A Semantic Approach to Integrating XML and Structured Data Sources

Peter M^{c}Brien[1] and Alexandra Poulovassilis[2]

[1] Department of Computing, Imperial College,
180 Queen's Gate, London SW7 2BZ
pjm@doc.ic.ac.uk

[2] Department of Computer Science, Birkbeck College, University of London,
Malet Street, London WC1E 7HX
ap@dcs.bbk.ac.uk

Abstract. XML is fast becoming the standard for information exchange on the WWW. As such, information expressed in XML will need to be integrated with existing information systems, which are mostly based on structured data models such as relational, object-oriented or object/relational data models. This paper shows how our previous framework for integrating heterogeneous structured data sources can also be used for integrating XML data sources with each other and/or with other structured data sources. Our framework allows constructs from multiple modelling languages to co-exist within the same intermediate schema, and allows automatic translation of data, queries and updates between semantically equivalent or overlapping heterogenous schemas.

1 Introduction

The presentation-oriented nature of HTML has been widely recognised as being an impediment to building efficient search engines and query languages for information available on the WWW. XML is a more effective means of describing the semantic content of WWW documents, with presentational information being specified using a separate language such as XSL. However, XML is still to some extent presentation-oriented, since the structuring of the data for a particular application is often made to suit the later presentation of that data. This is due to XML's hierarchical nature, with tags being nested inside each other, requiring document designers to make an *a priori* choice as to the ordering of the nesting. Whilst languages such as **DTD**s and **XML Schema** serve to structure XML documents, it is still the case that an essentially hierarchical model is being used.

For example, consider an application where a bank has records of customers, their accounts, and the bank site where the account is held. A site may have accounts belong to different customers and a customer may have accounts at several sites. Fig. 2(a) shows how one might list details of customers (called cust) in XML, detailing under each customer the account (called acc) and the site of the account. Alternatively, Fig. 2(b) shows how the same information

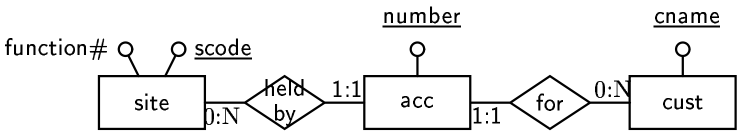


Fig. 1. ER schema of the Bank database

could be listed by site, with details of accounts, and the customer holding the account listed under each site.

The example appears to be that of a many-many relationship between customers and sites. Such choices of ordering as made in Fig. 2(a) and (b) do not arise in data models such as ER or UML, which are based on the classification of entities into types or classes, with relationships between them. For example, Fig. 1 shows an ER model for the same data as in Fig. 2(a) and (b). However, in XML the order of data can be significant and there may be semantic information embedded within XML documents which is assumed by applications but which is not deducible from the document itself. Looking at Fig. 2(a) for example, it might be the case that the first account listed for any customer is to be used for charging any banking costs to *e.g.* charges for customer Jones will be made to account 4411 and not to account 6676.

Apart from these variations in how the ER schema is navigated to produce a hierarchical data structure, there is also a choice as to whether to use XML elements or attributes. The examples given in Figures 2(a) and (b) use XML elements to represent data (as is usually the case in most of the literature) but for the 'leaf' nodes we could equally well use attributes. Fig. 2(c) takes this approach for the same data as shown in Fig. 2(a). XML (with DTDs) also supports a tuple-based representation of data as illustrated in Fig. 2(d) where duplication of data is avoided.

These variations in XML as to how the hierarchy is built, the possibility that ordering may or may not be significant, and the choice of using attributes or elements, has led us to investigate how a semantic data model can be used as the basis for integrating XML data sources with each other and/or with other structured data sources, rather than using XML itself. Note that this approach does not preclude the use of XML as the data transfer mechanism — it only precludes its use as the data modelling language.

The approach that we present in this paper extends our previous work on integrating structured data sources [16,12,11]. In this work, we have used as the common data model a low-level **hypergraph-based data model (HDM)**. We will see later in the paper that the separation in the HDM of data sets and constraints on data sets is useful for modelling XML data, since ordering of XML elements can be represented by extra node and edge information, leaving any other constraints on the data unchanged.

The remainder of the paper is structured as follows. In Section 2 we extend our previous work to show how XML can be represented in the HDM. This leads to the first contribution of the paper — providing a common underpinning for

```
⟨cust⟩
  ⟨cname⟩Jones⟨/cname⟩
  ⟨acc⟩
    ⟨number⟩4411⟨/number⟩
    ⟨site⟩
      ⟨scode⟩32⟨/scode⟩
    ⟨/site⟩
  ⟨/acc⟩
  ⟨acc⟩
    ⟨number⟩6976⟨/number⟩
    ⟨site⟩
      ⟨scode⟩56⟨/scode⟩
      ⟨function⟩Business⟨/function⟩
    ⟨/site⟩
  ⟨/acc⟩
⟨/cust⟩
⟨cust⟩
  ⟨cname⟩Frazer⟨/cname⟩
  ⟨acc⟩
    ⟨number⟩8331⟨/number⟩
    ⟨site⟩
      ⟨scode⟩32⟨/scode⟩
    ⟨/site⟩
  ⟨/acc⟩
⟨/cust⟩
```

(a) by customer, elements preferred

```
⟨site⟩
  ⟨scode⟩32⟨/scode⟩
  ⟨acc⟩
    ⟨number⟩4411⟨/number⟩
    ⟨cust⟩
      ⟨cname⟩Jones⟨/cname⟩
    ⟨/cust⟩
  ⟨/acc⟩
  ⟨acc⟩
    ⟨number⟩8331⟨/number⟩
    ⟨cust⟩
      ⟨cname⟩Frazer⟨/cname⟩
    ⟨/cust⟩
  ⟨/acc⟩
⟨/site⟩
⟨site⟩
  ⟨scode⟩56⟨/scode⟩
  ⟨function⟩Business⟨/function⟩
  ⟨acc⟩
    ⟨number⟩6976⟨/number⟩
    ⟨cust⟩
      ⟨cname⟩Jones⟨/cname⟩
    ⟨/cust⟩
  ⟨/acc⟩
⟨/site⟩
```

(b) by site, elements preferred

```
⟨cust cname="Jones"⟩
  ⟨acc number="4411"⟩
    ⟨site scode="32"/⟩
  ⟨/acc⟩
  ⟨acc number="6976"⟩
    ⟨site scode="56"
        function="Business"/⟩
  ⟨/acc⟩
⟨/cust⟩
⟨cust cname="Frazer"⟩
  ⟨acc number="8331"⟩
    ⟨site scode="32"/⟩
  ⟨/acc⟩
⟨/cust⟩
```

(c) by customer, attributes preferred

```
⟨cust cid="c1" cname="Jones"/⟩
⟨cust cid="c2" cname="Frazer"/⟩
⟨acc aid="a1" number="4411"
    cid="c1" sid="s1"/⟩
⟨acc aid="a2" number="6976"
    cid="c1" sid="s2"/⟩
⟨acc aid="a3" number="8331"
    cid="c2" sid="s1"/⟩
⟨site sid="s1" scode="32"/⟩
⟨site sid="s2" scode="56"
    function="Business"/⟩
```

(d) tuple-based

Fig. 2. Example XML data files for the Bank database

structured data models and XML, and hence the possibility of transforming between them and integrating them. In Section 3 we discuss how XML documents can be transformed into an ER representation, and from there into each other, thus providing a framework in which XML data sources can be integrated and queried in conjunction with each other and with other structured data sources. This leads to the second contribution of the paper — providing a method for transforming between ER and XML representations which allows complete control over whether the various elements of the XML representation have set or list-based semantics. In Section 4 we discuss related work. We give our concluding remarks in Section 5

2 Representing XML in the HDM

In [12] we showed how the HDM can represent a number of higher-level, structured modelling languages such as the ER, relational and UML data models. We also showed how it is possible to transform the constructs of one modelling language into those of another during the process of integrating multiple heterogeneous schemas into a single global schema By extending our work to specify how XML can be represented in the HDM, we are adding XML to the set of modelling languages whose schemas can be transformed into each other and integrated using our framework.

Structured data models typically have a set-based semantics *i.e.* there is no ordering on the extents of the types and relationships comprising the database schema, and no duplicate occurrences. XML's semi-structured nature and the fact that it is presentation-oriented means that lists need to be representable in the HDM, as opposed to just sets which were sufficient for our previous work on transforming and integrating structured data models. In particular, lists are needed because the order in which elements appear within an XML document may be significant to applications and this information should not be lost when transforming and integrating XML documents.

Thus we extend the notions of nodes and edges in HDM schemas (which respectively correspond to types and relationships in higher-level modelling languages) so that the extent of a node or edge may be either a set or a list. For reasons of space we refer the reader to our earlier work [16,12,11] for a full definition of the HDM. Here we give a simplified summary of it together with the extensions needed for representing XML:

A **schema** in the HDM is a triple (Nodes,Edges,Constraints). Nodes and edges are identified by their **schemes**, delimited by double chevrons $\langle\langle$ ... $\rangle\rangle$. The scheme of a node n consists of just the node itself, $\langle\langle n \rangle\rangle$. The scheme of an edge labelled l between nodes $n_1, \ldots, n_m$ is $\langle\langle l, n_1, \ldots, n_m \rangle\rangle$. Edges can also link other edges, so more generally the scheme of an edge is $\langle\langle l, s_1, \ldots, s_m \rangle\rangle$ for some schemes $s_1, \ldots, s_m$. Two primitive transformations are available for adding a node or an edge to an HDM schema, S, to yield a new schema:

addNode(s, q, i, c)
addEdge(s, q, i, c)

Here, s is the scheme of the node or edge being added and q is a query on S which defines the extent of s in terms of the extents of the existing schema constructs (so adding s does not change the information content of S)[1]. i is one of set or list, indicating the collection type of the extent of s, and c is a boolean condition on instances of S which must hold for the transformation to be applicable for that particular instance. Optionally, list may take an argument which determines the ordering of instances of s.

Often the argument c will simply be true, indicating that the transformation applies for all instances of S, and in our previous work i has always been set. In [11] we allowed q to take the special value void, meaning that s can not be derived from the other constructs of S. This is needed when a transformation pathway is being set up between non-equivalent schemas *e.g.* between a component schema and a global schema. For convenience, we use addNode(s,q,i) as a shorthand for addNode(s,q,i,true), addNode(s,q) for addNode(s,q,set,true), and expandNode(s) for addNode(s,void,set,true). We use similar abbreviations for adding edges.

There are also two primitive transformations for deleting a node or an edge from an HDM schema S, delNode(s, q, i, c) and delEdge(s, q, i, c), where s is the scheme of the node or edge being deleted and q is a query which defines how the extent of s can be reconstructed from the extents of the remaining constructs (so deleting s does not change the information content of S), and i the collection type of s. c is again a boolean condition on instances of S which must hold for the transformation to be applicable for that particular instance. Similar shorthands as for the add transformations are used. There are similarly two primitive transformations for adding/deleting a constraint from an HDM schema, addConstraint(s, *constraint*) and delConstraint(s, *constraint*).

Supporting a list collection type does not actually require the HDM to be extended and the above primitive transformations on the HDM can be viewed as 'syntactic sugar' for the primitive transformations we used in previous work. In particular, lists can be supported by introducing a reserved node order, the extent of which is the set of natural numbers. For any scheme s whose extent needs to be viewed as a list, an extra unlabelled edge $\langle\langle \_, s, \text{order} \rangle\rangle$ is used whose instances assign an ordinality to each instance of s. The instances of $\langle\langle \_, s, \text{order} \rangle\rangle$ do not necessarily need to be numbered consecutively, and the ordering of instances of s can be relative to the ordering of instances other schemes.

2.1 Specifying XML in Terms of the HDM

Table 1 summarises how the methodology we described in [12] can be used to build an HDM representation of an XML document. In particular:

1. An XML element may exist by itself and is not dependent on the existence of any other information. Thus, each XML element e is what we term a

[1] We first developed our definitions of schema equivalence, schema subsumption and schema transformation in the context of an ER common data model [9,10] and then applied them to the more general setting of the HDM [16]. A comparison with other approaches to schema equivalence and schema transformation can be found in [10].

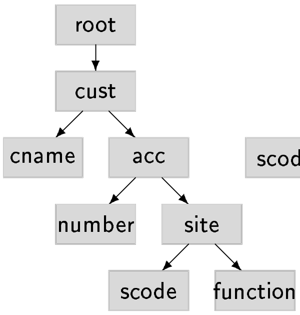
Table 1. Specifying XML constructs in the HDM

Higher Level Construct		Equivalent HDM Representation	
Construct	**element (Elem)**		
Class	nodal, set	Node	$\langle\langle \mathsf{xml}{:}e \rangle\rangle$
Scheme	$\langle\langle e \rangle\rangle$		
Construct	**attribute (Att)**	Node	$\langle\langle \mathsf{xml}{:}e{:}a \rangle\rangle$
Class	nodal-linking, constraint, list	Edge	$\langle\langle \_, \mathsf{xml}{:}e, \mathsf{xml}{:}e{:}a \rangle\rangle$
		Links	$\langle\langle \mathsf{xml}{:}e \rangle\rangle$
Scheme	$\langle\langle e, a \rangle\rangle$	Cons	$\mathsf{makeCard}(\langle\langle \_, \mathsf{xml}{:}e, \mathsf{xml}{:}e{:}a \rangle\rangle, 0{:}1, 1{:}N)$
Construct	**nest-list (List)**	Edge	$\langle\langle \_, \mathsf{xml}{:}e, \mathsf{xml}{:}e_s \rangle\rangle, \langle\langle \_, \langle\langle \_, \mathsf{xml}{:}e, \mathsf{xml}{:}e_s \rangle\rangle, \mathsf{order} \rangle\rangle$
Class	linking, constraint, list	Links	$\langle\langle \mathsf{xml}{:}e \rangle\rangle, \langle\langle \mathsf{xml}{:}e_s \rangle\rangle$
Scheme	$\langle\langle e, e_s \rangle\rangle$	Cons	$\mathsf{makeCard}(\langle\langle \_, \langle\langle \_, \mathsf{xml}{:}e, \mathsf{xml}{:}e_s \rangle\rangle, \mathsf{order} \rangle\rangle, 1{:}1, 0{:}N)$
Construct	**nest-set (Set)**	Edge	$\langle\langle \_, \mathsf{xml}{:}e, \mathsf{xml}{:}e_s \rangle\rangle$
Class	linking, set	Links	$\langle\langle \mathsf{xml}{:}e \rangle\rangle, \langle\langle \mathsf{xml}{:}e_s \rangle\rangle$
Scheme	$\langle\langle e, e_s \rangle\rangle$		

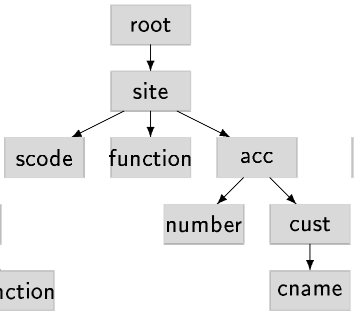
nodal construct [12] and is represented by a node $\langle\langle \mathsf{xml}{:}e \rangle\rangle$ in the HDM[2]. Each instance of e in an XML document corresponds to an instance of the HDM node $\langle\langle \mathsf{xml}{:}e \rangle\rangle$.

2. An XML attribute a of an XML element e may only exist in the context of e, and hence a is what we term a **nodal-linking** construct. It is represented by a node $\mathsf{xml}{:}e{:}a$ in the HDM, together with an associated unlabelled edge $\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e{:}a \rangle\rangle$ connecting the HDM node representing the attribute a to the HDM node representing the element e. A constraint states that each instance of the attribute is related to at least one instance of the element (we use a shorthand notation for expressing cardinality constraints using the function makeCard).
3. XML allows any number of elements $e_1, \ldots, e_n$ to be nested within an element e. This nesting of $e_1, \ldots, e_n$ is represented by a set of edges $\langle\langle \_,\mathsf{xml}{:}e, \mathsf{xml}{:}e_1 \rangle\rangle$, ..., $\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_n \rangle\rangle$. Each such edge is an individual **linking** construct, with scheme $\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_s \rangle\rangle$.
   Each such edge may have list or set semantics. For list semantics, there is an extra edge between the edge $\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_s \rangle\rangle$ and the node order, and a cardinality constraint which states that each instance of $\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_s \rangle\rangle$ is related to precisely one instance of order.
4. XML allows plain text to be placed within a pair of element tags. If a DTD is present, then this text is denoted as PCDATA. We thus assume there is an HDM node called pcdata whose extent consists of plain text instances. An element e can then be associated with a fragment of plain text by means of an edge $\langle\langle \_,\mathsf{e},\mathsf{pcdata} \rangle\rangle$.

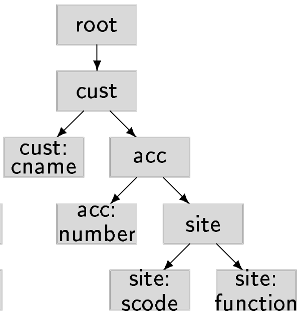
[2] Because it is possible to have present within the same HDM schema constructs from schemas expressed in different higher-level modelling notations, higher-level constructs are distinguished at the HDM level by adding a prefix to their name. This prefix is xml for XML constructs and er for ER constructs.



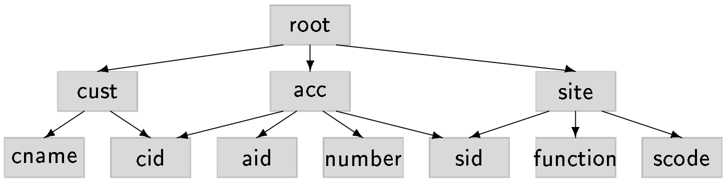
(a) HDM for Fig. 2(a)



(b) HDM for Fig. 2(b)



(c) HDM for Fig. 2(c)



(d) HDM for Fig. 2(d)

Fig. 3. HDM schemas for the XML Documents (a)-(d)

To illustrate this representation of XML documents in the HDM, we show in Fig. 3 the HDM schemas corresponding to the XML documents of Fig. 2 (we have not shown the constraints and the links that each edge has to the order node). In the HDM schema (d), the common child node cid between site and cust, and aid between site and acc, arise if we assume the presence of a DTD in XML document (d) with ID attributes cid and aid on cust and acc, and corresponding IDREF attributes on site.

In general, elements within an XML document may be repeated, with identical attributes and nested elements within them. Hence elements and attributes are uniquely identified by their position within the document. In representing an XML document as an instance of an HDM schema, we thus assume that all nodes are assigned unique identifiers which are generated in some way from their position within the document. For example, the XML elements cust, cname and acc of Fig. 2(a) can be represented by the following instance of the schemes ⟨⟨root⟩⟩, ⟨⟨cust⟩⟩, ⟨⟨cname⟩⟩, ⟨⟨pcdata⟩⟩, and ⟨⟨acc⟩⟩ of the HDM schema in Fig. 3(a), where r1, c1, c2, cn1, cn2, a1, a2, a3 are unique identifiers:

⟨⟨root⟩⟩={r1}
⟨⟨cust⟩⟩={c1,c2}
⟨⟨cname⟩⟩={cn1, cn2}
⟨⟨pcdata⟩⟩={Jones, Frazer}
⟨⟨acc⟩⟩={a1,a2,a3}

```
⟨site⟩
  ⟨scode⟩32⟨/scode⟩
  ⟨acc⟩
    ⟨number⟩4411⟨/number⟩
    ⟨cust⟩
      ⟨cname⟩Jones⟨/cname⟩
    ⟨/cust⟩
  ⟨/acc⟩
⟨/site⟩
```

```
⟨site⟩
  ⟨scode⟩32⟨/scode⟩
  ⟨acc⟩
    ⟨number⟩8331⟨/number⟩
    ⟨cust⟩
      ⟨cname⟩Frazer⟨/cname⟩
    ⟨/cust⟩
  ⟨/acc⟩
⟨/site⟩
```

```
⟨site⟩
  ⟨scode⟩56⟨/scode⟩
  ⟨function⟩Business⟨/function⟩
  ⟨acc⟩
    ⟨number⟩6976⟨/number⟩
    ⟨cust⟩
      ⟨cname⟩Jones⟨/cname⟩
    ⟨/cust⟩
  ⟨/acc⟩
⟨/site⟩
```

Fig. 4. Transformed XML

The nesting relationships between the XML elements cust, cname and acc are represented by the schemes ⟪_,root,cust⟫, ⟪_,⟪_,root,cust⟫,order⟫, ⟪_,cust,cname⟫, ⟪_, ⟪_,cust, cname⟫,order⟫, ⟪_,cname,pcdata⟫, ⟪_, ⟪_,cname,pcdata⟫,order⟫, ⟪_, cust, acc⟫, and ⟪_, ⟪_, cust,acc⟫,order⟫. The instances of these schemes are shown below. Note that the one root r1 of the XML document contains two ordered customers c1 and c2, that c1 contains, in order, one cname cn1 and two accounts a1 and a2, and that c2 contains, in order, one cname cn2 and one account a3:

⟪_,root,cust⟫= {⟨r1,c1⟩, ⟨r1,c2⟩}
⟪_,⟪_,root,cust⟫,order⟫={⟨⟨r1,c1⟩,1⟩, ⟨⟨r1,c2⟩,2⟩}
⟪_,cust,cname⟫={⟨c1,cn1⟩, ⟨c2,cn2⟩}
⟪_,⟪_,cust,cname⟫,order⟫={⟨⟨c1,cn1⟩,1⟩, ⟨⟨c2,cn2⟩,1⟩}
⟪_,cname,pcdata⟫={⟨cn1,Jones⟩, ⟨cn2,Frazer⟩}
⟪_,⟪_,cname,pcdata⟫,order⟫={⟨⟨cn1,Jones⟩,1⟩, ⟨⟨cn2,Frazer⟩,1⟩}
⟪_,cust,acc⟫={⟨c1,a1⟩, ⟨c1,a2⟩, ⟨c2,a3⟩}
⟪_,⟪_,cust,acc⟫,order⟫={⟨⟨c1,a1⟩,2⟩, ⟨⟨c1,a2⟩,3⟩, ⟨⟨c2,a3⟩,2⟩}

An HDM representation which assumed set-based semantics for element containment would simply omit the schemes linking edges to ⟪order⟫.

2.2 Primitive Transformations on XML

In [12] we describe how, once the constructs of some higher-level modelling language have been defined in terms of the HDM constructs, this definition can be used to *automatically derive* the necessary set of primitive transformations on schemas expressed in that language. Thus, from the specification of XML given in Table 1 and described in the previous section, the primitive transformations shown in Table 2 can be automatically derived. The left-hand column of this table gives the names and arguments of the XML transformations and the right-hand column gives their implementation as sequences of the primitive transformations on the underlying HDM representation.

To illustrate the use of these primitive transformations on XML, we give below a sequence of primitive transformations that transform the document in

Table 2. Derived transformations on XML

Transformation on XML	Equivalent Transformation on HDM
$\mathsf{renameElem_{xml}}(e,e')$	$\mathsf{renameNode}(\langle\langle \mathsf{xml}{:}e\rangle\rangle,\langle\langle \mathsf{xml}{:}e'\rangle\rangle)$
$\mathsf{addElem_{xml}}(e,q)$	$\mathsf{addNode}(\langle\langle \mathsf{xml}{:}e\rangle\rangle,q)$
$\mathsf{delElem_{xml}}(e,q)$	$\mathsf{delNode}(\langle\langle \mathsf{xml}{:}e\rangle\rangle,q)$
$\mathsf{renameAtt_{xml}}(a,a')$	$\mathsf{renameNode}(\langle\langle \mathsf{xml}{:}e{:}a\rangle\rangle,\langle\langle \mathsf{xml}{:}e{:}a'\rangle\rangle)$
$\mathsf{addAtt_{xml}}(e,a,q_{att},q_{assoc})$	$\mathsf{addNode}(\langle\langle \mathsf{xml}{:}e{:}a\rangle\rangle,q_{att})$; $\mathsf{addEdge}(\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e{:}a\rangle\rangle,q_{assoc})$; $\mathsf{addConstraint}(x \in \langle\langle \mathsf{xml}{:}e{:}a\rangle\rangle \rightarrow \langle \_,x\rangle \in \langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e{:}a\rangle\rangle)$
$\mathsf{delAtt_{xml}}(e,a,q_{att},q_{assoc})$	$\mathsf{delConstraint}(x \in \langle\langle \mathsf{xml}{:}e{:}a\rangle\rangle \rightarrow \langle \_,x\rangle \in \langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e{:}a\rangle\rangle;)$ $\mathsf{delEdge}(\langle\langle \_,\mathsf{xml}{:}\mathsf{e},\mathsf{xml}{:}e{:}a\rangle\rangle,q_{assoc})$; $\mathsf{delNode}(\langle\langle \mathsf{xml}{:}e{:}a\rangle\rangle,q_{att})$
$\mathsf{addList_{xml}}(\langle\langle e,e_s\rangle\rangle,q,p)$	$\mathsf{addEdge}(\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_s\rangle\rangle,q,\mathsf{list}(p))$
$\mathsf{delList_{xml}}(\langle\langle e,e_s\rangle\rangle,q,p)$	$\mathsf{delEdge}(\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_s\rangle\rangle,q,\mathsf{list}(p))$
$\mathsf{addSet_{xml}}(\langle\langle e,e_s\rangle\rangle,q)$	$\mathsf{addEdge}(\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_s\rangle\rangle,q)$
$\mathsf{delSet_{xml}}(\langle\langle e,e_s\rangle\rangle,q)$	$\mathsf{delEdge}(\langle\langle \_,\mathsf{xml}{:}e,\mathsf{xml}{:}e_s\rangle\rangle,q)$

Fig. 2(a) to that in Fig. 4. The HDM schema representation of the former is shown in Fig. 3(a) and of the latter in Fig. 3(b).

$\mathsf{addList_{xml}}(\langle\langle \mathsf{root,site}\rangle\rangle,\{\langle \mathsf{r1},x\rangle \mid \langle x\rangle \in \langle\langle \mathsf{site}\rangle\rangle\},\ \mathsf{after}(\langle\langle \mathsf{root,cust}\rangle\rangle))$
$\mathsf{addList_{xml}}(\langle\langle \mathsf{site,acc}\rangle\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\langle \mathsf{acc,site}\rangle\rangle\},\ \mathsf{after}(\langle\langle \mathsf{site,function}\rangle\rangle))$
$\mathsf{addList_{xml}}(\langle\langle \mathsf{acc,cust}\rangle\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\langle \mathsf{cust,acc}\rangle\rangle\},\ \mathsf{after}(\langle\langle \mathsf{acc,number}\rangle\rangle))$
$\mathsf{delList_{xml}}(\langle\langle \mathsf{acc,site}\rangle\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\langle \mathsf{site,acc}\rangle\rangle\},\ \mathsf{after}(\langle\langle \mathsf{acc,cust}\rangle\rangle))$
$\mathsf{delList_{xml}}(\langle\langle \mathsf{cust,acc}\rangle\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\langle \mathsf{acc,cust}\rangle\rangle\},\ \mathsf{after}(\langle\langle \mathsf{cust,cname}\rangle\rangle))$
$\mathsf{delList_{xml}}(\langle\langle \mathsf{root,cust}\rangle\rangle,\{\langle \mathsf{r1},x\rangle \mid \langle x\rangle \in \langle\langle \mathsf{cust}\rangle\rangle\},\ \mathsf{before}(\langle\langle \mathsf{root,site}\rangle\rangle))$

Notice that the above transformation consists of a 'growing phase' where new constructs are added to the XML model, followed by a 'shrinking phase' where the constructs now rendered redundant are removed. This is a general characteristic of schema transformations expressed within our framework.

The availability of a transformation pathway from one schema to another allows queries expressed on one schema to be automatically translated onto the other. For example, the following query on Fig. 3(a) finds the names of customers with accounts at site 32[3]:

$$\{n \mid \langle r,c\rangle \in \langle\langle \mathsf{root,cust}\rangle\rangle \wedge \langle c,n\rangle \in \langle\langle \mathsf{cust,name}\rangle\rangle \wedge \langle c,a\rangle \in \langle\langle \mathsf{cust,acc}\rangle\rangle \wedge \langle a,s\rangle \in \langle\langle \mathsf{acc,site}\rangle\rangle \wedge \langle s,sc\rangle \in \langle\langle \mathsf{site,scode}\rangle\rangle \wedge sc = 32\}$$

By substituting deleted constructs appearing in this query by their restoring query *i.e.* by the 3rd argument of the $\mathsf{delList_{xml}}()$ transformations above, it is possible to translate the query into the following equivalent query on Fig. 3(b) (see [11] for a general discussion of query/update/data translation in our framework):

[3] We do not consider here the issue of translating between different query languages, and assume a 'neutral' intermediate query notation, namely set comprehensions, into which queries submitted to local or global schemas can be translated as a first step.

$$\{n \mid \langle r,c\rangle \in \{\langle \mathsf{r1},x\rangle \mid \langle x\rangle \in \langle\!\langle \mathsf{cust}\rangle\!\rangle\} \wedge \langle c,n\rangle \in \langle\!\langle \mathsf{cust},\mathsf{name}\rangle\!\rangle \wedge$$
$$\langle c,a\rangle \in \{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\!\langle \mathsf{acc},\mathsf{cust}\rangle\!\rangle\} \wedge$$
$$\langle a,s\rangle \in \{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\!\langle \mathsf{site},\mathsf{acc}\rangle\!\rangle\} \wedge \langle s,sc\rangle \in \langle\!\langle \mathsf{site},\mathsf{scode}\rangle\!\rangle \wedge sc = 32\}$$

Any sequence of primitive transformations $t_1;\ldots;t_n$ is automatically reversible by the sequence $t_n^{-1};\ldots;t_1^{-1}$, where the inverse of an add transformation is a del transformation with the same arguments, and vice versa. Thus, the reverse transformation from Fig. 3(b) to Fig. 3(a) can be automatically derived to be:

$\mathsf{addList_{xml}}(\langle\!\langle \mathsf{root,cust}\rangle\!\rangle,\{\langle \mathsf{r1},x\rangle \mid \langle x\rangle \in \langle\!\langle \mathsf{cust}\rangle\!\rangle\}, \mathsf{before}(\langle\!\langle \mathsf{root,site}\rangle\!\rangle))$
$\mathsf{addList_{xml}}(\langle\!\langle \mathsf{cust,acc}\rangle\!\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\!\langle \mathsf{acc,cust}\rangle\!\rangle\}, \mathsf{after}(\langle\!\langle \mathsf{cust,cname}\rangle\!\rangle))$
$\mathsf{addList_{xml}}(\langle\!\langle \mathsf{acc,site}\rangle\!\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\!\langle \mathsf{site,acc}\rangle\!\rangle\}, \mathsf{after}(\langle\!\langle \mathsf{acc,cust}\rangle\!\rangle))$
$\mathsf{delList_{xml}}(\langle\!\langle \mathsf{acc,cust}\rangle\!\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\!\langle \mathsf{cust,acc}\rangle\!\rangle\}, \mathsf{after}(\langle\!\langle \mathsf{acc,number}\rangle\!\rangle))$
$\mathsf{delList_{xml}}(\langle\!\langle \mathsf{site,acc}\rangle\!\rangle,\{\langle x,y\rangle \mid \langle y,x\rangle \in \langle\!\langle \mathsf{acc,site}\rangle\!\rangle\}, \mathsf{after}(\langle\!\langle \mathsf{site,function}\rangle\!\rangle))$
$\mathsf{delList_{xml}}(\langle\!\langle \mathsf{root,site}\rangle\!\rangle,\{\langle \mathsf{r1},x\rangle \mid \langle x\rangle \in \langle\!\langle \mathsf{site}\rangle\!\rangle\}, \mathsf{after}(\langle\!\langle \mathsf{root,cust}\rangle\!\rangle))$

This reverse transformation can now be used to translate queries on Fig. 3(b) to queries on Fig. 3(a).

We finally observe the similarity of Fig. 4 to the document in Fig. 2(b), which in fact has the same schema, Fig. 3(b). It is not possible to capture the non-duplication of site 32 in Fig. 2(b) using the above XML-to-XML transformation. It is however possible to transform Fig. 2(a) to Fig. 2(b) going via the ER model of Fig. 1, and we discuss how in Section 3 below.

3 Transforming between ER and XML

In [12] we showed how ER schemas can be represented in the HDM, and we refer the reader to that paper for details. Here we recall that ER entity classes are nodal constructs represented by HDM nodes, ER relationships are linking constructs represented by an HDM edge and a cardinality constraint, and ER attributes are nodal-linking constructs represented by a node, an edge linking this node to the node representing the attribute's entity class, and a cardinality constraint. The primitive transformations on ER schemas consist of the operations add, del, expand, contract and rename on the constructs Entity, Attribute or Relationship.

3.1 Automated Generation of a Canonical ER Schema from XML

Using the sets of primitive transformations on XML and ER schemas, an XML document or set of documents can be automatically transformed into a 'canonical' ER schema by applying the rules given below. For ease of reading, we have specified these rules at the level of the HDM, and the primitive transformations on the HDM, rather than at the higher level of the XML and ER constructs. As with the example in Section 2.2 above, the transformation consists of a 'growing phase' where new ER constructs are added to the schema, followed by a 'shrinking phase' where the XML constructs now rendered redundant are removed:

1. Each node representing an XML element (*i.e.* is named xml:e for some e and is not order or pcdata) generates an ER entity class of the same name e, with a key attribute also named e (this explicit key attribute is needed because there is no implicit notion of unique object identifiers in the ER model):

   addNode($\langle\langle$er:$e\rangle\rangle$,$\langle\langle$xml:$e\rangle\rangle$)
   addNode($\langle\langle$er:e:$e\rangle\rangle$,$\langle\langle$xml:$e\rangle\rangle$)
   addEdge($\langle\langle$_,er:e,er:e:$e\rangle\rangle$, $\{\langle x,x\rangle \mid x \in \langle\langle$xml:$e\rangle\rangle\}$)
   addConstraint(makeCard($\langle\langle$_,er:e,er:e:$e\rangle\rangle$,1:1,1:1))

   Note that for the purposes of this rule, the root of the document should also be considered as a node, so there will always be a root entity in the ER model. Each instance of the root entity will represent a distinct XML document.
2. Each node representing an XML attribute (*i.e.* is named xml:e:a for some e and a) generates an attribute of the ER entity class e^4:

   addNode($\langle\langle$er:e:$a\rangle\rangle$,$\langle\langle$xml:e:$a\rangle\rangle$)
   addEdge($\langle\langle$_,er:e,er:e:$a\rangle\rangle$, $\langle\langle$_,xml:e,xml:e:$a\rangle\rangle$)
   addConstraint(copyCard($\langle\langle$_,xml:e,xml:e:$a\rangle\rangle$, $\langle\langle$_,er:e,er:e:$a\rangle\rangle$))

3. Each node linked by an edge to $\langle\langle$pcdata$\rangle\rangle$ has an attribute added called pcdata whose extent will consist of the instances of $\langle\langle$pcdata$\rangle\rangle$ with which instances of the node are associated:

   addNode($\langle\langle$er:e:pcdata$\rangle\rangle$, $\{x \mid \langle _, x\rangle \in \langle\langle$_, xml:$e$, xml:$e$:pcdata$\rangle\rangle\}$)
   addEdge($\langle\langle$_,er:e,er:e:pcdata$\rangle\rangle$, $\langle\langle$_,xml:e,xml:e:pcdata$\rangle\rangle$)
   addConstraint(copyCard($\langle\langle$_,xml:e,xml:e:pcdata$\rangle\rangle$, $\langle\langle$_,er:e,er:e:pcdata$\rangle\rangle$))

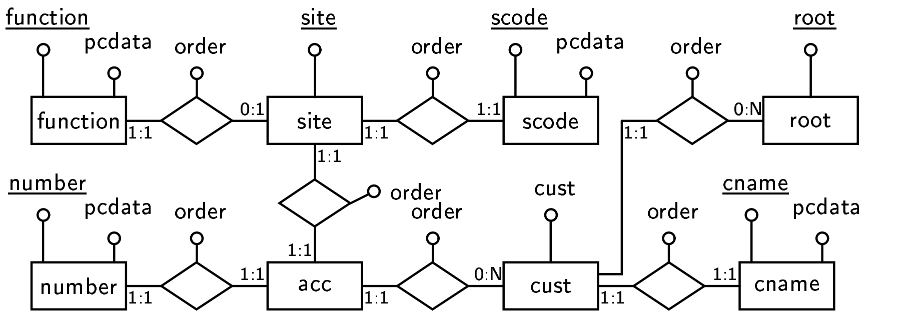
4. Each nesting edge between two XML elements generates an edge between the two corresponding ER entity classes representing a relationship between them:

   addEdge($\langle\langle$_,er:e,er:$e_s\rangle\rangle$, $\langle\langle$_,xml:e,xml:$e_s\rangle\rangle$)

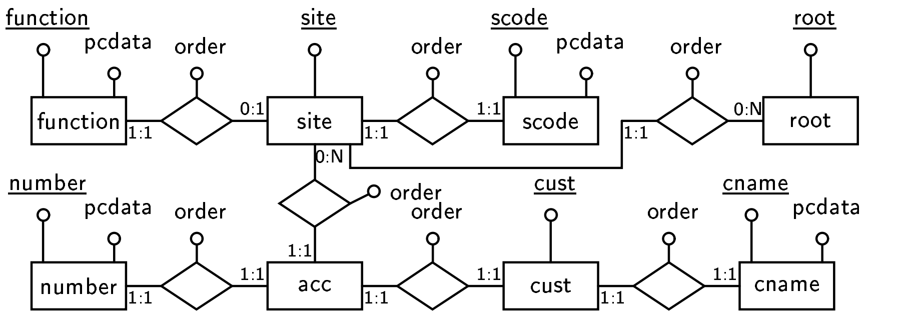
   For list-based nestings, each instance of the XML nesting edge will be linked to the order node. This information can be represented as an attribute order of the new ER relationship:

   addEdge($\langle\langle$_,$\langle\langle$_,er:e,er:$e_s\rangle\rangle$,order$\rangle\rangle$, $\langle\langle$_,$\langle\langle$_,xml:e,xml:$e_s\rangle\rangle$,order$\rangle\rangle$)

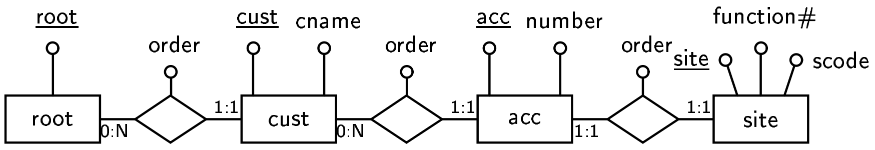
   Any constraint on the XML nesting edge should also be copied over onto the new ER relationship.
5. As a result of the above add transformations, all the XML constructs are rendered semantically redundant and can finally be progressively removed from the schema by applying a sequence of del transformations.



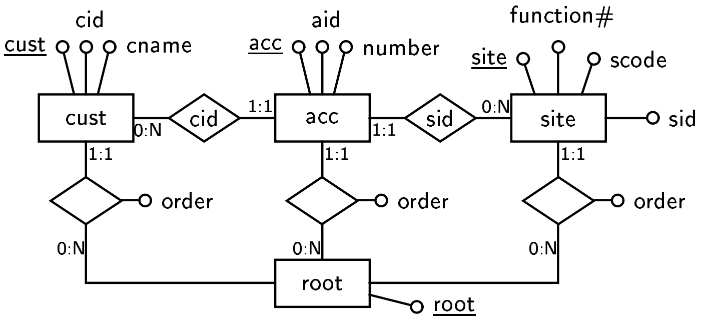
(a)



(b)



(c)



(d)

Fig. 5. Canonical ER representations of the XML documents (a)-(d)

The result of applying the above rules to the four HDM schemas representing the XML documents of Fig. 2 is shown in Fig. 5. We observe that each differs to a greater or lesser extent from the ER schema of Fig. 1 which we assumed was used to generate the original XML documents. We now study how to derive the original ER schema.

3.2 Manual Refinement of Canonical ER Schemas

Comparing Fig. 5(a) and (b) with Fig. 1, we notice the following characteristics of using XML to represent ER data, and hence possible refinements that may be made to the canonical ER schemas automatically generated by the method described in the previous subsection:

1. It is often the case that the ordering of elements in XML is not semantically significant, in which case the order attribute in the ER schema can be ignored. For example, if the order of site and number within acc is not significant in Fig. 2(a), then the order attribute in Fig. 5(a) can be removed from the relationship between acc and number, and from that between acc and site, as follows:

   $\mathsf{contractAttribute_{er}}(\langle\langle\langle\langle\_,\mathsf{acc},\mathsf{number}\rangle\rangle,\mathsf{order}\rangle\rangle)$
   $\mathsf{contractAttribute_{er}}(\langle\langle\langle\langle\_,\mathsf{acc},\mathsf{site}\rangle\rangle,\mathsf{order}\rangle\rangle)$

   In contrast if, as discussed in the introduction, the order of accounts within customers was significant, then the order attribute must remain on $\langle\langle\_,\mathsf{cust},\mathsf{acc}\rangle\rangle$.
2. Using XML elements to represent attributes in the original ER schema has resulted in entity classes being created in the new ER schema.
   For both Fig. 5(a) and (b) we see that function, number, cname and scode all have this property. The reason that this has occurred is because XML documents contain additional information regarding the position of constructs within the document, which is reflected by the order information being stored in the corresponding ER edges and the generation of a unique identifier for each XML element. If this ordering information need not be preserved in the ER schema (*i.e.* rule (1) above has been applied), these entity classes can be transformed into attributes, using a standard pattern of transformations which we give below for the case of the scode entity class:

   $\mathsf{addAttribute_{er}}(\langle\langle\mathsf{site},\mathsf{scode}\rangle\rangle,\ \{\langle x,z\rangle \mid \langle x,y\rangle \in \langle\langle\_,\mathsf{site},\mathsf{scode}\rangle\rangle \wedge$
   $\quad\langle y,z\rangle \in \langle\langle\mathsf{scode},\mathsf{pcdata}\rangle\rangle\},\ \mathsf{copyCard}(\langle\langle\_,\mathsf{site},\mathsf{scode}\rangle\rangle,\langle\langle\mathsf{site},\mathsf{scode}\rangle\rangle))$
   $\mathsf{contractAttribute_{er}}(\langle\langle\mathsf{scode},\mathsf{pcdata}\rangle\rangle)$
   $\mathsf{contractAttribute_{er}}(\langle\langle\mathsf{scode},\mathsf{scode}\rangle\rangle)$
   $\mathsf{contractRelationship_{er}}(\langle\langle\_,\mathsf{site},\mathsf{scode}\rangle\rangle)$
   $\mathsf{contractEntity_{er}}(\langle\langle\mathsf{scode}\rangle\rangle)$

[4] Here, the function copyCard() translates the constraint on an XML edge to the same constraint on an ER edge.

The result of applying these transformations to function, number, cname and scode entities in Fig. 5(a) results in the ER schema shown in Fig. 5(c) *i.e.* the ER schema generated from the XML document of Fig. 2(c).

3. The order in which elements are nested within each other in an XML document effects the cardinality of the relationships between the corresponding ER entity classes.
   For example, in Fig. 5(a) sites are repeated for each customer, and thus each site entity is associated with only one acc (there will be two sites with scode 32 in our example, each with one account). This can be corrected by changing the key of site from the automatically generated, position-related, key to be scode instead. This involves renaming the site entity to site', creating a new site entity with the new scode key, copying the attributes and relationships across from site' to site, and finally removing site':

```
renameEntity_er(⟨⟨site⟩⟩,⟨⟨site'⟩⟩)
addEntity_er(⟨⟨site⟩⟩,{x | ⟨_, x⟩ ∈ ⟨⟨site′, scode⟩⟩})
addAttribute_er(⟨⟨site,scode⟩⟩,{⟨x, x⟩ | ⟨x⟩ ∈ ⟨⟨site′, scode⟩⟩},1 : 1,1 : 1)
addAttribute_er(⟨⟨site,function⟩⟩,
        {⟨x, y⟩ | ⟨z, x⟩ ∈ ⟨⟨site′, scode⟩⟩ ∧ ⟨z, y⟩ ∈ ⟨⟨site′, function⟩⟩},
        copyCard(⟨⟨site',function⟩⟩,⟨⟨site,function⟩⟩))
addRelationship_er(⟨⟨_,site,acc⟩⟩,
        {⟨x, y⟩ | ⟨z, x⟩ ∈ ⟨⟨site′, scode⟩⟩ ∧ ⟨z, _, y⟩ ∈ ⟨⟨_, site′, acc⟩⟩},
        copyCard(⟨⟨_,site',acc⟩⟩,⟨⟨_,site,acc⟩⟩))
contractAttribute_er(⟨⟨site',function⟩⟩)
contractAttribute_er(⟨⟨site',scode⟩⟩)
contractAttribute_er(⟨⟨site',site⟩⟩)
contractRelationship_er(⟨⟨_,site',acc⟩⟩)
contractEntity_er(⟨⟨site'⟩⟩)
```

As already discussed, Fig. 5(c) is an intermediate stage of the transformations applied to Fig. 5(a) and only changing the key of entity classes from the automatically generated position-related key to another attribute remains to be done.

Finally, the ‘flat’ representation of information in the XML document of Fig. 2(d) results in the ER schema of Fig. 5(d). This is the closest to Fig. 1, with both the same attributes present and the same cardinality constraints. If they are not required for applications, we can simply drop the id nodes using contractAttribute$_{\text{er}}$(⟨⟨site,id⟩⟩). We can also apply the key transformation rules to change the key from the position-related key to another attribute.

In summary, we have shown in this section how four XML documents d_1 to d_4 can be automatically transformed into ER schemas er_1 to er_4 which may then be manually transformed into a single ER schema er. To transform an XML document d_i to another XML document d_j the forward transformation $d_i \rightarrow er_i \rightarrow er$ can be applied, followed by the reverse transformation $er \rightarrow er_j \rightarrow d_j$.

4 Related Work

There has been much work on translation (a) between XML and structured data models, and (b) between different XML formats. Considering first (a), a common approach is to use some XML query language to build a new XML document as a view of another XML document. In common with other approaches to representing XML or semi-structured data, *e.g.* [15,2,1,3,17,5], we use a graph-based data model which supports unique identifiers for XML elements. One difference with our approach is how ordering is represented. Our use of the order node in the HDM graph allows list semantics to be preserved with the data if desired. Alternatively, the links to order can be ignored for a set semantics. Generally, the provenance of the HDM as a common data model for structured data gives it a rather different flavour to models with an XML or semi-structured data provenance. For example, (1) there are constraints in an HDM schema, and (2) nodal, linking and nodal-linking constructs are used to represent XML documents as opposed to just nodes and edges.

Considering the issue of translating between structured data models and XML, this has also received considerable attention [4,7,8,18,1,14,3]. In contrast to this previous work, what we have proposed here is a general method for translating between structured and XML representations of data, not tied to any specific structured data model. Specifying XML in the HDM has allowed the set of primitive transformations on XML to be automatically derived in terms of sequences of primitive transformations on the HDM. Both high and low-level transformations are automatically reversible because they require the specification of a constructing/restoring query for add/del transformations. These two-way transformation pathways between schemas can be used to auomatically translate data, queries and updates in either direction. Work in schema translation and matching [1,14,3] can be utilised to enhance our framework by automatically or semi-automatically deriving the constructing/restoring query in add/del transformations where possible.

5 Concluding Remarks

We have developed our previous work which supports the integration of structured data sources to also handle XML documents. We have demonstrated how XML documents can be automatically transformed into ER schemas, and have discussed how to further transform such schemas so that they are semantically closer to the original source database schema that the XML documents may have been generated from. This restructuring is based on the application of well-understood schema transformation rules.

The process of restructuring the ER schema would be a more complex task for data sources where the information has never been held in a structured form *i.e.* 'really' semi-structured data. In [19] techniques are presented for discovering structured associations from such data, and we are studying how our framework can be adapted to use such techniques.

In a longer version of this paper [13] we show how to represent in the HDM the additional information available in an XML DTD, if present. We are also studying how the approach proposed in this paper may be adapted to use XML Schema definitions [6] in place of DTDs, which will enable some of the restructuring rules for ER models to be automatically inferred.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. In *Proceedings of ICDT'97*, 1997.
2. S. Abiteboul, *et al.* The Lorel query language for semistructured data. *Journal on Digital Libraries*, 1(1), 1997.
3. C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *Proceedings of ICDT'99*, 1999.
4. V. Christophides, S. Cluet, and J. Siméon. On wrapping query languages and efficient XML integration. *SIGMOD RECORD*, 29(2):141–152, 2000.
5. A. Deutsch, M. Fernández, D. Florescu, A. Levy, and D. Suciu. A query language for XML. In *Proceedings of 8th International World Wide Web Conference*, 1999.
6. D.C. Fallside. XML schema part 0: Primer; W3C working draft. Technical report, W3C, April 2000.
7. M. Fernández, W-C. Tan, and D. Suciu. SilkRoute: Trading between relations and XML. In *Proceedings of 9th International World Wide Web Conference*, 2000.
8. D. Florescu and D. Kossmann. Storing and querying XML data using an RDBMS. *Bulletin of the Technical Committee on Data Engineering*, 22(3):27–34, September 1999.
9. P.J. McBrien and A. Poulovassilis. A formal framework for ER schema transformation. In *Proceedings of ER'97*, volume 1331 of *LNCS*, pages 408–421, 1997.
10. P.J. McBrien and A. Poulovassilis. A formalisation of semantic schema integration. *Information Systems*, 23(5):307–334, 1998.
11. P.J. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — a schema transformation approach. In *Proceedings of ER99*, volume 1728 of *LNCS*, pages 96–113. Springer-Verlag, 1999.
12. P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Advanced Information Systems Engineering, 11th International Conference CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer-Verlag, 1999.
13. P.J. McBrien and A. Poulovassilis. A semantic approach to integrating XML and structured data sources. Technical report, Birkbeck College and Imperial College, November 2000.
14. T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of VLDB'98*, 1998.
15. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of ICDE'95*, 1995.
16. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
17. R.Goldman, J. McHugh, and J. Widom. From semistructured data to XML: Migrating the Lore data model and query language. In *Proceedings of WebDB*, 1999.
18. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and objectives. In *Proceedings of the 25th VLDB Conference*, pages 302–314, 1999.
19. K. Wang and H. Liu. Discovering structural association of semistructured data. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):353–371, 2000.