Towards Grouping Constructs for Semistructured Data

François Bry, Dan Olteanu, Sebastian Schaffert
Institute for Computer Science, University of Munich, 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 and to enhance query answering. 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 [15] 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 [13]) 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 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. Section 5 deals 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.

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 consider a more abstract example: Consider 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 it is 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 semistructured data, the previous course of studies example could be expressed as follows:

```
<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 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 semistructured 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>
```

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]

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

[1]Permitting indefinite as value for m allows to express the classical quantifiers "*", "+" and "?" as "n to m" facets

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 in the future.

Ontologies [3, 8, 11] have constructs similar to these grouping constructs, however they use them primarily for structuring meta-data. Also schemas for XML (DDML, XML-Data [12], XML-Schema [14] 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 using a combination between the *choice* group element and the *min-/ maxOccurs* facets). The **unordering** can be represented in XML-Schema by the *all* group element. The **repetition** and **exclusion** facets are covered by the *min-/ maxOccurs* facets.

4. Data Model: Trees with Grouping Facets

In this section, a data model of data trees with grouping facets is introduced. Let $\mathbf{N}$ denote a set of nodes, $\mathbf{E} \subseteq \mathbf{N} \times \mathbf{N}$ a set of edges and $\mathbf{L}$ a set of node labels. Furthermore, let $\mathcal{P}(X)$ denote the set of all (finite) repetition free lists with elements from X.

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 [6] or ACeDB [10].

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

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

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

- $name : \mathbf{N} \rightarrow \mathbf{L}$ *maps each node to its label*
- $children : \mathbf{N} \rightarrow \mathbf{Lists}(\mathbf{N})$ *maps each node* $n \in \mathbf{N}$ *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 enriched with grouping facets as follows:

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

- *(*$\mathbf{N}$*,* $\mathbf{L}$*,****root****,name, children) is an elementary data tree*
- $grouping : \mathbf{N} \rightarrow \mathcal{P}(\boldsymbol{G})$ *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, as assumed in 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). 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.

4.1. 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 $N \in \mathbf{N}(DT)$ *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.*

enriched subtree $N_{\mathcal{G}}$	interpreted as the elementary subtrees
$\mathbf{I}(N())$	{ N() }
$\mathbf{I}(N(T_1,\ldots,T_n))$	$\{N(T_1',\ldots,T_n') \mid T_i' \in \mathbf{I}(T_i), 1 \le i \le n\}$
$\mathbf{I}(N\{\})$	$\mathbf{I}(N())$
$\mathbf{I}(N\{T_1,\ldots,T_n\})$	$\bigcup\{\mathbf{I}(N(T_{\pi(1)},\ldots,T_{\pi(n)})) \mid \pi\, permutation\, of\, \{1,\ldots,n\}\}$
$\mathbf{I}(N_\epsilon())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_\epsilon(T_1,\ldots,T_n))$	$\mathbf{I}(N\{T_1,\ldots,T_n\})$
$\mathbf{I}(N_{AND}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{AND}(T_1,\ldots,T_n))$	$\mathbf{I}(N\{T_1,\ldots,T_n\})$
$\mathbf{I}(N_{OR}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{OR}(T_1,\ldots,T_n))$	$\bigcup\{\mathbf{I}(N\{P_1,\ldots,P_k\}) \mid \{P_1,\ldots,P_k\} \subseteq \{T_1,\ldots,T_n\}, 1 \le k \le n\}$
$\mathbf{I}(N_{ord.}())$	$\mathbf{I}(N())$
$\mathbf{I}(N_{ord.}(T_1,\ldots,T_n))$	$\mathbf{I}(N(T_1,\ldots,T_n))$
$\mathbf{I}(N_{unord.}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{unord.}(T_1,\ldots,T_n))$	$\mathbf{I}(N\{T_1,\ldots,T_n\})$
$\mathbf{I}(N_{repeat}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{repeat}(T_1,\ldots,T_n))$	$\bigcup\{\mathbf{I}(N\{T_1'\circ\ldots\circ T_n'\}) \mid T_i' = (T_i,\ldots,T_i),\ \lvert T_i'\rvert = k_i, 1 \le i \le n, k_i \ge 0\}$
$\mathbf{I}(N_{i\ to\ j}())$	$\mathbf{I}(N\{\})$
$\mathbf{I}(N_{i\ to\ j}(T_1,\ldots,T_n))$	$\bigcup\{\mathbf{I}(N\{P_1,\ldots,P_k\}) \mid \{P_1,\ldots,P_k\} \subseteq \{T_1,\ldots,T_n\}, i \le k \le j\}$
$1 \le i \le j \le n$	

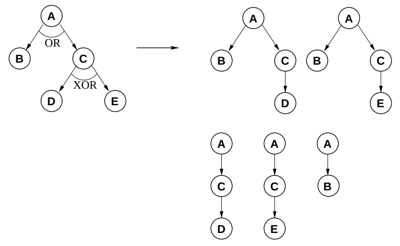
Table 1. Interpretation of grouping facets

$\mathbf{I}$ *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* $\mathbf{I}(DT)$.

In table 1 a void grouping facet is represented by ϵ. A node with an ϵ grouping facet can be viewed as without grouping facets. The formal definitions of other facets, like *XOR* or *exclude*, are presented in [1]. Their semantics comprises informations about missing or not allowed children of a node.

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.



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. Rather, a matching at the *semantic* level is more desirable. Such a matching is described below.

5. Matching Trees With Grouping Facets

Data trees with grouping facets are useful in both, queries and database items. In this section, it is investigated whether or not a given query pattern *matches* with a given data item, and if it does, what is the result.

Patterns and Databases. Patterns and data items differ inasmuch as (1) variables may occur in patterns(discussed in Section 6), not in data items, and (2) a pattern consist in a single data tree, a database in several.

Simulation as Basic Approach. The basic approach for pattern matching is based on a technique called simulation. Simulation is introduced in more detail in e.g. [5]. Simulation essentially is “walking down” through two graphs in parallel. I.e. for each node in the one graph, one tries to find the same node in the other graph. During the tree traversal, only edges having the same labels in both graphs are selected. The standard simulation [5] can easily be adapted to the elementary data trees considered in this paper (cf. [1]).

Two approaches for Data Trees with Grouping. In contrast, adapting simulation to data trees with grouping facets require significant modifications to the simulation definition and algorithms. Two approaches are possible (cf. [1]).

The first, naïve, approach defines a “simulation with grouping” with respect to the standard simulation without grouping: For two data trees with grouping DT_1 and DT_2, there exists a simulation with grouping if there is at least one elementary simulation between an interpretation of DT_1 and one interpretation of DT_2. This approach is straightforward. However in most cases it results in inefficient computations, especially in those cases where there is a perfect match, or no matches at all.

A second, more sophisticated approach is described in [1]. It computes the grouping simulation by just comparing the grouping facets of the two data trees. This approach first computes a “maximal simulation” (covering all of the possible other simulation) between the two data trees without considering grouping facets. If this succeeds, an aggregated answer is then determined by using a simple comparison table.

Answer Semantics. The primary interest is not in whether or not the pattern matches (it is assumed that it matches), but in retrieving a result out of the localization. Using the naïve approach, the result is given implicitly in the simulation, but needs to be reconstructed. Using the second approach on the other hand, one easily gets a third data tree aggregating the possibly many answers that one would have obtained by querying a “wider” database and with a “wider” pattern. Answer semantics is covered in [1].

6. Ongoing work

The data model we described in this paper is by no means complete. Many issues are still ongoing work:

Variables. For a full-fledged localization language it will be necessary to introduce “variables”. Possible approaches could be inspired by the techniques used in logic or functional programming languages.

Depth Facet. The depth facet is the most delicate of the grouping facets. A formal representation has already been suggested but has been left out of the paper for space reasons.

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 [15]).

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 trees. It would be desirable to extend this to databases having an arbitrary graph structure.

Non-Rooted Matching. In many applications it might be desirable to match patterns with substructures of the database. While the simulation technique allows such matching, it is necessary to investigate answer semantics.

7. Related work

A different approach to localization queries in SSD is used by XPath [13]. The difference between XPath and our localization approach is that the result usually is a set of nodes instead of a combined answer.

Inspirations for the topic have originated from the paper [7], 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 [4], where the ordered/unordered node-labeled tree model has been used. [4] 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. [9] and especially to ontologies and RDF [11, 8] (see Section 3).

References

[1] F. Bry, D. Olteanu, and S. Schaffert. Grouping constructs for semistructured data. Technical report, University of Munich, available at http://www.pms.informatik.uni-muenchen.de/publikationen, 2001.

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

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

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

[5] S. Abiteboul et al. *Data on the Web. From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[6] S. Chawathe et al. The TSIMMIS project: Integration of heterogenous information sources. In *Information Processing Society of Japan*, 1994.

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

[8] On-To-Knowledge IST Programme, http://www. onto-knowledge. org/oil/. *Ontology Inference Layer (OIL)*.

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

[10] J. Thierry-Mieg and R. Durbin. Syntactic defi nitions for the ACeDB data base manager. Technical report, MRC-LMB xx.92, MRC Laboratory for Molecular Biology, Cambridge, 1992.

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

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

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

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

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