Using Hypersets to Represent Semistructured Data

Piotr Kaminski[*]
piotr@ideanest.com

Abstract

This paper presents a new metamodel based on partially ordered hypersets. The metamodel is minimalist, its only primitives being atoms, potentially infinite hypersets and partial order. The type system is very loose and supports concurrent extensional and intensional definition of types. Every element of the metamodel can be directly referenced, thanks to a powerful automated reification facility. These characteristics make the metamodel well adapted to integrating semistructured data. The metamodel is also compared to a few other information representation formalisms to demonstrate that it subsumes them all and to exhibit some of their problems.

1 Introduction

A large number of information metamodels have been introduced over the last few decades. Each found its niche, according to the data representation needs and available computing power of that time, then was displaced by another, better adapted to the changing conditions. Complementary metamodels often coexisted, but data was rarely transferred between them and then only at great cost.

Today, the relational model is the undisputed leader for highly structured data, occasionally supplemented by object-oriented variants to help lower the impedance mismatch with the comparatively recent object-oriented programming languages. However, the advent of the web and the rise in popularity of document markup languages has caused a sharp increase in the quantity and importance of semi-structured data.

The changing conditions have elicited the usual stream of new (and not-so-new) metamodel contenders. While each has its technical or social advantages, in my opinion none is a superset of all the others that would allow natural integration of data from the various metamodels. In this paper, I present my entry into the fray: a metamodel based on partially ordered hypersets that is both simpler and more powerful than the competing formalisms.

The rest of this paper is divided as follows. Section 2 lays out the design goals and desired features of the metamodel. They will motivate the design of the hyperset-based metamodel (exposed in section 3), and orient its comparison to other metamodels (section 4). Section 5 concludes this paper, summarizing the contributions of this metamodel and listing directions for further research.

2 Goals and Features

The primary goal for this metamodel is that it be sufficiently flexible to be able to represent any kind of semi-structured data that might be encountered on the semantic web. It must be

[*] Dept. of Computer Science, University of Victoria, PO Box 3055, Victoria, BC, Canada, V8W 3P6

able to integrate information from all kinds of sources with no loss of meaning, since it seems rather unlikely (and perhaps undesirable) that everyone will agree on a single metamodel for all data. This means, among other things, that the metamodel cannot force a strict type system, since this would prevent data integration from untyped metamodels.

In order to be flexible, the metamodel should be as simple as possible. A small number of primitive constructs and the ability to combine them in many ways give it the best chance to be able to adapt to any kind of metamodel we might run into. For example, drawing an analogy to programming languages' data models, there should be no distinction made between primitive values and objects (as in C++ and Java). Rather, all information elements should be treated the same way so as to maximize flexibility (as in Smalltalk). The usual tradeoff is a loss of efficiency, which I consider acceptable.

Another feature critical to flexibility is the ability to reference anything in the model. Once again, by analogy to programming languages, we need powerful reflection facilities such as Lisp's ability to reference the program itself as a list, or Scheme's continuations. This leaves the metamodel open to manipulation from within the models themselves. An important aspect of this feature is that the metamodel must be closed under queries: the results of any query will "look" the same as any other part of the model, and can be referenced directly.

Since the new metamodel is meant to be used to integrate data from the semantic web, it must deal with issues of trust. At the very least, it must be easy to identify the provenance of each piece of information, and to filter out information that is not deemed trustworthy. The previous two goals dictate that this should not be a special-purpose mechanism (simplicity) and that it can be achieved by referencing all the data obtained from a source (universal references).

Finally, it is worth mentioning that ease of serialization was *not* considered a goal. If data needs to be exchanged, it can be exported into any of the multitude of other standard formats. Manual data manipulation will be done with the assistance of an application and not directly through a text file. Thus, no serialization format (XML or otherwise) is specified in this paper.

3 The Metamodel

This section introduces a data metamodel that satisfies the goals listed above. Much of it was inspired by the principles and ideas put forth in [5].

3.1 Foundations

The metamodel is composed of just two primitive elements: atoms and hypersets. Atoms are used to represent any piece of information that we cannot or do not wish to further decompose. Examples include values such as Boolean literals, numbers, characters, dates, etc. We also use atoms to stand for things in the world outside our system, such as books, people, web sites, files, etc. The only difference between the two kinds of atoms is that some may stand for things that have a representation in the programming language we happen to be using, while others may not.

The other kind of primitive element is the hyperset. A hyperset is a set, and can contain any mixture of atoms and hypersets. Hypersets differ from classical sets in that any given set is allowed to contain itself. This is achieved by replacing the Foundation Axiom with the Anti-Foundation Axiom [2], so hypersets are often called non-well-founded sets.

Figure 1 gives an example of a small model built using atoms and sets. Figure 1a employs a simple nesting notation. Sets are drawn as large ellipses, atoms as filled dots, and membership is indicated through containment. The atoms' labels merely indicate the value to which each atom corresponds; they are not part of the model. Instead, each atom that has a representation in the underlying system would be mapped to it in an implementation-dependent way.

Figure 1b shows the same model as figure 1a, but uses a custom-made direct graph notation. Sets are drawn as empty dots and atoms as filled dots. Edges point from sets to their members.

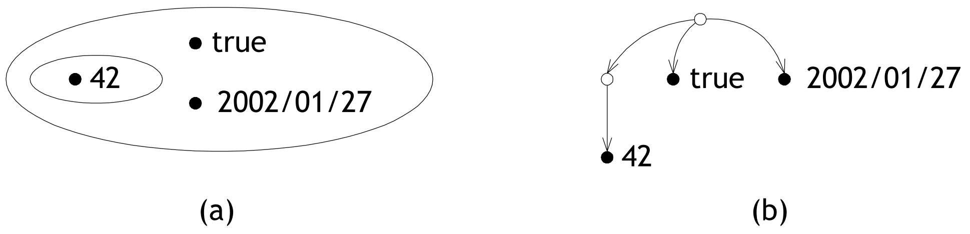

Figure 1

You can tell that both sets in figure 1 are classical because the graph in 1b is acyclic. Figure 2 shows an example of a recursive hyperset that generates an infinite expansion. As it is not practical to represent such models using nested diagrams, the rest of this paper will only use the directed graph representation. Section 4.2 explains why this is only a representation and how the metamodel differs from a traditional directed graph.

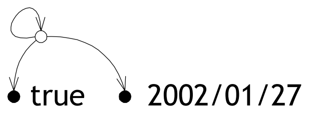

Figure 2

With only two kinds of primitive elements, the metamodel is very simple. We could pare it down further by defining atoms to be empty hypersets. There is no benefit in such a redefinition, though, since it doesn't make the metamodel any more expressive and it may be confusing to model atomic values as empty sets.

3.2 Types

While the metamodel does not enforce strict typing, it is useful to look at how types might be represented. The most natural approach is to consider a type as the set of its instances. The instances can be added to the type set manually, which would directly correspond to an extensional definition of the type. The contents of the type set can also be the result of a query, perhaps based on some properties of the type set, corresponding to an intensional definition. The definition could even be mixed, such as for a concrete supertype that automatically contains all the instances of its subtypes, but may also have explicit instances of its own.

Figure 3 shows an example of types. The set labels are there to make the graph readable and are not part of the model itself. Section 3.3 explains how to name elements.

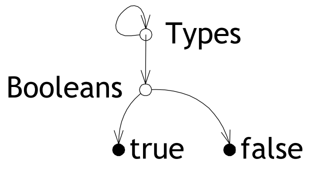

Figure 3

The *Booleans* type consists of two members, *true* and *false*. Since it is a type, it is an instance of the *Types* type, and therefore a member of that set. However, since *Types* is a type also, the set must be a member of itself to keep the model consistent, providing one reason why non-well-founded sets are needed.

This approach to typing, if employed, gives the metamodel a non-fixed layer architecture. The advantage is that types and instances are all part of the same model and can be treated uniformly, which is very important for semistructured data [1]. The disadvantages claimed in [9] include difficulties with formalizing the metamodel's semantics and the need to use hypersets, which can lead to Russell's paradox. However, those objections are irrelevant in view of our stated goals: since there are other metamodels that use a non-fixed layer architecture (for example RDF [7]), we must support it too to be able to integrate them.

3.3 Ordering and Duplicates

While the metamodel as presented above is complete, the models are meant for human consumption, and people often like to have data elements consistently ordered. It is possible to encode order directly with sets, so adding it as a primitive will not increase the expressive power of the metamodel but it will make ordering elements far more convenient and efficient.

The most common way to order elements is to impose a total order on the set, making a chain. This is not enough, though. When integrating data obtained from different sources, it is likely that we will have to merge ordered sets together. Since we expect the data to be semistructured, it is possible that the sets' members will not be comparable to each other. We must thus include partially ordered sets in our metamodel to satisfy the goal of smooth integration.

It is also useful to relax the set restriction and allow duplicates into the collections. In this way we obtain lists / n-tuples (when totally ordered) and bags (when unordered). The former are especially useful, since pairs can be interpreted as directed arcs and used to attach properties to elements.

Figure 4 demonstrates the use of pairs to give names to model elements. As usual, the labels are only informative and not part of the model. Order within n-tuples is indicated using tick marks, so the edge to the first member is adorned with a ', to the second member with '', etc.

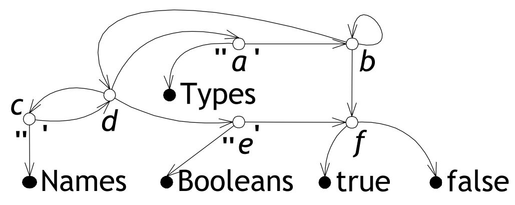

Figure 4

Figure 4 extends figure 3 by naming the type sets. The hypersets *b* and *f* are carried over. The pairs *a* and *e* associate a string atom to each of the two original type sets. By convention, the element being named is the first member, and the name is the second member. To indicate that these pairs are name properties we make them instances of the type set *d*, the set of name *relators*. Of course, this set has a name as well, indicated by the pair *c* that names the type set of which it's an instance. Finally, the set *d* is a type, which is indicated by its membership in *b* to avoid further complicating the sample model.

Table 1 summarizes the various kinds of collections in the metamodel, organized along the axes of ordering and duplication and stripped of the "hyper" prefix. We'll use the term *hypernest* (or *nest* for short) when referring to a collection that's part of a model but whose exact kind is irrelevant.

	Unordered	Partially Ordered	Totally Ordered
No Duplicates	set	poset	chain
Duplicates Allowed	bag	—	list / n-tuple

Table 1

3.4 Infinite Sets

There is still something missing from figure 4: the string atoms are not members of a type set. What would the *Strings* type set look like? According to our definition in section 3.2, it should contain all instances of the type—that is, all possible strings. Since the set of all possible strings is infinite (disregarding implementation limitations), it follows that the metamodel must allow for nests with an infinite number of members. Other examples of types with infinite extent are numbers, moments in time, and positions in space.

Notice that many of the infinite sets are thick, or even uncountable. They have a natural total ordering of their members, yet there exists no sequence that would enumerate them in the correct order. These infinite ordered sets require that ordering be a primitive feature that cannot be replaced by sequence index properties or linked lists, the only techniques considered in survey [8].

3.5 Membership Reification

Reification is the action of making some implicit object "real" within the model, and thus gaining the ability to refer to it. Most commonly used is reification of associations (statements), for example to make further statements about their origin. The metamodel naturally caters for this common case since associations are just nests (for example, the pairs in figure 4), and can already be referenced directly in other nests.

Reification of links is more rarely used. The intention is to reify the component objects of an association to make statements about them. In this metamodel, this is equivalent to reifying the membership of each element. For n-tuples, you can think of it as reifying the endpoints of the association arc.

The mechanism is as follows. Each model that requires reification includes a special membership nest. The contents of this nest are determined automatically by the system. It contains one container-member pair nest for each such pair present in the model. For bags and lists, each duplicate member gets its own reified membership pair. The membership nest is partially ordered: the pairs for each container match the order of its members, and pairs corresponding to different containers are unrelated.

Note that the membership nest is defined over the whole model, in which it is included. Consequently, if a model contains at least one non-empty nest, the membership nest will contain infinitely many membership reification pairs. The first pair reifies a member of the non-empty nest, which introduces a nest with two more members into the model. These members are reified, in turn, and produce two more pairs, etc. ad infinitum.

Of course, any implementation of this mechanism would have to use lazy evaluation to avoid infinite descent. Besides, link reification is rarely needed, and almost never past the first level. However, the metamodel must support arbitrary reification to be consistent.

4 Comparisons

This section draws some quick comparisons between the partially ordered hypernest metamodel introduced in this paper and a small group of other metamodels. The competitors were selected based on their ubiquity, expressive power, mind share, and possession of features similar to the hypernest metamodel.

4.1 Relational Databases

Although relational databases are not in the running in the race to model semistructured data, their popularity ensures that any metamodel aspiring to integrate data from every source will have to subsume the relational model. There is a straightforward mapping from relational tables to hypernests. Each table is viewed as a set, and each row as an n-tuple. Table definitions can be represented by properties attached directly to the table sets. With some extra information, foreign keys could be elided and mapped to associations between the appropriate rows. In any case, there is clearly no problem in assimilating the simple relational structure into the hypernest metamodel.

It is less clear how well SQL could be mapped to a hypernest query language; this is an area for future research.

4.2 Labeled Directed Graphs

Labeled directed graphs are the most popular way of modeling semistructured data. Some systems that use them are SHriMP [12] (Simple Hierarchical Multi-Perspective views), GOOD [6] (Graph-Oriented Object Database), OEM [10][1] (Object Exchange Model) and RDF [7] (Resource Description Framework). Although there are some variations in their application of this approach, this section will tackle the comparison at a very high level and list problems that are nearly universal.

All labeled directed graphs can be trivially transformed into a hypernest model. Each vertex is mapped to an atom; if it is labeled, the atom is given a name using the technique shown in section 3.3. Each edge is mapped to a nest. If the edge is directed, it's mapped to an n-tuple, and if it happens to be undirected, it's mapped to a bag. Any label can be associated with the edge nest using the standard technique, or it could be placed into an extra element of the n-tuple.

It is also true that any kind of information model can be represented using a directed graph, but a number of problems make them inappropriate for integrating semistructured data. First, they do not support universal references: it is impossible to refer to edges within the graph. If we want to reify an association (figure 5), or construct an n-ary association for n>2 (figure 6), we must create a new vertex to represent it.

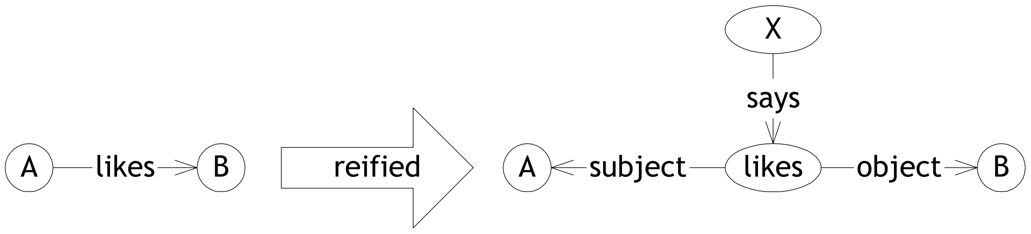

Figure 5

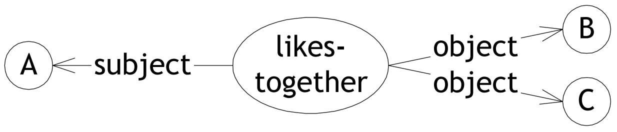

Figure 6

Similarly, if we want to state more than just the name of the links in a reified association, we have to reify it once again, so that the links become vertices. Not only is this annoying, but unless the whole model is at the same reification level we won't be able to write uniform queries across it. In other words, if we reify one association, we must reify them all, and rewrite all our queries to match the new structure. If we try to avoid the problem by keeping the original edges that were reified in the model, we would need to enforce consistent updates at all reification levels. The hypernest metamodel avoids these issues by ensuring that both associations and links to any depth can be referenced directly, without any changes to the model.

Another problem with labeled graphs is that the labels are an unnecessary primitive feature and cannot even be referenced. They are not first-class citizens of the model. When edge labels are used to indicate type, as is often done, any information about the type must be associated to the label in an ad-hoc manner, often by lexical matching on the label string. Labels also complicate query languages and often reduce their orthogonality.

Finally, most graph models have no intrinsic concept of edge order and must achieve this effect through indexing or linked lists. (TGraphs [4] are an interesting exception.)

4.3 Hypernode Model

The Hypernode Model is a nested-graph model employed by Hyperlog, as described in [11]. Each hypernode contains a non-well-founded set of hypernodes, and a separate set of directed (but unlabeled) edges. It is quite powerful and has a very well developed formal query and programming language. Programs and the type system are incorporated into the graph. It is the metamodel most similar to the one proposed in this paper.

Some differences with hypernests make it inappropriate for semistructured data integration, though. It is strongly typed and has no intrinsic concept of order. Worst of all, the edges cannot be referenced.

4.4 XML

XML [3] is the markup language currently sweeping the web. Its basic data model is an ordered tree with attributed nodes, though some specific vocabularies have extended these semantics. Due to its simplicity and the wide availability of generic parsers, it has become the language of choice for data exchange, and much semistructured data can be found in some dialect of it.

XML imports into hypernests very easily. Each element is a chain of other elements and string atoms. Attributes are attached to an element using, for example, 3-tuples. Specific vocabularies can include additional rules for creating cross-links through IDREF and including DTD, XSchema or RELAX-NG typing information.

Most any metamodel can integrate XML. The main advantage of hypernests is that the element chains are naturally ordered, which is important in some XML vocabularies.

5 Conclusions

By holding true to the twin design principles of extreme simplicity and universal references the partially ordered hypernests model presented in this paper achieves both elegance and expressive power. Some of its novel or rare features are:

- every model element can be directly referenced;
- implicit elements are automatically reified without upsetting the model structure;
- naturally supports nesting structures;
- order is intrinsic to the model and does not have to be emulated;
- allows infinite collections.

These features, combined with a loose type system, make for a very flexible metamodel that is perfectly suited to semistructured data integration. Further research work currently under way includes specifying a formal query language, producing formal bijective mappings to other metamodels, and implementing a model back-end as well as a reconfigurable visualization system.

References

1. Serge Abiteboul, Dan Suciu, and Peter Buneman: *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, October 1999, ISBN 155860622X.
2. Peter Aczel: *Non-Well-Founded Sets*. CSLI Publications, Stanford, 1988, ISBN 0937073229.
3. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, eds.: *Extensible Markup Language (XML) 1.0, 2nd ed*. W3C Recommendation, October 2000, *http://www.w3.org/TR/REC-xml.*
4. Jürgen Ebert and Angelika Franzke: *A Declarative Approach to Graph Based Modeling*. Technical Report 3-94, Universität Koblenz-Landau, 1994.
5. Robert L. Griffith: *Three Principles of Representation for Semantic Networks*. ACM Transactions on Database Systems, Vol. 7, No. 3, September 1982, pp. 417-442.
6. Marc Gyssens, Jan Paredaens, Jan Van den Bussche, and Dirk van Gucht: *A Graph-Oriented Object Database Model*. IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 4, 1994, pp. 572-586.
7. Ora Lassila and Ralph R. Swick: *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation, February 1999, *http://www.w3.org/TR/REC-rdf-syntax.*
8. Sergey Melnik and Stefan Decker: *A Layered Approach to Information Modeling and Interoperability on the Web.* Proceedings of the ECDL'00 Workshop on the Semantic Web, Lisbon, Portugal, September 2000.
9. Jeff Z. Pan and Ian Horrocks: *Metamodeling architecture of web ontology languages*. Proceedings of the Semantic Web Working Symposium, Stanford, July 2001, pp. 131-149.
10. Yannis Pakakonstantinou, Hector Garcia-Molina, Jennifer Widom: *Object Exchange Across Heterogeneous Information Sources*. Proceedings of ICDE'95, 1995.
11. Alexandra Poulovassilis and Stefan G. Hild: *Hyperlog: A Graph-Based System for Database Browsing, Querying, and Update*. IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No. 2, March/April 2001, pp. 316-332.
12. Margaret-Anne D. Storey: *A Cognitive Framework for Describing and Evaluating Software Exploration Tools*. PhD Thesis, School of Computing Science, Simon Fraser University, December 1998.