Mapping XML fragments to community web ontologies
Bernd Amann, Irini Fundulaki, Michel Scholl, Catriel Beeri, Anne-Marie Vercoustre

To cite this version:
Bernd Amann, Irini Fundulaki, Michel Scholl, Catriel Beeri, Anne-Marie Vercoustre. Mapping XML fragments to community web ontologies. WebDB 2001 - International Workshop on the Web and Databases, May 2001, Santa Barbara, California, United States. pp.97–102. hal-00948950

HAL Id: hal-00948950
https://hal.science/hal-00948950
Submitted on 18 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Mapping XML Fragments to Community Web Ontologies

B. Amann I. Fundulaki M. Scholl
Cedric-CNAM Paris and INRIA Rocquencourt, France
C. Beeri
Hebrew University, Israel
A-M. Veroustre
INRIA Rocquencourt, France and CMIS CARLTON Australia

(Extended Abstract)

1 Introduction

The C-Web (Community Web)\(^1\) project [2] aims at supporting the sharing, integration and retrieval of information in a specific domain of interest for a group of people who desire to access and exchange knowledge and information in this domain. A C-Web portal (or mediator) provides the infrastructure for (1) publishing information sources and (2) formulating structured queries by taking into consideration the conceptual representation of a specific domain in form of an ontology.

Query mediation has been extensively studied in the literature for different kinds of mediation models and for various source capabilities. Tsimmis [14], YAT [5], Infomaster [9], Information Manifold [13], Tukwila [15, 12] and PICSEL [10] are among the most prominent examples of mediation systems. In C-Web, query mediation is closely related to the last three of them which follow the local as view approach. These systems describe the source contents in terms of the mediator schema and the resulting query rewriting algorithms are based on efficient implementations for evaluating query subsumption and satisfiability. Similar to the above systems, in C-Web users formulate queries in terms of the ontology and the mediator translates these queries according to the source descriptions and the source query facilities.

In this paper we are interested in the publication and querying of XPath \(^6\) enabled XML resources. More precisely, we want to take advantage of the structure of XML documents (generally described by a DTD) for mapping pieces of information contained in XML fragments \(^11\) to domain specific ontologies. The objective here is to be able to forward user queries to diverse XML repositories while hiding their DTD heterogeneity to the end-user. Our contribution is two-fold: (1) we propose a simple mapping language describing sources by a set of rules relating XPath location paths to the concepts and roles of an ontology and (2) a query rewriting algorithm for translating user queries into queries expressed in an XML query language \([3, 4, 16]\) that are send for evaluation to XML sources.

XPath \([6, 17]\) is a tree pattern language which allows to characterise XML fragments according to their position in the document tree, their type and their contents. Whereas XPath does not have the full expressive power of XML query languages, the choice of using XPath as part of a mapping language for XML documents is interesting for several reasons. First, XPath is already part of several XML-related languages \([1]\) for the transformation (XSLT \([8]\)), linkage (XLink \([7]\)) and querying (XQL \([16]\), XQuery \([3]\) and Quilt \([4]\)) of XML documents. This implies that XPath is used by an important number of XML developers who do not have to learn a new language for writing mapping rules and that it is implemented in a variety of tools and can be integrated very easily in a standard Web server\(^2\).

In this paper we illustrate our approach considering the integration of cultural information sources accessible from Web servers. We show how such Web resources can be described according to an ontology (Section 2) by creating mapping rules between XML fragments (identified by XPath expressions) and ontology concepts and roles (Section 3).

\(^1\)http://cweb.inria.fr

Finally we illustrate in Section 5 how these mappings can be used for the rewriting of a restricted class of OQL queries presented in Section 4.

2 Ontologies and Schema Paths

Ontologies: An ontology is a quintuple \( O = (C, R, \text{source}, \text{target}, \text{isa}) \), where: (i) \( C \) is a set of concepts, (ii) \( R \) is a set of binary roles between concepts in \( C \), (iii) functions \( \text{source} \) and \( \text{target} \) map roles to their domain and target concepts, respectively, and (iv) \( \text{isa} \) is an inheritance relationship on \( C \) with the usual properties (hierarchy on \( C \)).

An ontology usually reflects a common understanding of a certain domain. In our case we are interested with domain-specific information stored in one or several XML databases and we define the semantics of an ontology by the databases that conform to it: a database for an ontology \( O \) contains a set of objects (instances) for each concept in \( C \) which are related by instances of roles in \( R \). The \( \text{isa} \) component of \( O \) is interpreted with a subset semantics, and we say that two concepts \( c \) and \( c' \) are \( \text{isa-related} \) if \( c = c' \) or \( c \ \text{isa} \ c' \) or \( c' \ \text{isa} \ c \) holds.

An example of an ontology for cultural artifacts is shown in Figure 1. Ten concepts and nine roles describe actors (persons) performing activities for producing artifacts. For example, concept \( \text{Person} \) collects all persons, is a subconcept of \( \text{Actor} \) and inherits role \( \text{performed} \) relating actors to activities.

![Figure 1: An Ontology for Cultural Artifacts](image)

It is possible to follow schema paths between concepts in the ontology graph. For example, the artifacts produced by an actor are obtained by traversing role \( \text{performed} \) (activities performed by actors), and then role \( \text{has produced} \) (artifacts produced by these activities). Observe also, that this traversal defines the subset of all artifacts produced by some actor. We distinguish between role paths defining derived roles and concept paths defining virtual concepts.

Role paths and derived roles: A role path of length \( n (n \geq 1) \) is a sequence \( r = r_1 \ldots r_n \), where \( r_i \) are roles, such that for all \( 1 \leq i < n \), \( \text{target}(r_i) \) and \( \text{source}(r_{i+1}) \) are \( \text{isa-related} \). The source and target of a role path are defined by the source and the target of its extremities: \( \text{source}(r) = \text{source}(r_1) \) and \( \text{target}(r) = \text{target}(r_n) \). Clearly, the composition of a role path \( r \) and a role path \( r' \), denoted \( r \circ r' \), is well-defined provided that \( \text{target}(r) \) and \( \text{source}(r') \) are \( \text{isa-related} \). Role \( \text{performed} \) is a role path of length 1 with source \( \text{Actor} \) and target \( \text{Activity} \). A role path \( r \) of length \( > 1 \) defines a derived role, \( \text{rol}(r) \) from instances of its source concept to instances of its target concept. For example, role path \( \text{technique}.\text{used_material} \) of length 2 defines a derived role, \( \text{rol(technique}.\text{used_material)} \) between concept \( \text{Man Made Object} \) and concept \( \text{Material} \). A sequence \( r_1, r_2 \) can be viewed as a derived role whose every instance connects an instance \( o \) of \( \text{source}(r_1) \) with an instance \( o' \) of \( \text{target}(r_2) \), through an intermediary \( o'' \) that must be an instance of both \( \text{target}(r_1) \) and \( \text{source}(r_2) \). Instance \( o'' \) can only exist if \( \text{target}(r_1), \text{source}(r_2) \) are \( \text{isa-related} \), and \( o'' \) must be an instance of both concepts \( \text{target}(r_1) \) and \( \text{source}(r_2) \).

Concept paths and virtual concepts: A concept path \( p \) is either of the form \( c \), or a sequence \( c.r \), where \( c \) is a concept and \( r \) is a role path, such that \( \text{source}(r) \) and \( c \) are \( \text{isa-related} \). The length of \( p \) is 0 in the first case, and the length of the role path \( r \) in the second case. The source and target of \( c \) are \( c \). The source and target of \( p = c.r \) are defined as: \( \text{source}(p) = c \) and \( \text{target}(p) = \text{target}(r) \). The composition of a concept path \( p \) and a role path \( r \), denoted \( p \circ r \), is well-defined provided that \( \text{target}(p) \) and \( \text{source}(r) \) are \( \text{isa-related} \). A concept path \( p = c.r \) can be viewed as defining a virtual concept denoted by \( \text{con}(p) \), standing for “the instances of \( \text{target}(p) \) that can be reached from \( \text{source}(p) \) by following the roles in \( p \), in order”. For example, concept path \( \text{Person}.\text{performed}.\text{has_produced} \) denotes the artifacts (instances of concept \( \text{Man Made Object} \)) produced by some person. We also consider that the suffixes of a concept
path $p$ define superconcepts of the virtual concept defined by $p$ : a suffix of a concept path $p$ is obtained by removing a prefix, and adding an appropriate concept in the beginning. For example, if $p = c_1.r_1.r_2.r_3$, then $source(r_3).r_3$ and $source(r_2).r_2.r_3$ are its suffixes. It is evident that the extent of $p$ is a subset of the extent of any of its proper suffixes. Thus, we also relate a concept path and its suffixes (as virtual concepts) by isa.

Given an ontology, a set of role paths $\mathcal{R}$, and a set of concept paths $\mathcal{C}$ defined on this ontology, we can define a view ontology, $\mathcal{O}_V = (C_V, R_V, source_V, target_V, isa_V)$ where $C_V$ is the set of virtual concepts defined by each concept path in $\mathcal{C}$ along with its superconcepts (as previously defined), $R_V$ is the set of derived roles defined by each role path in $\mathcal{R}$, and $isa_V$ is the isa relationship between virtual concepts.

3 A Mapping Language for XML

In the following we present a language for establishing mappings between XML fragments and ontology concepts and roles. Suppose that http://www.art.com is a web server of XML documents about artists and art in general. The corresponding DTD is shown in Figure 2.

```
<!--ELEMENT ARTIST (NAME,NATIONALITY,ARTIFACT*)> <!--ELEMENT NAME (#PCDATA)>
<!--ELEMENT ARTIFACT (TITLE,PROCEDURE,LOCATION)--> <!--ELEMENT NATIONALITY (#PCDATA)>
<!--ELEMENT TITLE (#PCDATA)> <!--ELEMENT MATERIAL (#PCDATA)>
<!--ELEMENT LOCATION (#PCDATA)> <!--ELEMENT STYLE (#PCDATA)>
<!--ELEMENT PROCEDURE (MATERIAL,STYLE)>
```

Figure 2: A simple XML DTD for Artists and Art in general

The mapping rules in Figure 3 map XPath location paths to schema paths in the ontology of Figure 1:

- $R_1$: http://www.art.com//ARTIST as $u_1$ ← Person
- $R_2$: $u_1$/NAME ← has_name
- $R_3$: $u_1$/ARTIFACT as $u_2$ ← performed.has_produced
- $R_4$: $u_2$/TITLE ← has_title
- $R_5$: $u_2$/PROCEDURE as $u_3$ ← technique
- $R_6$: $u_3$/MATERIAL ← used_material
- $R_7$: $u_3$/NATIONALITY ← born.took_place_at
- $R_8$: http://www.art.com//ARTIFACT as $u_2$ ← Man Made Object
- $R_9$: $u_2$/LOCATION ← located_at

Figure 3: Set of Mapping Rules

For example, rule $R_1$ states that all fragments of element type ARTIST which are descendants of the root of document http://www.art.com (see [6, 17] for the definition of XPath patterns/location paths) are instances of concept Person. In the same way, rule $R_8$ states that all elements of element type ARTIFACT, descendants of the root of document http://www.art.com, are instances of concept Man Made Object. Rule $R_3$ creates instances of the derived role, $rol(performed.has_produced)$ connecting each element $x$ of type ARTIST obtained by rule $R_1$, to all elements of type ARTIFACT that can be obtained from $x$ by applying XPath /ARTIFACT.

Mapping Rules: A mapping rule is an expression of the form $R : u/q [as v] ← p$, where 1) $R$ is the rule’s label, 2) $u$, the rule’s root, 3) $q$ is an XPath location path, 4) $[as v]$ is an optional binding of variable $v$ and 5) $p$ is a schema path. More precisely, the root ($u$) is either a variable, or a URL and $p$ is a role path if $u$ is a variable and a concept path otherwise. Rule $R$ is called a relative mapping rule if its root is a variable, and an absolute mapping rule otherwise. Let $lp(R), cp(R)$ denote $R$’s location path $q$ and schema path $p$, respectively.

A rule $r : u/q [as v] ← p$, is interpreted as follows: (1) apply location path $q$ to all instances ($x_1$) of variable $u$; (2) the obtained fragments ($x_2$) are added to the set of instances of variable $v$; (3) if $u$ is a variable, add ($x_1, x_2$) to the set
of instances of the derived role \( p \); (4) if \( u \) is a URL, add \( x_2 \) to the set of instances of virtual concept \( p \). Observe that we allow multiple bindings for the same variable. Such bindings reduce the number of mapping rules, while making the structure easier to understand. In the opposite way, it is possible to map the same location path to different variables in order to distinguish between two interpretations of the same set of fragments. Observe that rule \( R_3 \) binds the same variable \((u_2)\) as rule \( R_8 \) which allows the application of rules \( R_4 \) and \( R_5 \) on the instances “found” by both rules to obtain the artifacts’ title and technique.

Given a set of mapping rules, we define reachability for rules and variables, as follows: Each rule whose root is a URL, or a reachable variable is reachable, and each variable bound in a reachable rule is reachable. A mapping \( M \) over an ontology \( \mathcal{O} \) is a set of mapping rules such that 1) labels are unique (that is, no two rules have the same label), 2) all rules are reachable (hence so are all variables) and 3) the concepts and roles used in its rules occur in \( \mathcal{O} \). Note that the binding-use relationships between variables in a mapping may be cyclic. The simplest case of a cycle is a rule whose left-hand-side contains \( v/A \) as \( v \) (provided that \( v \) can be reached from a URL by some other rules). A mapping is cyclic if it contains a binding-use cycle.

**Concatenation of mapping rules**: Two rules \( R_1 : a/q_1 \: \text{as} \: v_1 \: \leftarrow \: p_1, \: R_2 : v_1/q_2 \: \text{as} \: v_2 \: \leftarrow \: p_2 \), can be concatenated, if the composition of their schema paths, \( p_1 \circ p_2 \) is well defined\(^3\). Note the constraint that the root of \( R_2 \) is bound in \( R_1 \) and that concatenation is possible only if \( p_2 \) is a role path. The concatenation is the rule \( R_1 \circ R_2 : a/q_1/q_2 \: \text{as} \: v_2 \: \leftarrow \: p_1 \circ p_2 \). Given a mapping \( M \), its closure, denoted by \( M^* \), is the set of all rules that can be obtained from \( M \) by repeated concatenation. Its expansion, denoted \( M^* \), is the set of absolute rules in \( M^* \) (\( M \subseteq M^* \)). Given an ontology \( \mathcal{O} \), a mapping \( M \) over \( \mathcal{O} \) defines a view ontology, \( \mathcal{O}_M = (C_M, R_M, \text{source}_M, \text{target}_M, \text{isa}_M) \) where \( C_M \) is the set of virtual concepts defined by the concept paths of absolute rules in \( M \) and \( R_M \) is the set of derived roles defined by the relative rules in \( M^* \).

### 4 Tree Queries

The user views the C-web portal as a single database of fragments without knowledge of the source on which each fragment is located. We might then consider each fragment as an object whose identity is the location path of the fragment. A mapping \( M \), allows us to organise the fragments into collections of instances of concepts in \( \mathcal{O}_M \), and also to view certain XML paths as representing roles of \( \mathcal{O}_M \). It follows that one can in principle query this database of fragments using a query language, such as OQL. The answer of a query is defined using the semantics of queries on object bases. However, even though the answer is well-defined, an efficient evaluation requires that we use effectively the mapping \( M \) to translate the query into one or more queries on the sources. Our ability to do so may depend on the form of the query. We introduce a tree query language, as a restricted version of OQL, and discuss how mapping rules can be used in the evaluation of queries in this language.

Tree queries are based on select-from-where clauses on schema paths. Figure 4 shows two tree queries \( Q \) and \( Q_1 \) where \( Q \) is the general form of a tree query and \( Q_1 \) is an example of such a query that looks for “the titles and location of all man made objects where clay was used in the applied technique”.

![Tree Queries](image)

Tree query \( Q \) is defined as follows. The \( x_i \)'s are variables. Each \( e_i \) in the from clause is either (1) a concept path

\(^3\)We do not define any restriction on the concatenation of location paths (rule left-hand-sides).
c_i/c_i,q_i, defining a (virtual) concept or (2) a variable x_i followed by a derived role q_i. In the first case, c_i and c_i,q_i, and in the second case, q_i, are called the binding path of x_i, denoted bp(x_i). < is a partial order on the variables, such that if x_i, q_i, x_j occurs in the from clause, then x_j < x_i (x_j is called the parent of x_i). The where clause is a conjunction of simple predicates, of the form c_k ≡ x_iθd in which θ ∈ {=, <, >, ≤, ≥} and d is an atomic value. It is not possible to express joins by equalities between variables. Schema paths appear exclusively in the from clause, a syntax that simplifies the presentation of our rewriting algorithm. The language has no quantifiers, aggregates, or subqueries but a variable x_j, present in the from clause but not in the select or where clauses, is implicitly existentially quantified.

5 Query Rewriting and Evaluation

We present here a query rewriting algorithm that transforms a tree query Q according to a given acyclic mapping M into a query expressed in an XML query language. The algorithm consists of two phases, namely the binding and the rewriting phases. In the first phase we try to find all rules whose schema paths “map” to query paths and in the second phase we rewrite the initial query into a set of XPath expressions.

Binding Variables to Rules: Take, for example, query Q1 and the mapping shown in Figure 3. Intuitively, rules R_1,R_3 and R_5 might be used to find man made objects for variable a. The former finds all man made objects that have been produced by a person and the latter can be used to find man made objects in general. Rule R_4 can then be used to find the titles for those man made objects and values for variable b. In the same way, rule R_5,R_6 finds the material used to produce these objects and obtains instances for variable c.

Formally, a variable to rule binding, or shortly variable binding, for a query Q is a mapping β on a subset of variables in Q, denoted dom(β). More precisely, if dom(β) is not empty, then β associates each variable in dom(β) with a rule in closure M*, such that the following holds:

1. if x is the root of query Q, then β(x) is an absolute mapping rule such that the query binding path bp(x) of variable x denotes a superconcept of the virtual concept, con(cp(β(x))) in O_M, which is the case when bp(x) is equal to or a suffix of cp(β(x)) or a superconcept of the target of β(x);
2. else the declaration of x in Q has the form x’.q x, i.e. x is the child of variable x’ and the binding path of x is q. Answers for x can be obtained from answers for x’, by following the binding path q of x if (1) the root variable of rule β(x) is bound in rule β(x’), (2) the role denoted by the derived role of the rule β(x), rol(cp(β(x))), is equal to the role denoted by the binding path of x and (3) the composition of the schema paths of the rules β(x’) and β(x) is well-defined, i.e. the concatenation of the two rules is well-defined.

For example, for query Q1 above, the query root variable a is bound in the query to the (real) concept Man Made Object. In this case the concept path of bp(a) is a suffix of cp(R_1,R_3)=Person.performed.has.produced and is equal to the concept path of cp(R_3). Now, the binding path bp(b) of variable b in Q1 is has.title, so we need a rule whose schema path leads from Man Made Object to the virtual concept Man Made Object.has.title; Rule R_4 is such a rule. Similar reasoning applies for the binding variables c, d.

Binding Algorithm: A variable binding is full if it is defined on all variables of Q, and partial otherwise. Given a query, we need to find all full variable bindings according to some mapping M. Indeed, each such binding provides, as shown below, a subset of the answer. By taking the union of all these answers, we have a maximal answer, with respect to the given sources and the given mapping. We assume, w.l.o.g. that variables are arranged in pre-order: x_1,...,x_n. This ensures that when we try to extend a binding to a variable, it is defined on its parent. A binding is represented as a vector of associations of variables to rules or concatenation of rules, in that order, namely \{x_1 ↦ r_1,...,x_n ↦ r_n\}.

We sketch here the binding algorithm for acyclic mapping rules. We assume that since the closure M* is finite, it has been calculated in advance. In the first step of this algorithm, we extend the empty binding to the root variable x_1. For each absolute rule r in M* such that cp(r) is a subconcept of the binding concept path bp(x_1), we create a binding \{x_1 ↦ r\}. Then, we iterate through the sequence of variables, from the left. Assume we have constructed a set of partial bindings that are defined on all variables up to and including x_i, we show how to extend it to x_{i+1}. Let
Let all previous partial bindings can be dropped. x′ traversed in this step, and only in this step. After all bindings that are defined up to and including x are computed, all previous partial bindings can be dropped. When M is cyclic, M* may be infinite. In a longer version of this paper we show how to compute a useful finite representation of M* that can be used to compute a finite representation of the set of bindings.

We illustrate variable binding for the mapping shown in Figure 3 and for query Q1 above. In the first step, we need to extend the empty binding with a binding for the root variable a, associated with virtual concept Man Made Object. An obvious one is a → R8, but since Person.performed.has_produced defines a subconcept, the binding a → R1.R3 is another solution and we obtain two (partial) bindings β1 and β2 defined on a. It is easy to see that each of them can be extended afterwards by b → R4, c → R5.R6, d → R9 to a complete binding.

Query Rewriting: Let B be a set of full variable bindings calculated by the foregoing algorithm for some query Q and mapping M. Then for each of the variable bindings in B, we create a query QS where we replace the schema path for a query variable with the XPath location path of the rule associated to it. For example, for query Q1, the algorithm returns bindings β1 and β2. For binding β1 = \{a → R1.R3, b → R4, c → R5.R6, d → R9\}, the obtained query is:

\[
\text{QS1: select } b, d \\
\text{from } \text{http://www.art.com/ARTIST/ARTIFACT}_a, \\
a/TITLE b, a/PROCEDURE/MATERIAL c, a/LOCATION d \\
\text{where } c = \text{“clay”}
\]

This query can be easily expressed in any XML query language using XPath for binding variables such as Quilt [4], XQL [16] and XQuery [3]. However, XPath is a restricted language. For example, an XPath query can only return one kind of node (and for each node its XML subtree). Thus, the query QS1 cannot be expressed as a single XPath expression since it returns a set of pairs \{(b, d)\}. But, we can express in XPath a request for the node of a, including in it the conditions that the paths in the XML document leading to b, c, d exist, and that the value of node for c is equal to “clay”. When a subtree that corresponds to a is returned, we can locally project it on the nodes for b, d.

An algorithm that decomposes a query QS into a partially ordered set of XPath expressions to be evaluated by the sources, where the partial order implies information passing, followed by an XML query to be evaluated by the mediator on the results of these subqueries is presented in a full version of this paper.

References


