots, P_k\}) \mid \{P_1, \ldots, P_k\} \subseteq \{T_1, \ldots, T_n\}, i \leq k \leq j\}$
$1 \leq i \leq j \leq n$	

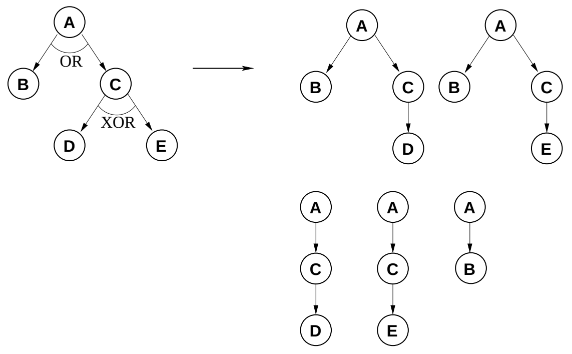
Table 1. Interpretation of grouping facets

In table 1 ϵ denotes a void grouping facet. A node with an ϵ grouping facet can be viewed as a node without grouping facets.

The number of possible interpretations of a data tree DT grows exponentially with the number of grouping facets in the tree, because it is necessary to combine each of the grouping facets of the parent node with the ones of the children. This shows the expressive power of grouping facets: It is possible to express informations that would usually require a large number of elementary data trees in one single data tree with grouping facets.

Example 4.1
A simple data tree with grouping facets and its set of elementary presentations is shown in the figure. Each of the elementary data trees is a model for the enriched tree.

enriched subtree $N_{\mathcal{G}}$	interpreted as
$\mathbf{I}(N_{AND}())$	$\mathbf{I}(N_{AND}()\neg(\emptyset))$
$\mathbf{I}(N_{AND}(T_1,\ldots,T_n))$	$\mathbf{I}(N_{AND}(T_1,\ldots,T_n)\neg(\emptyset))$
$\mathbf{I}(N_{exclude}()\neg(M))$	$\mathbf{I}(N\{\}\neg(M))$
$\mathbf{I}(N_{exclude}(T_1,\ldots T_n))$	$\{\mathbf{I}(N\{\}\neg(M\bigcup(T_1,\ldots,T_n)))\}$
$\mathbf{I}(N_{XOR}()\neg(M))$	$\mathbf{I}(N\{\}\neg(M))$
$\mathbf{I}(N_{XOR}(T_1,\ldots,T_n)$ $\neg(M))$	$\bigcup\{\mathbf{I}(N\{T_i\}\neg(M\bigcup T_j) \mid$ $1\leq i,j\leq n, j\neq i\}$



However, the localization process for databases and patterns based on data trees with grouping facets should not consist in a systematic generation of the elementary data trees representing the patterns and databases. This would be inefficient and would have the drawbacks mentioned at the end of Section 1. Rather, a matching at the *semantic* level is more appropriate and desirable. Such a matching is described below.

5. Matching trees with grouping facets

An important issue for our model is to allow querying with these richer semantics, considering both patterns and databases as data trees with grouping facets. We investigate the relationship between data trees with grouping facets in order to decide whether or not a given pattern *matches* with a given database, and if it matches, what is the result.

5.1. Patterns and Databases

The difference between a pattern and a database comes from the usage and from the fact that a pattern may have variables (discussed in Section 7). While a **database** will usually consist of one or more data trees that *contain* some (useful) data, a **pattern** is used as a query to such a database that either matches with the database (and possibly binds variables) or not. Hence a **pattern** is a declarative way to pose a (localization) query.

For introducing the notion of *matching* between two data trees we use a simulation-based approach, as defined in the following subsection.

5.2. Simulation for DTs

A relationship between two graphs may be expressed in terms of a relation, called *simulation*[1]. It is worth noting its usage, in addition to querying, for classifying all objects (nodes) from a database wrt to a schema [1], or for establishing whether there is a subsumption between two different schemas [3]. In the followings the notion of *simulation* is adjusted to our needs. node-labeled trees, instead of edge-labeled graphs, are considered without losing the power of the notion. We define *matching* between two data trees as a simulation between the trees. Furthermore, we extend this matching in order to cope with grouping facets. Here we consider a matching approach between unordered trees. The order feature can be achieved by adding an ordering relation on the tree nodes, as in [19].

Definition 5.1 (elementary simulation)
Given two elementary data trees DT_1 and DT_2, a binary relation $\mathcal{R} \subseteq \mathbf{Nodes}(DT_1) \times \mathbf{Nodes}(DT_2)$ is an ***elementary simulation*** *on DT_1 and DT_2 if it satisfies*

- *if $n_1\ \mathcal{R}\ n_2$, then name(n_1) = name(n_2)*
- $\forall n_1, n_1' \in \mathbf{Nodes}(DT_1)\ \forall n_2 \in \mathbf{Nodes}(DT_2)$
  $(n_1 \mathcal{R} n_2 \wedge n_1' \in children(n_1) \Rightarrow$
  $\exists n_2' \in \mathbf{Nodes}(DT_2)$
  $(n_1'\ \mathcal{R}\ n_2' \wedge\ n_2' \in children(n_2)))$

If $\mathcal{R}$ is a simulation on two elementary data trees DT_1 and DT_2, then we shall write $DT_1 \mathbf{sim}_{\mathcal{R}} DT_2$.

If the roots r_1 and r_2 of DT_1 and DT_2 are in the simulation ($r_1 \mathcal{R} r_2$), then the simulation is called rooted.

Definition 5.1 states a simple property of preserving the structure of DT_1 in DT_2, although the nodes in DT_2 might have additional child nodes that don't have correspondences in DT_1.

The extension of the simulation for elementary data trees will also have to take into consideration the interpretations for the grouping facets.

5.3. Naïve Matching with Grouping

For data trees with grouping, we will first present a simulation approach that seems straightforward for the task. In Section 6.2, we will see that this approach needs further refinement.

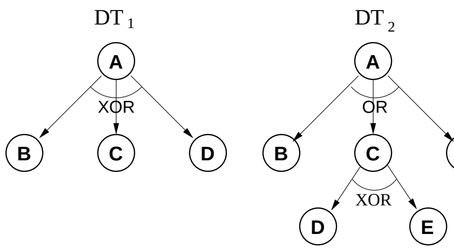
Definition 5.2 (grouping simulation)
Given two enriched data trees DT_1 and DT_2 with grouping facets, an elementary relation $\mathcal{R} \subseteq Nodes(DT_1) \times Nodes(DT_2)$ is a ***grouping simulation*** *on DT_1 and DT_2 if it satisfies*

$\exists\, I_1 \in \mathcal{I}_{\mathcal{G}}(DT_1)\, \exists\, I_2 \in \mathcal{I}_{\mathcal{G}}(DT_2)\; (I_1 \mathbf{sim}_{\mathcal{R}} I_2 \Rightarrow DT_1 \mathbf{sim}_{\mathcal{R}} DT_2)$

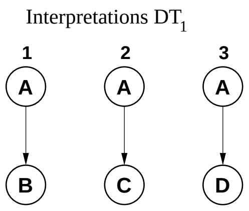
If $\mathcal{R}$ is a grouping simulation on DT_1 and DT_2 with grouping, then we shall write $DT_1 \mathbf{sim}^g_{\mathcal{R}} DT_2$ instead of $DT_1 \mathbf{sim}_{\mathcal{R}} DT_2$.

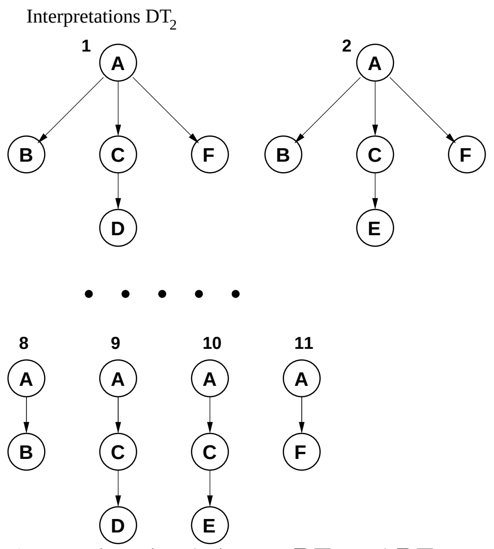
The enhanced simulation defined above maps not only nodes from DT_1 to nodes from DT_2, but also considers the possible grouping facets of DT_1 and DT_2, making use of the interpretations of the facets. Informally speaking, it states that, in order to have a matching between two nodes from DT_1 and DT_2, at least one of the interpretations of the grouping facets attached to the first node should match to at least one of the interpretations of the grouping facets attached to the second. Although this may be refined with techniques similar to branch-and-bound search, the complexity is still exponential in both space and time. Therefore we suggest an enhancement in Section 6.2. For a better understanding, refer to Example 5.1, where the pattern and the database are represented as data trees with grouping facets.

Example 5.1
Following Definition 5.2, we will use two data trees DT_1 and DT_2 to demonstrate the grouping simulation.



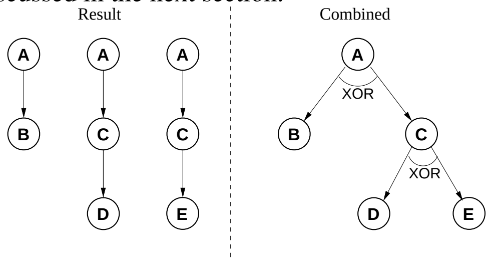
The interpretations of the two data trees are given in the next figure. Of the 11 interpretations of DT_2, only the first two and the last 4 are given in this figure.





A grouping simulation on DT_1 and DT_2 corresponds to an elementary simulation on one interpretation of DT_1 and one interpretation of DT_2. In this example, there are three elementary simulations on interpretations of DT_1 and DT_2, $(1, 8)$, $(2, 9)$ and $(2, 10)$. Note that the semantics of the XOR facet forbids elementary simulations such as $(1, 1)$.

Using the elementary simulations $(1, 8)$, $(2, 9)$, and $(2, 10)$ one can construct the following results of the matching of DT_1 and DT_2. Note that a combined presentation of these results using grouping constructs, as shown in the figure, would be preferable. An improved matching technique for efficiently computing such a combined result is discussed in the next section.



6. Answer semantics

In order to allow querying the data, it is necessary to generate answers containing the fragments from the database that match with the pattern. Recall Example 5.1, where the result of matching is computed with respect to the simulation between pattern and database.

This section introduces a method to compute such results. Furthermore, based on these results, an efficient matching is presented.

In a first step, we refine what has already been used without further investigation in the previous section: Simulation will be used as a result for matching patterns on *elementary* data trees.

6.1. Simulation as Result

Since the simulation that specifies a match between a pattern and a database already contains the references to the matching parts, it seems straightforward to use this information as result and generate the matching fragment out of it.

However, this approach has some deficiencies:

- in general, there is more than one simulation between a pattern and a database
- usually not only the tree specified by the pattern is expected as a result, but also the context of its nodes, i.e. the whole branch of the database that matched (without this it would be a "what you get is what you already have")

The first problem can be addressed with a technique called *maximal simulation*. The *maximal simulation* is the union over all possible simulations for a given pattern and database:

Proposition 6.1 (see [1], page 136)
If $DT_1 \mathbf{sim}_{\mathcal{R}_1} DT_2$ *and* $DT_1 \mathbf{sim}_{\mathcal{R}_2} DT_2$ *then* $DT_1 \mathbf{sim}_{\mathcal{R}_1 \cup \mathcal{R}_2} DT_2$.

Computing the maximal simulation is not difficult and will result in the largest matching fragment of the database. A simple algorithm that generates the maximal simulation by searching for all possible simulations is presented in [1]. Refinements are possible that find the maximal simulation immediately.

The second problem could be overcome by giving the user the option to chose for each node in the pattern whether it should "return" all children or only the matched children.

6.2. Grouping Inheritance: Non-Naïve Matching with Grouping

Using the maximal (rooted) simulation for elementary data trees presented before, we now introduce an efficient method to calculate the result for data trees *with grouping* for the case when the matching is *rooted*.

In Section 5.3, it was necessary to compute all of the interpretations for a tree with grouping when matching two trees based on simulation. Although it is possible to restrict this similar to a branch and bound search, the complexity is nonetheless exponential in both time and space.

Furthermore, it appears difficult to combine the simulations from data trees with grouping in the same way as can be done for an elementary simulation with maximal simulation. The combination would end up in recalculating the grouping that was already there (before it was resolved to elementary data trees).

Grouping Facet in the		
database	pattern	combined result
ϵ	ϵ	ϵ
AND	ϵ	AND
OR	ϵ	OR
XOR	ϵ	XOR
ϵ	AND	AND
AND	AND	AND
OR	AND	AND
XOR	AND	-[1]
ϵ	OR	OR
AND	OR	OR
OR	OR	OR
XOR	OR	XOR
ϵ	XOR	XOR
AND	XOR	-[1]
OR	XOR	XOR
XOR	XOR	XOR
unordered	ϵ	unordered
ordered	ϵ	ordered [2]
ϵ	unordered	unordered
unordered	unordered	unordered
ordered	unordered	ordered [2]
ϵ	ordered	ordered
unordered	ordered	ordered
ordered	ordered	ordered [2]
i to k	l to m	- [3]
i to k	l to m	- [4]
i to k	l to m	$max(i,l)$ to $min(k,m)$

Table 2. Grouping Inheritance

[1]AND and XOR will not generate a match if the number of elements is larger than 1

[2]if children in pattern appear in the same order as in the database, - otherwise

[3]if result contains less than $max(i,l)$ children

[4]if $l < k$ or $m < i$

The new approach treats the grouping facets on a more abstract level by comparing them and is otherwise based on the maximal simulation between elementary data trees.

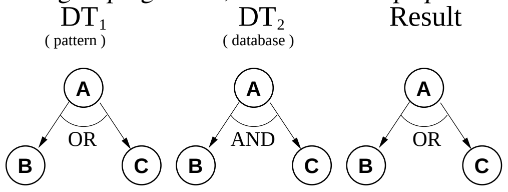
The following steps will generate the desired result for two data trees with grouping:

1. Generate the result from the maximal simulation between the two trees without taking into consideration the grouping properties of the two data trees
2. For each node in the resulting tree, inherit the grouping facet according to the relationships in table 2

In table 2, the relationships are currently given only for the connector facet.

In example 6.1, the matching between two data trees with grouping facets provides an enriched result, which is of the same kind as the input data trees.

Example 6.1
The result of matching of DT_1 in DT_2 is a data tree with grouping facets, wrt relationships provided in table 2.



7. Ongoing work

The data model we described in this paper is by no means complete. Many issues are still ongoing work. This section provides a quick overview over this ongoing work.

Variables

For a full-fledged localization language it will be necessary to introduce "variable" as used in declarative languages like Prolog or Haskell.

The formal introduction of variables is currently investigated and more or less stable. However, there are some issues that still have to be resolved. Among these are multiple occurrences of the same variable in a tree, variables in both the pattern and the database, etc.

Possible approaches could be inspired by the techniques used in logic or functional programming languages.

Depth Facet

The depth facet is probably the most delicate of the grouping facets. A formal representation has already been suggested but has been left out of this paper for space reasons. However the implementation is a topic with a broad range of possibilities for optimization etc.

Inspirations for this could come from the area of graph and search algorithms. Related work is done in the fields of query languages for XML (see [21]).

Combining Grouping Facets

A topic that has not been addressed in this paper is the combination of several grouping facets for the same group of nodes. Combining facets can give very different meanings to a set of nodes (consider e.g. the AND-connector and the depth facet). Therefore, refining the semantics presented in Section 4 so as to accommodate multiple grouping is worth investigating.

Arbitrary Graph Structures

In this paper we restricted the model to tree structured databases. However, it would also be desirable to extend this to databases having an arbitrary graph structure.

Non-Rooted Matching

In many applications it might be desirable to match some pattern with some substructure of the database. While the simulation technique allows such matching, research is necessary in the field of answer semantics and implementation.

Implementations

One of the main issues is to bring the presented ideas into an algorithmic form. Currently finished is an implementation of matching with and without grouping, but generating results like presented in Section 6 is not yet possible.

8. Related work

A different approach to localization queries in SSD is used by XPath [18]. The difference between XPath and our localization approach is that the localization is done by a path instead of a tree and the result usually is a set of nodes instead of a combined answer.

Inspirations for the topic have originated from the paper [10], where matching for elementary data trees with *aggregated answers* has been proposed. However, our work goes beyond and presents an enriched SSD data model based on adding grouping constructs, i.e. aggregated trees also for databases and patterns.

A collection of tree matching problems, called tree inclusion problems, has been addressed in [8], where the ordered/unordered node-labeled tree model has been used. [8] provides also an extension of tree inclusion problems by logical variables used to extract substructures of the pattern instances and to express equality constraints on them.

The work presented here is also related to semantic modeling in general, see e.g. [12] and [14] and especially to ontologies and RDF(see Section 3).

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.

[2] P. Buneman, S. Davidson, and D. Suciu. Programming constructs for unstructured data. In *DBLP*, 1995.

[3] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Modeling and querying semi-structured data. *Network and Information Systems*, 2(2):253–273, 1999.

[4] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogenous information sources. In *Information Processing Society of Japan*, 1994.

[5] Defense Advanced Research Projects Agency. *The DARPA Agent Markup Language (DAML)*, 2000.

[6] R. D. J. Thierry-Mieg. Syntactic definitions for the ACeDB data base manager. Technical report, MRC-LMB xx.92, MRC Laboratory for Molecular Biology, Cambridge, 1992.

[7] D. F. Jonathan Robie, Don Chamberlin. *QUILT: an XML query language*. http://www.almaden.ibm.com /cs/people/chamberlin/quilt_euro.html, March 2000.

[8] P. Kilpeläinen. *Tree matching problems with application to structured text databases*. PhD thesis, Department of Computer Science, University of Helsinki, 1992.

[9] P. Kröger. Modeling of biological data. Master's thesis, Institute for Computer Sciences, University of Munich, http://www.pms.informatik.uni-muenchen.de/lehre/projekt-diplom-arbeit/biological-data.html, 2001, to appear.

[10] H. Meuss, K. Schulz, and F. Bry. Towards aggregated answers for semistructured data. In *International Conference on Database Theory*, 2001.

[11] On-To-Knowledge IST Programme, http://www.ontoknowledge.org/oil/. *Ontology Inference Layer (OIL)*, 1999-2002.

[12] R. K. Richard Hull. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[13] J. Robie. *XQL: XML Query Language*. http://metalab.unc.edu/xql/xql-proposal.xml, August 1999.

[14] B. Thalheim. *Entity-Relantionship Modeling. Foundations of Database Technology*. Springer, 2000.

[15] W3 Consortium, http://www.w3c.org/RDF/. *RDF*, 1999.

[16] W3C, http://www.w3.org/TR/1998/NOTE-XML-data-0105/. *XML-Data*, Jan. 1998.

[17] W3C, http://www.w3.org/TR/NOTE-ddml. *Document Definition Markup Language (DDML) Specification, Version 1.0*, Jan. 1999.

[18] W3C, http://www.w3.org/TR/xpath. *XML Path Language (XPath)*, 1999.

[19] W3C, http://www.w3.org/TR/query-datamodel/#section-Order-operators. *XML Query Data Model*, Feb 2001.

[20] W3C, http://www.w3.org/XML/Schema. *XML Schema*, March 2001.

[21] W3C, http://www.w3.org/TR/xquery/. *XQuery: A Query Language for XML*, Feb 2001.