A Case Tool for Designing XML Views

Ya Bing Chen, Tok Wang Ling, Mong Li Lee
School of Computing, National University of Singapore
(chenyabi, lingtw, leeml)@comp.nus.edu.sg

Abstract
XML views are essential for managing XML data on the Web. Like views in traditional databases, XML views allow users to see data from different perspectives. Moreover, XML views are able to offer a structured interface over semistructured data and make it possible to integrate heterogeneous data source. In this paper, we describe a graphical case tool for designing XML views with the following novel features. First, it supports validation of views by checking if a view conforms to the semantics in the underlying source schemas. The feature is critical for XML views design because it avoid producing meaningless views. Second, the case tool provides a scheme to store XML data into an object-relational database that reduces redundancies and facilitates efficient processing of queries over XML views.

1. Introduction

Although XML [XML] (eXtensible Markup Language) is originally a document mark-up language, it is becoming a standard for publishing and exchanging data on the Web. Unlike HTML, XML enforces explicit structuring and separates presentation from data content. If data sources contain information with more structure, it will be more appropriate to use XML rather than HTML to export their data to the Web and exchange it with other data sources.

If an information source chooses to use XML to export their data, then it becomes important to manage the XML data efficiently, and provide for the systematic import and/or export of XML data. As in traditional databases, users often need to view data from different perspectives. XML views are able to offer a structured interface over semistructured data and make it possible to integrate heterogeneous data source [A99].

Several systems such as ActiveViews [AAC+99] and MIX [BGL+99] have been developed to support XML views. The ActiveViews system defines views using the object-oriented model. MIX system integrates heterogeneous data sources and offers views based on the underlying data sources. While both systems provide for the definition of XML views, they do not guarantee that the views designed are valid.

In this paper, we offer a novel case tool for designing XML views – XML Views System (XVS), which has important new features. XVS offers a graphical views design facility besides query expression, which is more straightforward than designing views only using query expression. The designed views will also be validated so that no meaningless views will be produced. The feature is critical for designing XML views because it guarantees all designed views are meaningful. Unfortunately it is not supported by the other systems. Moreover, our system stores XML data into an object-relational database using a particular method by mapping XML data to an ORA-SS schema [LLD01], which is more efficient than storing in an object database.

The rest of the paper is organized as follows. In section 2, we will give some background of our system, particularly, the data model and the view definition language. Section 3 gives an overview of XVS. We use an example to demonstrate the workings of the system. Section 4 discusses some related work and we conclude in Section 5.

2. Preliminaries

2.1 ORA-SS Data Model

The ORA-SS (Object-Relationship-Attribute model for Semistructured data) data model has three basic concepts: object classes, relationship types and attributes. An object class is similar to a set of entities in the real world, an entity type in an ER diagram, a class in an object-oriented diagram or an element in XML document. A relationship type describes a relationship among object classes. Attributes are properties, and may belong to an object class or a relationship type. The ORA-SS data model consists of four diagrams: the schema diagram, the instance diagram, the functional dependency diagram and the inheritance diagram. A full description of the data model can be found in [DWLL00]. In this paper, we will focus on the schema diagram.

Figure 1 shows the ORA-SS schema diagram of an instance diagram depicted in Figure 2. An object class is represented as a labeled rectangle. A relationship type between two object classes in an ORA-SS schema diagram can be described by *name, n, p, c,* where *name* denotes the name of the relationship type, *n* is an integer indicating the degree of the relationship type (n = 2 indicates binary, n = 3 indicates ternary, etc.), *p* is the participation constraint of the parent object class in the relationship type, and *c* is the participation constraint of the child object class. The participation constraints are defined using the min:max notation.

Attributes are denoted by labeled circles. Keys are filled circles. Attributes of an object class can be distinguished from attributes of a relationship type. The former has no label on its incoming edge while the latter has the name of the relationship type to which it belongs on its incoming edge.

ORA-SS is a semantically rich data model. It not only reflects the nested structure of semistructured data, but also distinguishes between object classes, relationship types and attributes. Moreover, ORA-SS makes it possible to specify the participation constraints of object classes in relationship types and distinguish between attributes of relationship types and attributes of object classes. Such information is lacking in existing semistructured data models. However, it is essential for designing valid XML views because the labels on attributes and relationship types convey necessary semantics to distinguish between meaningful views and views that are not meaningful. Therefore, we adopt ORA-SS as the data model for valid XML views design. The difference between invalid views and valid views will be illustrated in section 3.2.

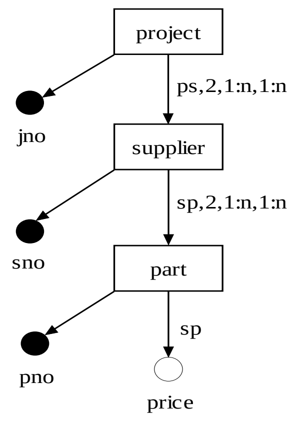

Figure 1: ORA-SS Schema Diagram

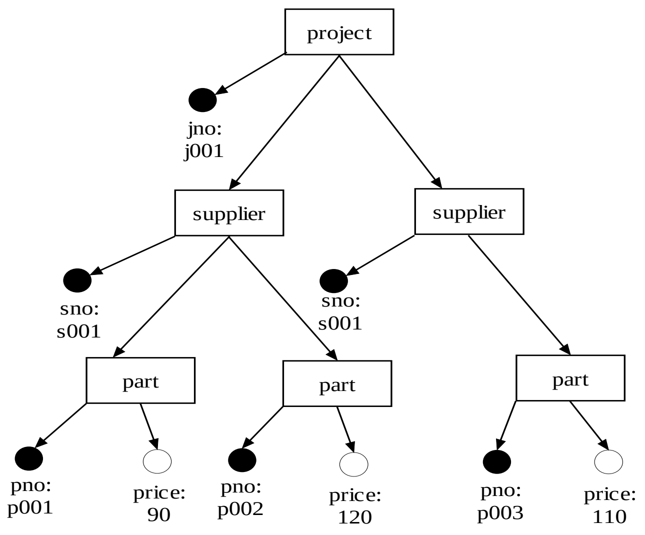

Figure 2: ORA-SS Instance Diagram

2.2 View Definition Language

The World Wide Web Consortium recently proposed an XML query language called XQuery [XQ]. XQuery provides flexible query facilities to extract data from real and virtual documents on the Web. Although XQuery do not provide for the definition of views, we can extend it easily to include the definition.

The basic form of a view definition clause consists of a For, Let, Where and Return (FLWR) expression. This is in fact the core part of XQuery. The definition of a view is as follows:

"Create View As view name" followed by a FLWR expression.

The For clause and/or Let clause serve to bind values to variables. The values to be bound to variables are represented by path expression. The For clause is used when iteration is needed, while the Let clause is used when there is no iteration. The Where clause is a set of predicate expressions, which are used to further filter the binding-tuples, generated by the For and Let clauses. The Return clause generates the structure of the view in XML notation. A full description of XQuery can be found in [XQ]. We will adopt XQuery not only as the view definition language, but also as the query language used to query views.

3. Architecture of XVS

In this section, we give an overview of our XML Views System – XVS. We will first introduce the high level architecture of our system, and then we will illustrate the work process of our system with an example. XVS offers new features that cannot be supported by other view systems. First, it supports graphical facility for designing views, which is straightforward to use. Based on graphical source schema, users can dynamically design views by selecting, dropping or transforming the structure. Secondly, XVS supports validation of views, which checks if a view conforms to semantics in source schemas. The feature therefore offers a guarantee that each designed view is meaningful. XVS also supports storing XML data into an object-relational database, which adopts a novel mapping method [LLD01]. The method stores an XML document into sets of object tables and sets of relation tables, which takes advantage of ORA-SS and reduces redundancies.

The architecture of the XML Views System (XVS) is depicted in Figure 3.

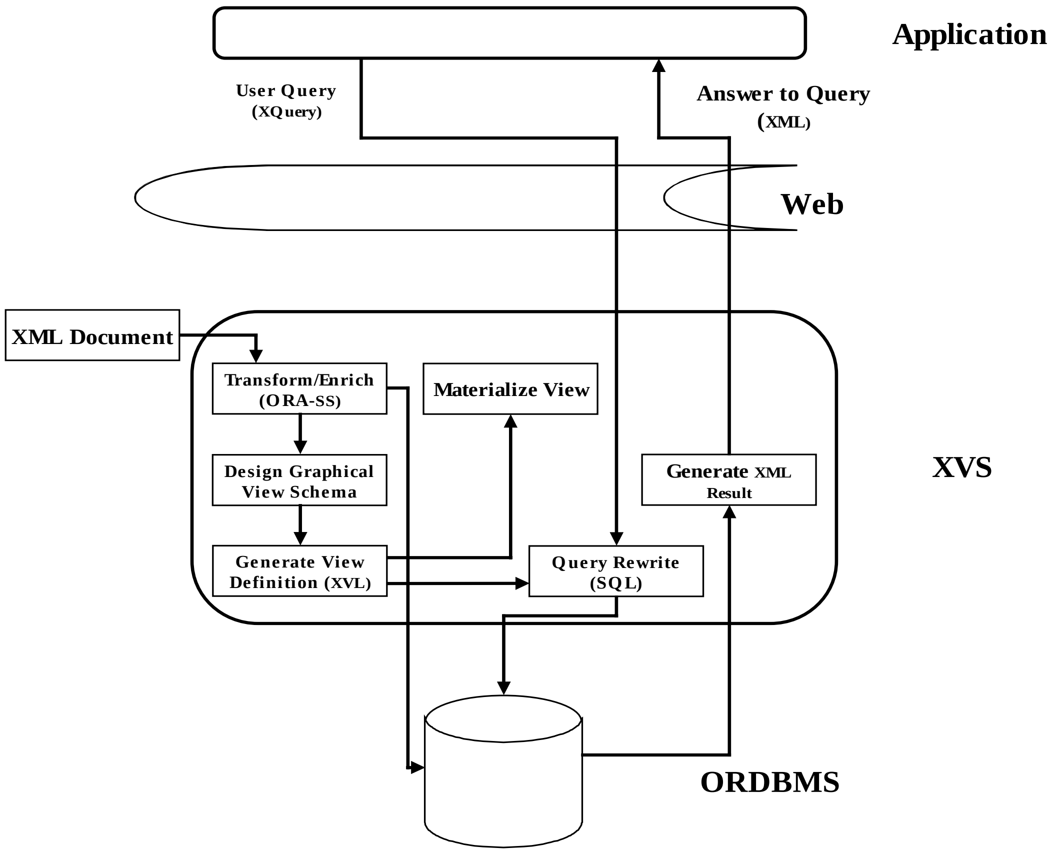

Figure 3. The Architecture of XVS

Each module of the system is described as follows:

- **Transform/Enrich (ORA-SS):** This module takes an XML document and generates the corresponding ORA-SS graphical schema diagram. It also allows users to enrich the ORA-SS schema diagram to add necessary semantic information for XML views design. For example, whether an attribute belongs to an object or a relationship can be indicated in the diagram. After that, these XML documents will be stored into an object-relationship database by mapping the ORA-SS schema diagram into a series of objects and relationship tables [LLD01].
- **Design graphical view schema:** This allows the user to design a view graphically over a source schema. Objects and/or attributes can be selected or dropped, and the structure of the source schema can be changed.
- **Generate view definition:** A view definition in FLWC expression is generated from the graphical view schema.
- **Query rewrite (SQL):** This module rewrites a user query based on the view definition and generates an equivalent SQL query. The SQL query will be passed

to the underlying object-relationship database and executed. The result will be returned to the XVS in a table form.

- **Generate XML result:** An XML document is generated from the result table and sent to some application in the Web.
- **Materialize view:** This module generates an XQuery corresponding to the view definition and passes it to the underlying object-relational database to generate the result of the view. This allows users to materialize the defined view.

3.1 Mapping the underlying XML to ORA-SS

Now we will illustrate the work process of our system. First, XVS takes an XML document as its input. The document is transformed to a tree structure based on its hierarchy. Then it is mapped to an ORA-SS schema diagram. Elements in the document are mapped to object classes. Attributes of the elements are mapped to attributes of object classes. Users may input semantic information on the original XML tree to transform some object classes to attributes and/or add relationship types among object classes. Users may also indicate if an attribute belongs to an object class or a relationship type.

Example 1

Consider an XML document that describes project, supplier and part, which is depicted in Figure 4. The document is input into XVS and transformed into an ORA-SS schema, depicted in Figure 5. Then XVS will allow users to enrich the schema diagram with semantics. For example, relationship types among the object classes may be added in the diagram, such as *js* between project and supplier, *sp* between supplier and part, and *spj* among project, supplier and part. These relationship types are essential semantics for XML views design. Moreover, the element price in XML document is labeled as an attribute of the relationship type *sp*, as shown in figure 5. Then we have such a functional dependency: *supplier, part → price.*

```
<db>
  <project jno=”j001”>
    <supplier sno=“s001”>
      <part pno=“p001”>
        <price> 100</price>
      </part>
    </supplier>
    <supplier sno=“s002”>
      <part pno=“p001”>
        <price> 100</price>
      </part>
    </supplier>
  </project>
</db>
```

Figure 4: An XML document “spj.xml” on Project-Supplier-Part

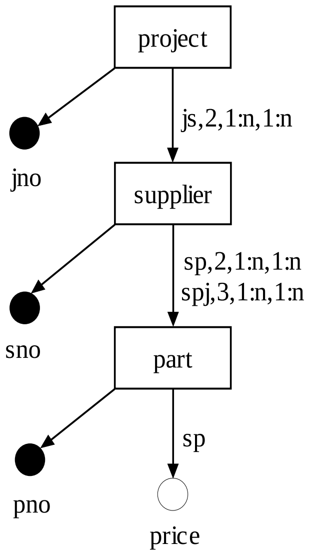

Figure 5: An ORA-SS schema diagram on Project-Supplier-Part

After that, XVS will store the enriched document into an object-relational database. In general, each object class in the ORA-SS schema is mapped to an object table. Similarly, each relationship set in ORA-SS is mapped to a relationship table.

Example 2
For example, XVS stores the document in Example 1 into a schema in Figure 6. The three object classes in the ORA-SS schema diagram are mapped to three object relations. Their respective key attributes in the diagram are still keys in the relations. The three relationship types in the ORA-SS schema diagram are also mapped to three relationship relations. The key of each relationship relation is a combination of key attributes of the participating object classes. As an attribute of relationship type, price is also stored in the relation supplier_part.

Object relation:
Project(jno); supplier(sno); part(pno);
Relationship relation:
Project_supplier(jno,sno); supplier_part(sno, pno, price); spj(jno, sno, pno)

Figure 6: Storage schema of ORA-SS schema diagram in Figure 4

3.2 Defining View

XVS provides a graphical interface for users to design views. Based on the enriched ORA-SS schema diagram, users can operate it directly to design a graphical view schema. That is, users can choose to select or drop some object classes, or restructure some parent-child relationship sets on the graphical schema diagram. The final view schema will be passed to the view definition generator. The view definition generator will first check whether the view schema is valid or not. That is, it will determine if the view conflicts with any semantic information provided in ORA-SS source schema. If the view is invalid, XVS will ask users to redesign the view. If the view is valid, XVS will generate a view definition expressed in XQuery expression.

Example 3
Consider the ORA-SS schema diagram in Figure 5. Suppose we design a view that swaps the location of object classes supplier and part. That is, supplier will become a child of part, while part becomes the parent of supplier. After swapping, we may design two different views, as shown in Figures 7 and 8.

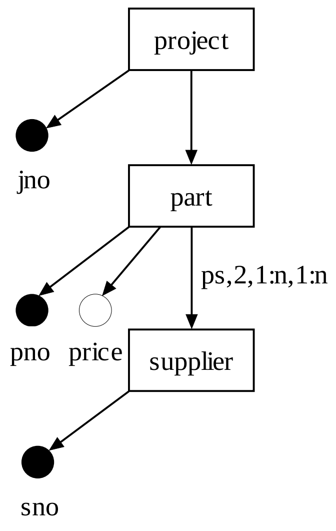

Figure 7: invalid view which swaps supplier and part

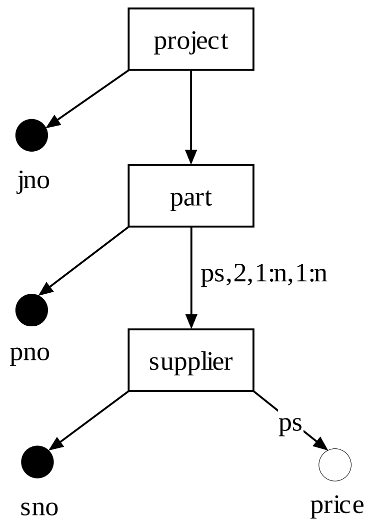

Figure 8: valid view which swaps supplier and part

In Figure 7, the attribute price is placed under the object class part. It therefore becomes an attribute of part. In this case, the functional dependency that is hold in the source schema: *supplier, part* $\rightarrow$ *price* has been changed to *part* $\rightarrow$ *price*. Then it violates the original semantics in the source schema and is called an invalid view. The system will inform users that the view is invalid and requires redesigning it.

Figure 8 shows a valid view in which the attribute price is placed under the new child object class – supplier. It conforms to the functional dependency: *supplier, part* $\rightarrow$ *price*, and is thus a meaningful view.

The example shows that ORA-SS enables us to avoid designing invalid XML views. It is because ORA-SS expresses necessary semantics for valid XML views design, as shown in the example above. Other models, such as OEM [AQM+97], XML DTD and XML Schema [XS] are not able to express those necessary semantics. Therefore, they cannot support valid XML views design. Existing tools, such as ActiveViews and MIX, which are based on an object model and XML DTD respectively, are also not able to support valid XML views design.

After validating the ORA-SS view, XVS will generate a view definition for it (Figure 9). The view definition first declares the name of the view as swap-supplier-part. Then it binds $j to each project object. For each project, it fetches every distinct part. For each part, it fetches every supplier that supplies it in the given project. Finally, it fetches the price of the part supplied by each supplier. This view definition can be rewritten in XQuery by removing the view declaration clause. By executing the XQuery, we obtain a materialized view document.

The process of the mapping between the graphical representation and the output is divided into two steps. In the first step, an intermediary view definition will first be generated based on the graphical representation of the view schema. The intermediary view definition adopts a novel declarative query language, which is in the SELECT-FROM-WHERE notation. Besides that, the intermediary language is specially designed for ORA-SS schema diagram. Therefore the case tool can easily generate a view definition using the language according to the view schema diagram. In the second step, the case tool will translate the intermediary view definition into a formal view definition in XQuery notation. The resulting view definition will be generated step by step according to what operation is contained in the intermediary view definition.

```
Create View As swap-supplier-part
For $j In document("spj.xml")//project
Return
        <project jno={$j/@jno}>
        {  for $pn In distinct($j//part/@pno)
           Return
              <part pno={$pn}>
              {  For $s In $j/supplier[part/@pno=$pn]
                 Return
                    <supplier sno={$s/@sno}>
                    { For $p In $s/part[@pno=$pn]
                      Return
                       <price> {$p/price/text()} </price>
                    }
                    </supplier>
              }
              </part>
        }
        </project>
```

Figure 9: View definition for the view in Figure 8.

3.3 Rewriting Query

After the view definition is generated, XVS allows users to issue queries over the view. The XQuery is re-written as an SQL query over the source database. The queries over views are limited within the form of FLWR expression, because XQuery may take other form of expression. A full description of XQuery can be found in [XQ]. Within the scope of FLWR expression, XQuery over views can be mapped into SQL query blocks in a straightforward way.

Example 4
For example, users may issue an XQuery as shown in Figure 10 on the view in Figure 8. This query will retrieve all suppliers that supply part “p001” in project “j001”. The corresponding SQL query is shown in Figure 11. The result of the query is a flat table, which will be transformed into the XML document shown in Figure 12.

```
For $s In
    view("swap-supplier-part")//project[@jno="j001"]/part[@pno="p001"]/supplier
Return
   <supplier sno={$s/@sno}>
        {$s/price}
   </supplier>
```

Figure 10: An XQuery against the view that retrieves all suppliers that supplies part "p001" in project "j001"

```
Select    supplier.sno supplier_part.price
From      supplier, supplier_part , spj
Where     spj.jno='j001' and spj.pno='p001'
And       supplier.sno=spj.sno and supplier_part.sno=s.sno
And       supplier_part.pno='p001'
```

Figure 11: SQL query that retrieves supplier supplying part p001 in project j001

```
<result>
  <supplier sno="s001">
     <price>100</price>
   </supplier>
   <supplier sno="s002">
     <price>120</price>
   </supplier>
   <supplier sno="s003">
     <price>110</price>
   </supplier>
</result>
```

Figure 12: Result of the XQuery in Figure 10.

4. Related Work

There have been several prototype systems developed to design XML views. The ActiveViews system [AAC+99] is built on top of ArdentSoftware's XML repository [AS], which is based on the O2 system – an object-oriented database system. In the system, an active view is presented as an object, which includes both data and methods [AB91, SDA94]. Since the underlying database of the system stores XML data as objects, the object id associated with each object increases the cost of data management and affects the performance of queries. Moreover, the ActiveViews system only supports the definition of simple views that do not require any restructuring, because the underlying data model cannot express necessary semantics for those more flexible XML views design.

The MIX (Mediation of Information using XML) System [BGL+99] provides a virtual view of underlying heterogeneous sources. The system utilizes XML DTD as its data model. Unfortunately, the semantics allowed in DTDs are limited. It is unable to indicate if an attribute belongs to an object or a relationship. Again, the views supported in MIX only involve selecting some elements based on certain conditions. Comparing with our system, the main problem of these two system is that they cannot guarantee valid XML views design, because they do not express enough semantics explicitly in their data models.

	ActiveViews system [AAC+99]	MIX system [BGL+99]	XVS (our system)
Data model	XML	XML DTD	ORA-SS
Storage	O2 system	N/A	ORDBMS
View definition language	OQL-style language	XMAS language	XQuery language
Query language	Lorel language	XMAS language	XQuery language
Validation of views	No	No	Yes
Support projection, join and restructuring of views	No	No	Yes
Support graphical views design	No	No	Yes

Table 1. Comparison of ActiveViews system, MIX system and XVS

Table 1 lists and compares the features of ActiveViews, MIX and our system – XVS. The ActiveViews system and MIX system use XML and XML DTD as their data model respectively, while XVS adopts a semantically rich data model – ORA-SS to express source schema and view schema. With a semantically richer model, XVS is able to support more flexible views. Moreover, XVS stores XML data into an ORDBMS by mapping ORA-SS schema into object-relational tables, while ActiveViews system stores the XML data into the O2 system – an object-oriented database. The latter requires that each object have an object id, which are used as reference pointers. MIX then uses heterogeneous data sources as its storage. It does not store XML data into those different databases, but only retrieve XML data from them.

The ActiveViews system uses the Object Query Language as a view definition language, and the Lorel language [AQM+97] as its query language over the views. This requires the users to be familiar with two different languages. MIX develops its own XMAS language as the view definition language and query language. In contrast, XVS directly adopts the W3C standard, XQuery as the view definition language and the query language over the views. A view definition is differentiated from a query by its additional view declaration clause before FLWR expression.

Both the ActiveViews system and MIX system do not provide for the validation of views. As a consequence, these two systems cannot support valid projection and join views, and views that involve restructuring.

5. Conclusion

In this paper, we have described our system for designing XML views – XVS. XVS adopts the core part of XQuery – FLWR expression as the view definition language and the query language over views, and adopts the semantically rich ORA-SS to express the necessary semantic information that is critical for designing valid XML views. XVS allows more flexible XML views design than other existing systems. The views that can be designed in our system may include projection, join and restructuring operations. Moreover, the designed views in XVS can be validated first. If they are validated successfully, then they can be used to generate the view definition. Otherwise, redesigning these views is required. Finally, XVS supports mapping and storing XML data in an object-relational database, which is essential to extend other functions in our system. Future work includes providing support to update XML views.

References

[A99]. S. Abiteboul. On Views and XML. In Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems, pages 1--9. ACM Press, 1999.
[AAC+99] S. Abiteboul, B. Amann, S. Cluet, A. Eyal, L. Mignet, and T. Milo. Active views for electronic commerce. *In Int. Conf. on Very Large DataBases (VLDB),* Edinburgh, Scotland, pages 138-149, September 1999.
[AB91]. S. Abiteboul and A. Bonner. Objects and Views. *In Proc. ACM SIGMOD Conference on Management of Data*, pages 238-247, 1991.
[AQM+97]. S. Abiteboul, D. Quass, J. McHugh, J.Widom, and J. L. Wiener. The lorel query language for semistructured data. *International Journal of Digital Libraries*, pages 68-88, volume 1:1, 1997.
[AS]. Ardent Software. http://www.ardentsoftware.com.
[BGL+99]. C. Baru, A. Gupta, B. Ludaescher, R. Marciano, Y. Papakonstantinou, and P. Velikhov. XML-Based Information Mediation with MIX. *In Demo Session, ACM-SIGMOD'99,* Philadelphia, PA, pages 597-599, 1999.
[LLD01]. Tok Wang Ling, Mong Li Lee, Gillian Dobbie. Application of ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. *In IIWAS,* 2001.
[SDA94]. C. Souza dos Santos, C. Delobel, and S. Abiteboul. Virtual Schemas and Bases. *In Proceedings of the International Conference on Extending Database Technology*, pages 81-94, March 1994.
[XML]. http://www.w3.org/XML
[XQ]. http://www.w3.org/TR/xquery
[XS]. http://www.w3.org/XML/Schema.