Extreme Markup Languages 2003

Montréal, Québec
August 4-8, 2003

Object mapping for markup semantics

David Dubin
University of Illinois

Abstract

The BECHAMEL system is a knowledge representation and inference environment for expressing and testing semantic rules and constraints for markup languages. Written in Prolog, the system provides predicates for processing the syntactic structures that emerge from a SGML/XML parser, defining object classes, instantiating object instances, assigning values to properties, and establishing relationships between or among object instances. BECHAMEL uses Prolog's built-in capabilities to derive inferences from these facts.

Part of the ongoing development of BECHAMEL involves experimenting with strategies for mapping syntactic relations to object relations and properties. This paper describes the current strategy, based on a blackboard model. Advantages of this approach include context free rules and the potential to exploit parallel processing for scalability. It has the drawback, however, of not permitting evidence to be described in ways people are likely to find natural or familiar. By using the current approach to produce formal accounts of the semantics of popular markup languages, we hope to learn a great deal about the ways markup syntax typically cues semantic relationships. That advance in our understanding will inform the development of more usable languages for object mapping.

Object mapping for markup semantics

Table of Contents

EXTREME MARKUP LANGUAGES 2003

Object mapping for markup semantics

David Dubin

§ Motivation and background

What is the meaning or interpretation that underlies an author's decision to tag an SGML or XML document instance in a particular way? On the surface this would seem a fairly straightforward question, since the generic identifiers for the elements have meaningful names: a `list` element is used to mark a list structure, a `para` element for a paragraph, and so on. But on reflection one recognizes a range of problems from subtle to crude. At the subtle end, it has been shown that authors and readers effortlessly draw inferences using facts and rules that aren't explicit in the markup itself [cmsmcq00] [cmsmcq02] [renear02]. At the crude end, we have the fact that authors often use markup language constructions in ways other than intended by the language designer. To discourage such practices by calling it "tag abuse" is not to dismiss the constructions as meaningless.

This paper reports some recent developments with the BECHAMEL Markup Semantics Project. BECHAMEL's logic programming environment [dubin03] includes a syntactic layer supporting operations similar to features of XSLT and an object layer supporting class definitions like those one can express in ontology languages for the Semantic Web. This paper is the first to discuss our work at the mapping level where instances in the object layer emerge from processing at the syntactic layer. Functionally, the mapping process is similar to the operation of an "extraction language" for Web documents [sahuguet01] or techniques used to "populate" topic maps [pepper02].

However, the BECHAMEL KR/inferencing system is not a Semantic Web application. Our goal is not to create a tool for transforming XML documents or extracting data from them. Rather, the aim is a working environment in which to express formal theories of what markup means to the designers and users of ordinary SGML and XML applications. The kind of practical benefits we hope to contribute include the ability for software to reliably report how many itemized lists a document contains, or to perform an operation on every element containing text in German. Realizing such benefits requires document processing software to be equipped with representations that make markup semantics explicit, and with deductive capabilities that can emulate the inferences that humans are able to draw.

§ Introduction

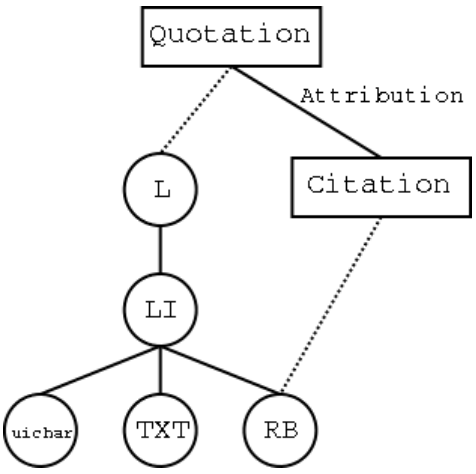
In SGML and XML-based markup languages, elements are often interpreted as representing content objects, and attributes as properties of those objects. The relationship between parent and child elements is typically interpreted as a containment or whole/part relationship. In practice, however, it may often be more reasonable to interpret elements as representing properties of objects or even relations among objects, rather than content objects themselves. The generic parent-child relationship between elements may signal any number of specific, named relationships that hold between the objects or properties to which the parent and child correspond.

```
<SUBSEC>
<HE>The Development of Structured Markup</HE>
<P>Document markup might arguably be considered part of <em>any</em>
communication system, including early writing, scribal publishing,
and printing, but with the emergence of digital text processing and
typesetting the use of markup became self-conscious and explicit
<cite src="coombs87"> <cite src="spring89">. </P>
```

In the example above, the subsection, paragraph, and emphasized phrase are each identifiable content objects. However, the heading does not seem to be part of the subsection in the same sense that the emphasized phrase is a part. It would appear to invite a distinction, such as that the heading *introduces* the section. Furthermore, the appearance of the `heading` element licenses the inference that its content represents the *title* of the section (a property of the section).

One way to interpret the `citation` elements is that they too represent content objects. But one could just as reasonably decide that these elements don't signal a new instance of some object called a citation;

Figure 1



Block quote tagged as a list

instead, the citation elements could be understood to represent relations that hold between the paragraph and sources (*i.e.*, the paragraph cites the source).

The lack of a *particular* semantic interpretation is, of course, a strength of SGML and XML-based languages, not a weakness [cover98]. But, as has been argued elsewhere, the lack of a formal framework with which to express or communicate these distinctions is problematic [cmsmcq00] [cmsmcq02] [renear02]. In other papers, we have described BECHAMEL, a knowledge representation and inference environment for expressing and testing semantic rules and constraints of the kind discussed here [dubin03]. One level of that system concerns mechanisms and rules for mapping syntactic markup structures to instances of objects, properties, and relations. The remainder of this paper discusses challenges in the development of the mapping level in BECHAMEL, and how those challenges are currently being addressed.

§ Mapping problems

The development of a mapping strategy requires answers to several questions:

1. In what form is the evidence that triggers a mapping?
2. What syntax-level functions are needed to select the nodes or subtrees needed for a mapping?
3. What kind of execution model is most appropriate?
4. How does one ensure that mapping computations scale to large data sets?
5. How can mapping rules be expressed in ways that will be useful and understandable to document designers and engineers?

We illustrate our approach to answering these questions using examples where the problems are crude: instances of tag abuse in a journal article [ndumu97] retrieved from the University of Illinois DLI Testbed [cole95] [schatz96] [cole00]. The article conforms to a DTD based on ISO 12083.

In the first example, elements for marking lists and list items have been to tag what is clearly a block quote:

```
In this paper, we adopt the loose definitions of an agent as
<L><LI><uichar class="default" entity="&lsquo;" unicode="&#x2018;">
</uichar>a self-contained program capable of controlling its own
decision-making and acting, based on its perception of its environment,
in pursuit of one or more objectives
<uichar class="default" entity="&rsquo;" unicode="&#x2019;"></uichar>
<RB RID="BR11" OCC="1"></RB></LI></L>
```

The article has several block quotes marked this way, and the evidence available for recognizing them is that they are all single element lists, where the list item begins with a left single quote character (the `uichar` element) rather than a bullet or alphanumeric item label. Figure 1 illustrates the desired mapping from element nodes to object instances: the element tagged as a list is semantically a block quotation, and the `RB` element maps to a citation object. In this case, the generic parent/child relationship between the `LI` node and the `RB` node represents an attribution relationship: the citation attributes the quotation.

The first stage of mapping this fragment is to parse the document into a familiar tree structure. The BECHAMEL system is written in Prolog and represents documents at the syntactic level using predicates such as the following:

```
node(N)                 N is an element node or text node.
gi(N,G)                 Node N has generic identifier G
content(C,T)            Text node N has atom T for content.
first_child(N,C)        Node C is the first child of node N.
parent(N,P)             Node N has node P as a parent.
nsib(N,S)               Node N's next sibling is node S.
atval(N,A,V)            Attribute A in node N has value V:
type_id(E,A)            Attribute A of element E is an id.
type_idref(E,A)         Attribute A of element E is idref.
def_value(E,A,V)        Attribute A of element E takes value
                        V as a default value.
```

Structures at the object level are represented using predicates such as these:

```
object(O)               O has been instantiated as an object.
obj_class(O,C)          Object O is of class C.
class(C)                C has been declared an object class.
subclass(Sub,Super)     Class Sub is a subclass of class
                        Super. Subclasses can take the same
                        properties and participate in the
                        same relations as their Superclasses.
property_of(C,P,T)      Property P of type T has been declared
                        for class C.
opv(O,P,V)              Object O has value V on property P:
relation(R,L)           Relation R can hold among objects of
                        classes listed in ordered list L.
construct(C,N,O)        Construct object O of class C modeling
                        node list N.
apply_property(O,P,V)   Object O takes value V on property P.
```

Object instances are connected via a "models" relation to the nodes from which they are mapped. The problem is how to execute predicates like `construct` and `apply_property` in an effective manner.

One could, of course, proceed as in a conventional programming language, by traversing the parse tree depth-first or breadth-first, and executing callback procedures on nodes of different types. However, understanding code written in that way requires one to consider the context in which a particular procedure or function is invoked, and that's is precisely the situation that BECHAMEL is intended to change. The mapping rules should be as explicit and declarative as possible, not require reverse engineering to be understood.

For that reason, we have adopted a blackboard model [winston84] as our first experimental approach to working at this level. The knowledge to accomplish mappings from syntactic structures to object, properties, and relations is expressed in production rules, each of which is able to examine the parse tree and the current state of the objects. When the conditions for the execution of a rule are met, it fires and makes some change, *e.g.*, instantiating a new object, destroying one, setting or altering the value of a property, or establishing a named relationship between two or more object instances.

One of the rules for realizing Figure 1's mapping is illustrated below. Its interpretation is as follows: if there exists an `L` node and its first child has no next sibling (*i.e.*, if something is tagged as a one-element list), and if the first child of the list item is a `uichar` that represents a left single quote, and if no quotation object has been mapped to this `L` node, then construct such an object instance. Other rules that participate in this mapping include one that discovers when the same `L` node is modeled by both a list object and a quotation object: when it fires, the list object is destroyed.

```
mrule :- node_exists(Lnode, 'L'),
   first_child(Lnode,Inode),
```

```
   not(nsib(Inode,_)),
   first_child(Inode,Cnode),
   gi(Cnode,uichar),
   atval(Cnode,entity,Char),
   char(lsquo, Char),
   not(exists(_,quotation,[Lnode])),
   construct(quotation,[Lnode],_).
```

When no more mapping rules can fire, the blackboard predicate reports that the mapping is complete. This approach mapping via independent production rules not only allows the rules to be kept context free and easy to manage, it also invites the application of parallel processing to address the problem of scalability. There are drawbacks to this approach, however. Because one rule can undo a change made by another rule, careless rule definition can easily lead to an infinite loop. And as will be illustrated in the next example, even a fairly independent set of mapping rules can be difficult to interpret.

Not only does our illustrative document exhibit quotations tagged as lists, there are also lists that aren't tagged. A fragment of one such example is shown below:

```
which society will have to deal with through
various pieces of legislation. They include</para>
<para><uichar class="Symbol" entity="&bull;" isocode="&#183;"
unicode="&#x2022;"></uichar>
privacy: how do you ensure your agents maintain your privacy when acting on
your behalf?</para>
```

This sequence of paragraphs, each beginning with a bullet character, is clearly an itemized list. However, no syntactic element encloses the list items. The needed mapping from the syntactic to the object layer is illustrated in figure 2: the `P` nodes that begin with a bullet character must be mapped to list item objects, and pairs of such objects mapped from sibling paragraph nodes need a "follows" relation between them. These list item objects stand in part/whole relation to a list object that corresponds to no syntactic element. That list takes a value of `"Itemized"` for its `style` property.

A number of independent mapping rules are needed to realize the mapping shown in Figure 2. Two such rules are shown below. The first discovers paragraphs that begin with a bullet character, and maps them individually to list items that are part of lists. Its interpretation is as follows: if a `P` node exists whose first child is a `uichar` representing a bullet character, and no list item has been mapped to that paragraph node, construct such an object and let it stand in part/whole relation to a new list object. A separate rule (not shown) discovers and merges list objects that contain list items that follow on one another.

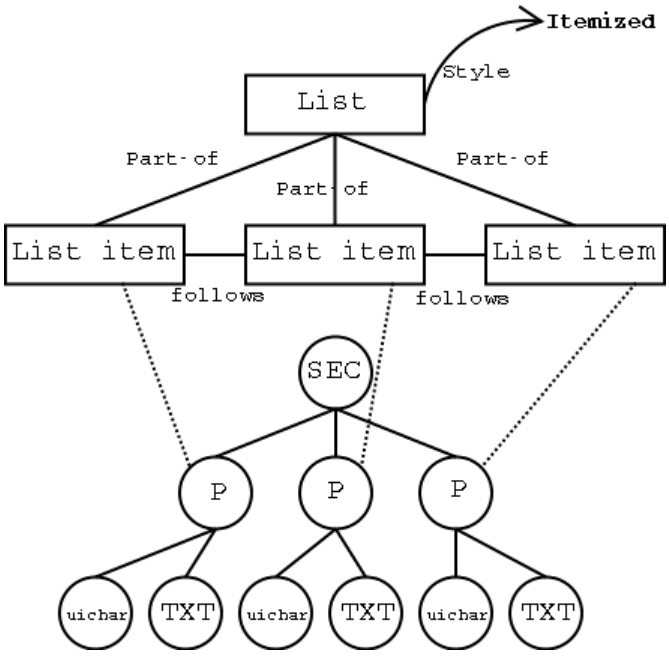
```
mrule :- node_exists(Pnode, 'P'),
   first_child(Pnode, Bnode),
   gi(Bnode, uichar),
   atval(Bnode,entity,Char),
   char(bull,Char),
   not(exists(_,listitem,[Pnode])),
   construct(listitem,[Pnode],Listitem),
   construct(list,[Pnode],List),
   apply_relation(part_of,[Listitem,List]).
```

The second rule discovers list item objects that have been mapped from sibling nodes and establishes a "follows" relation between them, if no such relation has yet been created:

```
mrule :- exists(Item1, listitem, [Node1]),
   exists(Item2, listitem, [Node2]),
   nsib(Node1, Node2),
   not(relation_applies([follows,Item2,Item1])),
   apply_relation(follows,[Item2,Item1]).
```

These rules successfully discover the sequence of paragraphs that need to be mapped to a list object. But the bottom-up, distributed manner in which the mapping is accomplished does not permit the rules to easily communicate their semantics to a human reader. It would be better to have a way of simply declaring that a sequence of elements meeting particular conditions must be mapped to an object of a particular class, having particular properties, etc. A notation incorporating, similar to, or derived from XPath might be suitable, given that the current mapping rule syntax already shares axes and some predicates with XPath. However, it's not clear yet whether a single context node will be enough for the kinds of deictic expressions the mapping rules will need to include. Ultimately, the definition of a more

Figure 2



List object must be inferred

natural language for expressing mappings must be informed by extensive surveys of and experimentation with markup languages that are in common use. The current blackboard-based approach seems robust enough to support research at the next stage: projects aimed at producing formal accounts of the semantics of popular markup languages. Results of those experiments will speak to the question of how the mapping language can be made more intuitive.

Bibliography

[cmsmcq00] Sperberg-McQueen, C. M., Huitfeldt, C., and Renear, A. “Meaning and interpretation of markup”. *Markup Languages: Theory and Practice* 2, 3 (2000), 215-234.

[cmsmcq02] Sperberg-McQueen, C. M., Dubin, D., Huitfeldt, C., and Renear, A. Drawing inferences on the basis of markup. In *Proceedings of Extreme Markup Languages 2002* (Montreal, Canada, August 2002), B. T. Usdin and S. R. Newcomb, eds.

[cole00] Cole, T. W., Mischo, W. H., Ferrer, R., and Habing, T. G. Using XML, XSLT, and CSS in a digital library. In *ASIS 2000: Proceedings of the 63rd ASIS Annual Meeting* (Medford, NJ, 2000), D. H. Kraft, ed., American Society for Information Science, Information Today, pp. 430-439.

[cole95] Cole, T., and Kazmer, M. “SGML as a component of the digital library”. *Library High Tech* 13, 4 (1995), 75-90.

[cover98] Cover, R. XML and semantic transparency. Technology report, Cover Pages, 1998. Published on the Worldwide Web at **http://www.oasis-open.org/cover/xmlAndSemantics.html**.

[dubin03] Dubin, D., Sperberg-McQueen, C. M., Renear, A., and Huitfeldt, C. “A logic programming environment for document semantics and inference”. *Journal of Literary and Linguistic Computing* 18, 1 (April 2003), 39-47.

[ndumu97] Ndumu, D. T., and Nwana, H. S. Research and development challenges for agent-based systems. In *IEEE Proceedings on Software Engineering* 144, 1 (1997), 2-10.

[pepper02] Pepper, S. and Garshol, L. M. The XML Papers: Lessons on Applying Topic Maps. In *Proceedings of XML 2002* (2002).

[renear02] Renear, A., Dubin, D., Sperberg-McQueen, C. M., and Huitfeldt, C. Towards a semantics for XML markup. In *Proceedings of the 2002 ACM Symposium on Document Engineering* (McLean, VA, November 2002), R. Furuta, J. I. Maletic, and E. Munson, eds., Association for Computing Machinery, pp. 119-126.

[sahuguet01] Sahuguet, A. and Azavant, F. "Building intelligent Web applications using lightweight wrappers". *Data and Knowledge Engineering* 36, 3 (2001), 283-316.

[schatz96] Schatz, B., Mischo, W. H., Cole, T. W., Hardin, J. B., Bishop, A. P., and Chen, H. "Federating diverse collections of scientific literature". *Computer* 29, (May 1996), 28-36.

[winston84] Winston, P. H. *Artificial Intelligence*. Addison-Wesley, Reading, MA, 1984, ch. 5.

The Author

David Dubin
University of Illinois, Graduate School of Library and Information Science
501 E. Daniel Street
Champaign
IL
61820
USA
ddubin@uiuc.edu
tel: 217-244-3275
fax: 217-244-3302

David Dubin is a senior research scientist on the staff of the Information Systems Research Lab at the University of Illinois Graduate School of Library and Information Science. He is a member of the Electronic Publishing Research Group.

Extreme Markup Languages 2003
Montréal, Québec, August 4-8, 2003
This paper was formatted from XML source via XSL
by Mulberry Technologies, Inc.