Data Transformation for Warehousing Web Data

Yan Zhu Christof Bornhövd*
Alejandro P. Buchmann

Department of Computer Science, Darmstadt University of Technology
64283 Darmstadt, Germany
{zhu, buchmann}@dvs1.informatik.tu-darmstadt.de
christof_bornhoevd@hp.com

Abstract

In order to analyze market trends and make reasonable business plans, a company's local data is not sufficient. Decision making must also be based on information from suppliers, partners and competitors. This external data can be obtained from the Web in many cases, but must be integrated with the company's own data, for example, in a data warehouse. To this end, Web data has to be mapped to the star schema of the warehouse. In this paper we propose a semi-automatic approach to support this transformation process. Our approach is based on the use a rooted labeled tree representation of Web data and the existing warehouse schema. Based on this common view we can compare source and target schemata to identify correspondences. We show how the correspondences guide the transformation to be accomplished automatically. We also explain the meaning of recursion and restructuring in mapping rules, which are the core of the transformation algorithm.

1 Introduction

Information from the Web has already become of major importance in helping individuals and companies to follow the current development in many areas, analyzing market developments and making business decisions. In an online bookshop, a data warehouse, for example, can be used to manage business transaction data, such as customer orders and promotions. The implementation of OLAP on the data warehouse will help to gain an insight into customer behavior, perform buy and replenish analysis, and design focused promotions. However, in order to analyze market trends and make new business plans, a company's own data is not sufficient, the bookshop manager also needs information from his suppliers, partners, and about his competitors. For example, discount book information or information about new publications from his competitors is important to help him to better plan own production lines or offer new promotions. Such information can be acquired from the Web. Integrating Web data and a companys data, materializing them in a data warehouse for implementing OLAP on them, making business plans based on them, and mining historic data to deduct business rules will greatly benefit e-commerce.

In our previous work [17, 18] we discussed a framework and an approach for warehousing Web data. This approach is already used to combine Web information and company data, and materializing them in a data warehouse. In [17] a language has been designed for describing mapping rules and procedures. The advantage of this language is its simple syntax. But because a mapping procedure is defined together with the correspondences between Web data and warehouse tables, this makes the mapping procedure dependent on the concrete Web data and the concrete warehouse tables. Therefore, when a new application domain is integrated the transformation program must be modified manually to adapt to the new application.

In order to automate the mapping process in this paper we propose a semi-automatic transformation approach based on the comparison of source- and target-schema and tree restructuring. First, Web data is integrated based on a common structural and semantic basis by using a self-describing object model, called MIX [6, 7]. We refer to this step as the Web data representation and integration phase, and the MIX model serves as a Web data representation model. Next is the transformation phase. In this step our source data has already become MIX objects representing the Web data, while the target schema is the relational data warehouse schema. The transformation task is to map MIX objects to relational warehouse tables. MIX objects representing complex data objects of a certain semantic concept can be understood as an arbitrarily deep rooted labeled tree. The Rooted Labeled Tree (RLT) or the Labeled Graph (LG) representation are already used in many integration, transformation and

*Current affiliation: Hewlett-Packard, e-Solutions Division, Palo Alto

Proceedings of the Third Int'l Workshop Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'01)
1530-1354/01 $10.00 © 2001 IEEE

query systems for semistructured/unstructured data, such as [1, 4, 5, 9, 10, 11, 13, 14, 15]. RLTs serve as a view of MIX objects in our approach. Besides, a relational database table can be easily represented as a tree with all leaves having the same depth (fixed-depth tree). The online bookshop data warehouse in our example is designed based on the relational data model, so that it can also be represented using RLTs.

Once the source schema and the target schema are viewed as RLTs, we can compare source and target schemata to find correspondences between them. We have observed that the tree representation of a relational table can be achieved through extracting subtrees from a MIX object tree and constructing a new tree from these subtrees. Therefore the transformation from semi-structured or unstructured Web data to well-structured relational data warehouse data can be implemented through tree restructuring.

We define the correspondences between MIX objects and warehouse tables in a mapping rule definition file. This file is parsed to produce a node list representing route information and a list of subtrees representing needed MIX subobjects. The mapping processor traverses the MIX object tree to an arbitrary depth through matching the nodes on the route, extracts needed subtrees used to generate a new fixed-depth tree, which represents a warehouse table. The mapping process designed following the tree traverse and construction is common for all transformation tasks, therefore it can be done automatically. For a new application we only need to specify the correspondences between semantic concepts of new MIX objects and column names in new warehouse tables in the mapping rule definitions, while the actual transformation process will not be changed, i.e., is application independent.

Our transformation rule definition file can be written using a language for querying semi-structured or unstructured data. In our prototype, we use UnQL [3, 4] as the rule definition language. The reasons are: first, that language is designed for querying semi-structured and unstructured data organized as Rooted Labeled Trees (RLTs) or Labeled Graphs (LGs) based on structural recursion. Second, UnQL's internal algebra, UnCAL, is close in spirit to the relational algebra and thus can express all relational algebra queries. Therefore, a query on warehouse data can be translated into an UnQL query on MIX objects. And third, UnQL supports restructuring queries, which can define the construction from an arbitrarily deep tree to a new tree. This is what we need, because the transformation from MIX objects to the relational data in a warehouse can be achieved by generating fixed-depth trees representing relational tables through restructuring an arbitrarily deep MIX object tree.

There are also other languages that have been proposed for querying semi-structured or unstructured data organized as RTLs and LGs, for example, Lorel and XML-QL. However Lorel [1] does not provide restructuring operations which are needed in our transformation tasks. XML-QL [10] is designed for querying XML, and has been developed based on UnQL and Strudel [11]. Although a first working draft of the XML Query Algebra was just published by W3C's XML Query Working Group [16], there is still no formal algebra for XML that can be used to formally describe the semantics the queries and to support query optimization. In contrast, UnQL provides both restructuring operations and a clear algebra for describing the meaning of mapping definitions. In fact, our approach is not limited to the use of UnQL, any language for querying semi-structured or unstructured data can also be used in our transformation framework if it provides the two features mentioned above.

The main contributions of this paper are:

- it introduces a semi-automatic transformation approach based on source- and target schemata comparison and tree restructuring for materializing Web data into a data warehouse;
- it describes the exact meaning of transformation rules, i.e., the recursion and restructuring specified by them.

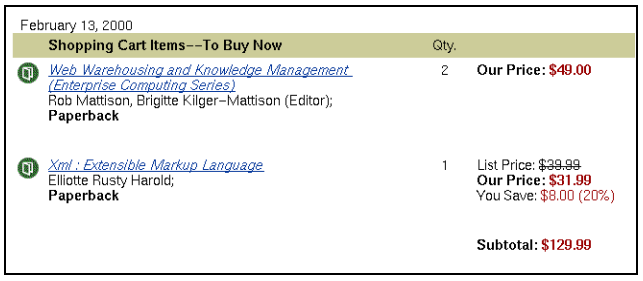
The rest of the paper is organized as follows: Section 2 provides an overview of our system for warehousing Web data. A motivating example is also shortly introduced in this section. Section 3 discusses tree-based representation of MIX objects and warehouse tables. The mapping rule definitions and the transformation algorithm are presented in Section 4. In Section 5 basic mapping rules are analyzed. Section 6 discusses the semantics of transformation rules. Section 7 describes the implementation of the automatic transformation process. Finally, we consider related work in Section 8.

2 An Overview of the System for Warehousing Web Data

2.1 An Application Scenario

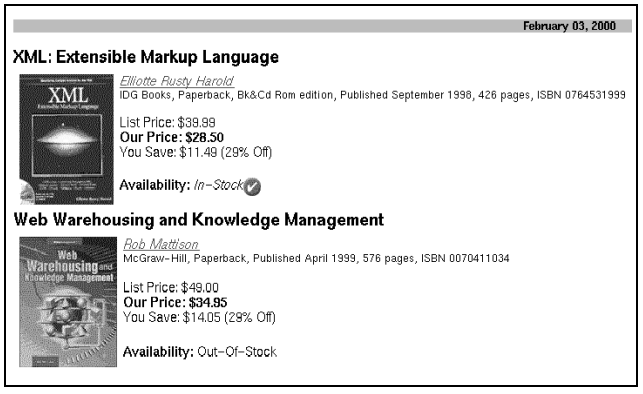
Online book shopping is a very active e-commerce area. A large amount of customer orders is produced every day, and can be recorded in a bookshop data warehouse for OLAP and decision making. Figure 1 shows such a customer order. In addition, the book shop manager may also integrate discount book information from his competitors' in this data warehouse, in order to compare pricing schemes, analyze market trends and make new business plans. This information can be obtained from related Web pages. Figure 2 shows discount book information from an online provider given as a HTML page.

The bookshop data warehouse in our example is based on the relational data model. The star schema of a simplified bookshop data warehouse is defined in Figure 3. This data warehouse manages e-commerce data of an online bookshop and discount book information from competitors Web pages. E-business data populates the book shop fact table, while Web data populates the discount fact table. The two fact tables can share several well-conformed dimensions, e.g., Book Dimension and Time Dimension table, or they may also have their own dimension table, e.g., Customer Dimension and BookStore Dimension.


February 13, 2000

Shopping Cart Items--To Buy Now	Qty.	
Web Warehousing and Knowledge Management (Enterprise Computing Series) Rob Mattison, Brigitte Kilger-Mattison (Editor); Paperback	2	Our Price: $49.00
Xml : Extensible Markup Language Elliotte Rusty Harold; Paperback	1	List Price: ~~$39.99~~ Our Price: $31.99 You Save: $8.00 (20%)
		Subtotal: $129.99

Figure 1: A customer order


February 03, 2000

XML: Extensible Markup Language

Elliotte Rusty Harold
IDG Books, Paperback, Bk&Cd Rom edition, Published September 1998, 426 pages, ISBN 0764531999

List Price: $39.99
Our Price: $28.50
You Save: $11.49 (29% Off)

Availability: In-Stock

Web Warehousing and Knowledge Management

Rob Mattison
McGraw-Hill, Paperback, Published April 1999, 576 pages, ISBN 0070411034

List Price: $49.00
Our Price: $34.95
You Save: $14.05 (29% Off)

Availability: Out-Of-Stock

Figure 2: Discount book information from source A

2.2 System Architecture and the MIX Model

Our system framework that provides a platform for integrating Web data and materializing it into a relational data warehouse has been introduced in [17]. The implementation of our framework is outlined in Figure 4. Components such as Transformation Processor, Federation Manager, Wrappers and Ontology Server are located at the organization possessing the data warehouse. Web data sources are available via the Internet.

We represent Web data using a special data model called MIX (Metadata based Integration model for data X-change) [2, 6, 8]. MIX is a self-describing data model in the sense

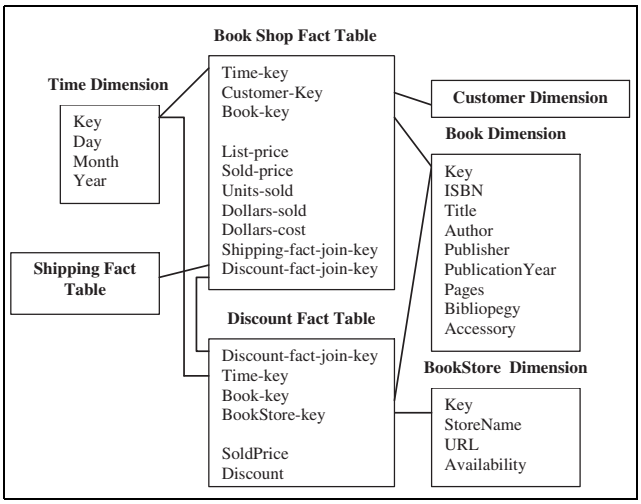


Figure 3: Star schema of the online bookshop data warehouse

that information about the structure and semantics of the data is given as part of the available data itself, not in the form of a separate schema. This model represents data together with a description of their underlying interpretation context and uses domain-specific ontologies in order to enable a semantically correct interpretation of the available data and metadata. Thus, it supports the integration of Web-based data.

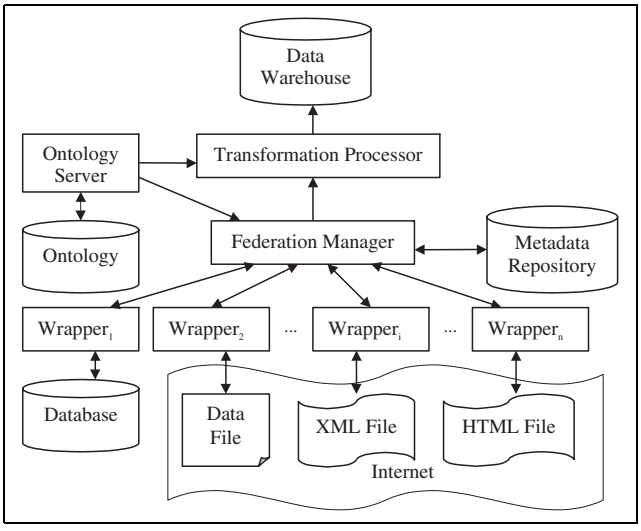


Figure 4: System architecture for warehousing Web data

The model is based on the concept of a semantic object. A simple semantic object representing an atomic data value v is a triple of the form:

$$SemObj := < C, \acute{v}, \$ > .$$

Where $C \in$ Ontology denotes the ontology concept to which the semantic object adheres, and $\acute{v} \in$

$Dom(RepType(C)$ is the physical representation of value v according to the physical representation type of C. $ specifies the semantic context associated with *SemObj* that provides additional information about the assumed meaning of v.

In contrast, a complex semantic object is defined as:

$$CompSemObj := < C, \mathbb{A} > .$$

Where $C \in$ Ontology denotes the ontology concept underlying *CompSemObj*, and $\mathbb{A}$ corresponds to the set of semantic objects associated with it that provide a representation of its subobjects.

The data given in Figure 2 can be represented as MIX objects of concept *BookOffer*. Using conversion functions, MIX objects from different sources can be further integrated by converting them to a common semantic context in the Federation Manager. In our example, we assume dates represented in the form *Mon DD, YYYY*, author names given by *Last Name, First Name Second Name*, and prices specified in *Euro* as the common semantic context. Figure 5 shows a sample of the integrated data corresponding to the books of Figure 2.

SemObj$_1$ = <BookOffer, {
<StoreName, "Source A">,
<URL, "http://www.bookpool.com/">,
<OfferDate, "Feb 03, 2000", {<DateFormat, "Mon DD, YYYY">} >,
<SoldPrice, 28.50, {<Currency, "EUR">} >,
<Book, {
<ISBN, 0764531999>,
<Title, "XML: Extensible Markup Language">,
<Author, "Harold, Elliotte Rusty",{<NameFormat, "Last, First Second">}>,
<Publisher, "IDG Books">,
<PublicationYear, "1998">,
<Pages, 426>,
<Bibliopegy, "Paperback">,
<Accessory, "CD Rom"> }>,
<ListPrice, 39.99>, {<Currency, "EUR">} >,
<Discount, "29%">,
<Availability, "In-Stock"> }>

SemObj$_2$= <BookOffer, {
<StoreName, "Source A">,
<URL, "http://www.bookpool.com/">,
<OfferDate, "Feb 03, 2000", {<DateFormat, "Mon DD, YYYY">} >,
<SoldPrice, 34.95, {<Currency, "EUR">} >,
<Book, {
<ISBN, 0070411034>,
<Title, "Web Warehousing and Knowledge Management">,
<Author, "Mattison, Rob", {<NameFormat, "Last, First">} >,
<Publisher, "McGraw-Hill">,
<PublicationYear, "1999">,
<Pages, 576>,
<Biblopegy, "Paperback">, }>,
<ListPrice, 49.00>, {<Currency, "EUR">} >,
<Discount, "29%">,
<Availability, "Out-Of-Stock"> }>

Figure 5: Converted Web data in MIX representation

3 Rooted Labeled Trees as a Common View

As introduced in Section 2, MIX is used in our system as a Web data representation model. A complex semantic object in MIX can be understood as a heterogeneous set of semantic objects that are grouped under a corresponding ontology concept. These semantic objects can be either complex semantic objects or simple semantic objects. A simple semantic object is a triple containing an ontology concept, a physical value of this object and a semantic context of this object. Such a containing relationship can be represented using a DAG (Directed Acyclic Graph). When all non-root nodes have only one input edge, we can depict such a DAG as a tree. The structure of MIX objects can therefore be represented as a Rooted Labeled Tree (RLT). Figure 6 gives such a RLT representation of the complex MIX objects from Figure 5. $SemObj$ is the root which links all objects of concept *BookOffer*, i.e., $SemObj_1$ and $SemObj_2$. Ontology concepts of complex and simple semantic objects are used as the node labels, leaves will be physical values and semantic contexts of MIX objects, respectively. Semantic contexts are not shown in Figure 6, but they are a part of the data.

A relational database can also be represented as a set of fixed-depth trees. The data warehouse in our system is based on the relational data model and can be represented as a set of rooted, labeled trees. The table name is the root of a tree, internal nodes of the tree are columns of the table, leaves represent physical values. Figure 7 gives an example of the RLT representation of two data warehouse tables shown in Figure 3. In Figure 7 and the figures in the next sections, the label *tuple* is a special label, which is used to indicate the encoding of a tuple, rather than a column of the table. We use it only for illustration.

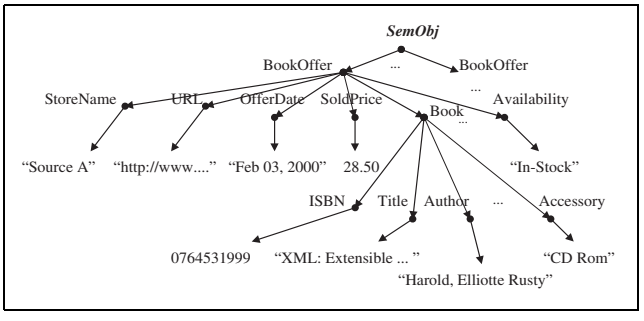


Figure 6: RLT representation of a BookOffer MIX object

4 Constructing Warehouse Tables through Restructuring MIX Object Trees

4.1 Mapping Rule Definition

Based on the tree representations in Figure 6 and Figure 7, we can observe that the schemata of source and target have correspondences. Constructing a warehouse table can be achieved by generating fixed-depth database trees through restructuring an arbitrarily deep MIX object tree

following these correspondences. We specify correspondences explicitly in a mapping rule definition file, where we specify not only the schema correspondences between MIX objects of a certain concept and columns of warehouse tables, but also define the mapping rules from data values of objects to values of columns.

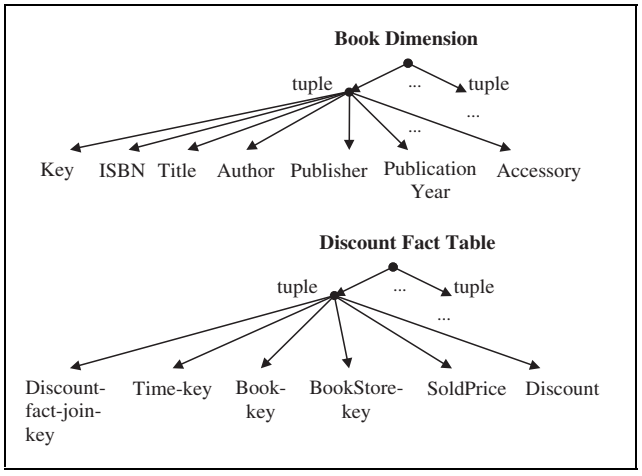


Figure 7: RLT representation of data warehouse tables

Our mapping rule definitions are written in UnQL [3, 4]. UnQL consists of tree constructors, function definitions and query definitions:

- tree constructors $\{\}$, $\{l : t\}$, $t_1 \cup t_2$, $\{l_1 : t_1, ..., l_n : t_n\}$
- functions defined by structural recursion:

```
let sfun h1({country:C}) =
  let sfun h2({name:N}) = {result:N}
  in h2(C)
in h1(db)
```

- queries defined with select-where and pattern matching, which can be translated into structural recursion:

```
select e where ({PE:T} in e' rest)→
  let sfun h({PE:T}) = (select e where rest)
  in h(e')
select e where () → e
```

When defining correspondences between MIX objects and warehouse tables in queries, the pattern in the where clause can be viewed as subtrees of an input MIX object tree. They denote needed subobjects and paths to them. The pattern in the select clause denotes a constructed output result, which is the tree representation of a relational table. Node labels and leaf variables used in the two clauses specify the correspondences between MIX objects and columns of a warehouse table. For example, Query 1 defines the correspondence between objects of *BookOffer* shown in Figure 6 and columns of the *Book Dimension* table shown in Figure 7. Query 2 defines the relationship between objects of *BookOffer* and columns of the *BookStore Dimension* table. Query 3 specifies the correspondence between objects of *BookOffer* and columns of the *Discount Fact* table.

```
Query1 :=
  select {BookDim : {ISBN : isb, Title : tit, Author : aut,
      Publisher : pub, PublicationYear : puy, Pages : pag,
      Bibliopegy : bib, Accessory : acc}}
  where {BookOffer : {Book : {ISBN : isb , Title : tit,
      Author : aut, Publisher : pub, PublicationYear : puy,
      Pages : pag, Bibliopegy : bib, Accessory : acc}}}
  in SemObj

Query2 :=
  select {BookStoreDim : {StoreName : sna , URL : url,
    Availability : ava}}
  where {BookOffer : {StoreName : sna , URL : url,
    Availability : ava}}
  in SemObj

Query3 :=
  select {DiscountFact : {SoldPrice : spr, Discount : disc}}
  where {BookOffer : {SoldPrice : spr, Discount : disc}}
  in SemObj
```

select-where with pattern matching can be translated into structural recursion. Structural recursion can be understood as queries that can be recursively applied on an object tree and its children, since the structure of a tree supports recursive traverse and ensures that the recursion always terminates. Therefore, the above mapping definitions can be written in the following form using structural recursion following UnQL's syntax [4]. We will use mapping definitions in the structural recursion form in later sections, because this form can help to understand the semantics of transformation rules, which will be discussed in Section 6.

```
Q1:
let sfun f1 ({BookOffer : bo}) =
  let sfun f2 ({Book : bk}) = {BookDim :
    let sfun f3 ({ISBN : isb} ∪ {Title : tit}
        ∪ {Author : aut} ∪ ... ∪ {Accessory : acc}) =
    {ISBN : isb} ∪ {Title : tit} ∪ {Author : aut} ∪ ...
        ∪ {Accessory : acc}
    in f3(bk)}
  in f2(bo)
in f1(SemObj)

Q2:
let sfun f1 ({BookOffer : bo}) = {BookStoreDim :
  let sfun f2 ({StoreName : sna} ∪ {URL : url}
      ∪ {Availability : ava}) =
  {StoreName : sna} ∪ {URL : url} ∪ {Availability : ava}
  in f2(bo)}
in f1(SemObj)
```

Q3:
let sfun f_1 ({*BookOffer : bo*}) = {*DiscountFact :*
  let sfun f_2 ({*SoldPrice : spr* } ∪ {*Discount : disc*}) =
    {*SoldPrice : spr*} ∪ {*Discount : disc*}
  in f_2*(bo)*}
in f_1*(SemObj)*

These queries produce the following three subtrees from the *BookOffer* tree:

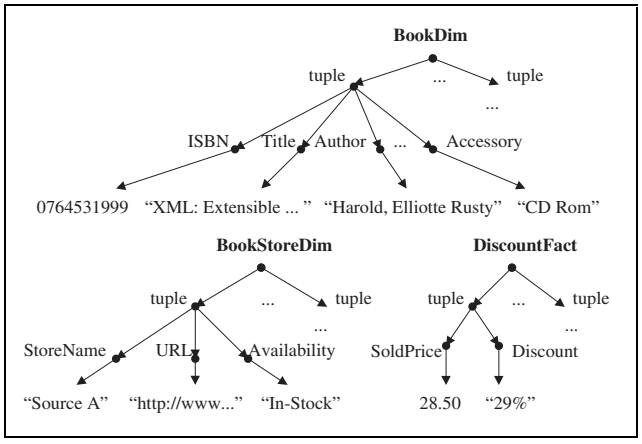


Figure 8: Output trees through restructuring an input tree

4.2 Transformation Algorithm

After mapping rules are specified, a parser will parse the mapping rules and produce the traversing route list and a list of subtrees needed in constructing the tree representing the relational table. This data provides the input parameters of the mapping algorithm. The input tree in the algorithm can be any MIX object tree of the corresponding concepts. The output trees will be warehouse tables.

The key part of the transformation algorithm is a procedure for recursively traversing an object tree from the root to the leaves until needed subtrees are reached. At first, the transformation processor does a breadth-first traverse on a MIX object tree. If a node label matches the route information from the mapping rules, the search will be applied on its subtrees (subobjects). If two or more node labels match with the same route node label, the subobjects of that first MIX object matched will be searched first. This procedure works recursively until all needed subtrees (MIX subobjects) are found. Once a needed subtree is found it will be extracted, transformed and merged into the corresponding output tree. The transformation algorithm is outlined below:

Transformation Algorithm:

- Given a MIX object which can be illustrated as a tree T, t: the root node of tree T;
- P: a list of MIX objects, their ontology concepts as node labels indicate routes from the root node to the needed subtrees;
- C: a list of column names of a warehouse table;
- Output: a relational table which can be represented as a fixed-depth tree RT.

```
Procedure transform ( t, P, C )
Var   Mark : sign of visited
      m, n : nodes
      p : an ontology concept in P
      c : a column name in C
Begin
  if t is not visited then Mark[t] : = visited;
  for each m adjacent to t do
 Begin
    Mark[m] : = visited;
    if  m matches one p in P then
   Begin
      if  p does not match any c in C then
        transform (m, P, C);
      else  // p matches c
      Begin
        for each n adjacent to m do
       Begin
          if n is not visited then
         Begin
           Mark[n] : = visited;
           do whatever processing is necessary on n;
           merge (n, RT);
         End
       End
      End
    End
  End
End
```

5 Basic Mapping Rules

1. **Composition relationships of concepts:**
   The relationships between a MIX object and its subobjects can be understood as part-of relationships. When mapping a complex MIX object to warehouse tables, we must distinguish between two cases: First, if the complex semantic object populates only a dimension table, the hierarchy of the tree will be flattened, see mapping definition Q1. Second, when the complex MIX object populates a fact table as well as dimension tables, the object must be decomposed and subobjects are separately mapped to facts in the fact table and attributes in dimension tables, such as in the case of Q2 and Q3.

2. **Identifying attributes or surrogate keys:**
   In the MIX model each complex object is identified

through a single or multiple identifying attributes, similar to key attributes in the relational model. For instance, in Figure 5 a complex semantic object *BookOffer* is identified by its attributes: *StoreName, URL, OfferDate, Price* and *Book* which are underlined. A complex MIX object of concept *Book* is identified by only the attribute *ISBN*. Two complex objects of the same ontology concept are identified by the same set of attributes. We can use surrogate keys in data warehouses, as already shown in Figure 3. An additional column for a surrogate key is created when a warehouse table is designed. The system can generate an unique key for these tables when transforming the corresponding data.

3. **One MIX object to one/many columns:**
When mapping objects of an ontology concept to table columns we must distinguish between three cases. In the easiest case we have a direct 1:1 mapping, i.e., a concept is directly mapped to a column, and the values of the corresponding MIX objects are directly assigned to the corresponding table columns. In the second case, we must calculate the values of a column by applying a specified function on the respective MIX object. For instance, *Euro* is converted to *US dollar*, for this a conversion function *euroToUsd()* will be used in the corresponding mapping rule:

Q4-1:
let sfun f_1 ({*BookOffer : bo*}) = {*DiscountFact :*
  let sfun f_2 ({*SoldPrice : spr*}) =
    {*SoldPrice : euroToUsd(spr)*}
  in f_2*(bo)*}
in f_1*(SemObj)*

Finally, when values of a MIX object must be decomposed to multiple table columns (1:n mapping), decomposition functions have to be applied to calculate suitable values. For example, *OfferDate* in Figure 5 is in the form of *"Mon DD, YYYY"*. In the *Time Dimension* of the data warehouse in Figure 3 we have *Day, Month*, and *Year* columns. Therefore, we must use decomposition functions, like:

getYearfromDate("Feb 03, 2000") $\Rightarrow$ *"2000"* ,

to generate values for these columns:

Q4-2:
let sfun f_1 ({*BookOffer : bo*}) = {*TimeDim :*
  let sfun f_2 ({*OfferDate : odate*}) =
    {*Day : getDayfromDate(odate)*}
  ∪ {*Month : getMonthfromDate(odate)*}
  ∪ {*Year : getYearfromDate(odate)*}
  in f_2*(bo)*}
in f_1*(SemObj)*

4. **MIX objects of different concepts to one column:**
When multiple MIX objects of different concepts are mapped to one column, i.e., values from multiple attributes are used to derive a column value (n:1 mapping), an aggregation function has to be applied to calculate the value. For example, calculating the *TotalCost* paid by a customer in one book purchase, MIX objects of concept *Subtotal* and *ShippingCost* are involved. The required calculation function is given by:

$F_{TotalCost}(Subtotal,\ ShippingCost) = Subtotal + ShippingCost \Rightarrow TotalCost$

The corresponding mapping rule is:

Q5:
let sfun f_1({*CustomerOrder : co*}) =
  {*let sfun* f_2({*Subtotal : st*}) *in* f_2*(co)*}∪
  {*let sfun* f_3({*ShippingItem : si*}) =
    let sfun f_4({*ShippingCost : sc*}) = { *totalcost : st + sc*}
    in f_4*(si)*
  in f_3*(co)*}
in f_1*(SemObj)*

5. **Default values:**
When we map MIX objects to the tables of the data warehouse, we sometimes do not have values for all attributes. Thus, it may be necessary to use some default value in these places, such as *"Paperback"* as the default value of *Bibliopegy* (the art of bookbinding). For example, we can use the following definition:

Q6:
let sfun f_1({*BookOffer : bo*}) =
  let sfun f_2({*Book : bk*}) = {*BookDim :*
    let sfun f_3({*ISBN : isb, ...,Author : aut, ...,*
        Bibliopegy : bib, Accessory : acc}) =
      if (bib = null)
      then {*ISBN : isb, Author : aut, ...,*
        Bibliopegy : defaultValue, Accessory : acc}
      else {*ISBN : isb, Author : aut, ..., Bibliopegy : bib,*
        Accessory : acc}
    in f_3*(bk)*}
  in f_2*(bo)*
in f_1*(SemObj)*

In fact, we don't specify any check of values in mapping rules. All values from MIX objects will be automatically checked for being null in the transformation process.

6. **Multi-valued attributes problem:**
When different subobjects of the same concept occur in a MIX object, a multi-valued attribute problem arises. For example, a book may have more than one author. Many relational database systems do not support multi-valued attributes.

If the maximal cardinality of such multi-valued attributes is known in advance, multiple columns can be used to store them. Otherwise, separate rows have to be used to store them. However, in general it is impossible to exactly know the maximal cardinality of such an attribute, for instance the amount of the authors of a book, before we design a data warehouse schema. Therefore, one method is to arrange several columns in advance, say 5 for author attributes. The problem is sparsity or lost information. On the other hand, when we create one tuple for each value of the multi-valued attribute, the problem is information redundancy.

If we use the first approach to handle the multi-valued attributes problem, we must see a MIX object tree as a partly ordered tree: first author appears first, second author appears thereafter and so on. In this case, the author subtree is interpreted as a list depicted as $\{||\, l : t \,||\}$, and concatenation @ replaces union $\cup$, $f(T_1 @ T_2)$ replaces $f(T_1 \cup T_2)$. Then we define, for example, 5 columns in advance for storing each value of the author. Given a book which has two authors, the mapping rules of such a case are shown in query Q7-1. Figure 9 shows the result of Q7-1.

Q7-1:
let sfun f_1*({BookOffer : bo}) =*
  let sfun f_2*({Book : bk}) = {BookDim :*
    let sfun f_3*({ISBN : isb, ..., Author :* aut_1*,*
      Author : aut_2*, Publisher : pub, ..., Accessory : acc})*
    = $\{ISBN : isb\} \cup ... \cup \{||Author_1 : aut_1||\}$ @
      $\{||Author_2 : aut_2||\}$ @ $\{||Author_3 :$ " "$||\}$ @ ... @
      $\{||Author_5 :$ " "$||\} \cup ... \cup \{$*Accessory : acc*$\}$
    in f_3*(bk)}*
  in f_2*(bo)*
in f_1*(SemObj)*

If we use two separate rows to store each value of author attributes, the mapping rule is as specified by query Q7-2.

Q7-2:
let sfun f_1*({BookOffer : bo}) =*
  let sfun f_2*({Book : bk}) = {BookDim :*
    let sfun f_3*({ISBN : isb, ..., Author :* aut_1*,*
      Author : aut_2*, ..., Accessory : acc}) =*
      $\{||$*ISBN : isb, ..., Author :* aut_1*, ...,*
      Accessory : acc$||\}$ @
      $\{||$*ISBN : isb, ..., Author :* aut_2*, ...,*
      Accessory : acc$||\}$
    in f_3*(bk)}*
  in f_2*(bo)*
in f_1*(SemObj)*

6 Semantics of Transformation Rules

In Section 4.2 we have discussed our transformation algorithm. The core of this algorithm is determined by a set

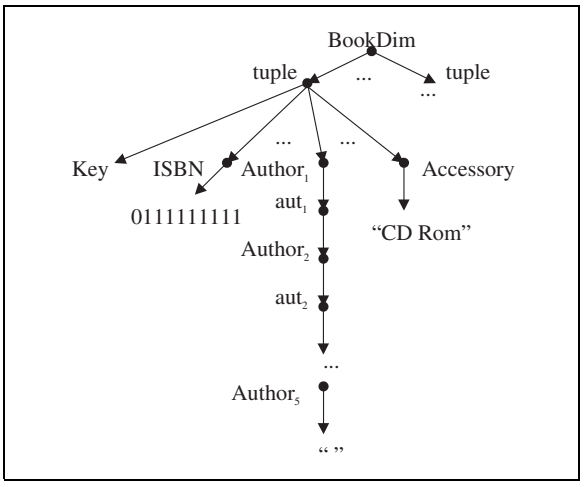


Figure 9: One of the resulting trees of the set values transformation

of mapping rules that define recursive traversal on an object tree and the restructuring of its subtrees. In order to make explicit the exact meaning, i.e., the specified processing of the mapping rules, we use a calculus called UnCAL [3, 4], which is UnQL's internal algebra in the sense of lambda calculus (a formalism with variables and functions) rather than in the sense of relational calculus (a logic with variables and quantifiers). For structural recursion on trees its syntax is *rec(λ(l, t).e)* or simply $rec(e)(t)$., here: λ means lambda, l and t are understood as label and tree variables, and $rec(e)$ denotes the structural recursion on edges. By using UnCAL to explicitly describe the semantics of transformation rules, we want to reveal how the pattern matching, recursive search and subtrees restructuring take effect in the mapping rules, and thus to make mapping rules better understandable. In our approach, we evaluate the structural recursion based on labeled nodes. This is because each non-root node in the RLT view of MIX objects has only one input edge, thus labeling a node is equivalent to labeling an edge. Furthermore, we extended UnCAL by adding the transformation function definition: $E ::= f(E; ...; E)$. This function definition can explain the semantics of external function application. Following are some notations needed in this section.

The syntax of extended UnCAL is:

E ::= {} | {L : E} | E ∪ E | &x := E | &y | ( ) | E ⊕ E |
  E @ E | cycle(E) | Var | if B then E else E |
  f(E; ... ; E) | rec(λ(LabelVar, Var).E)(E)
L ::= LabelVar | a, a ∈ Label
B ::= isempty(E) | L = L
f ::= function

a) Q1 in Section 4 (tree flattening) has the following semantics:

rec(λ(l_1*, bo).*

```
if l1 = "BookOffer"
then {rec(λ(l2, bk).
  if l2 = "Book"
  then {BookDim.rec(λ(l3, isb).
    if l3 = "ISBN" then {ISBN : isb} else {})(bk)
  ∪ BookDim.rec(λ(l4, tit).
    if l4 = "Tilte" then {Title : tit} else {})(bk)
    ...
  ∪ BookDim.rec(λ(li, acc).
    if li = "Accessory" then {Accessory : acc} else {})(bk)}
  else {})(bo)}
else {})(SemObj)
```

b) Q2 in Section 4 (complex MIX object decomposition) has semantics:

```
rec(λ(l1, bo).
  if l1 = "BookOffer"
  then {BookStoreDim.rec(λ(l2, sna).
    if l2 = "StoreName" then {StoreName : sna}
    else {})(bo)
  ∪ BookDim.rec(λ(l3, url).
    if l3 = "URL" then {URL : url}
    else {})(bo)
  ∪ BookDim.rec(λ(l4, ava).
    if l4 = "Availability" then {Availability : ava}
    else {})(bo)}
  else {})(SemObj)
```

c) the meaning of Q4-1 in regard to calculation mapping is:

```
rec(λ(l1, bo).
  if l1 = "BookOffer"
  then {DiscountFact.rec(λ(l2, spr).
    if l2 = "SoldPrice" then {SoldPrice : euroToUsd(spr)}
    else {})(bo)}
  else {})(SemObj)
```

d) the semantics of Q4-2 in regard to 1:n mapping is:

```
rec(λ(l1, bo).
  if l1 = "BookOffer"
  then {TimeDim.rec(λ(l2, odate).
    if l2 = "OfferDate"
    then {Day : getDayfromDate(odate)}
      ∪ {Month : getMonthfromDate(odate)}
      ∪ {Year : getYearfromDate(odate)}
    else {})(bo)}
  else {})(SemObj)
```

e) Q5 about n:1 mapping has the meaning:

```
rec(λ(l1, co).
  if l1 = "CustomerOrder"
  then {rec(λ(l2, st).
    if l2 = "Subtotal" then {subtotal : st} else {})(co)}
  ∪ {rec(λ(l3, si).
    if l3 = "ShippingItem"
    then {rec(λ(l4, sc).
      if l4 = "ShippingCost"
      then {totalcost : subtotal + sc} else {})(si)}
    else {})(co)}
  else {})(SemObj)
```

f) the meaning of Q6 (using a default value) is:

```
rec(λ(l1, bo).
  if l1 = "BookOffer"
  then {rec(λ(l2, bk).
    if l2 = "Book"
    then {BookDim.rec(λ(l3, isb).
      if l3 = "ISBN" then {ISBN : isb} else {})(bk)}
    ∪ ...
    ∪ {BookDim.rec(λ(l4, aut).
      if l4 = "Author" then {Author : aut} else {})(bk)}
    ∪ ...
    ∪ {BookDim.rec(λ(li, bib).
      if li = "Bibliopegy"
      then if bib = " " then {Bibliopegy : "paperback"}
           else {Bibliopegy : bib}
      else {})(bk)}
    else {})(bo)}
  else {})(SemObj)
```

g) the semantics of Q7-1 (one possible solution of the multi-valued attribute problem) is:

```
rec(λ(l1, bo).
  if l1 = "BookOffer"
  then {rec(λ(l2, bk).
    if l2 = "Book"
    then {BookDim.rec(λ(l3, isb).
      if l3 = "ISBN" then {ISBN : isb} else {})(bk)}
    ∪ ...
    ∪ {BookDim.rec(λ(li, setOfValues).
      if li = "Author"
      then {|| Author1 : 1st value in setOfValues* ||} @ ...
           @ {|| Author5 : 5th value in setOfValues ||}
      else {})(bk)}
    ...
    ∪ {BookDim.rec(λ(ln, acc).
      if ln = "Accessory" then {Accessory:acc} else {})(bk)}
    else {})(bo)}
  else {})(SemObj)
```

h) the semantics of Q7-2 (another solution of the multi-valued attribute problem) is:

```
rec(λ(l1, bo).
```

* $Author_i : i^{th}$ value in the setOfValues, if the setOfValues $\neq \emptyset$; $Author_i$: " ", if setOfValues $= \emptyset$

```
if l1 = "BookOffer"
then {rec(λ(l2, bk).
  if l2 = "Book"
  then {|| BookDim.rec(λ(l3, isb).
    if l3 = "ISBN" then {ISBN : isb} else {})(bk)
  ∪ ...
  ∪ BookDim.rec(λ(li, aut1).
    if li = "Author" then {Author : aut1}
    else {})(bk)
    ...
  ∪ BookDim.rec(λ(ln, acc).
    if ln = "Accessory" then {Accessory : acc}
    else {})(bk) ||}
  @ {|| BookDim.rec(λ(l3, isb).
    if l3 = "ISBN" then {ISBN : isb} else {})(bk)
  ∪ ...
  ∪ BookDim.rec(λ(li, aut2).
    if li = "Author" then {Author : aut2} else {})(bk)
    ...
  ∪ BookDim.rec(λ(ln, acc).
    if ln = "Accessory" then {Accessory : acc}
    else {})(bk) ||}
  else {})(bo)}
else {})(SemObj)
```

7 Implementation

The architecture of the Transformation Processor is composed of 6 components:

- A *Mapping Definition File* consisting of a set of user-specified transformation rules which describe the required mappings, such as default mapping, 1:n mapping or the solution of set values. These different transformation cases and the semantics of mapping rules have been discussed in Sections 5 and 6.

- A *Parser* analyzing mapping rules to produce correspondences between MIX objects and columns of a data warehouse table. In addition to the correspondences, path information and information about needed subtrees are deduced to support the mapping.

- A *Transformation Function Library* consisting of a set of mapping functions. These functions are Java methods accomplishing aggregated, decomposed and calculated mapping. These functions can be called by the *Mapper* component at run time in the corresponding cases.

- A *Tree Traverser* working under the guide of the mapping correspondences. It traverses complex MIX objects through matching path nodes recursively till it gets to the needed subtrees. Simple MIX objects represented by needed subtrees are extracted and sent to the *Mapper*.

- A *Mapper* receiving simple MIX objects from the *Tree Traverser*, separating semantic contexts from the MIX objects, and mapping these MIX objects to columns of a table in the data warehouse following the given mapping rules. Then this component constructs values for each column according to different transformation definitions. At the end the *Mapper* connects dimension tables with the fact table and sends the data to the *Loader*.

- A *Loader* receiving data from the *Mapper*, connecting to the data warehouse, and loading the data into the data warehouse tables via JDBC. Figure 10 shows the relationships between these components.

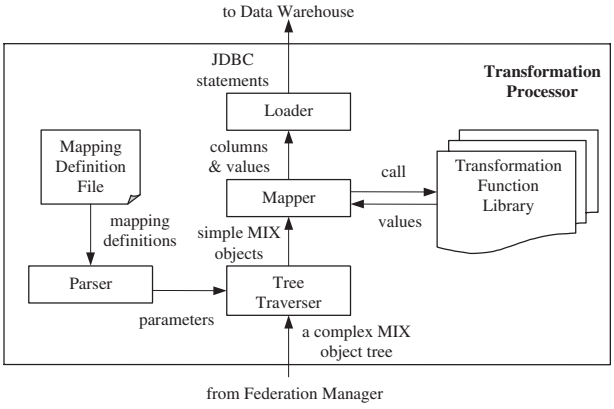


Figure 10: Architecture of the Transformation Processor

In the Transformation Processor, the Parser analyzes the given mapping definition file and produces a set of parameters, which include path information, a list of needed subtrees and transformation functions for value mappings. The Tree Traverser reads a complex MIX object tree from the Federation Manager, and gets parameters from the Parser, then it walks along the specified path to search needed simple MIX objects (subtrees). Once all simple MIX objects are found, they are sent to the Mapper.

The Mapper separates the semantic contexts from the objects and extracts the data values from them. The values are transformed according to corresponding rules, e.g., by calling transformation functions from the Transformation Function Library at run time. Some data formats may be changed when transforming data. For example, the *OfferDate* in a MIX object may be decomposed as *day, month, year* into three values for the time dimension table of the data warehouse, this will also be accomplished by the Mapper.

Besides, the Mapper assigns a unique key value for a tuple in each dimension table and links the fact table with dimension tables using these keys at the end. The process

of reading a complex MIX object, mapping attributes of objects to columns of target tables, transforming values following the mapping rules will be repeated until no new MIX object comes from the Federation Manager.

After data has been prepared, the Mapper sends them to the Loader. The Loader constructs SQL statements, connects to the data warehouse, and executes the prepared statements to load transformed data values into the corresponding warehouse tables. Then the Loader closes the connection to the data warehouse, and the loading process is completed.

The Transformation Processor is written in Java and all mapping rules discussed in our paper have been implemented in a prototype.

8 Related Work and Conclusions

In this paper, we propose a semi-automatic transformation approach for materializing Web data into a data warehouse. In this approach we first compare the structure of MIX objects and the star schema of data warehouse tables based on a common view (rooted, labeled tree) to identify semantic correspondences between them. These correspondences are explicitly described as mapping rules, based on which the transformation can be accomplished automatically via tree restructuring.

Our approach is related to the work of Milo, Beeri and Zohar [5, 13]. They define common schema and data models for the source and target data. Using a rule-based method, they match components in the source schema with components in the target schema. The matching identified is then used for translating instances of the source schema to instances of the target schema. In our framework, Web data is first represented based on the MIX model, then the Rooted Labeled Tree is used as a common view to represent the structure of MIX objects and the schema of warehouse tables. Similar to their approach, the MIX objects and the warehouse tables are compared based on the RLT representation, the correspondences between them can be formalized, and the transformation can be performed automatically.

However, there are several differences between our work and theirs. At first, they use two models for their data translation task. One is the schema model (graph) used in the schema matching process, the other is a data model (labeled forest) used in the data translation step. Different from that, we use only one model (Rooted Labeled Trees) as a common comparison basis for defining the correspondences of source objects and target warehouse tables, and for required value transformations. Because schema information is already provided as a part of MIX objects, we can use RLTs not only for the schema comparing process but the data mapping task as well.

Second, the TranScm system introduced in [13] uses rules to match schemata. Each rule uses match and descendents functions to handle schema matching, and uses translation functions to handle data translation. In contrast to their descendents functions for handling children of a vertex, we use path pattern, which can be translated into structural recursion. Therefore, our transformation rules are easier to understand and evaluated automatically.

Third, TranScm uses rules and tries to find for each component (vertex) of the source schema a unique best matching component (vertex) in the target schema, or determine that the component should not be represented in the target. There are cases where the matching process may fail. In our prototype, the target schema of the data warehouse determines what data should be extracted from Web sources. According to queries from the data warehouse the Federation Manager integrates data from heterogeneous Web sources. When some attributes of the data warehouse acquire no data from the Web, we can use default values for these attributes.

Another related system is the YAT system [9]. In their work, data from heterogeneous sources is integrated using an ODMG object view and materialized into an object-oriented database. A rule-based language called YATL serves as the conversion language, which has pattern matching, restructuring facilities and skolem functions. Similar to our approach, they use rules to filter the input data and construct a set of output patterns.

However, some major differences exist between our work and theirs. In our system data from the Web are integrated as MIX objects and are then materialized into a relational data warehouse. The mapping rules and algorithm in our approach are designed to map Web objects into existing warehouse tables efficiently and automatically. While in the YAT system data from heterogeneous sources are integrated using an ODMG object view and then stored in an object-oriented database without additional transformation. The rules in the YAT system describe the conversion from relational and SGML data to ODMG objects and the translation from ODMG data to HTML files.

Several projects on storing data from heterogeneous sources, especially XML documents, in a relational database can also be compared with our work.

Schmidt et al. presents a data model and an execution model that allow for efficient storage and retrieval of XML documents in a relational database [14]. In their approach, XML documents are represented as syntax trees which are then decomposed into binary associations. Associations that provide semantically related information are stored together in the binary relationships of the database repository. The greatest difference between our approach and theirs is that we extract needed leaves (subobjects) from a MIX object tree and store the physical value of these subobjects in the warehouse tables. The paths from the root to the leaves

are used to guide how to traverse the MIX object tree. While in [14] they store a lot of vertical path fragments in tables of a relational database. These path fragments represent routes from the root to internal nodes, from the root to CDATA and to character data.

Shanmugasundaram et al. in [15] use a standard relational database system to evaluate powerful queries over XML documents. They process a DTD to generate a relational schema, then parse XML documents conforming to the DTD and load them into tuples of the created relational tables. Following this, they translate semi-structured queries over XML documents into SQL queries over corresponding relational data and convert the results back to XML. One important difference between our approach and theirs is that the target schema is assumed to already exist in our system before Web data is integrated. We focus on representing MIX objects and warehouse schema based on a common view (RTL) and identify similarities between schemata to specify and accomplish our transformation. While in [15] their main focus is on the transformation from a DTD to a relational schema and the translation from semi-structured queries over XML documents to SQL queries over the corresponding relational data.

Florescu and Kossmann studied ad-hoc schemata for storing XML using RDBMS [12]. They represent XML data as ordered labeled directed graphs. Each XML element is represented by a node, element-subelement relationships are represented by edges, and values (e.g., strings) of an XML document are stored as leaves. They proposed three approaches to store edges of the graph in one/more tables and proposed two approaches to store values of XML documents in relational tables. Compared with their approach, the most significant difference is that our relational database schema is designed to store data values coming from the Web, while they primarily study several relational schemata to store the topological information of a XML document.

The approach proposed in this paper focuses on how to simplify and automate the transformation task for storing Web data into an existing data warehouse. The star schema of the data warehouse is based on the relational model. As future work we will consider how to map Web data into multi-dimensional cubes for OLAP.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener: *The Lorel Query Language for Semistructure Data*, International Journal on Digital Libraries 1(1): 68-88, 1997

[2] C. Bornhövd, A. P. Buchmann: *A Prototype for Metadata-based Integration of Internet Sources*, CAiSE'99, Heidelberg, Germany, 1999

[3] P. Buneman, S. Davidson, G. Hillebrand, D. Suciu: *A Query Language and Optimization Techniques for Unstructured Data*, SIGMOD'96, Montreal, Canada, 1996

[4] P. Buneman, M. Fernandez, D. Suciu: *UnQL: a Query Languag and Algebra for Semistructured Data Based on Structural Recursion*, VLDB Journal 9(1): 76-110, 2000

[5] C. Beeri and T. Milo: *Schemas for Integration and Translation of Structured and Semi-structured Data*, ICDT'99, Jerusalem, Israel, 1999

[6] C. Bornhövd: *MIX – A Representation Model for the Integration of Web-based Data*, Technical Report, DVS99-1, Department of Computer Science, Darmstadt University of Technology, 1999

[7] C. Bornhövd: *Semantic Metadata for the Integration of Web-based Data for Electronic Commerce*, WECWIS'99, Santa Clara, USA, 1999

[8] C. Bornhövd: *Semantic Metadata for the Integration of Data Source from the Internet*, Ph.D. thesis, Darmstadt University of Technology, Germany, Jan. 2001

[9] S. Cluet, C. Delobel, J. Simeón and K. Smaga: *Your Mediators Need Data Conversion!*, SIGMOD'98, Seattle, WA, USA, 1998

[10] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, D. Suciu: *A Query Language for XML*, WWW8 / Computer Networks 31(11-16): 1155-1169, 1999

[11] M. Fernandez, D. Florescu, J. Kang, A. Levy, D. Suciu: *STRUDEL : A Web Site Management System*, SIGMOD'97, Tucson, USA, 1997

[12] D. Florescu, D. Kossmann: *Storing and Querying XML Data Using a RDBMS*, Bulletin on Data Engineering 22(3): 27-34, 2000

[13] T. Milo and S. Zohar, *Using Schema Matching to Simplify Heterogeneous Data Translation*, VLDB'98, New York City, USA, 1998

[14] A. Schmidt, M. Kersten, M. Windhouwer, F. Waas: *Efficient Relational Storage and Retrieval of XML Documents*, WebDB'00, Dallas, USA, 2000

[15] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton: *Relational Databases for Querying XML Documents: Limitations and Opportunities*, VLDB'99, Edinburgh, Scotland, 1999

[16] P. Fankhauser, M. Fernndez, A. Malhotra, M. Rys, J. Siméon and P. Wadler (Eds): *The XML Query Algebra*, www.w3.org/TR/query-algebra/, W3C Working Draft, 2000

[17] Y. Zhu, C. Bornhövd, D. Sautner, A. P. Buchmann: *Materializing Web Data for OLAP and DSS*, WAIM'00, Shanghai, China, 2000

[18] Y. Zhu: *A Framework for Warehousing the Web Contents*, ICSC'99, Hong Kong, 1999