Querying Heterogeneous XML Sources through a Conceptual Schema

Sandro Daniel Camillo, Carlos Alberto Heuser[1] and Ronaldo dos Santos Mello[2]

[1] Universidade Federal do Rio Grande do Sul
Instituto de Informática
Caixa Postal 15064 Av. Bento Goncalves, 9500 - Bloco IV - Prédio 43412
CEP: 91501-970 - Porto Alegre - RS - Brazil
e-mail: camillo,heuser@inf.ufrgs.br
[2] Universidade Federal de Santa Catarina
Centro Tecnológico Departamento de Informática e de Estatística
Campus Universitário Trindade
Caixa Postal 476 CEP: 88040-900 - Florianópolis - SC - Brazil
e-mail: ronaldo@inf.ufsc.br

Abstract. XML is a widespread W3C standard used by several kinds of applications for data representation and exchange over the *web*. In the context of a system that provides semantic integration of heterogeneous XML sources, the same information at a semantic level may have different representations in XML. However, the syntax of an XML query depends on the structure of the specific XML source. Therefore, in order to obtain the same query result, one must write a specific query for each XML source. To deal with such problem, a much better solution is to state queries against a global conceptual schema and then translate them into an XML query against each specific data source. This paper presents *CXPath (Conceptual XPath)*, a language for querying XML sources at the conceptual level, as well as a translation mechanism that converts a *CXPath* query to an *XPath* query against a specific XML source.

1 Introduction

XML is a common W3C standard for semi-structured data representation, being used by *web* applications for data exchange [1, 2]. In this paper, we focus on the problem of performing queries on heterogeneous XML data sources related to some specific domain. Examples of applications that require solutions to such problem are federated information systems [3] and semantic *web* applications [4]. In this context, the main challenge is to deal with different XML representations of semantically equivalent data. To manage this problem, it is necessary to provide [5]:

1. a global (unified) representation for all XML source schemata, avoiding that the user must know the schema of each source to formulate queries;
2. a translation mechanism to convert a global query into queries in accordance to the schema of each XML source, avoiding that several queries must be formulated against each XML source;

3. an instance integration mechanism to unify query results coming from several XML sources into a single query result in accordance to the global schema.

This paper particularly focuses on the second point. We propose a mechanism to deal with the problem of translating global queries to XML sources. This mechanism is supported by:

- a *global schema* that is a *conceptual abstraction* of several XML schemata (such a conceptual schema is constructed by a semantic integration process that is out of the scope of this paper [6]);
- a *language* to formulate queries over this conceptual model, called *CXPath (conceptual XPath)*;
- *mapping information* for concepts at the conceptual schema level into concepts at the XML level.

We adopt a *conceptual* model for defining a global schema instead of the *logical* XML model, because the XML model is unable to abstract several XML schemata at the same time. This is due to the inherent hierarchical nature of XML data. The schema of an XML source must define a specific hierarchical structure for representing relationships between XML elements. Therefore, heterogeneous XML sources belonging to a same application domain may define different hierarchical representations for the same many-to-many relationship. Considering the example of a library, a many-to-many relationship between *article* and *author* may be represented by two different XML schemata: (i) one *article* (ancestor element) associated to many *authors* (descendent elements); or (ii) one *author* (ancestor element) associated to many *articles* (descendent elements). A global XML schema would be able to represent only one of these possibilities. However, a conceptual model directly represents many-to-many relationships (many *articles* associated to many *authors*, and vice-versa), without imposing a strict navigation order between them.

This paper defines the query language used at the conceptual level. During the definition of this language we had two objectives in mind: (i) to simplify the process of translation of a query at the conceptual level to a query at the XML level; and (ii) to simplify the learning process of the language to those acquainted with the query languages of the XML standard (*XPath* and XQuery) [7].

One way to simplify the process of translation between two languages is to narrow the semantic gap between these languages. The target language *XPath 1.0* is based on the concept of *path expression* for navigating through the hierarchical relationships of an XML instance. Thus, we have chosen for the conceptual level a language that is also based on the concept of path expression, and discarded languages like *entity-relationship* algebras [8, 9] and *SQL* [10] that are based on the *join* operation.

Examples of query languages that are based on the concept of *path expression* are OQL for the object-oriented model [11], *Lorel* for semi-structured data [12] and *XPath* for the XML model. As we aim at simplifying the process of learning of the proposed language for those acquainted with the XML, we chose to base the conceptual level query language on the *XPath 1.0* language [13].

We defined *CXPath (Conceptual XPath)* as a variant of *XPath 1.0* [13]. The main difference between *CXPath* and *XPath 1.0* is that, whereas in *XPath 1.0* a path expression specifies the navigation through hierarchical relationships, in *CXPath* a path expression specifies the navigation through a web of relationships among entities in a graph-based conceptual base.

The translation process of a *CXPath* query to an *XPath 1.0* query is based on mapping information associated to each concept and relationship in the conceptual schema. A mapping information is an *XPath path expression* that maps each conceptual construct into an XML source. For a concept at the conceptual schema, this expression specifies how to reach the corresponding XML construct (element or attribute) in the source. For a relationship at the conceptual schema, this expression specifies the traversal path in terms of the XML source that corresponds to the traversal of the relationship at the conceptual level. The use of the same language as the target language and as the language for expressing mapping information further simplifies the process of query translation from the conceptual into the XML level.

The *CXPath* language and the proposed translation mechanism may be applied to the context of mediation-based systems that provides integrated access to heterogeneous XML sources related to a same application domain [14–17]. Such systems are responsible for the semantic integration of XML schemata and global query processing to XML sources. A semantic integration module generates the conceptual schema and the mapping information. A query processing module receives a *CXPath* query based on the conceptual schema, translates it to *XPath* queries for each XML source and performs the integration of query results.

This paper is organized as follows. Section 2 presents the *CXPath* language: its syntax, the considered conceptual model and some query examples. Section 3 presents the adopted approach to represent the mapping information between the conceptual and the XML levels. Section 4 describes the *CXPath* into *XPath* translation mechanism, providing some examples of query translation. Section 5 is dedicated to the conclusion. The Appendix contains the *CXPath* grammar.

2 CXPath

CXPath (Conceptual XPath) is a language for querying a conceptual base that results from the integration of XML sources. The considered conceptual model is based on ORM/NIAM [18] and defines *concepts* and *relationships* between concepts. We use this model because it is the model used in the schema integration module that we have developed in previous work [19, 20]. However, the language and the translation process do not depend on this particular data model and should be easily adapted to other variants of the ER model. Two types of concepts are supported in this model: lexical and non-lexical. A *lexical concept* models information that has an associated textual content, like #PCDATA elements and attributes. A *non-lexical concept* models information that is composed by other information, like elements that have sub-elements.

An *association relationship* is a binary relationship with cardinality constraints. Relationship names and roles may be optionally defined on it.

Figure 1 shows a conceptual schema for a domain of bibliographical references. Article is a non-lexical concept (solid rectangle), being composed by information about *Title, Year* and *Person* (authors and referees). *Title, Year* and *Name* are lexical concepts (dotted rectangles), holding textual information. An association relationship is defined between *Article* and *Year*, denoting that an article has one associated year information, and a year is associated to one or more articles. A named association relationship *referee* is defined between *Article* and *Person*, denoting that a person may be a referee of several articles.

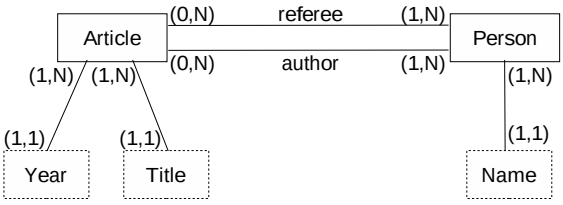

Fig. 1. An example of a conceptual schema

A *CXPath* query specifies an *XPath 1.0*-like *path expression* to reach information that must be retrieved, with optional selection predicates that may be defined on this path. Although they are based on the *XPath 1.0* syntax, *CXPath* and *XPath 1.0* have different semantics because they are applied to different data models. *XPath 1.0* is suitable for navigating in XML documents that are tree-based structures, whereas *CXPath* is suitable for navigating in the conceptual base that is a graph based structure. These differences in the data model impose several differences on the languages, as described below.

- *concept names instead of element names*
  In *XPath*, XML elements are referred by their labels (element names). In *CXPath*, concept names are used instead of element names. A concept name refers to all instances of that concept in the conceptual base.

- *root element and absolute path expressions*
  An XML instance has a *root element*. An *XPath* expression that begins with slash (an *absolute* path expression) starts navigating from the root element. The conceptual base does not have this root element. The *CXPath* semantics for the absolute path expression is that the navigation source is the entire conceptual base, i.e. the navigation may start at any concept in the conceptual base.
  Example 1. Retrieve all instances of concept Article:

```
/Article
```

– *navigation operator (slash operator) and relative path expressions*
In *XPath*, the slash operator, when not appearing at first place in a path expression, has the semantics of "navigate to the child elements". As the conceptual base is a graph and not a tree, this semantics has been changed to "navigate to the related elements".
Example 2. Retrieve all instances of concept Title that are related to the instances of concept Article:

```
/Article/Title
```

Further, in *XPath* a path expression may contain other path expressions. For example, the path expression `/Article[Year = "2003"]/Title` contains the path expression `Year`. This is a *relative* path expression. In *XPath* the context of evaluation of a relative path expression is the set of child elements of an Article instance. As with the slash operator, in *CXPath* the context of evaluation in which a relative path expression is evaluated has been changed from "all child elements" (*XPath*) to "all related elements" (*CXPath*).
Example 3. Retrieve all instances of concept Title that are related to those instances of concept Article that are related to instances of concept Year with value 2003:

```
/Article[Year = "2003"]/Title
```

In this path expression, the context of evaluation of the path expression `Year` is the set of all concepts related to *Article*. Thus, `Year` refers to the set of all *Year* instances that are related to *Article* instances.

– *qualified navigation operator - relationship name*
When more than one relationship relates two concepts, the identification of a specific relationship to be navigated may be needed. For example, the expression `/Article/Person` refers to all instances of Person that are related to instances of Article, thus including authors and referees. If a specific relationship (author or referee in the example) is to be navigated, the name of the relationship may appear in curly brackets after the slash operator.
Example 4. Retrieve all names of persons that are authors of articles produced in the year 2003:

```
/Article[Year = "2003"]/{author}Person/Name
```

In this example, the qualified navigation operator `"/{author}"` specifies the navigation to only those instance that are related through the relationship named author.

– *qualified navigation operator - role name*
In the case of self-relationships, the *role name* may be needed to indicate the direction of navigation.
Example 5. Consider the conceptual schema depicted in Figure 2.
The following *CXPath* query retrieves the names of all husbands in current marriages:

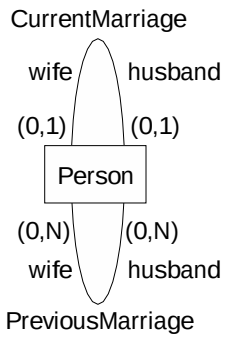

Fig. 2. Examples of self-relationships in a conceptual schema

```
/Person/{CurrentMarriage.husband}Person/Name
```

– *hierarchical operators*
  All *XPath* operators that refer explicitly or implicitly to the navigation to ancestors (e.g.: "..” or "ancestor” operator) or descendents (e.g.: "//” operator) do not appear in *CXPath*. As stated before, just the *XPath* slash operator remains, but its semantics has been changed from "navigate to the child elements” to "navigate to the related elements”.
  The production rules of the *CXPath* grammar are shown in the Appendix. This grammar does not contain all language operators but only those that are relevant to the understanding of the proposed approach.

3 Mapping Information

In order to translate a *CXPath* query to an *XPath* query, mapping information between the conceptual and XML schemata is required. In the literature of database integration, two approaches are usually employed to define mapping information: a *mapping catalog* or *views* [5]. In the mapping catalog approach, constructs in the global schema are mapped into constructs in the logical schema. In the view approach, a query statement (view) for each global concept and relationship is defined describing which corresponding data must be retrieved from local sources. We chose the view approach. For each construct at the conceptual level an *XPath* query that retrieves the set of instances in the XML source is defined. This approach was chosen because it simplifies the translation mechanism, as each reference to a concept in a *CXPath* query is directly replaced by the corresponding *XPath* expression.

XPath expressions are associated to each *concept* and to each *traversal direction of each relationship* in the conceptual schema. One *XPath* expression is associated to each XML source.

To exemplify the mapping between *CXPath* and *XPath*, consider that the conceptual schema of Figure 1 is a unified abstraction of the XML sources 1 and 2, shown at Figure 3 (a) and (b), respectively.

Table 1 depicts the mapping information from concepts and relationships of the conceptual schema to the XML sources 1 and 2.

Table 1. Mapping information to the conceptual schema of Figure 1

Concept/Relationship	*Mapping to XML Source 1*	*Mapping to XML Source 2*
Article	/references/paper	/publications/author/article
Person	/references/paper/* [local-name(.)='author' or local-name(.)='referee']	/publications/author
Title	/references/paper/@title	/publications/author/ article/heading
Year	*not applicable*	/publications/author/ article/year
Name	/references/paper/* [local-name(.)='author' or local-name(.)='referee']/name	/publications/author/name
Article$^{\{author\}}$→Person	Author	..
Article←$^{\{author\}}$Person	[local-name(.) = "author"]/..	article
Article$^{\{referee\}}$→Person	Referee	*not applicable*
Article←$^{\{referee\}}$Person	[local-name(.) = "referee"]/..	*not applicable*
Article→Title	@title	heading
Article←Title	..	..
Article→Year	*not applicable*	year
Article←Year	*not applicable*	..
Person→Name	name	name
Person←Name	..	..

The mapping information from a *concept* to an XML source is an absolute *XPath* expression that defines the hierarchy of elements that must be searched to reach the corresponding element or attribute. The mappings from *Article* and *Title*[3] to the XML source 1 are examples (see Table 1). When a concept has more than one corresponding element or attribute in an XML source, the *XPath* mapping expression must refer to the union of the corresponding elements. One example is the concept *Person* that matches *author* and *referee* in XML source 1 (the `local-name` function returns the name of an element). Another example is the mapping of the concept *Name* to the XML source 1.

The mapping information from a *relationship* C_1-C_2 to an XML source is a *relative XPath expression* that denotes how to navigate from the element corresponding to concept C_1 to the element corresponding to concept C_2 (and vice-versa) in this source. Mappings for both traversal directions are defined. The mapping of the relationship *Person-Name* to the XML source 2 is an example. In this source, the element `name` (corresponding to concept *Name*) is a sub-element of the element `author` (corresponding to concept *Person*). The Person→Name navigation direction is mapped to this XML source by the relative path `"name"`. This path expression allows the navigation to *name* elements,

[3] A reference to an attribute A in *XPath* is denoted by *@A*

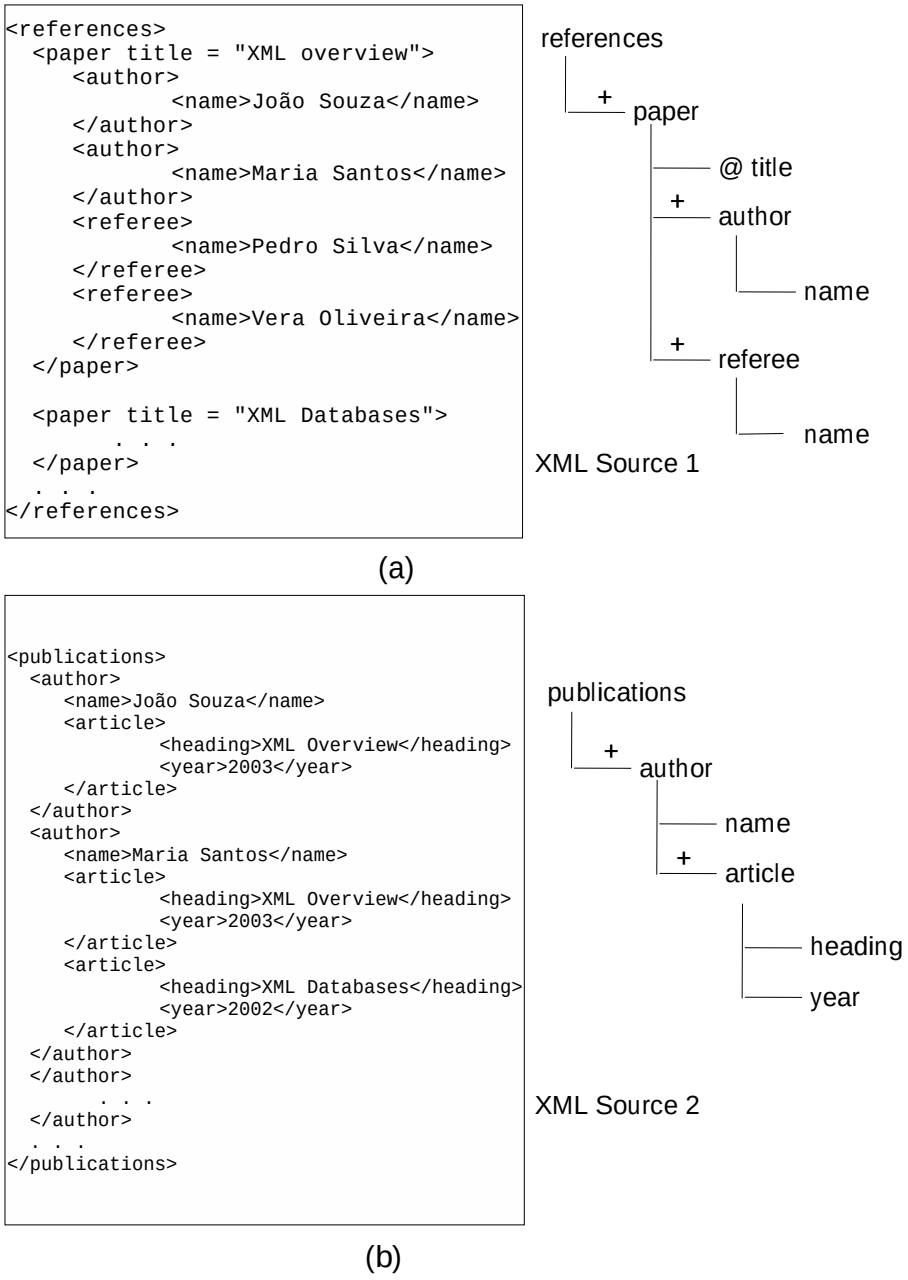

Fig. 3. An example of two XML sources (a) and (b): document and schema

when the context is an `author` element. Analogously, the *Person*←*Name* navigation direction is mapped to *"..."*. This path expression allows the navigation in the reverse order, i.e., to `author` elements, when the context is a `name` element.

A special case on the mapping definition is a concept C that has more than one corresponding element in an XML source S, one for each relationship R_i it takes part. A mapping from a relationship $C \rightarrow^{\{Ri\}} C_x$ to S must select only those elements in S that correspond to C and take part of relationship R_i. One example is build by concept *Person* and the relationships *referee* and *author* when source 1 is considered. In this source the concept *Person* maps into two elements, `author` and `referee`. A *Person* instance that maps to an `author` element takes part of the *author* relationship, whereas a *Person* instance that maps to an `referee` element takes part of the *referee* relationship. If a navigation starts at an element representing *Person*, the *XPath* expression must assure that the correct element (in the example, an `author` or a `referee`) is used as the source of the navigation. This is achieved by the *XPath* function local-name(), that results in the name of an element. Then, the mapping from $Article^{\{referee\}} \leftarrow Person$ to XML source 1, as shown in Table 1, is the following *XPath* expression:

```
[local-name(.) = "referee"]/..
```

This predicate guarantees that the source element of the navigation is an element of type referee, before navigating to the *paper* element (that represents to concept *Article*).

In general, *XPath predicates* must be defined in a mapping information every time a specific semantic intention must be checked to provide a correct relationship mapping. To exemplify another situation, suppose that a concept *Person* has two relationships with a concept *Address* named *Home* and *Office* (home and office addresses of a person, respectively). If an XML source defines an element `person` with two sub-elements `address`, where the first one is the home address, the mapping from the relationship $Person^{\{home\}} \rightarrow Address$ to this source should be `Address[position() = 1]`. The *XPath* function `position()` verifies if the occurrence of the sub-element `Address` to be searched is equal to 1.

We assume that the mapping information for the XML sources is semi-automatically generated by a *semantic integration module* when the XML schemata of these sources are integrated. Such a module is discussed elsewhere [19, 20].

4 CXPath to XPath Translation

The *CXPath* to *XPath* translation applies a *rewriting strategy*, i.e., each reference to a concept as well as each relationship traversal found in the *CXPath* expression are replaced by their corresponding mapping information to the considered XML source.

The translation process of an input *CXPath* query to an output *XPath* query for an XML source basically proceeds as follows:

- The input is analyzed from left to right.
- Tokens found in the input are written to the output, with some exceptions:
  - the first slash of an absolute *CXPath* expression is not written to the output;
  - each *concept* found in the input will be substituted by an *XPath* expression as shown below;
  - a qualified navigation operator in the form /{*Rel*} or /{*Rel.role*} is substituted in the output by the slash operator.
- When the first concept C_a of an *absolute CXPath* expression is found, the *XPath* expression that maps the concept C_a to the source is written to the output. This is always an absolute expression.
- Each other concept C_r found in the input has a context, i.e., it is *relative* to some other concept C_c. When such a relative concept C_r is found in the input, the *XPath* expression that maps the traversal of the relationship from the context concept C_c to the relative concept C_r ($C_c \rightarrow C_r$) is written to the output. If the concept C_r is preceded by an expression in the form {*Rel*} or {*Rel.role*}, this expression is used to select which mapping expression will be used.

In the following, some examples of query translations are presented. Such examples take into account the mapping information shown in Table 1.

Translation 1. The *CXPath* query defined in Figure 4 retrieves titles of articles from the 2003 year. This translation is performed to XML source 2.

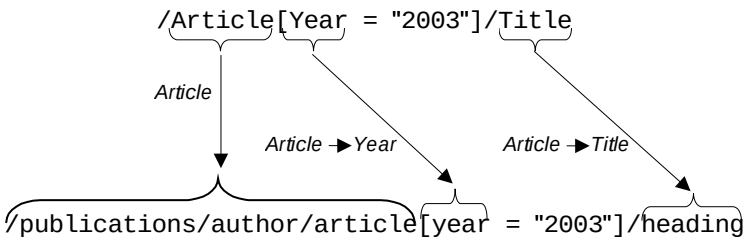

Fig. 4. Example of *CXPath* to *XPath* query translation (I)

The left-to-right analysis of the *CXPath* expression starts with the concept *Article*. As *Article* is the first concept in an absolute *CXPath* path expression, the *XPath* expression `/publications/author/article` that maps the concept *Article* is written to the output. In the sequence, the construct "[" is written to the output and the concept *Year* is found. *Year* is a relative concept and its context is *Article*. Thus, the mapping for the relationship navigation *Article*→ *Year* (year) is written to the output. The remaining tokens of the predicate are written to the output, as well as the construct "`/`" that follows. Finally, the relative

concept *Title* is found. The context of this concept is still *Article*. Therefore, the mapping of the relationship navigation *Article*→*Title* (heading) is written to the output.

Translation 2. A more complex example of query translation is shown in Figure 5. It illustrates the translation of a *CXPath* query that contains a qualified navigation expression (/{referee}). This *CXPath* query retrieves titles of refereed articles. This query is translated to the XML source 1.

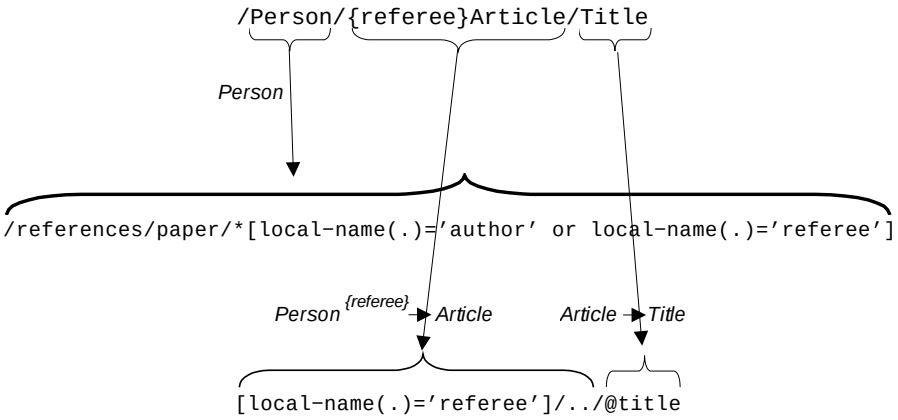

Fig. 5. Example of *CXPath* to *XPath* query translation (II)

The query translation starts with the mapping of the absolute *CXPath* path expression `/Person`. When the relative concept *Article* is found, the context concept is *Person*. As the concept *Article* is preceded by the qualification `referee` the mapping for the relationship navigation $Person^{\{referee\}} \rightarrow Article$ is written to the output.

Notice in this example the use of the local-name(.) function as described in Section 3.

Translation 3. This last example of query translation, shown at Figure 6, illustrates the recursive translation of an embedded absolute *CXPath* query. The presented query retrieves titles of articles that were written in the same year of the article with title *"XML Overview"*. This query is translated to the XML source 2.

Initially, the absolute *CXPath* path expression `/Article` is identified and the mapping of the concept *Article* is performed. In the sequence, the "[" operator is written to the output and the relationship *Article*→*Year* is identified and mapped to XML source 2. At this point, the embedded absolute *CXPath* expression `/Article[Title = "XML Overview"]/Year` is identified. Thus, the translation process is recursively applied to it.

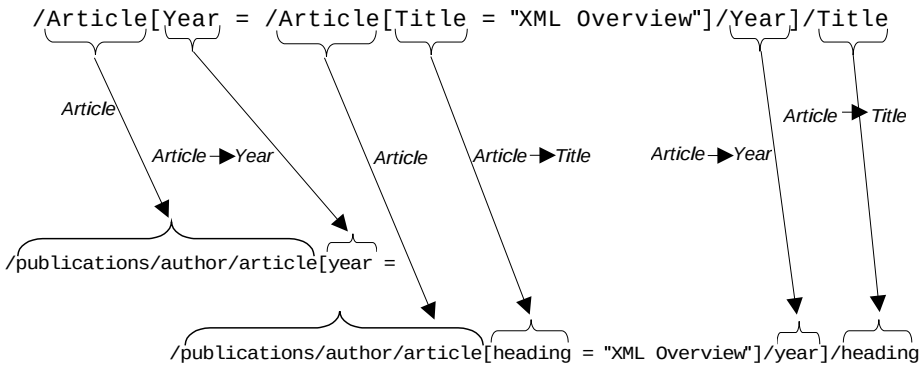

Fig. 6. Example of *CXPath* to *XPath* query translation (III)

5 Conclusion and Future Work

This paper proposes an approach to deal with the problem of querying several XML sources through a conceptual schema that unifies the schemata of these sources. The contributions of this work are the following:

- *CXPath*, an *XPath* based language for building queries over a conceptual schema that is an abstraction of several XML sources;
- a strategy for defining mapping information related to concepts and relationships of the conceptual schema that is also based on *XPath* views; and
- a translation mechanism from *CXPath* queries to *XPath* queries that deals with the problem of schematic heterogeneity when a high level query over a conceptual schema must be converted to a query over an XML source.

However, other issues related to the problem of querying heterogeneous data sources through a global unified schema are still open.

In this paper, the translation process considers a single XML source. The problem of *query decomposition*, i.e., the problem of deciding how a query against a global schema that unifies several XML sources is translated to queries against those several sources must be investigated.

The *XPath* query generated by the translation of a *CXPath* query is not optimized. For example, the *XPath* query which results from the translation process in Figure 5:

```
/references/paper/*[local-name(.)='author' or local-name(.)=
       'referee'][local-name(.)="referee"]/../@title
```

could be optimized to:

`/references/paper/referee/../@title.`

Another future work is the application of W3C *XQuery* [7] or a variant of it as the language for querying the conceptual schema. *XPath* itself is more a language to refer to parts of XML documents than properly a query language able to build new XML instances. This role is played in the W3C standard by *XQuery*. As *XQuery* embeds *XPath*, the use of *XQuery+CXPath* as the conceptual level language should be investigated.

References

1. *Extensible Markup Language* - XML. Available at: *http://www.w3.org/XML*.
2. Bradley, N. The *XML Companion*. Addison-Wesley Longmann Limited, 2ed., 2000. 435p.
3. Busse, S.; Kutshce R.; Leser, U.; Weber, H. *Federated Information Systems: Concepts, Terminology and Architectures*. Technical Report 99-9, Universitt Berlin, 1999.
4. *W3C Semantic Web*. Available at: *http://www.w3.org/2001/sw*.
5. Elmagarmid, A.; Rusinkiewicz, M.; Sheth, *A. Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann Publishers, Inc., 1999. 413p.
6. Mello, R.S.; Heuser, C.A. A Bottom-Up Approach for Integration of XML Sources. In: *International Workshop on Information Integration on the Web (WIIW'2001)*, pp. 118-124, Rio de Janeiro, Brazil, apr, 2001.
7. *XQuery 1.0 and XPath 2.0 Data Model*. Available at: *http://www.w3.org/TR/query-datamodel*.
8. Parent, C.; Spaccapietra S. An Entity-Relationship Algebra. *In: 1st International Conference on Data Engineering (ICDE)*, pp. 500-507, Los Angeles, USA, IEEE Computer Society, apr, 1984.
9. Campbell, D. M.; Embley, D. W.; Czejdo, B. D. A Relationally Complete Query Language for an Entity-Relationship Model. In: *4th International Conference on Entity-Relationship Approach*, pp. 90-97, Chicago, USA, North-Holland, oct, 1985.
10. *ANSI/ISO/IEC 9075-2* - Information Technology - Database Languages - SQL. 1999.
11. ODMG Home Page. Available at: *http://www.odmg.org*.
12. Abiteboul, S.; Quass, D.; McHugh, J.; Widom, J.; Wiener, J. L. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, v.1, n.1, 1997, pp. 68-88.
13. *XML Path Language - XPath*. Available at: *http://www.w3.org/TR/xpath*.
14. Rodriguez-Gianolli, P.; Mylopoulos, J. A Semantic Approach to XML-Based Data Integration. In: *20th International Conference on Conceptual Modeling (ER'2001)*, pp. 117-132, Yokohama, Japan, Springer-Verlag, nov, 2001.
15. Bergamaschi, S.; Castano, S.; Beneventano, D.; Vincini, M. Semantic Integration of Heterogeneous Information Sources. *Data Knowledge Engineering*, v.36, n.1, mar, 2001, pp. 215-249.
16. Vdovjak, R.; Houben, G. RDF-Based Architecture for Semantic Integration of Heterogeneous Information Sources. In: *International Workshop on Information Integration on the Web (WIIW'2001)*, pp. 51-57, Rio de Janeiro, Brazil, apr, 2001.

17. Jensen, M.R.; Moller, T.H.; Pedersen, T.B. Converting XML Data to UML Diagrams for Conceptual Data Integration. In: *1st International Workshop on Data Integration over the Web (DIWeb) at 13th Conference on Advanced Information Systems Engineering (CAISE'01)*, Interlaken, Switzerland, jun, 2001.
18. Halphin, T. Object-Role Modeling (ORM/NIAM). *Handbook on Architectures of Information Systems*. Springer-Verlag, 1998. p.81-102.
19. Mello, R.S.; Heuser, C.A. A Rule-Based Convertion of a DTD to a Conceptual Schema. In: *20th International Conference on Conceptual Modeling (ER'2001)*, pp. 133-148, Yokohama, Japan, Springer-Verlag, nov, 2001.
20. Mello, R.S.; Castano, S.; Heuser, C.A. A Method for the Unification of XML Schemata. *Information and Software Technology*, v.44, n.4, mar, 2002, pp. 241-249.

6 Appendix: CXPath Grammar

This appendix contains the syntax of the *CXPath* language. The initial symbol of grammar is *CXPathExpr*. The names of the majority of productions rules are identical to those used in the *XPath 1.0* grammar available at the W3C site *(http://www.w3.org/TR/xpath)*

```
CXPathExpr ::= ("/" RelativePathExpr?) | RelativePathExpr
RelativePathExpr ::= StepExpr ("/" StepExpr)*
StepExpr ::= (ForwardStep) Predicates
ForwardStep ::= ("{"RelationshipName(("}")|("." RoleName "}"))?
               NodeTest
Predicates ::= ( "[" (CXPathExpr) (GeneralComp
                                   (literal|CXPathExpr))? "]" )  *
GeneralComp    ::=    "=" |  "!=" |  "<" |  "<=" |  ">" |  ">="
Literal ::= NumericLiteral |  StringLiteral
NumericLiteral ::= IntegerLiteral | DecimalLiteral | DoubleLiteral
StringLiteral ::= ('"'(('"' '"')|[^"])*'"')|("'"(("'"
                  "'")|[^'])*"'")
IntegerLiteral ::= Digits
DecimalLiteral ::= ("." Digits) |  (Digits "." [0-9]*)
DoubleLiteral ::= (("." Digits)|(Digits("."[0-9]*)?))("e"|
                  "E")("+" | "-")? Digits
Digits ::= [0-9]+
NodeTest ::= QName
RelationshipName ::= QName
RoleName ::= QName
VarName ::= QName
QName ::= (Letter) (NCNameChar)*
NCNameChar ::= Letter | Digit | "_"
Letter ::= [a-z|A-Z]+
```