Semistructured Data [*]

Peter Buneman
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
peter@cis.upenn.edu

Abstract

In semistructured data, the information that is normally associated with a schema is contained within the data, which is sometimes called "self-describing". In some forms of semistructured data there is no separate schema, in others it exists but only places loose constraints on the data. Semistructured data has recently emerged as an important topic of study for a variety of reasons. First, there are data sources such as the Web, which we would like to treat as databases but which cannot be constrained by a schema. Second, it may be desirable to have an extremely flexible format for data exchange between disparate databases. Third, even when dealing with structured data, it may be helpful to view it as semistructured for the purposes of browsing. This tutorial will cover a number of issues surrounding such data: finding a concise formulation, building a sufficiently expressive language for querying and transformation, and optimization problems.

1 The motivation

The topic of semistructured data (also called unstructured data) is relatively recent, and a tutorial on the topic may well be premature. It represents, if anything, the convergence of a number of lines of thinking about new ways to represent and query data that do not completely fit with conventional data models. The purpose of this tutorial is to to describe this motivation and to suggest areas in which further research may be fruitful. For a similar exposition, the reader is referred to Serge Abiteboul's recent survey paper [1].

The slides for this tutorial will be made available from a section of the Penn database home page
`http://www.cis.upenn.edu/~db`.

[*]This work was partly supported by the Army Research Office (DAAH04-95-1-0169) and the National Science Foundation (CCR92-16122).

1.1 Some data really is unstructured

The most obvious motivation comes from the need to bring new forms of data into the ambit of conventional database technology. Some of these, such as documents with structured text [3, 2] and data formats [9, 17], while they may call for increasingly expressive query languages and new optimization techniques, only require mild extensions to the existing notion of data models such as ODMG [13]. However these extensions still require the prior imposition of structure on the data, and there are some forms of data for which this is genuinely difficult.

The most immediate example of data that cannot be constrained by a schema is the World-Wide-Web. As database researchers we would like to think of this as a database, but to what extent are database tools available for querying or maintaining the web? Most web queries exploit information retrieval techniques to retrieve individual pages from their contents, but there is little available that allows us to use the structure of the web in formulating queries, and since the web does not obviously conform to any standard data model, we need a method of describing its structure.

Another example, little known to the database community but responsible for piquing the author's interest in this topic, is the database management system ACeDB, which is popular with biologists [36]. Superficially it looks like an object-oriented database system, for it has a schema language that resembles that of an object-oriented DBMS; but this schema imposes only loose constraints on the data. Moreover the relationship between data and schema is not easily described in object-oriented terms, and there are structures that are naturally expressed in ACeDB, such as trees of arbitrary depth, that cannot be queried using conventional techniques.

1.2 Data Integration

A second motivation is that of data exchange and transformation, which is the starting point for the Tsimmis project [33, 21] at Stanford. The rationale here is that none of the existing data models is all-embracing, so that it is difficult to build software that will easily convert between two disparate models. The Object Exchange Model (OEM) offers a highly flexible data structure that may be used to capture most kinds of data and provides a substrate in which almost any other data structure may be represented. In effect, OEM is an internal data structure for exchange of data between DBMSs, but having such a structure invites the idea of querying data in OEM format directly.

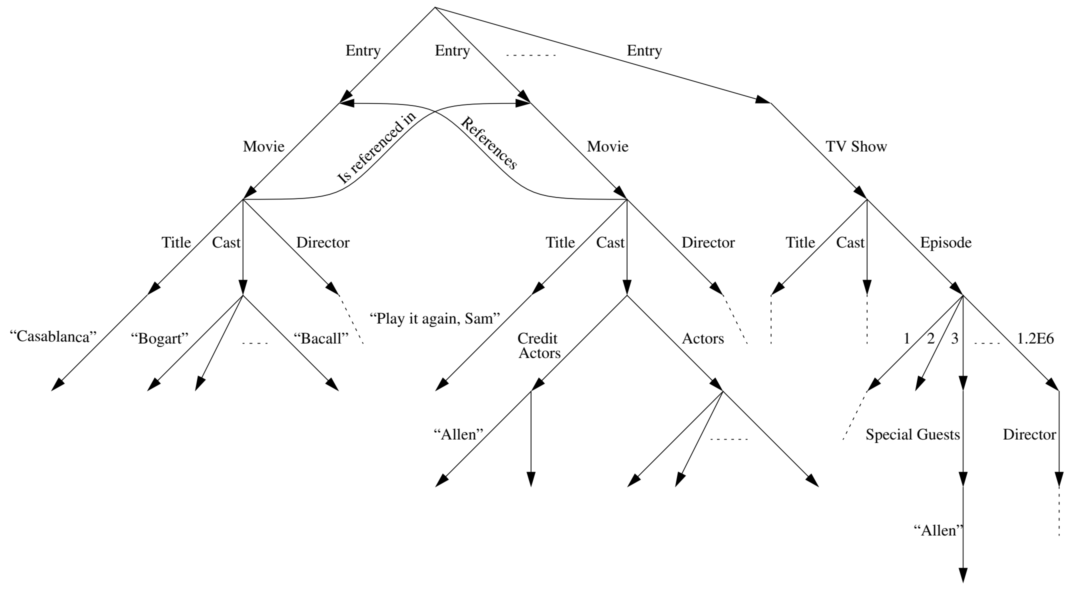

Figure 1: An example movie database.

1.3 Browsing

A final motivation is that of browsing. Generally speaking, a user cannot write a database query without knowledge of the schema. However, schemas may have opaque terminology and the rationale for the design is often difficult to understand. It may help in understanding the schema to be able to query data without full knowledge of the schema. For example the queries,

- Where in the database is the string `"Casablanca"` to be found?
- Are there integers in the database greater than 2^{16}?
- What objects in the database have an attribute name that starts with `"act"`

Such questions cannot be answered in any generic fashion by standard relational or object-oriented query languages. While languages have been proposed that allow schema and data to be queried simultaneously [24] in the context of relational and object-oriented database systems, these languages do not have the flexibility to express complex constraints on paths, and it is not clear how their implementation will work on the structures described below.

2 The Model

The unifying idea in semi-structured data is the representation of data as some kind of graph-like or tree-like structure. Although we shall allow cycles in the data, we shall generally refer to these graphs as trees. The example in figure 1 is taken from [10] in which the data model is formalized as an *edge labeled graph.* The structure is taken (with some inaccuracies) from a well-known web database [23] that provides a good example of semistructured data. There are several things to note about it. If one confines ones attention to the parts of the database below `Movie` edges, the data appears fairly regular except that there are two ways of representing a cast. That is, the data does not quite fit with some relational or object-oriented presentation. Edges are labeled both with data, of types such as `int` and `string` and possibly other base or external abstract types (video, audio etc.). Edges are also with names such as `Movie` and `Title` that would normally be used for attribute or class names. We shall refer to such labels as *symbols.* Internally they are represented as strings. Note that arrays may be represented by labeling internal edges with integers. We can formulate the type of this kind of labeled tree as:

$$\begin{aligned}&\textit{type label} = \textit{int} \mid \textit{string} \mid \ldots \mid \textit{symbol}\\&\textit{type tree} = \textit{set}(\textit{label} \times \textit{tree})\end{aligned}$$

The first line describes a tagged union or variant, the second says that a tree is a set of label/tree pairs. The edges out of nodes in our trees are assumed to be unordered.

There are a number of variations on this basic model, and it is worth briefly reviewing them. In [5] leaf nodes are labeled with data, internal nodes are not labeled with meaningful data, and edges are labeled only with symbols

$$\begin{aligned}&\textit{type base} = \textit{int} \mid \textit{string} \mid \ldots\\&\textit{type tree} = \textit{base} \mid \textit{set}(\textit{symbol} \times \textit{tree})\end{aligned}$$

The differences between the two models are minor and give rise to minor differences in the query language. It is easy to define mappings in both directions.

Another possibility is to allow labels on internal nodes, for example:

$$\begin{aligned}&\textit{type base} = \textit{int} \mid \textit{string} \mid \ldots \mid \textit{symbol}\\&\textit{type tree} = \textit{label} \times \textit{set}(\textit{label} \times \textit{tree})\end{aligned}$$

The problem with using this representation directly is that it makes the operation of taking the union of two trees difficult to define. However, by introducing extra edges, this represaentation can be converted into one of the edge-labelled representations above.

A final and more complex issue is that of object identity, by which we mean node labels – or possibly edge labels – that, apart from an equality test, are not observable in the query language. In OEM, object identities are used as node labels and place-holders to define trees. While object-identities provide an efficient way to define and test equality *within* a database, they pose problems when comparing data *across* databases. See [10, 25, 32] discussions and related work.

It is straightforward to encode relational and object-oriented databases in this model, although in the latter case one must take care to deal with the issue of object-identity. However, the coding is not unique, and the examples in [10] and [5] show some differences in how tuples of sets are treated.

The term "self describing" is often used to describe unstructured data. In each of the models we have described, the data is a tagged union type, and one can imagine a program whose behavior is dynamically determined by "switching" on the type. For example, a program's behavior may be altered by whether it finds an integer or string as a label, and one would expect any language for dealing with semistructured data to incorporate predicates that describe the type of an edge or node. The situation is similar to that in programming languages. Lisp and many interpreted and scripting languages are *dynamically* typed. Predicates are available to determine (at run time) type of a value or class of an object. Languages in the Algol tradition (Pascal, C, ML, Modula) are *statically* typed. Predicates are not needed to determine the type of a value because it is known from the source code of the program and hence to the programmer. There is a good analogy between dynamic type systems and semistructured data on one hand, and static type systems and databases with schemas on the other

3 Query Languages

There appear to be two general approaches to devising query languages for semistructured data. First, take SQL (or perhaps OQL[14, 13]) as a starting point and add enough "features" to perform a useful class of queries. The second approach is to start from a language based on some formal notion of computation on semistructured data then to massage that language into acceptable syntax. It is remarkable that the two approaches appear to end up with very similar languages.

Let us start with the first approach to see what what kinds of queries are useful. The following SQL-like syntax suggests itself:

```
select Entry.Movie.Title
from DB
where Entry.Movie.Director ...
```

However the syntax does not make clear how much of the two paths `Entry.Movie.Title` and `Entry.Movie.Director` are to be taken as the same. The solution is to introduce variables to indicate how paths or edges are to be tied together. These variables can then be used in other expressions to form new structures. Label variables, tree variables and possibly path variables are needed to express a reasonable set of queries.

The next problem is that one wants to specify paths of arbitrary length to find, for example, all the strings in the database. This requires us to be able to express arbitrary paths in our syntax. Even this is not enough. Consider the problem of finding whether `"Allen"` acted in `"Casablanca"`. One might try this by searching for paths from a `Movie` edge down to an `"Allen"` edge, but one would *not* want this path to contain another `Movie` edge. These problems indicate that one would like to have something like regular expressions to constrain paths.

The "select" fragment of UnQL[10] and the Lorel query language [5] solve these problems with very similar syntactic forms. Lorel, which is a component of the Lore project [27] requires a rich set of overloadings for its operators for dealing with comparisons of objects with values and of values with sets. These are avoided in UnQL by not having object identity and exploiting a simple form of pattern matching. Other languages that use a SQL-like syntax include a precursor to Lorel [34], and WebSQL [29, 7] which contains a number of constructs specific to web queries. A language for web site management is proposed in [18].

Having asked what the surface syntax should look like, one wants to ask what the underlying computational strategy should be. Here there appear to be two principled strategies. The first is to model the graph as a relational database and then exploit a relational query language. In our labeled graph model this is remarkably simple. We can take the database as a large relation of type (node-id, label, node-id) and consider the expressive power of relational languages on this structure, but this apparently simple approach has a number of complications:

1. Our labels are drawn from a heterogeneous collection of types, so it may be appropriate to use more than one relation.
2. If information also is held at nodes, one needs additional relations to express this.
3. The node identifiers may only be used as temporary node labels, and one may want to limit the way they can appear in the output of the query. How they are used is related to the discussion of object identity.
4. We are concerned with what is accessible from a given "root" by forward traversal of the edges, and one may want to limit the languages appropriately.

Some forms of unbounded search will require recursive queries, i.e., a "graph datalog", and such languages are proposed in [26, 16] for the web and for hypertext. Theoretical treatments of queries that deal with computation on graphs or on the web appear in [6, 30]. It should also be mentioned that this model of computation is used in [5, 15] as a starting point for optimization.

The second strategy is adopted in the basis for UnQL [11, 10]. Here the starting point is that of *structural recursion*, and is an extension of a principle put forward in [12] that there are natural forms of computation associated with the type. For semistructured data one starts with the natural form of recursion associated with the recursive datatype of labeled trees. However, some restrictions need to be placed for such recursive programs to be well-defined: we want them to be well-defined on graphs with cycles. These restrictions give rise to an algebra that can be viewed as having two components: a "horizontal" component that expresses computations across the edges of a given node (and from this, computations to a fixed depth from the root); and a "vertical" component that expresses computations that go to arbitrary depths in the graph. A property of this algebra is that, when restricted to input and output data that conform to a relational (nested relational) schema, it expresses exactly the relational (nested relational) algebra. Hence an SQL-like language is a natural fragment of UnQL.

The SQL or OQL like languages we have mentioned typically bring information to the surface, but they are not capable of performing complex or "deep" restructuring of the data. Simple examples of such operations include deleting/collapsing edges with a certain property, relabeling edges, or performing local interchanges. Both "graph datalog" and UnQL are capable of various forms of restructuring. For example, in UnQL one can write a query that corrects the egregious error in the `"Bacall"` edge label. One can also perform a number of global restructuring functions such as deleting edges with certain properties or adding new edges to "short-circuit" various paths. The the relationship between the restructuring possible in UnQL and what can be done in "graph datalog" is not understood. Some simple forms of restructuring are also present in a view definition language proposed in [4].

4 Implementation and Optimizations

This topic is very much in its infancy and again depends on the underlying representation of the data. Moreover the optimization prblems differ depending on whether one is using a semistructured model as an interface to existing data or one is building a data structure to represent semistructured data directly [28]. In the former case the extensions of existing techniques for optimization of object-oriented or relational query languages mentioned above may be exploited together with the addition of path or text indices on labels and strings. In the second case, disk layout and clustering, together with appropriate indexing, is also important.

In [10] a large class of computations can be shown to be translatable into a basic graph transformation technique which, in turn, allows some simple optimizations. Also some of the basic optimizations of the relational algebra can be applied to the "vertical" computations. In [35] it is shown how an analysis of the query, combined with some segmentation of the graph into local "sites" can be used to decompose a query into independent, parallel sub-queries. In [5] and [15] extensions to optimization techniques for object-oriented query languages are exploited. In [19] a translation is specified for a fragment UnQL into a an underlying relational structure.

5 Adding Structure

One of the main attractions of semistructured data is that it is unconstrained. Nevertheless, it may be appropriate to impose (or to discover) some form of structure in the data. In [8] a schema is defined as a graph whose edges are labeled with predicates and the property of simulation is used to describe the relationship between data and schema. In [31, 22] the schema is also an edge labeled graph and the stronger relationship of automata equivalence is used. In [20] schemas are used for further optimization.

Schemas are useful for browsing and for providing partial answers to queries. They will also be needed for the passage back from semistructured to structured data, for which a richer notion of schema is necessary. This is an area in which much further work is needed.

6 Acknowledgments

I would like to thank Susan Davidson and Dan Suciu for their collaboration and for stimulating my interest in this area. I am greatly indebted to Serge Abiteboul for most constructive discussions on a number of issues.

References

[1] Serge Abiteboul. Querying semi-structured data. In *Proceedings of ICDT*, Jan 1997.

[2] Serge Abiteboul, Sophie Cluet, Vassilis Christophides, Tova Milo, and Jérôme Siméon. Querying documents in object databases. In *Journal of Digital Libraries*, volume 1:1, 1997.

[3] Serge Abiteboul, Sophie Cluet, and Tova Milo. Querying and updating the file. In *Proceedings of 19th International Conference on Very Large Databases*, pages 73–84, Dublin, Ireland, 1993.

[4] Serge Abiteboul, Roy Goldman, Jason McHugh, Vasilis Vassalos, and Yue Zhuge. Views for semistructured data. Technical report, Stanford University, 1977.

[5] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Weiner. The lorel query language for semistructured data. In *Journal of Digital Libraries*, volume 1:1, 1997. To appear. See `http://www-db.stanford.edu/pub/papers/`.

[6] Serge Abiteboul and Victor Vianu. Queries and computation on the web. In *Proceedings of ICDT*, Jan 1997.

[7] Gustavo O. Arocena, Alberto O. Mendelzon, and George A. Mihaila. Applications of a Web query language. In *Proc. 6th. Int'l. WWW Conf.*, April 1997. In press.

[8] P. Buneman, S. Davidson, Mary Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of ICDT*, January 1997.

[9] P. Buneman, S.B. Davidson, K. Hart, C. Overton, and L. Wong. A data transformation system for biological data sources. In *Proceedings of VLDB*, Sept 1995.

[10] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 505–516, Montreal, Canada, June 1996.

[11] Peter Buneman, Susan Davidson, and Dan Suciu. Programming constructs for unstructured data. In *Proceedings of 5th International Workshop on Database Programming Languages*, Gubbio, Italy, September 1995. To appear.

[12] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, September 1995.

[13] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, California, 1996.

[14] Sophie Cluet and Claude Delobel. A general framework for the optimization of object oriented queries. In M. Stonebraker, editor, *Proceedings ACM-SIGMOD International Conference on Management of Data*, pages 383–392, San Diego, California, June 1992.

[15] Sophie Cluet and Guido Moerkotte. Query processing in the schemaless and semistructured context. Technical report, INRIA, 1997.

[16] Mariano P. Consens and Alberto O. Mendelzon. Expressing structural hypertext queries in graphlog. In *Proc. 2nd. ACM Conference on Hypertext*, pages 269–292, Pittsburgh, November 1989.

[17] Susan B. Davidson, Christian Overton, Val Tannen, and Limsoon Wong. Biokleisli: A digital library for biomedical researchers. In *Journal of Digital Libraries*, volume 1:1, November 1996. See `http://www.cis.upenn.edu/ db`.

[18] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. STRUDEL: A Web Site Management System. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, Tuscon, May 1997.

[19] Mary Fernandez, Lucian Popa, and Suciu Dan. A structure based approach to querying semistructured data, 1997. Manuscript available from `http://www.research.att.com/info/{mff,suciu}`.

[20] Mary Fernandez and Dan Suciu. Query optimizations for semi-structured data using graph schemas, 1996. Manuscript available from `http://www.research.att.com/info/{mff,suciu}`.

[21] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The tsimmis approach to mediation: Data models and languages. In *Proceedings of Second INternational Workshop on Next Generation Information Technologies and Systems*, pages 185–193, June 1995.

[22] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. Technical report, Stanford, 1977.

[23] The internet movie database. `http://us.imdb.com/`.

[24] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In M. Stonebraker, editor, *Proceedings ACM-SIGMOD International Conference on Management of Data*, pages 393–402, San Diego, California, June 1992.

[25] Anthony Kosky. Observational properties of databases with object identity. Technical Report MS-CIS-95-20, University of Pennsylvania, 1995.

[26] Laks V.S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A declarative language for querying and restructuring the world-wide-web. In *Post-ICDE IEEE Workshop on Research Issues in Data Engineering (RIDE-NDS'96)*, New Orleans, February 1996. See also `ftp://ftp.cs.concordia.ca/pub/laks/papers`.

[27] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. Technical report, Stanford University Database Group, February 1997.

[28] J. McHugh and J. Widom. Integrating dynamically-fetched external information into a dbms for semistructured data. Technical report, Stanford University, 1997.

[29] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. In *Proc. PDIS '96*, pages 80–91, December 1996. Available as `tp.db.toronto.edu/pub/papers/pdis96.ps.gz`.

[30] Alberto O. Mendelzon and Tova Milo. Formal models of web queries. In *Proc. PODS '97*, May 1997. In press, available in `ftp.db.toronto.edu/pub/papers/pods97MM.ps`.

[31] S. Nestorov, J Ullman, Weiner J, and S. Chawathe. Representative objects: Concise representations of semistructured hierarchical data. In *Proceedings of the Thirteenth International Conference on Data Engineering*, Birmingham, England, April 1997.

[32] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc 22nd. VLDB conference*, September 1996.

[33] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogenous information sources. In *Proceedings of IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

[34] D. Quass, A. Rjaraman, Y. Sagiv, and J. Ullman. Querying semistructured heterogeneous information. In *Proceedings of the Fourth International Conference on Deductive and Object-oriented Databases*, pages 319–344, dec 1995.

[35] Dan Suciu. Query decomposition for unstructured query languages. In *Proc 22nd. VLDB conference*, September 1996.

[36] Jean Thierry-Mieg and Richard Durbin. ACeDB — A C. elegans Database: Syntactic definitions for the ACeDB data base manager, 1992.