Naming in XML Documents

Ramon Lawrence

IDEA Lab, Department of Computer Science, University of Iowa
Iowa City, IA, USA, 52242
ramon-lawrence@uiowa.edu
http://www.cs.uiowa.edu/~rlawrenc/

Abstract. XML is now an established standard for data communication and representation. There has been considerable work on XML querying, modeling, and type definition. However, one of the most important aspects of XML, standardized tag naming for conveying semantics, has been almost ignored by the research community. This paper argues that the naming aspects of XML are important to consider and presents a naming methodology for XML tags that captures increased context information. Using semantic tag names opens up the possibility of semantic querying of XML documents, which simplifies query formulation by reducing the reliance on path expressions. A semantic query facility allows XML documents with similar semantics, but organized using different DTDs, to be queried without modifying the original query formulation. Finally, we demonstrate an algorithm for converting semantic queries to structural queries by disambiguating incomplete path expressions.

1 Introduction

As XML becomes a *de facto* standard for data communication and representation, research has focused on XML querying, modeling, and type definition. However, there has been little focus on the naming challenge of XML. Without the ability to define unique tag sets for XML documents, XML as a modeling language is very similar to the hierarchical model [11] of years past. XML provides the ability to model and query semistructured data. Its usefulness as an integration and communication tool is founded on the ability to define tags with standardized meaning. Consequently, **semantics**, not just structure, may be communicated.

Unfortunately, name construction for XML tags has devolved into the often poor naming characteristics used in the database community where it is not uncommon to build names for attributes and relations by various combinations of abbreviation, concatenation, and hyphenation. The result are names that although concise, are often ambiguous, especially when extracted from surrounding context. Exploiting algorithmic naming has beneficial properties for documentation, human readability, and simplifying querying. Our contribution is a semantic naming methodology that can be applied to existing XML DTDs. After enumerating some of the challenges of XML modeling and naming in Section 2, a naming model is presented in Section 3. A major benefit of the model

is that the tag names become *context independent* in that they encode sufficient context information that their semantic meaning can be determined regardless of surrounding context. Further, a major advantage of exploiting the naming model is that semantic querying can be performed on XML DTDs. Semantic querying provides the ability to re-arrange XML documents and DTDs without re-writing queries, and simplifies XML querying by reducing the reliance on path expressions. Section 4 overviews an algorithm that maps a semantic query to a structural query on a particular DTD. The algorithm uses the naming of DTD elements to determine a complete path expression. The paper closes with future work and conclusions.

2 Modeling Challenges in XML

Despite its wide-spread adoption, XML is actually not a very powerful modeling language. An XML document has a hierarchical structure based on the nesting of elements. Without considering the use of ID/IDREF linking within a document or XLink [10] or XPath [7] between documents, the DTD for an XML document is very similar to the original hierarchical model [11] in that it can only model tree structured data.

A working example used throughout this paper is based on the automobile XML DTDs originally proposed in [16]. For this paper, the two DTDs from separate organizations will be combined into one source, while preserving most of the original elements and semantics. We only examine querying from a single source, although the work may be extended to handle querying across sources (XML documents)[1]. An ER-model for the combined information is given in Figure 1.

Construction of a DTD modeling the information in the ER diagram requires a decision on a hierarchical ordering of the information. Thus, there are multiple possible DTDs that can be used to encode the exact same semantic information present in the ER model. Each of these DTDs can be considered as a view of the original model. Two possible DTDs are given in Figure 2. DTD1 selects a hierarchical representation from the manufacturer perspective listing manufacturer, then model, then vehicle. DTD2 is from the vendor perspective and lists vendors, then vehicles with their associated manufacturer and model information. Note that there are both minor naming differences in the DTDs (`option` versus `op-name`) and structural differences besides nesting. For example, manufacturer is represented as a composite object in DTD1, and as a simple attribute in DTD2. In most cases DTD2 is a suboptimal choice because of the duplication of manufacturer and model information. However, DTD2 is just as valid as DTD1.

After the hierarchical ordering of elements in a DTD is selected, the nesting of XML elements has ambiguous semantics. A nesting relationship between two XML elements may encode: a specialization/generalization (IS-A) relationship,

[1] The two XML documents can be considered as views of a global schema as given by the ER diagram.

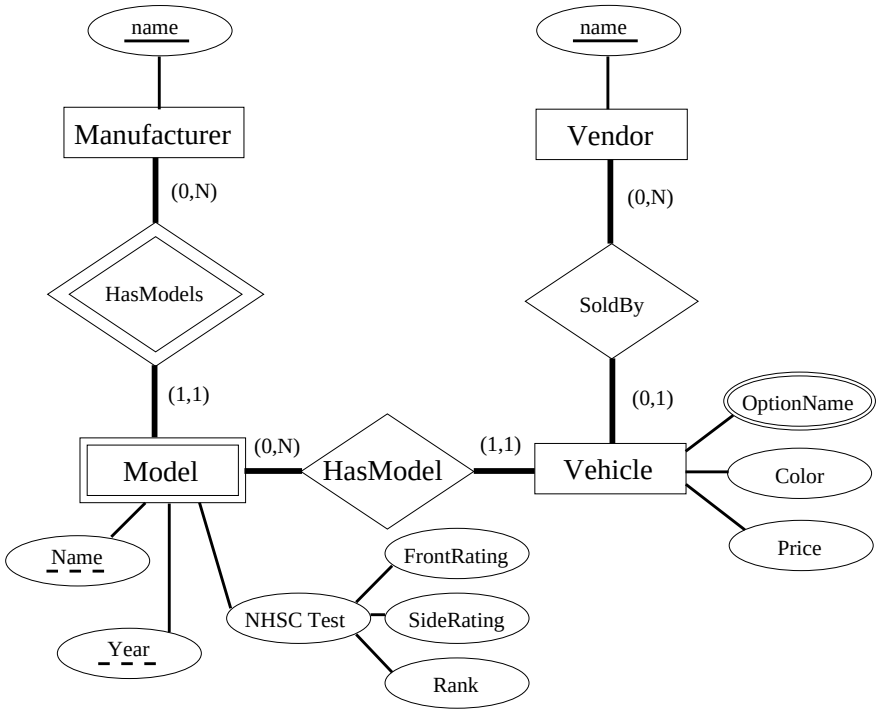

Fig. 1. ER Diagram for XML Example

DTD1

```
<!ELEMENT list-manufacturer (manufacturer+)>
<!ELEMENT manufacturer (mn-name, model+)>
<!ELEMENT mn-name #PCDATA>
<!ELEMENT model (mo-name, year, front-rating
                 side-rating, rank, vehicle+)>
<!ELEMENT mo-name #PCDATA>
<!ELEMENT year #PCDATA>
<!ELEMENT front-rating #PCDATA>
<!ELEMENT side-rating #PCDATA>
<!ELEMENT rank #PCDATA>
<!ELEMENT vehicle (color, price, vendorName, option+)>
<!ELEMENT color #PCDATA>
<!ELEMENT price #PCDATA>
<!ELEMENT vendorName #PCDATA>
<!ELEMENT option #PCDATA>
```

DTD2

```
<!ELEMENT list-vendor (vendor+)>
<!ELEMENT vendor (vendorName, vehicle+)>
<!ELEMENT vendorName #PCDATA>
<!ELEMENT vehicle (color, price, op-name,
                   mn-name, model)>
<!ELEMENT color #PCDATA>
<!ELEMENT price #PCDATA>
<!ELEMENT op-name #PCDATA>
<!ELEMENT mn-name #PCDATA>
<!ELEMENT model (mo-name, year, front-rating
                 side-rating, rank)>
<!ELEMENT mo-name #PCDATA>
<!ELEMENT year #PCDATA>
<!ELEMENT front-rating #PCDATA>
<!ELEMENT side-rating #PCDATA>
<!ELEMENT rank #PCDATA>
```

Fig. 2. Two different DTDs for the same ER Model

a meronym/holonym (PART-OF/HAS-A) relationship, an ordering/grouping of elements, or a general relationship (join). Thus, given only the structure of an XML document without considering the tag name semantics, it is impossible to determine the relationship between nested elements. Further, structural ambiguity is present when attributes are used to encode information instead of just elements. For this paper, only elements are used to encode information.

XML query languages [4] rely on the structure of the document (DTD) for querying. XML queries are specified by providing path expressions that navigate through the document to retrieve the appropriate elements. A parallel can be drawn between path expressions and navigating through records in hierarchical databases. Although path expressions can be given declaratively in SQL syntax, the formulation of the query is so intertwined with document structure that structural transparency is not achieved. Consider the query: *"Return the manufacturer name and vehicle price for all vehicles with price < 30,000, and the vehicle model is in the top 10 for safety tests".* [16] The query specified in the Lorel [2] query language for DTD1 (Q_1) and DTD2 (Q_2) is given below:

select M.mn-name, M.model.vehicle.price
from list-manuf.manufacturer M
where M.model.rank $\leq$ 10 **and** M.model.vehicle.price $<$ 30000 (Q_1)

select V.manufacturer.mn-name, V.price
from list-vehicles.vehicle V
where V.manufacturer.model.rank $\leq$ 10 **and** V.price $<$ 30000 (Q_2)

Notice that despite both DTDs modeling exactly the same information, the hierarchical nature of XML causes considerably different queries to obtain an identical answer. Although it is possible to create a single query that functions over both XML DTDs (see Q_3), the complexity of such a query is clearly undesirable. The addition of the selection operator ("|") and the Kleene closure operator ("*") unnecessarily complicate the query formulation and add to possible interpretation ambiguity for the user.

select T._*.mn-name, T._*.vehicle.price
from (list-vehicles | list-manufacturer).(vendor | manufacturer) T
where T._*.model.rank $\leq$ 10 **and** T._*.vehicle.price $<$ 30000 (Q_3)

Further, the naming in each DTD demonstrates some undesirable properties. In both cases, an outer tag name (`list-vehicles` or `list-manufacturers`) must be defined and used in querying to group elements in the document. Second, naming for a common concept such as a name of an entity is not systematic. For instance, `mn-name` and `mo-name` are concatenated abbreviations for manufacturer name and model name. Given these two tag names out-of-context of

the DTD or original document[2], it is extremely difficult to determine their semantics. The `vendor` tag stores a name attribute, but that information can only be determined by examining the actual values in the XML document and using domain knowledge to recognize them as names.

From this brief overview, it should be apparent that modeling and naming in XML are complicated issues. The goal of this work is to present a systematic naming mechanism for XML DTDs. We then define a canonical semi-structured model that allows queries to be posed on semantically equivalent DTDs without query re-formulation. Given a particular DTD and a semantic query, a mapping algorithm translates the semantic query into a structural query for the DTD.

3 XML Naming Model

Every element in a DTD is associated with a tag name. For example, some of the tag names in DTD1 (see Figure 2) are: {`list-manufacturer, manufacturer, mn-name, year, ...`}. Let N_x be the tag name set for a given DTD x. Each tag name $tn \in N_x$ is selected during the design of the DTD to represent the semantics of the element it is naming. Typically, an element's semantics can be adequately captured by selecting a term t from some ontology D. This term t "captures" the appropriate semantics based on its standard usage in the language. For example, the term `manufacturer` has a defined meaning that can be determined using a database of lexical relations such as WordNet [17].

The complexity in naming arises when a single term t cannot adequately convey the semantics of an element. In this case, a set of terms T can be used to provide the additional semantic information. Each $t_i \in T$ has a unique definition in the ontology D. Given a set of terms T the challenge is to combine these terms appropriately to preserve the intended semantics of the element. Common techniques include selecting the most representative term[3] t_i of T and discarding the other terms, or selecting some set $S \subseteq T$ and applying to each $t_i \in S$ the operators of abbreviation and concatenation. Concatenation may be performed with or without using separators such as underscore ("_"), hypen ("-"), or changes in capitalization.

Naming schemes are already used in the naming of classes, variables, and methods in programming languages. We present a naming scheme for XML elements that preserves semantic information to an arbitrary degree of accuracy.[4] Further, the systematic naming scheme allows for the construction of unique tag names as required by XML.

[2] Determining semantics of tag elements out-of-context is not a rare occurrence. Consider an XML query that extracts only the manufacturer name and model name from the original document to produce a new document.

[3] The most representative term is determined by the user.

[4] Increased accuracy is achieved by using more terms.

3.1 Constructing XML Tag Names

Given a set of concepts C to be modeled in an DTD x, each concept $c_i \in C$ must be associated with a tag name $tn_i \in N_x$. The structure of each tag name tn_i is as follows:

$$tag\_name ::= [CT\_Term] \mid [CT\_Term].PN$$
$$CT\_Term ::= CT \mid CT\ ;CT\_Term \mid CT\ ,CT\_Term$$
$$CT ::= < context\ term >$$
$$PN ::= < property\ name >$$

The *context term(s)* and *property name* are selected from some ontology D. We will not discuss the actual construction of the ontology. The ontology may be considered as a "light-weight ontology"[5] that defines a concept hierarchy using IS-A and HAS-A relationships and provides terms and definitions for the modeled concepts. A database such as WordNet [17] may be used, but in small domains, custom-built concept hierarchies are also practical. For our purposes, the ontology provides terms that have an *accepted meaning* to a human user.

A tag name consists of one or more context terms and an optional property name term. Context terms are related by either IS-A or HAS-A relationships (represented using "," or ";" respectively). In general, tag names are not assigned based on the nesting of elements, but rather the semantics of the element they represent. The following rules dictate the naming for a concept c_i to produce a tag name tn_i:

- The XML document root element (view root) is always assigned a name V.
- Every tag name must have at least one context term.
- A tag name for a composite element has only context terms.
- An atomic element name (#PCDATA) or attribute name must always have a property name that represents the semantics of the attribute. The tag name must also contain all the context terms from its associated entity.
- A weak entity contains the context terms of its parent entity.
- A non-weak entity is named based on its semantics independent of its structural representation and nesting in the document.
- Additional context terms can be added for clarity, but then they must always be used consistently.

The above rules dictate a naming methodology for the construction of tn_i, but not an exact algorithm. There are two major degrees of freedom that are not restricted by the naming method:

- The selection of the ontology D, and the individual terms chosen from D to represent each concept.
- The number of context terms chosen for a tag name. Additional terms can be used to capture various levels of semantic precision.

[5] The term "light-weight ontology" is used to differentiate between knowledge base ontologies which in addition to a concept hierarchy also define a rule-base to relate concepts.

It is these two degrees of freedom that complicate naming. The first dictates that it may be impossible to create a single ontology D to name all concepts precisely in any domain. The English language is full of ambiguity and multiple terms with similar meaning. The tag names chosen may differ between designers if different ontological terms are chosen. That is, two different designers may select two different term sets E_1, E_2 where $E_1 \subset D$, $E_2 \subset D$, and $E_1 \neq E_2$. Applying the name construction rules will result in two tag names tn_1 and tn_2 where $tn_1 \neq tn_2$. This degree of freedom can be limited in a given domain by reducing ontology size and the choice of terms.

The second degree of freedom relates to the fact that there is no "one correct way" to capture semantics in a name. Any name we chose will miss some of the related semantics. When assigning names, a designer must make the trade-off between adding more context terms to improve the semantic description and increasing the size of the tag name. One example of this is the decision to introduce the term `NHSC Test` into DTD1 (see Figure 3) to provide more semantics to the model rating statistics. This is an optional addition that improves human readability.

The naming methodology, and later the semantic query facility built on top of it, is not significantly hindered by these two degrees of freedom. One of the reasons for this is the names are defined for a given domain. Given prior agreement on naming in the domain, conflicts may be minimized.[6] Note however that it is the existence of these degrees of freedom that make integration of structures and ontologies between domains so complex. We will not consider integration and querying between domains in this work.

Assigning semantic names to DTD1 gives the resulting DTD in Figure 3. DTD2 is similar. Note that since tag names in an XML document cannot contain the characters `"["`,`"]"`, `";"`, and `","` the semantic names when represented in a DTD have been translated such that: `"["` and `"]"` are removed, and the relationship characters (`";"`, `","`) are substituted with the valid characters (`"-"`, `"_"`) respectively. Finally, the `"."` used to separate the property name is replaced with `"--"`, and spaces within a term are removed. For clarity of presentation in the text, the tag names use the original format presented.

An important difference between regular DTDs and those with semantic naming is that the semantics of every tag name can be determined without examining the nesting inherent in the DTD. That is, given a tag name, we can uniquely determine its semantics. Further, inconsistent naming issues such as the tag names for manufacturer name, model name, and vendor name are avoided. Thus, this methodology contributes by reducing the ambiguity inherent in XML naming by systematically combining additional context terms.

Constructing semantic names for elements increases the string length of the tag name without modifying the document structure. This may be an issue when typing queries or even in the physical size of the resulting XML documents.[7] These issues are discussed in a following section.

[6] Effectively, the term sets E_i for each concept c_i are agreed upon *a priori*.
[7] XML compression algorithms exist to reduce document sizes.

```
<!ELEMENT V (Manufacturer+)>
<!ELEMENT Manufacturer (Manufacturer--Name, Manufacturer-Model+)>
<!ELEMENT Manufacturer--Name #PCDATA>
<!ELEMENT Manufacturer-Model (Manufacturer-Model--Name, Manufacturer-Model--Year,
    Manufacturer-Model-NHSCTest--FrontRating, Manufacturer-Model-NHSCTest--SideRating,
    Manufacturer-Model-NHSCTest--Rank, Vehicle+>
<!ELEMENT Manufacturer-Model--Name #PCDATA>
<!ELEMENT Manufacturer-Model--Year #PCDATA>
<!ELEMENT Manufacturer-Model-NHSCTest--FrontRating #PCDATA>
<!ELEMENT Manufacturer-Model-NHSCTest--SideRating #PCDATA>
<!ELEMENT Manufacturer-Model-NHSCTest--Rank #PCDATA>
<!ELEMENT Vehicle (Vehicle--Color, Vehicle--Price, Vendor--Name, Vehicle-Option--Name+)>
<!ELEMENT Vehicle--Color #PCDATA>
<!ELEMENT Vehicle--Price #PCDATA>
<!ELEMENT Vendor--Name #PCDATA>
<!ELEMENT Vehicle-Option--Name #PCDATA>
```

Fig. 3. DTD1 with Semantic Naming

4 Semantic Querying

Although systematic naming captures increased semantics in XML names, many users would see the increased size of the XML document and tag size as sufficiently negative to avoid systematic naming. However, by exploiting systematic naming it is possible to define a semantic query facility that reduces the negative features of longer semantic names and the need for path expressions in XML querying.

A *path expression* in XML querying is a sequence of edge labels starting from the root that enumerates a set of nodes. Complex path expressions may contain regular expressions both on the edge label strings or on the path itself. Operators such as selection ("|") and Kleene closure ("*") can be used.

Path expressions often complicate querying by focusing on the structure of the document rather than realizing the semantics of the query. The frustration users encounter with joins in the relational model is magnified considerably with XML querying where their required awareness of the structure of the document is increased further. In effect, the structure of the XML document **impedes** user querying as simple queries may be unnecessarily complicated by path expressions. Even graphical query languages such as XML-GL [8] focus on structural querying of documents rather than semantic data extraction. Our query language uses the semantic names assigned during systematic naming to produce a canonical query tree that is later mapped to the exact structure of a specific XML document.

Although we exploit the previously presented semantic naming system, semantic querying can be performed with any naming system as long as it has two important properties:

- A tag (semantic) name must be uniquely identifiable by a human user without presenting surrounding structure or names.

– The same tag name must be used consistently to represent the identical concept even though the concept may be organized using different structural representations in various DTDs.

The combination of these two naming features allows the construction of an algorithm that dynamically searches structural representations for particular concepts by name.

An XML document can be modeled as an edge-labeled tree [1] where the label of each edge is the tag name. We will use an edge-labeled tree to model a DTD. Formally, a DTD $x = (N_x, L_x, E_x, v_x)$ where N_x is a set of nodes, L_x is a set of edge labels, $E_x \subset N_x \times N_x \times L_x$, and v_x is the root node of the tree. For our purposes, we will initially consider DTDs with no ID or IDREF attributes (i.e. those XML documents that can be modeled as a tree). The two example DTDs modeled as trees are given in Figure 4 with leaf nodes represented as empty circles, and interior nodes as filled circles.

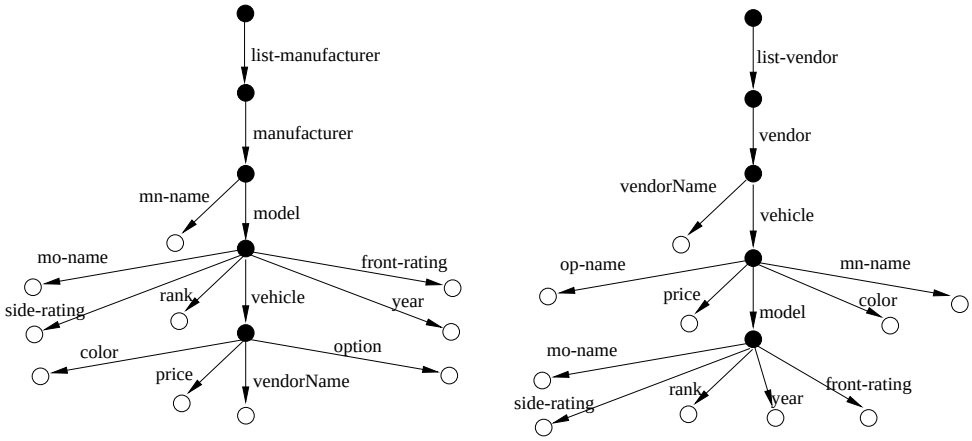

Fig. 4. Tree representations of DTDs

By examining the tree structures, it is apparent why the two DTDs require different XML queries: path expressions denote paths from the root to target nodes. Hence, re-organizing the tree changes the path expressions. Now, consider a canonical tree structure of the same data given in Figure 5. This canonical tree structure is built by extracting the semantic tag names from the DTDs and nesting them according to the term nesting in each tag. For the following discussion, the term semantic name is used interchangeably for tag name.

Definition 1 *Define a* **context view** $CV = (N_v, L_v, E_v, v_0)$ *as a rooted, edge-labeled graph that consists of a set of nodes* N_v*, a set of edge labels* L_v*, and a set of edges* $E_v \subset N_v \times N_v \times L_v$*, and a distinguished root* $v_0 \in N_v$*. CV can be built algorithmically given DTD x by using the tag set* N_x*. The root node* v_0 *is referred to as V or the* view root *in the text.*

Given the canonical form of the context view (see Figure 5), users specify their query on the context view. Note that to simplify presentation the edge label only displays the *term at depth*. The full edge label is constructed by constructing the path expression from root to node and concatenating the edge labels. For example, the tag name `Year` under `Model` has an edge label of `[Manufacturer;Model].Year`.

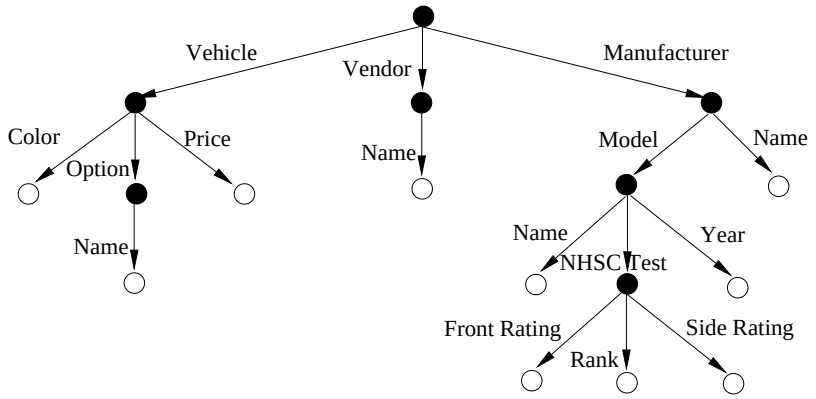

Fig. 5. Context view

The query procedure is straight-forward:

- The user is given the context view $CV = (N_v, L_v, E_v, v_0)$ in tree form.
- The user selects a set of edges $E' \subseteq E_v$ for inclusion into their query either as conditions (WHERE) or selection clauses. Each edge $e = (u, v, tn) \in E'$ has a unique tag name tn.
- Given the actual DTD $x = (N_x, L_x, E_x, v_x)$ and the edge set E' selected by the user, the system attempts to match each edge $e = (u, v, tn) \in E'$ with an edge $f = (u', v', tn') \in E_x$ where $tn = tn'$.
- This matching produces a set of edges $E'_x \subseteq E_x$. Each edge $f \in E_x$ has a unique path from the root. The path replaces the tag name to produce a structural query.

The example query posed on the context view is in Figure 6. The canonical query does not need a `FROM` clause in this case. In general, the introduction of a `FROM` clause is not required if the query can be answered using a single query iterator variable. Notice how path expressions are removed from the query. Thus, the query will be correct even if the DTD changes.

To map from a canonical query to an XML query requires that each element referenced in the query be converted into the appropriate path expression for the DTD. For a tree-structured DTD with no internal linking, this mapping can be easily achieved by performing a breadth-first search from the DTD root to

select [Manufacturer].Name, [Vehicle].Price
where [Vehicle].Price $<$ 30000 **and** [Manufacturer;Model;NHSC Test].Rank $\leq$ 10 (Q_4)

Fig. 6. Semantic Query on Context View

the appropriate tag names. Since the DTD is a tree structure, there is a unique path from root to tag name for each tag, and there is a single path that relates all concepts.

4.1 Advanced Querying

Although simple XML documents can be modeled as tree structures, linking between XML elements is possible using ID and IDREF attributes. The ID attribute is used to assign a unique key to an element, and IDREF is used to reference an ID of another element. This allows data to be hierarchically organized in the base XML document, but also include cross-links between elements to model other relationships in which they participate. ID and IDREF attributes can be easily modeled in the context view by introducing ID and IDREF nodes. Consider modifying DTD2 to produce DTD3. DTD3 contains IDs for models so that they may be listed separately and eliminate duplication. The modification to the DTD results in an attribute ID added to model element, and attribute IDREF added to vehicle element (see Figure 7). The context view now becomes a canonical graph structure with directed links between ID and IDREF nodes (see Figure 8).

There are several new challenges when introducing links via ID/IDREF. The first challenge is naming the IDREF values appropriately. In the example, we could not use the tag name `Model` to refer to both the model element itself, and the sub-element introduced under vehicle to provide a link to the actual model element. In our system, the standard name for IDREF attributes is the name of the parent element (`[Vehicle]`) plus the name of the linked element (`[Manufacturer;Model]`) plus the generic term `Id`. Thus, the name for the IDREF attribute for vehicle is `[Vehicle;Manufacturer;Model].Id`. This guarantees unique IDREF attribute naming.

The second challenge is handling path ambiguity now that we have a canonical graph. For example, querying for `[Manufacturer;Model].Name` now has two potential paths:

- A semantically direct path of `[Manufacturer;Model].Name` that corresponds to model names whether or not there is a vehicle in the XML document with that model. The particular path for DTD3 is `V.Manufacturer-Model.Manufacturer-Model--Name`.
- A semantically indirect path through `[Vehicle]` with the constraint that we only want model names if there is a vehicle in the XML document with that

```
<!ELEMENT V (Vendor+, Manufactuer-Model+)>
<!ELEMENT Vendor (Vendor--Name, Vehicle+)>
<!ELEMENT Vendor--Name #PCDATA>
<!ELEMENT Vehicle (Vehicle--Color, Vehicle--Price, Vehicle-Option--Name+,
                   Vehicle-Manufacturer-Model)>
<!ELEMENT Vehicle--Color #PCDATA>
<!ELEMENT Vehicle--Price #PCDATA>
<!ELEMENT Vehicle-Option--Name #PCDATA>
<!ELEMENT Vehicle-Manufacturer-Model #EMPTY>
   <!ATTLIST Vehicle-Manufacturer-Model Vehicle-Manufactuer-Model--Id IDREF #IMPLIED>
<!ELEMENT Manufacturer-Model (Manufacturer--Name,Manufacturer-Model--Name,
      Manufacturer-Model--Year, Manufacturer-Model-NHSCTest--FrontRating,
      Manufacturer-Model-NHSCTest--SideRating, Manufacturer-Model-NHSCTest--Rank)>
   <!ATTLIST Manufacturer-Model Manufacturer-Model--Id ID #REQUIRED>
<!ELEMENT Manufacturer--Name #PCDATA>
<!ELEMENT Manufacturer-Model--Name #PCDATA>
<!ELEMENT Manufacturer-Model--Year #PCDATA>
<!ELEMENTManufacturer-Model-NHSCTest--FrontRating #PCDATA>
<!ELEMENT Manufacturer-Model-NHSCTest--SideRating #PCDATA>
<!ELEMENT Manufacturer-Model-NHSCTest--Rank #PCDATA>
```

Fig. 7. DTD with ID/IDREF Attributes

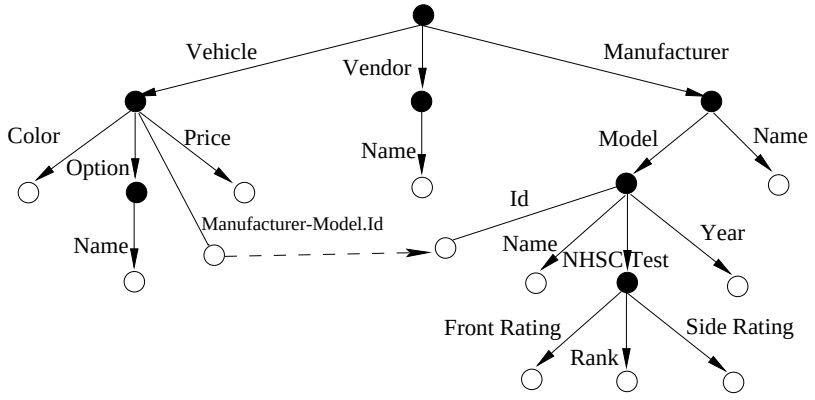

Fig. 8. Context View with Linking

model. This path is `[Vehicle;Manufacturer;Model].Name`, and the physical path in DTD3 is `V.Vendor.Vehicle.Vehicle-Manufacturer-Model` then linked to `V.Manufacturer-Model.Name` using ID/IDREF.

The first semantic path is a direct path in the canonical graph. The second semantic path represents a derived tag name by linking `[Vehicle]` and `[Manufacturer;Model]`. The user can select the derived path by giving its full derived tag name. The system resolves path ambiguity by selecting the path that reflects the hierarchical organization of the document (avoids using ID/IDREF) if possible. A full discussion of semantic querying in the presence of ID/IDREF is beyond the scope of this paper. Further, we have not considered links between documents using XLink [10] or XPath [7], although the context view model could be extended to handle them.

Most XML query languages allow path expressions to contain wild cards such as: selection ("a | b"), match one edge ("_"), Kleene closure ("*"), at least one occurrence ("+"), and optional occurrence ("?"). Although such wild card operators give path expressions more flexibility to hide some of the structure of the document, they do not provide the same flexibility as the context view. Further, the context view can be queried using the same wild card operators.

4.2 Mapping Algorithm

This section overviews the mapping algorithm that converts a semantic query S expressed on the context view CV to an XML query Q on a given DTD x. The element tag names for x are defined according to the ontology used to define CV. DTD x may contain linking via ID/IDREF. Query translation is performed by mapping every semantic name to a path expression for x. The algorithm performs the following steps:

- Perform a breadth-first traversal of DTD x starting from the root to build a mapping table T.
- Each entry in the mapping table contains a semantic tag name tn and an associated set of path expressions P. Each path expression $p \in P$ provides a path in DTD x to an element with tag name tn.
- If DTD x is a tree, then each tag name tn has only one path, and replacing a semantic name in the query with a path expression is a 1:1 mapping.
- If DTD x is a graph, there may be multiple possible paths $p_1, p_2, ..., p_n$ to tn. The multiple mappings can be used to return the union of all paths, or the possible paths can be given to the user to select one.
- After a set of mappings to path expressions have been determined, the system must verify that the path expressions are connected. A set of path expressions is *connected* if it is possible to build a minimal spanning tree with a defined root connecting all nodes involved in the query **after** the root node of DTD x is removed.
  - DTDs with no linking always have a connected spanning tree for all queries.

- DTDs with ID/IDREF linking many have more than one possible spanning tree for certain queries. The system enumerates the possibilities and has the user select one.

The performance of the algorithm is primarily determined by the cost to perform the mappings. The cost to construct the mapping table is $O(|E_x|)$ where $|E_x|$ is the number of edges in the graph representing DTD x. However, this cost can be amortized across multiple queries by pre-computing the mapping table. At query time, the query processor must only lookup the appropriate mappings in the table which can be performed quite efficiently. If the mappings yield a set of paths that is not connected, the system must search for connections to produce a spanning tree.

For example, the query in Figure 6 is translated for DTD1 by searching for $E' = \{$`[Manufacturer].Name, [Manufacturer;Model;NHSC Test].Rank, [Vehicle].Price`$\}$. Parsing the DTD gives a path to `[Manufacturer].Name` as `V.Manufacturer.Manufacturer--Name`. The path for `[Vehicle].Price` is `V.Manufacturer.Manufacturer-Model.Vehicle.Vehicle--Price`. Searching for the rank attribute is similar. Note that once the first attribute is found (in this case `[Manufacturer].Name`) all other attributes contain as part of their path, a path through the parent node of the first attribute. Performing substitution of semantic tag names for paths results in a Lorel query. The parent node of the first attribute found is used to define a variable in the FROM clause. In this case, this produces `FROM V.Manufacturer M`, and then `M` is used to specify relative paths to the remaining elements.

The same query on DTD3 results in two possible paths for several tag names, including `[Manufacturer].Name`, that can be accessed through a direct path or through an indirect path using the IDREF under the `Vehicle` element. In this case, there is only one mapping for `[Vehicle].Price`, so this mapping is chosen. This selection forces the mapping for `[Manufacturer].Name` to use the IDREF mapping, so there is no ambiguity. Using the IDREF mapping produces a spanning tree with `Vendor` as the root element. Note that if the query just specified `[Manufacturer].Name` then the mapping selected would be the one without using IDREF. For cases where multiple IDREF links are possible, the user must chose a correct mapping (method of relating the individual trees of the forest) to produce a single spanning tree.

5 Related Work

There has been a comparison [4] between the numerous XML query languages. All the query languages studied, including graphical query languages such as XML-GL [8] focus on extracting data based on structural path expressions. Although this is sufficient for users with intimate knowledge of XML querying and the DTD itself, it is challenging for users with less domain and structural DTD knowledge to formulate queries. We have proposed a semantic query facility built on top of systematic naming of XML elements. The user queries a

canonical view, and the system translates semantic tags to path expressions on the actual DTDs. Further, after semantic naming of the document, it is still possible to query the document using normal XML query languages. Thus, the semantic naming can be considered as a virtual, overlay semantics on top of the original structure.

Related work on XML querying includes the MIX system [3], integrating keyword search into XML query processing [12], and the definition of a `meet` operator [20]. These systems allow simpler user querying by allowing navigation of XML documents, allowing imprecise queries by using keyword search, or by relating concepts in a document by structure. Our work improves on these systems by providing a general, semantic querying layer on top of existing XML structure that hides the entire structure from the user. Further, the semantic layer can be browsed independently of the XML documents, and queries can be formulated graphically over the context view [14]. XML Namespaces [23] can be used to create unique tag names, but there is no requirement for their systematic usage. Further, comparing semantics of names across namespaces encapsulates many of the challenges in system integration. This work presents a methodology for naming that does not rely on the construction of namespaces, and the use of XML namespaces is mostly orthogonal to this work.

One of the motivations for our work is that XML documents (views) allow access to a relational database [21] by multiple parties. In these cases, it is beneficial for users to be able to graphically browse and semantically query information stored in XML documents without in-depth detail of its organization.

Others areas of similar work include integration and querying using ontologies such as Ontobroker [9]. At its basis, semantic naming relies on an ontology (the canonical view) for naming XML elements. However, the ontology defined for this work does not formally encode axioms or relationships between elements. This is a design decision as the relationships can be dynamically realized by matching element names. The Ontobroker ontology is more powerful than the one we use for semantic querying. However, it does not provide a naming system for entities, nor was it designed for structure-independent querying.

In the relational database world, there has been extensive work in the development of graphical query tools [5, 22] to aid in the formulation of SQL queries, query languages such as query by example (QBE) [24], and semantic query approaches [15, 18] to reduce the semantic burden on users. Systems such as Query-by-Example (QBE) [24] and Kaleidoscope [6] allow users to query relational databases without using SQL. There also have been examples of semantic query languages for the relational model such as SemQL [15], SemanticAccess [19], and conceptual query language (CQL) [18]. Our approach works on XML as well as the relational model, and does not require wrappers or advanced database knowledge. For less experienced users, querying using names is simple due to is similarity with keyword searching. Further, we have implemented a graphical query tool [13] that allows querying by context and provides an easy-to-use query interface.

6 Future Work and Conclusions

In conclusion, we have presented a semantic naming methodology to construct element tag names for XML DTDs. The semantic naming procedure produces names with regular structure that contain sufficient context to identify element semantics. Systematic naming allows for increased human readability and documentation. Given a semantically named DTD, it is possible to build a canonical context view for semantic querying. Semantic queries are then mapped to structural queries on specific DTDs and ambiguity is resolved by exploiting the hierarchical structure of the document itself. Since the document structure is unchanged, regular structural queries can still be performed, albeit with longer tag names. The drawback of longer tag names is mitigated by graphical user interfaces for querying and browsing the context view, and by lessening the requirement for path expressions which are automatically determined for semantic queries. This work represents one of the first attempts at providing systematic naming and querying without specifying path expressions.

Future work includes developing a formal query algebra for semantic queries and mapping semantic queries to relational and object-oriented models.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers, 2000.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.
3. C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-Based Information Mediation with MIX. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 597–599, 1999.
4. A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–79, 2000.
5. T. Catarci and G. Santucci. Query by Diagram: A Graphical Environment for Querying Databases. *SIGMOD Record*, 23(2):515–515, June 1994.
6. S. Cha and G. Wiederhold. Kaleidoscope Data Model for An English-like Query Language. In *17th International Conference on Very Large Data Bases*, pages 351–361, 1991.
7. J. Clark and S. DeRose. XML Path Language (XPath), November 1999.
8. S. Comai, E. Damiani, and P. Fraternali. Computing Graphical Queries over XML Data. *ACM Transactions on Information Systems*, 19(4):371–430, 2001.
9. S. Decker, M. Erdmann, and R. Studer. ONTOBROKER: Ontology based access to distributed and semi-structured information. In *Database Semantics - Semantic Issues in Multimedia Systems*, volume 138, 1998.
10. S. DeRose, E. Maler, and D. Orchard. XML Linking Language: XLink, 1995.
11. R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 2000.
12. D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. *WWW9 / Computer Networks*, 33(1-6):119–135, 2000.

13. R. Lawrence and K. Barker. Integrating Relational Database Schemas using a Standardized Dictionary. In *SAC'2001- ACM Symposium on Applied Computing*, March 2001.
14. R. Lawrence and K. Barker. Using Unity to Semi-Automatically Integrate Relational Schema. In *18th International Conference on Data Engineering (ICDE 2002)*, pages 329–330, March 2002.
15. J. Lee and D. Baik. SemQL: A Semantic Query Language for Multidatabase Systems. In *Proceedings of the 8th International Conference on Information Knowledge Management (CIKM'99)*, pages 259–266, Kansas City, MO, November 1999.
16. D. Maier. Database desiderata for an XML query language, 1998.
17. G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Five Papers on WordNet. Technical Report CSL Report 43, Cognitive Systems Laboratory, Princeton University, 1990.
18. V. Owei and S. Navathe. Enriching the conceptual basis for query formulation through relationship semantics in databases. *Information Systems*, 26(6):445–475, 2001.
19. N. Rishe, J. Yuan, R. Athauda, S. Chen, X. Lu, X. Ma, A. Vaschillo, A. Shaposhnikov, and D. Vasilevsky. Semantic Access: Semantic Interface for Querying Databases. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 591–594, 2000.
20. A. Schmidt, M. Kersten, and M. Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proceedings of the 17th International Conference on Data Engineering*, pages 321–329. IEEE Computer Society, 2001.
21. J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. Querying XML views of relational data. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases*, pages 261–270, 2001.
22. M. Stonebraker, J. Chen, N. Nathan, C. Parson, A. Su, and J. Wu. Tioga: A Database-Oriented Visualization Tool. In *Proceedings of the Visualization '93 Conference*, pages 86–93, San Jose, CA, October 1993.
23. T.Bray, D. Hollander, and A. Layman. Namespaces in XML, January 1999.
24. M. Zloof. Query-by-Example: the Invocation and Definition of Tables and Forms. In *Proceedings of the International Conference on Very Large Data Bases*, pages 1–24. ACM, 1975.