A Three-Layer Approach to Semistructured Data

André Bergholz* Johann Christoph Freytag

Abstract

The field of semistructured data arises from the fact that the amount of data available and the variety of representations has grown tremendously over the past couple of years. We propose a graph-based three-layer model for querying semistructured data. The bottom layer consists of syntactically well-formed objects, the middle layer of different sorts of schemata and the top layer of the database-related constructs such as queries. For the middle layer we suggest predicate and path schemata. These schema sorts serve as the basis for the queries. The modularity of the approach and the more direct relationship between schema and query lead to more flexibility and interactivity when moving from content-based to structure-based data handling, which is to us the core issue of semistructured data.

1 Introduction and Motivation

The amount of electronically available data material has grown tremendously during recent years. The variety of data representations is vast. This has led to the research area of semistructured data. One of the main problems is that defining schema information for some data in advance turns out to be very difficult and that thus a reverse engineering approach has to be considered. In this young research area the term semistructured data has no precise definition yet. In general, semistructured data arises when less effort is put into structurizing and homogenizing data. Consequently, more effort is necessary when retrieving the data. Abiteboul calls it "data that is neither raw data nor strictly typed" [1]. Examples for semistructured data include HTML files, BibTeX files or genome data. They share the following properties:

1. The structure may be irregular. The data material may be over- or underspecified in relationship to the schema.
2. The structure may vary. Over a discourse world the granularity of the structure depends on the view.
3. The structure of the data may be implicit. The structure becomes clear only after analyzing the actual value of the data. In particular, structured text such as HTML or BibTeX files contain structural information that is implicit.

Additionally, Buneman remarks that some data really is unstructured, e. g. because the possibly existing structure is not discovered yet [2]. He also points out that the research area of semistructured data will provide a significant contribution to the field of data integration.

In summary, the core problem in semistructured data is that the structure of the data is not fully known. This leads to the fact that querying the data is often content-based as opposed to the structure-based querying e. g. in relational systems. Furthermore, this has led to the fact that users often browse through data instead, because no structural knowledge (or *schema information*) is necessary.

*This research was supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316)

2 A three-layer approach: The global picture

In this section we describe the layers of the proposal. We believe in the idea that semistructured data is about moving from *content-based* data handling to *structure-based* data handling. Both data and user behaviour evolve over time. On the one hand the structure of the data becomes clearer as time commenses; irrelevant aspects are removed, other aspects occur. On the other hand users, when first confronted with some data, will start with simple browsing in order to get some idea about the data and will be able to formulate queries based on structure at some later time.

We emphasize a modular approach. It allows queries solely based on content but benefits from already known structure and also allows dynamic handling of structural information.

We propose three layers, the *objects*, the *schemata* and the *database-related constructs*. The objects represent the *content meaning* of the database. They have per default no additional, structural or procedural meaning nor do parts of them. The only thing we require is that an object conforms to a particular syntax, i. e. , that it is *well-formed*. For an example please refer to Figure 3. Schemata are also well-formed objects; they do have a *structural meaning* but no content meaning. A schema describes a set of objects. Later in this paper we will propose different sorts of schemata. On the top layer we have the database-related constructs such as queries, views and constraints. They are expressed by the means of the middle layer, the schemata. Therefore, we can guarantee higher expressiveness by adding better sorts of schemata to the middle layer. Figure 1 illustrates the approach.

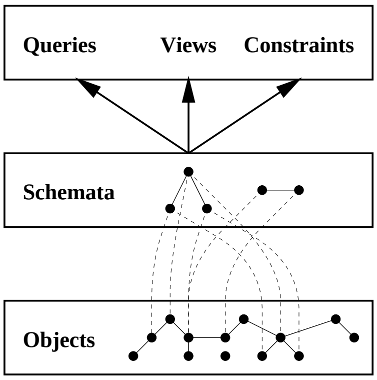

Figure 1: The proposed three-layer approach

The layers interact in different ways, every layer is linked only to the one directly above or below. Schemata and objects are linked by the notion of conformity. It is important to notice that an object can conform to various schemata and, of course, many objects can conform to one schema. In this way we provide a base for recognizing interesting and less interesting parts of a database and also interesting and less interesting schemata. Queries, views and constraints can be formulated using schemata. Additionally, if there is no appropriate schema in advance queries also induce schemata.

Figure 2 summarizes the query scenario based on the proposed layer model. The query process is divided into four steps: (i) The user submits the query. (ii) The query component consults the schema collection to see whether anything appropriate for the processing of the query can be found and then turns to the object database to answer the query. (iii) The answer is sent to the user. (iv) The schema induced by the user query may be integrated into the schema collection. This reintegration as well as occasional reorganization of the schema collection is an open issue and completely omitted in the rest of the paper.

The goal of this project is:

1. to provide a flexible and extendible query mechanism for data stored in semistructured form,

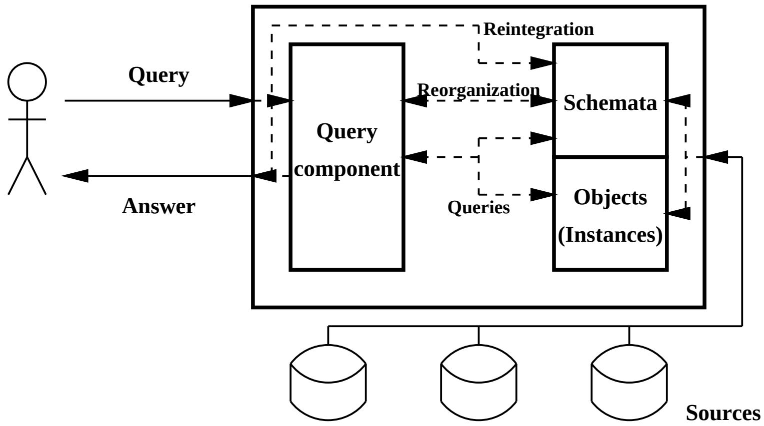

Figure 2: The query scenario

2. to optimize the processing of queries,
3. to establish a connection to recent technologies (such as XML),
4. and to address the maintenance of schema information.

In the following sections we will describe the three layers of the model in more detail.

3 Objects - The bottom layer

This section introduces the basic data model that is used throughout the work. Since this will be a graph oriented model a couple of notions from graph theory are adapted. We use the data structure of a labeled graph, the "unifying idea in semi-structured data" [2], for representing information.

Let $\mathcal{L}$ be a set of labels. An ($\mathcal{L}$-)*labeled directed graph* G is a tuple (V, A, s, t, l) where (V, A, s, t) is a total directed graph (with total source and target functions s and t for the arcs) and $l : V \cup A \longrightarrow \mathcal{L}$ is the label function assigning each element in the graph a label from $\mathcal{L}$.

We call an arc sequence $(a_1, \ldots, a_n)$ a *simple path* in the graph (V, A, l) if all a_i are distinct and there exist nodes $v_0, \ldots, v_n$ such that for all a_i $s(a_i) = v_{i-1}$ and $t(a_i) = v_i$. Note that we do not require the $v_0, \ldots, v_n$ to be distinct. We denote the set of all simple paths in a graph by P and, if necessary, a labeled graph by (V, A, P, s, t, l). The source and the target function can be extended to cover non-empty paths $p = (a_1, \ldots, a_n)$ in the way that $s(p) = s(a_1)$ and $t(p) = t(a_n)$.

An *object* is a labeled directed graph. We will from denote objects by lower-case letters (o_1, o_2 etc.) and arbitrary sets of objects by upper-case letters ($\mathcal{O}_1, \mathcal{O}_2$ etc.).

Example 3.1. Throughout the paper we will use the example in Figure 3 showing an extract from a genome database. It shows an entry from the DNA collection 'EMBL' that occurs in humans. Furthermore, it is illustrated that humans are primates and eukaryotes in the classification of species. The hierarchy between the classes is given. We illustrate a semistructured flavor by omitting a 'classifiedBy'-arc between the species and the vertebrata class and by a different level of information given for the species classes.

4 Two sorts of schemata - The middle layer

In this section we introduce the notions of schema and conformity between object and schema. In fact, we introduce two different sorts of schemata, namely predicate and path schemata. With

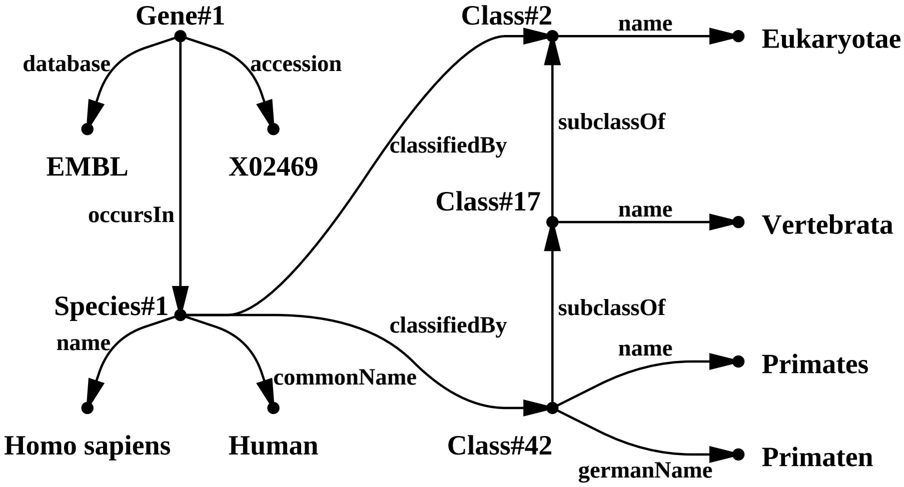

Figure 3: A database example

the chosen modular approach any improvement of the schema language will automatically result in an improvement of the top-level concepts.

Informally, a schema is an object that describes a set of objects. In the simpler syntactic framework of the label world the schema concept certainly exists as well. One label can describe a set of other labels. This is frequently done, data types, predicates and regular expressions are examples.

Now the first idea for schemata in the graph world would be to assign schemata from the label world to the elements of the graph. We choose predicates to be the label world schemata. Given a set of unary predicates $\mathcal{P}$, a *predicate schema* (over $\mathcal{P}$) is an object $s = (V, A, l)$ where the elements are labeled with predicates ($l : V \cup A \longrightarrow \mathcal{P}$).

In order to establish a relationship between the schema and the objects described by it we have to establish the notion of *conformity* between schema and object. Depending on the direction of the mapping we say that we *match* the schema into the object or we *interpret* the object into the schema. A *match* of a predicate schema s into an object o is an isomorphic embedding of s into o, i. e. a total, injective function $m : s \longrightarrow o$ that maps nodes to nodes, arcs to arcs and respects the morphism property for the source and target functions. We do not require type compatibility in the traditional graph transformation sense but rather we require that for all $x \in V_s \cup A_s$ the predicate $l_s(x)$ holds for $l_o(m(x))$. If there exists a match of the schema s into the object o we call o an *instance* of s.

Example 4.1. The example in Figure 4 gives a predicate schema that is intended to conform to all species. In fact, it conforms to anything having a 'name' and a 'commonName' arc. Note, that we use label constants c as abbreviations for the predicate $X = c$ and the predicate $true()$ for expressing that we don't care about the actual label in the object.

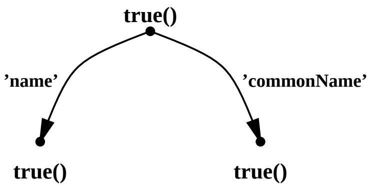

Figure 4: A predicate schema

However, in addition to making statements about the labels in an object there is also a structural level. Let us for a moment follow the intuition that nodes represent some kind of entities and arcs represent the relationships between them. It is true, especially in semistructured data, that exact relationships between entities are not completely known, it may even not be known, whether they are expressed directly or indirectly. This is why paths play an important role and are featured in all existing query languages. In our approach we introduce a new schema sort, namely the path schema.

Informally, we first introduce a set of quantifiers which will be used to indicate length constraints on paths. We denote the set of quantifiers by $\mathbf{Q}$, note, that we use a bold letter here to indicate that it is the set of all quantifiers. We can also give $\mathbf{Q}$ explicitely, it is the union of the sets $\{[m,n]\}$, $\{[m,]\}$, $\{[,n]\}$ and $\{*,+,?\}$ for arbitrary natural numbers m and n, $m \leq n$ (n may also be $+\infty$).

Given a set of unary predicates $\mathcal{P}$ a *path schema* is then an object $s = (V, A, s, t, l)$, where the nodes are labeled with predicates as before and the arcs are labeled with predicates together with a quantifier ($l : V \cup A \longrightarrow \mathcal{P} \cup \mathcal{P} \times \mathbf{Q}$ and $l|_V : V \longrightarrow \mathcal{P}$). For arcs we will denote the sublabels indicating the predicate and the quantifier with $\pi_{\mathcal{P}}(l)$ and $\pi_{\mathbf{Q}}(l)$, respectively.

Obviously, we have to define a new notion of match now. A *match* of a path schema $s = (V_s, A_s, s_s, t_s, l_s)$ into an object $o = (V_o, A_o, P_o, s_o, t_o, l_o)$ is a total, injective function $m : s \longrightarrow o$ such that nodes are mapped to nodes and arcs are mapped to simple paths in P_o. Since we have also a source and a target function for non-empty simple paths we derive the *structure-preserve requirement* from the morphism property and demand $\forall a \in A_s : m(s_s(a)) = s_o(m(a)) \wedge m(t_s(a)) = t_o(m(a))$. As for predicate schemata we require that the predicates in the schema hold for the respective labels in the object. Additionally, the lengths of the paths $m(a)$ have to conform to the semantics of the quantifier $\pi_{\mathbf{Q}}(l_s(a))$, that is for a quantifier $[m,n]$ the length of the respective path has to be no smaller than m and no bigger then n. In this context $[m,]$, $[,n]$, $*$, $+$ and $?$ are shorthands for $[m,\infty]$, $[0,n]$, $[0,\infty]$, $[1,\infty]$ and $[0,1]$, respectively. Please note that every predicate schema is also a path schema and that it does not matter which of the two kinds of match you use when you match a predicate schema into an an object.

Example 4.2. In Figure 5 we have a path schema that conforms to all the subclasses of the eukaryotes. The dashed lines indicate the match function, only the relevant part of it and the database is given.

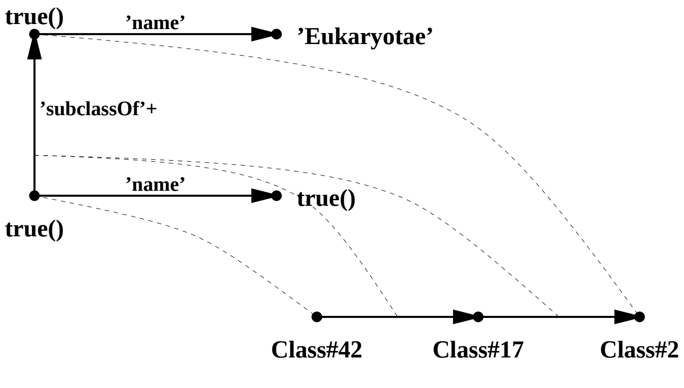

Figure 5: A path schema

5 Queries - The top layer

Schemata of the middle layer can be used for a variety of purposes, e. g. for specifying views or constraints on data. However, we want to use them to formulate queries first. It is important to note that, unlike in other graph transformation approaches, the underlying graph, the database, is not changed. We postulate three kinds of queries: the pattern query, the focus query and the transformation query.

A schema in itself is already a query, the answer being the instances of the schema. We call this kind of query a *pattern query*.

With the focus query we give an equivalent to the relational projection operator. A *focus query* is a tuple (s, s'), where s' is a subschema of s. We adopt the usual subgraph notion here. The answer to a focus query (s, s') is the answer to the pattern query (s), but projected only to the match of s'.

Example 5.1. In Figure 6 we query for the vertebrates class. The schema s consists of the two nodes and the arc at the top of the figure. The subschema s' is indicated by circles, in this case it consists only of the one node on the left. We are thus interested only in the identifier of the vertebrates class. The match to the relevant part of the database graph is indicated by the dashed line.

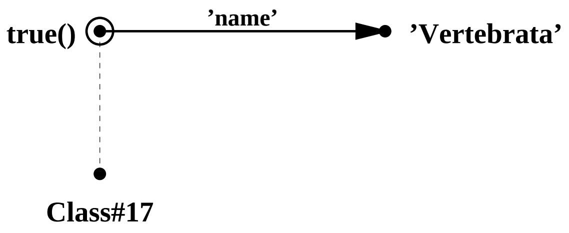

Figure 6: A focus query

A *transformation query* consists of a schema l as left-hand-side, an object labeled with terms r as right-hand-side and a possibly partial mapping t between l and r. The motivation behind it is to be able to restructure the answer presented to the user. This is in the flavor of the single-pushout approach [3], although it is not the database graph, but rather the matches of the left-hand side, that are transformed.

Transformation queries can also be used to restructure the whole database. Their semantics must then be adapted towards a more traditional graph transformation concept. This adaption is not covered in this work.

6 Related work

Two international projects are closest to our work, namely the Lore project [4] at Stanford and the UnQL project at UPenn [5].

The Lore project aims at storing and querying semistructured data. It originated from the TSIMMIS project [6] that provided tools for integration of heterogeneous data. Both projects use a simple and flexible data model, the Object Exchange Model (OEM) [7], to represent the data. All objects are self-describing, there is initially no need for classes or schemata. For this model a query language named Lorel ("Lightweight Object Repository Language") [8] has been developed. Syntactically, Lorel is similar to SQL/OQL. It supports simple queries, boolean connectors, subqueries and label markers to distinguish prefixes in paths. The semistructured flavor comes through by the introduction of general path expressions. They serve two purposes: One can use wildcards for paths, and one can define regular expressions over the labels. In a next step some schema information in the form of dynamically created and maintained DataGuides [9] is introduced. This helps the user to get a better view on the data, e. g. for query formulation. Additionally, DataGuides are useful for optimization, since better query evaluation plans may be developed.

A similar, somewhat more theoretical project is the University of Pennsylvania's UnQL project. A query language UnQL ("Unstructured Query Language") [5] has been developed. As a data model the concept of an edge-labeled tree is used. The query language UnQL has a similar functionality as Lorel. As query results edge-labeled trees and labels can occur. One interesting aspect is the explicit treatment of restructurings of a database by introducing the `traverse`-command. Again, post-defined schemata for the purpose of optimization are introduced [10].

Much related work arise in connection with information extraction from the WWW. A main focus lies on query languages suited for the Web ([11], [12], [13], [14]). A very fundamental work on data stored in files is the one by Abiteboul, Cluet and Milo [15]. Structured files are transformed to databases so that file querying and manipulating by using database technology becomes possible.

Graph transformations address the dynamic aspects of graphs and are thus interesting as a means to manipulate query answers and the whole database. The basic idea is the rule-based manipulation of graphs. These rules, together with an initial work graph, are called a graph grammar. Graph transformations have been a subject since the early 70s ([16], [17]). To name two systems, RWTH Aachen's PROGRES [18] focuses on an executable specification language based on graph rewriting systems. It uses EER-like class diagrams for the definition of complex object structures. Among its applications are process modeling and reengineering tools. TU Berlin's AGG [19] provides a visual programming environment that allows visual specification of the topmost object structures and its dynamics.

7 Conclusion

The field of semistructured data offers many new challenges to database research. We have proposed a three-layer model that separates syntax and semantics in order to form queries over semistructured data. Our modular proposal allows us to introduce various sorts of schemata. The expressiveness of queries is guided by these schema sorts. At this stage we suggested predicate and path schemata. Predicate schemata put constraints on the labels in an object. Path schemata are used if a sequence of arcs between nodes cannot be fully specified. We use concepts derived from the area of graph transformations in order to formulate queries over the data. We introduced pattern, focus and transformation queries.

In contrast to other approaches we provide a modular framework that provides a base for a more flexible and interactive data handling. Through a more direct relationship between schema and query and by introducing more than one sort of schema (as opposed e. g. to the relational model) we believe that a better query mechanism respecting the degree of schema information known is possible. Schema information is induced by user queries. Querying works without any schema information, but the processing of queries may benefit from it. On the more pragmatic side, because our approach is based on schema matching, we do not need to specify entry points to a database and can thus adopt a more general graph model than other approaches for semistructured data.

Optimizing query processing, making full use of the given schema information and detecting "good" schemata are challenges of the future.

Acknowledgment

The authors wish to thank Lukas Faulstich, Felix Naumann, Myra Spiliopoulou and Gabriele Taentzer for many interesting discussions and useful comments.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proceeding of the International Conference on Database Theory*, 1997.

[2] P. Buneman. Semistructured data. In *Proceedings of the PODS Conference*, 1997.

[3] M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.

[4] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[5] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[6] 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 *Proceedings of the Information Processing Society of Japan (IPSJ) Conference*, pages 7–18, October 1994.

[7] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogenous information sources. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

[8] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases*, pages 319–344, 1995.

[9] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the VLDB Conference*, August 1997.

[10] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of the International Conference on Database Theory*, 1997.

[11] D. Konopnicki and O. Shmueli. W3qs: A query system for the world-wide web. In *Proceedings of the VLDB Conference*, pages 54–65, 1995.

[12] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the world-wide web. In *Proceedings of the Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.

[13] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the web. In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering (RIDE)*, February 1996.

[14] S. Abiteboul and V. Vianu. Queries and computation on the web. In *Proceedings of the International Conference on Database Theory*, 1997.

[15] S. Abiteboul, S. Cluet, and T. Milo. Querying the file. In *Proceedings of the VLDB Conference*, 1993.

[16] J. L. Pfaltz and A. Rosenfeld. Web grammars. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 609–619, 1969.

[17] H. Ehrig, M. Pfender, and H. Schneider. Graph grammars: an algebraic approach. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180, 1973.

[18] A. Schuerr. Programmed graph replacement systems. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1. World Scientific, 1997.

[19] The AGG-Homepage, `http://tfs.cs.tu-berlin.de/agg/`.