Ontology-Based Integration of XML Web Resources*

Bernd Amann[1], Catriel Beeri[2], Irini Fundulaki[1], and Michel Scholl[1]

[1] Cedric-CNAM and INRIA-Rocquencourt
(amann|fundulak|scholl)@cnam.fr
[2] The Hebrew University, Israel
beeri@cs.huji.ac.il

Abstract. This paper deals with some modeling aspects that have to be addressed in the context of the integration of heterogeneous and autonomous XML resources. We propose an integration system, but the emphasis of this paper is neither on its algorithmic aspects nor on its technical details. Instead, we focus on the significance of offering appropriate high-level primitives and mechanisms for representing the *semantics* of XML data. We posit that support for such primitives and mechanisms is a pre-requisite for realizing the goals of the *semantic Web*.

1 Introduction

The last decade has seen the emergence of the Web as the central forum for data storage and exchange, and as the infrastructure for a large part of human communications and information-based activities in many domains, from art and medicine to business. The utility of the Web depends, however, on the development of models and paradigms for the representation and the manipulation of data, that enable the development of flexible and expressive applications.

Towards this end, XML [1] has been proposed as a standard for data exchange, possibly also for storage. Compared to the relational model, the de-facto standard for database systems, its structural primitives for building trees of elements with attributes offer much more flexibility in data organization and format.

Clearly, such standardization efforts must be accompanied by formal and experimental studies of XML and its associated mechanisms: formal studies, to understand their expressive power and the computational complexity of the algorithms they require, and experimental studies, to better understand the requirement of potential application domains, and the support they require for conceptual modeling and manipulation of data. For example, the shortcomings of DTD's as the mechanism for specifying schematic properties of XML have been identified, and an effort to overcome those limitations has lead to the definition of XML Schema [16].

In this paper we report on *XML data integration* issues, encountered during the C-Web project [13]. One of our goals was to design and implement a portal architecture or *mediator* [27] which can be considered as an experimental study on the use of XML

 - This work was partly supported by the EC project C-Web (IST 1999-13479).

for data integration. The resulting prototype [17], follows the *local as view approach* (LAV) [18] and offers to its users a virtual data repository in a given domain. The repository is virtual in that the actual data resides in some external sources. However, the users of the repository are not concerned with source location and source data organization which are taken care of by the integration portal. We posit XML-enabled data sources, with some support for XML querying, either by using XPath [10], or possibly (in the near future) by using XQuery [7].

Our emphasis in this paper is neither on the algorithmic details nor on the technical details of the system, which are thoroughly presented in [5, 3, 17]. Rather, we concentrate on the *data model* of the mediator and the mechanism for *describing* XML sources at the mediator. We also discuss at length *conceptual modeling* issues that need to be addressed in the context of a data integration project, and our approach to tackling them. Our study emphasizes the significance of offering appropriate high-level primitives and mechanisms for representing semantics of XML data. We posit that support for such primitives and mechanisms is a pre-requisite for realizing the goals of the *semantic Web*.

The outline of the paper is as follows. Section 2 is an overview of the approach to data integration using cultural XML resources. In Section 3 we present the integration model and discuss the studied problems and justify the decisions we have made. We conclude in Section 4 and present the future work.

2 System Overview

We present in this section a general overview of our system architecture, its main ideas, and main components. Detailed discussion of technical concepts and algorithms can be found in [3]. Our goal here is to provide sufficient understanding of the system and its underlying ideas, as a basis for the discussion of issues in subsequent sections.

2.1 XML Resources

We illustrate our approach using an example concerning the integration of two Web-accessible XML-based cultural information sources. The first source, located at URL *http://www.paintings.com*, contains information about painters and their paintings. The DTD of this source defines two element types (Painter and Painting) where each of them contains one XML attribute :

```
<!ELEMENT Painter  (Painting+)>
<!ATTLIST Painter  name  CDATA #REQUIRED>
<!ELEMENT Painting EMPTY>
<!ATTLIST Painting title ID #REQUIRED>
```

The second source, located at URL *http://www.all-about-art.com*, is more complete than the first source and describes paintings, artists and museums. Its DTD is the following :

```
<!ELEMENT Art      (Painting|Artist|Museum)*>
<!ELEMENT Painting Title>
<!ATTLIST Painting painter #IDREF #REQUIRED
                   museum  #IDREF #IMPLIED>
<!ELEMENT Artist   EMPTY>
<!ATTLIST Artist   name    #CDATA #REQUIRED
                   id      #ID    #REQUIRED>
<!ELEMENT Title    (#PCDATA)>
<!ELEMENT Museum   (Name,City)>
<!ATTLIST Museum   id      #ID    #REQUIRED>
<!ELEMENT Name     (#PCDATA)>
<!ELEMENT City     (#PCDATA)>
```

Although the two sources deal with the same subject, they differ in terms of both contents and terminology. The first source contains information about painters and their paintings and the data is organized by painter. The locations of the paintings in museums are deemed irrelevant and are not described. In the second source, painters, artists and museums are described. It organizes its data differently from the first source in that paintings and artists are described independently. To designate the painter and the museum of a painting the XML ID/IDREF mechanism is used.

As is common in integration scenarios, each of the sources may supply only *part* of the information sought by a user. For the information about painters, the sources use different terminologies (the first one uses the term `Painter`, the second `Artist` to designate a painter). Observe that information concerning the same entity types might be structured differently in different sources. For example, while in the first, paintings are arranged "below" the painters, the second source prefers to organize the same data by painting, with painters below their respective paintings.

2.2 The Global Schema

We base our integration project on the *local as view* approach [18]. That is, we assume a hypothetical global repository that contains all the information of interest to the user in a given subject area described by a global schema.

Each of the sources is a local view of this global repository. Being a view means that it contains only part of the relevant data. It may contain only information about some entity types, but not about others. Even concerning entity types that it contains, it may not contain information about all the entities of that type that are present in the global repository; or, it may not contain all information about an entity. And, its structure may differ from that of the global repository.

In our case, the local views (sources) are represented in XML. However, the data model used for the global schema is not XML, but rather an *ontology*. For us, an ontology is essentially a simple object-oriented schema describing concepts with typed multi-valued attributes and connected by binary, symmetric, many-to-many roles. Attributes and roles can be inherited through inheritance (isa) links between classes [4].

An example of an ontology for cultural artifacts, inspired by the ICOM/CIDOC Reference Model [14] is shown in Figure 1 as a labeled directed graph.

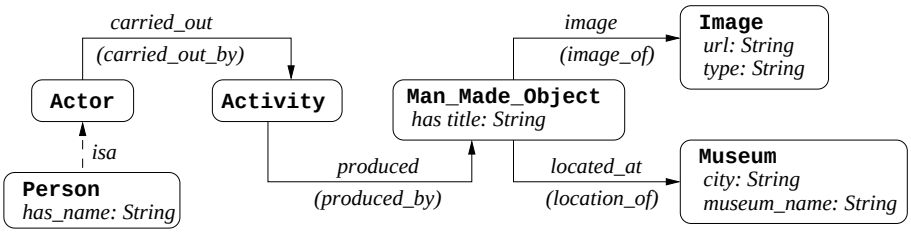

Fig. 1. An Ontology for Cultural Artifacts

Nodes of the graph represent the *concepts* of the ontology and their attributes. Each concept has a unique name represented in bold-face followed by a possibly empty list of attributes (name and type). The ontology describes six concepts for the representation of cultural data : Actor, Person, Activity, Man_Made_Object, Image and Museum. An example of an attribute is *has_name* of type String in concept Person.

Concepts are related to each other by *binary roles* depicted by solid arcs. *Inheritance* (isa) links between concepts are depicted by dashed arcs. The fact that an actor performs an activity (instance of concept Activity) to produce a man made object is represented by roles *carried_out* and *produced*. We postulate that each role has an *inverse* which is depicted in the figure in parentheses (e.g. for role *carried_out* its inverse is *carried_out_by*).

The high level of detail in this ontology is due to the desire to enable the modeling of as many sources as possible in the art domain. We defer to Section 3 the detailed discussion of why we use ontologies, rather than XML, for the global data model. Additional assumptions about ontologies and their components, as well as additional explanations are presented in Section 3.

2.3 Derived ontologies

Each role has a $source$ and a $target$. Roles r_l, r_2 can be concatenated, provided the target of r_1 and the source of r_2 are compatible, taking into account the isa relationships. For example, the $source$ of role *carried_out* is concept Actor and its $target$ is concept Activity. A role r can also be concatenated with an attribute a, under similar conditions. Concatenations of role/attribute sequences are referred to as *role paths*, and can be viewed as *derived roles* (or *derived attributes*, if the last member is an attribute). For example, role path *carried_out.produced* defines a derived role from concept Actor to concept Man_Made_Object. Each derived role has a $source$ and $target$, and an inverse. For example, the inverse role path of *carried_out.produced* is *produced_by.carried_out_by*.

A *concept path* p is either of the form c, or of the form $c.r$, where c is a concept and r is a role path, where the $source$ of r is c or a subconcept of c. A concept path $c.r$ defines a *derived concept*, standing for "*the instances of the* $target$ *of* r *that can be reached from instances of its* $source$ *by following* r*"*. For example, Person.*carried_out.produced* defines a subconcept of Man_Made_Object and stands for all the instances of concept Man_Made_Object that are reached from an instance of

concept Person, following the role path *carried_out.produced*. Adding the derived concepts and derived roles to an ontology defines a *derived ontology* that properly contains the given one.

Reasoning about subset relationships between derived concepts, one can also derive isa relationships between them. For example, the derived concept Person.*carried_out.produced* is a subconcept of Activity.*produced* (in general, *suffixes* of derived concepts define more general derived concepts). Given a repository that conforms to a given ontology, extents of derived roles/attributes/concepts are uniquely defined, so a repository for the derived ontology is well-defined.

Our interest in the derived ontology is motivated by the fact that some sources may provide data only for derived concepts. The isa relationships in the derived ontology enable us to use these sources to provide answers in terms of the original concepts. For example, even if a source provides only information about Person.*carried_out.produced*, this allows us to obtain some instances of Man_Made_Object, although not necessarily all. Note that answers obtained from sources in the LAV approach are partial answers in any case.

2.4 Source Descriptions

To evaluate a user query expressed in terms of the ontology, we have to translate it into one or more queries on the XML sources. For this, we need to establish a correspondence between each source and the global ontology. This correspondence is described by a *mapping*, which is a collection of *mapping rules*. Mapping rules for XML are considered in [11], where several options of granularity for mapping rules between trees are mentioned (node-to-node, path-to-path, tree-to-tree, etc.). We have chosen the path-to-path approach. Specifically, the rules we use map a restricted sublanguage of XPath [10] (XPath *patterns* without predicates) to *schema paths* in the ontology. For example, the following rules map XML fragments of source *http://www.paintings.com* to the ontology of Figure 1 :

R_1: `http://www.paintings.com/Painter` as u_1 ← Person
R_2: u_1`/@name` as u_2 ← has_name
R_3: u_1`/Painting` as u_3 ← carried_out.produced
R_4: u_3`/@title` as u_4 ← has_title

A rule has the form $r : u/q \ as \ v \ \leftarrow p$, where r is the rule's label, u is either a URL or a variable, v is a variable, q is an XPath pattern and p is an ontology schema path. The variable v is *bound* (or defined) in the rule; if u is a variable, then it is a *use* of the variable, and we assume it is bound in some rule. The path q is called the *source path* of the rule and is an XPath pattern using only the *child* and *descendant* axis. The path p is a concept or role path in the ontology, called the *schema path* of the rule.

Rules define instances of derived concepts : XML fragments obtained by the rules are viewed as object instances of concepts. For example, mapping rule R_1 states that the elements of type `Painter` (bound in variable u_1), root elements of the XML documents in *http://www.paintings.com*, are instances of concept Person. Rules also define instances of (possibly derived) attributes and roles. For example rule R_2 specifies that

the XML attribute `name` corresponds to the concept attribute *has_name*. More precisely, it tells that for any instance x of concept Person, we can obtain a value for concept attribute *has_name* by following path `@name` from the root of x (remind that x is an XML fragment). In the same way, rule R_3 defines instances of the derived role *carried_out.produced* connecting each instance of concept Person obtained by rule R_1 to all fragments, instances of concept Man_Made_Object, obtained by R_3 when evaluating location path `Painting` (which is an abbreviation of `child::Painting`).

Rule concatenation: Variables serve as a glue, that allows us to concatenate mapping rules. They are also convenient in the formulation and implementation of query processing. Finally, they can be used to express a *semantic relationship* between different XML elements. For example, the following mapping contains four rules from the source *http://www.all-about-art.com* to our ontology:

S_1: `http://www.all-about-art.com/Painting` as v_1 ← Man_Made_Object
S_2: v_1`/id(@painter)` as v_2 ← produced_by.carried_out_by
S_3: `http://www.all-about-art.com/Painter` as v_2 ← Person
S_4: v_2`/@name` as v_3 ← has_name
S_5: `http://www.all-about-art.com/Sculpture` as v_1 ← Man_Made_Object

We see that one can reach v_2 (that binds instances of concept Person) by two different routes, and then continue one step down `@name` to obtain the value for attribute *has_name* for a person. In this example, the two routes, namely the *rule concatenation* $S_1 \circ S_2$ and the rule S_3 lead to the same element type. Observe also that S_1, and S_5 bind the same variable v_1, allowing us to relate different element types (`Painting` and `Sculpture`) that are semantically related, and have a similar structure.

2.5 Query Rewriting and Evaluation

The description of the global schema in terms of the ontology allows users to formulate structured queries, without being aware of the source specific structure. We illustrate querying with a simple sub-language of OQL defining *tree queries*. These queries allow no explicit joins and aggregate operators, but are sufficiently powerful to illustrate the issues of answering queries from source data, using the mapping rules. Given a user query q and a set of sources S, we need to get *all* possible answers that satisfy q. Since each source s may provide a subset of the possible answers for q, we need to evaluate query q over all (applicable) sources $s \in S$. Consider the query Q_1 below that requests *"the title of the objects created by Van Gogh"*:

Q_1: **select** c
from Person a,
a.name b,
a.carried_out.produced.has_title c,
where b = "Van Gogh"

In query Q_1, the variables and the paths that connect them form a tree. To evaluate this query over a source we need to rewrite it into a query that the source can answer. For that, we consider the binding paths of the query variables, and compare them to the

schema paths of the mapping rules for a given source. For example, if we consider the mapping for source *http://www.paintings.com*, we can bind the variable a (associated with Person) with rule R_1, the variable b with rule R_2 , and the variable c with the path obtained by the *concatenation* of rule R_3 with rule R_4 ; we obtain a *variable to rule binding* (in this example there is exactly one; in the general case there may exist a set of such bindings, even for one source.)

The variable to rule binding is now used to replace the variable binding paths in the query **from** clause by the location paths of the mapping rules to which they have been associated. For the example query Q_1, this replacement produces the query $Q_1(a)$ illustrated below.

$Q_1(a)$: **select** c
from `http://www.paintings.com/Painter` a,
a`./@name` b,
a`./Painting/@title` c,
where b = "Van Gogh"

If the source supports some general XML query language such as XQuery [7], then this query can be rewritten to the XQuery expression $Q_1(b)$ illustrated below, and sent to the source for evaluation.

$Q_1(b)$: **FOR** $a **IN** document("http://www.paintings.com")/Painter,
$b **IN** $a/@name,
$c **IN** $a/Painting/@title
WHERE $b = "Van Gogh"
RETURN $c

If the source supports only restricted query facilities such as XPath 1.0 [10], then the query needs to be rewritten into XPath. This is rather easy for this example, since we are requesting only the value of title elements, with some conditions on the paths leading to them. In the general case, an XQuery expression may need to be decomposed into several XPath queries, or the XPath query may return a larger tree that needs to be further filtered at the integration site to obtain the answer XML fragments (see [5] for more details).

Of course, such a rewriting of a user query to a source query should be attempted for each source. Some problems that arise when attempting to discover such rewritings are described later. However, assuming we have performed several rewritings, and obtained several answer sets, these have to be *merged* and presented to the user. The merge may be just a simple union, but often we can and should do better. If we obtain (possibly partial) information about the same entity from two sources, a *join* rather than union is called for.

To decide on, and to perform join, one needs to ensure that the entities represented by two elements, from different sources, are identical. This requires the use of *keys*, both in the global schema and in the sources. We will discuss keys in more detail in Section 3.

In some cases we cannot obtain a full answer from one source. Then, our rewriting algorithm [3] tries to find the *largest subquery*, the main subquery, of the user query that can be answered by the source. *Remainder subqueries* that cannot be answered by the source are identified, and processed against other sources. Results of the main and

remainder subqueries are then joined at the integration site. Assume the following query Q_2 that requests *"the title of the objects created by Van Gogh, as well as the name and the city of the museum where they are exposed"* :

Q_2: **select** d, f, g
from Person a, a.name b,
a.carried_out.produced c, c.has_title d,
c.located_at e, e.museum_name f, e.city g
where b = "Van Gogh"

The source *http://www.paintings.com* does not contain any mapping rule whose schema path matches variable's e binding path (*located_at*). Consequently we can only obtain incomplete answers from this source: for an object created by Van Gogh, we do not get the museum where it is located, nor its name and city. To get this information, we identify the remainder subquery asking for *"the name and city of the museum of man made objects"*, that is, the subquery that involves the variables c, e, f, g, and process it against the other sources. The variable c is included in the remainder subquery since it is the *join variable* between the two queries. We will glue the results of the main and the remainder subqueries by joining on c. The two queries are:

$Q_2(a)$: **select** d, **c**
from Person a, a.name b
a.carried_out.produced c,
c.has_title d
where b = "Van Gogh"

$Q_2(b)$: **select** f, g, **c**
from Man_Made_Object c,
c.located_at e,
e.museum_name f,
e.city g

There is one issue that needs to be considered here. The values bound to c are elements which might be of different type. How do we know which elements represent the same man made object? Here also the answer is the use of *keys*. If a key for man made objects is available in the source *http://www.paintings.com*, on which the first subquery was evaluated, and the same key is available in whatever source against which the remainder subquery was evaluated, then a join can be performed. Otherwise, the best we can do is to present to the user only the partial results from the first source. Note that if such keys are available, then we can optimize the remainder subquery by first evaluating the first subquery, then using the values bound to c, and the corresponding keys to generate the remainder subquery for the other source. Obviously, this specific query may return a very large set of answers, so optimization is called for. The issue of optimization is described with other algorithmic issues in [3].

3 The Integration Model

In this section two issues are addressed: (i) we discuss the choice of *ontology* as the *integration data model* : a user is presented with and poses queries on an ontology that represents a certain domain of interest; (ii) we introduce the notion of *keys* as an essential feature for the querying and integration of data from heterogeneous XML sources.

3.1 XML Integration

Most previous data integration projects and the relevant theory were presented in the framework of the relational model [26, 19, 22]. When the sources are relational, using the relational model for integration has some advantages; in particular the same query language can be used to define the *source-global schema* mappings. However, we are concerned with sources that use XML.

Which model to use for the global data model in such a context is not an easy decision to make. An obvious question is "*Why not XML?*" Indeed, Xyleme [11], MIX [20], Nimble [25] and Agora [21] use XML. Nimble and MIX employ the *global as view approach* (GAV) and are not directly comparable to our approach. The major advantage of this approach is that the mediator is defined as a traditional view (query) on different sources and user queries can be rewritten to source specific queries by unfolding the view definitions.

However, as in Xyleme and Agora, we use the *local as view* (LAV) approach [18] where this advantage does not exist and query rewriting becomes necessary.

In the Agora system [21] an XML global schema is used for the integration of relational and XML resources. This schema is represented by a *generic relational* schema that faithfully captures the XML document structure. Then resources are represented as relational views over this generic schema. User queries are XQuery expressions over the XML global schema, which are then translated to SQL queries in terms of the generic relational schema; these SQL queries that are evaluated by the sources. Although XML is used as the global data model, an extended use of the relational model is made for source-global schema mappings as well as for query rewriting.

Xyleme [11] defines a global schema as a simple XML DTD (called *abstract DTD*). As in our approach, XML resources are described using path-to-path mappings where absolute source paths (starting from the document root) are mapped to absolute paths in the abstract DTD. Source paths *and* abstract paths can only follow descendant relationships. In addition, by using XML DTDs for the global user's data model, it is not possible to distinguish entities from relationships, which leads to less precise mappings.

Another usage of ontologies to access sets of distributed XML documents on a conceptual level is presented in [15]. The integration is achieved by the assumption that all documents respect a canonical DTD derived from the ontology. The expressive power of XML and DTD grammars is compared to more powerful ontology models.

3.2 Conceptual Integration Model for XML

In the choice of an integration model, several problems have to be dealt with when the sources are XML : two entities of the same type can be represented by two elements with different structure even in a single source; different sources with the same structure may use different tags; the structure of the same elements may differ between sources; as aforementioned, a piece of data can be represented by an element in one source, an attribute in another.

We use *concepts* to represent entity types. Each has a unique name; XML elements from sources, with different tags, may be mapped to the same concept. Thus, a user sees one concept name, and does not need to worry about the diversity of tags used in

different sources. These are taken care by the mapping rules used by source authors to describe their source in the mediator. Each entity type in the global schema has a well defined structure. Sources may contain, for a corresponding entity type, only a subset of the entities of that type and also only a part of the entity's structure. This is commensurate with the LAV approach.

One might argue that a global data model based on XML could as well correctly take into account the above features [11]. In the following we briefly discuss the shortcomings of XML as a user-oriented, conceptual, data model. For simplicity, the discussion assumes DTD's are used to describe XML. Although in the XML Schema proposal [16], some of these shortcomings are partially solved, we believe that the following arguments merit attention and also help for a better understanding of XML Schema.

Let us first state some nice properties expected from a conceptual model. Such models distinguish between *entities* (or objects) and *values*, that may be simple or complex. Entities are related to each other by *relationships*, that may be constrained by various cardinality constraints. Often, the relationships are symmetric. Entities also have *attributes* that relate them to values. Entities are identified by *keys*. Finally, entity types may be related by isa relationships, that emphasize commonality of structure. All of these provide a user with a simple yet expressive model of a domain, with support for visual presentation of schemas, and both declarative and visual query languages.

In contrast, XML is a hierarchical data model, in which most relationships are directed from parent to child. There is no notion of a symmetric relationship, and unlike the case in the ODMG standard, there is no notion of inverse relationship to represent symmetry. Not all relationships in XML are hierarchical. The ID/IDREF mechanism allows for horizontal relationships, and these may be one-to-one or one-to-many. Still, these relationships are untyped and there is no mechanism that allows one to declare a symmetric relationship, that is represented by a pair of directional links.

Attributes and relationships Conceptual models usually distinguish between attributes and relationships. In XML there is a distinction between attributes, for which only a single occurrence is allowed (but an attribute may be multi-valued!), and elements, for which either one or any number of occurrences are allowed. However, there are no clear guidelines when data should be represented in one of these forms and not the other. As a matter of fact, it is well-known that XML serves two masters: the *document community* and the *database community*. The distinction between attributes and elements is straightforward for documents: The sectional units of a book are represented as elements. Extra information about the book, such as publication data, is represented as attributes. This distinction, however, is quite meaningless for data. It is not clear at all that the notions of attributes and elements in XML correspond to attributes and relationships in conceptual models.

***Isa* relationships** Type inheritance described by isa relationships - not supported by DTD's but introduced in the XML Schema proposal-, conveniently summarizes similarity of structures. In a database context, these relationships also represent containments of sets of entities. If Person isa Actor, then if a query asks for actors with certain properties, and a source offers information about Person, it makes sense, in the LAV

approach, to generate a query on this source to retrieve the persons that have these properties. These form a subset of the answer to the query.

In addition to its simplicity for representing common structure and its querying power (a query can request information about a concept or a subconcept there of) this feature also helps to overcome terminological differences between sources.

Symmetric relationships With XML, a symmetric binary relationship is modeled by an asymmetric parent-child relationship between two elements. A source can choose either of the two element types as being the parent of the other. For example, Painting is a child of Painter in the source *http://www.paintings.com*. Another source might as well choose to invert this relationship, so Painting becomes a parent of Painter. Of course, binary relationships may as well be represented by the XML ID/IDREF mechanism. However, as of today, there is no established methodology that directs XML authors to generate XML documents in some *normal form*, in which entities are represented as top-level elements, and relationships between them as horizontal references. A first step in such a direction has recently been made in [6]. However, currently, in an integration scenario, we have to accept all possibilities.

If XML was chosen as the global data model, and element nesting as the representation of relationships, this problem of inverse hierarchies between a global schema and a source would force the use of the ancestor axis of XPath in mapping rules. This would not only render the source description complex but also significantly complicate query processing. Our symmetric representation of relationships, where each relationship has an inverse, and either direction can be used, is much more intuitive. It also allows us to view sources as simple hierarchies, whose description requires only the child and descendant axis of XPath.

In summary, the use of a simple conceptual model for the global data model has the advantages of simplicity and expressive power. While it can possibly be viewed as some veneer for an XML-based data model, we believe that these advantages are significant, and should not be given up.

3.3 Semantic Keys for XML

Keys are essential for data integration. In fact they are the only way to decide whether two XML fragments of two different sources are identical or not when considered as concept instances.

The XML resources we are dealing with are heterogeneous and autonomous. Consider for example our two sources *http://www.paintings.com* and *http://www.all-about-art.com*. The first source uses the `ID` attribute `title` to identify a painting (there might not exist two paintings of the same title). In the second source, `ID` attributes are used to identify museums and painters. Paintings are *not* identified by their title (there might exist two paintings with the same title in the second source). So, two distinct sources might provide us with different local key definitions and, for integration, we have to define keys at the global schema.

Key paths and identity We define a *key on concept c* as a set of role paths with source c, called *key paths*. While, in general, a concept may have zero, one or more keys, we assume for simplicity that each concept has exactly one key, denoted $key(c)$. Observe that the key of a concept might be empty, which means that instances of this concept cannot be identified. We also assume for simplicity that isa-related concepts all share the same key.

For example, persons and actors are identified by their names: key(Actor) = key(Person) = $\{has\_name\}$. The key of concept Man_Made_Object is {*has_title, produced_by.carried_out_by*}, which means that man made objects are identified by their title and the artist who produced them. Instances of concept Activity are not identifiable: key(Activity)=$\emptyset$.

Calculating key values In order to decide whether two instances α and β of some concept c are identical, we have to calculate and compare their *key values*. These values can be obtained by considering $key(c)$ as a *key query* evaluated on α and β. If all key paths $key(c)$ are attribute paths, this query simply follows all key paths starting from the corresponding instance. In the case of key paths ending in some concept, the query is obtained by replacing all target concepts by their key queries. For example, the following key query returns the key value for some instance of concept Man_Made_Object represented by the variable α:

select t, n
from α.has_title t,
α.produced_by.carried_out_by p,
p.has_name n

A concept c is a *joinable concept* if and only if it has a non-empty key and all concepts c' which are the target of a key path in $key(c)$ are joinable concepts. It is easy to see that key values can only be calculated for instances of joinable concepts (our notion of joinable concept is very similar to the notion of *value-representable* concept in [23]).

As in [2] and other semi-structured query languages, we assume that all attributes and roles in the global data model may be multi-valued and optional. This means that the result of a key query (the key value of an instance) is a *set of tuples* and we have the choice of defining the identity of an object by the whole set of tuples or just one tuple. As in most semi-structured query languages, we have chosen the second solution and define two fragments to be identical, if their key values are not disjoint.

Like user queries, key queries have to be rewritten w.r.t. the corresponding source mapping. This rewriting consists essentially in replacing schema paths by the rules' location paths that can be applied to some XML fragment of the source (similar to XML keys as defined in [9]). There might exist zero, one or several such rewritings which only return a *subset* of the "complete" key value (it is for example possible that our two sources identify the same artist by two different names). In other words, by the LAV approach, we might miss some tuples of the *real* key value of α in order to conclude that α is equal to some other instance β.

Observe that, unlike user queries, a key query is issued against a *single* source. If we do not obtain a complete binding for a source (i.e. some variables are not bound),

key queries are not decomposed. We will end our discussion about keys with one other issue concerning the rewriting of key queries.

An existing mapping might not allow the system to find the instances of some concept c but be sufficient to find the key values of these instances. For example there might be no rule for binding path α.*produced_by.carried_out_by* p of the previous key query, but some other rule returning the name of the artist who has produced the artifact.

In this case, the key query might be *unfolded* by replacing all key paths p with target concept c' by the set of paths $p.q$ where q is a key path in $key(c')$. The previous key query would then be rewritten into the following query :

select t, n
from α.has_title t,
α.produced_by.carried_out_by.has_name n

Observe that this rewriting is correct since the key of the replaced concept contains only one key path. If an artist would have been defined by a first name x and a last name y, unfolding would have relaxed the condition that x and y correspond to the same person.

4 Conclusions

This paper addressed the problems encountered, and the design choices made in the context of the integration of heterogeneous XML sources in the C-Web project. In particular we justified our choice of an ontology instead of XML as the mediator data model, and discussed the benefits we draw out of it.

In the context of the semantic Web, an applicable data model that could be used to represent our ontologies is RDF Schema [8] or DAML+OIL [12] which define all (DAML+OIL) or almost all (RDF Schema lacks the notion of inverse roles) the properties that we have identified to be necessary in the context of XML data integration. Authors in [24] advocate the need to develop a unified model for XML and RDF in order to bridge the gap between the syntax (XML) and the semantics (RDF) for Web applications. Our approach fits well into their context.

Another important problem raised by our cultural application example, is the exploitation of *semantic metadata*. By this, we denote information concerning the contents of a source, that is not present in the actual data, and thus cannot be represented by the *source-global schema* mappings. As far as we are aware of, all data integration projects, assume that all the information necessary for query processing is available in those mappings. The presence of other semantic metadata can be taken into account in our system in two different ways : first, to process queries that request information that is not present in the actual data and, secondly, to filter the sources which do not satisfy the query conditions before starting the rewriting process. We are working towards the choice of a language to define such semantic metadata, as well as an algorithm that exploits the metadata as well as the source-global schema mappings for query rewriting.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data On the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, October 1999.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.
3. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. XML Data Integration. Technical report, CEDRIC/CNAM, Paris, France, January 2002.
4. B. Amann, C. Beeri, I. Fundulaki, M. Scholl, and A-M. Vercoustre. Mapping XML Fragments to Community Web Ontologies. presented at WebDB, May 2001. http://cedric.cnam.fr/PUBLIS/RC255.pdf.
5. B. Amann, C. Beeri, I. Fundulaki, M. Scholl, and A-M. Vercoustre. Rewriting and Evaluating Tree Queries with XPath. Technical report, CEDRIC/CNAM, Paris, France, November 2001.
6. M. Arocenas and L. Libkin. A Normal Form for XML Documents. To appear, PODS 2002.
7. S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, J. Simon, and M. Stefanescu (eds.). XQuery 1.0: An XML Query Language. W3C Working Draft, December 2001. URL: http://www.w3.org/TR/xquery.
8. D. Brickley and R.V. Gupta (eds.). Resource description framework (RDF) schema specification 1.0. W3C Candidate Recommendation, March 2000.
9. P. Buneman, S.B. Davidson, W. Fan, C.S. Hara, and W. Chiew Tan. Keys for XML. In *Proc. WWW10*, pages 201–210, 2001.
10. J. Clark and S. DeRose (eds.). XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999. http://www.w3c.org/TR/xpath.
11. S. Cluet, P. Veltri, and D. Vodislav. Views in a Large Scale XML Repository. In *Proceedings VLBD*, Rome, Italy, September 2001.
12. D. Connoly, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. Daml+oil (march 2001) reference description. W3C Note 18 December 2001, mar 2001.
13. CWeb. The CWeb (Community Webs) Project. http://cweb.inria.fr/.
14. M. Doerr and N. Crofts. Electronic organization on diverse data - the role of an object oriented reference model. In *Proceedings of 1998 CIDOC Conference*, Melbourne Australia, October 1998.
15. M. Erdmann and R. Studer. How to structure and access XML documents with ontologies. *Data Knowledge Engineering*, 36(3):317–335, 2001.
16. D. Fallside. XML Schema Part 0: Primer. W3C Recommendation, May 2001. http://www.w3.org/TR/XML-schema-0.
17. I. Fundulaki, B. Amann, C. Beeri, and M. Scholl. STYX : Connecting the XML World to the World of Semantics. Demonstration at EDBT'2002.
18. A.Y. Levy. Answering queries using views: a survey. *VLDB Journal*, 2001.
19. A.Y. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of VLDB*, Bombay, India, September 1996.
20. B. Ludäscher, Y. Papakonstantinou, and P. Velikhov. A Framework for Navigation-Driven Lazy Mediators. In *Proc. of WebDB*. Philadelphia, USA, 1999.
21. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries over Hterogeneous Data Sources. In *Proceedings of VLDB*, Rome, Italy, September 2001.
22. R. Pottinger and A. Levy. A Scalable Algorithm for Answering Queries using Views. In *Proc. VLDB*, Cairo, Egypt, September 2000.
23. K.-D. Schewe, J. W. Schmidt, and I. Wetzel. Identification, genericity and consistency in object-oriented databases. In *Database Theory–ICDT '92*, pages 341–356, 1992.

24. J. Simeon and P. Patel-Schneider. The Ying/Yang Web : XML Syntax and RDF Semantics. To appear in Proc. of WWW Conference, 2002.
25. Nimble Technology. URL: http://www.nimble.com/.
26. J.D. Ullman. Information integration using logical views. In *Proc. ICDT*, pages 19–40, Delphi, Greece, 1997.
27. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, pages 38–49, March 1992.