USING SQL/XML FOR EFFICIENTLY TRANSLATING QUERIES OVER XML VIEW OF RELATIONAL DATA

Fernando Lemos
PRiSM Lab, Université de Versailles, 45 avenue des Etats-Unis, Versailles, France

Clayton Costa, Vânia Vidal
Departament of Computing, Federal University of Ceará, S/N av. Eng. Humberto Monte, Fortaleza, Brazil

Keywords: XML, Web Services, Relational Database.

Abstract: XML and Web services are becoming the standard means of publishing and integrating data over the Web. A convenient way to provide universal access (over the Web) to different data sources is to implement Web services that encapsulate XML views of the underlying relational data (Data Access Web Services). Given that most business data is currently stored in relational database systems, the generation of Web services for publishing XML view of relational data has special significance. In this work, we propose *RelP*, a framework for publishing and querying relational databases through XML views. The main contribution of the paper is an algorithm that translates XML queries over a published XML view schema into a single SQL/XML query over the data source schema.

1 INTRODUCTION

A convenient way to provide universal access (over the Web) to different data sources is to implement *data access Web services*, i.e., Web services that publish XML views of data stored in a data source. Given that, a user can query the data source through XML views. Without an XML view, the XML application developer must be aware of the relational schema and must write and maintain all the queries to transform the SQL data into the desired XML structure. In this case, the use of XML views causes the developer to know only the exported XML schema, over which he can write less complex XML queries to retrieve the relevant date.

This strategy not only offers a flexible and transparent way to publish underlying data but also encapsulates details of access and transformation between the application and the data. Moreover, with the Semantic Web, the use of Web services and ontologies will make the data store content machine-processable and machine-interpretable.

The two main problems resulting from XML publishing are: *defining the XML view* (how to specify the mappings from source to target schemas) and *query answering* (how to use the mappings to answer correctly the queries posed on the XML view schema). Additionally, approaches for publishing relational data to XML must meet three requirements (Fernández, 2002): (i) be *general*, i.e., the mapping language must be flexible enough to allow specifying complex XML schemas over the relational schema; (ii) be *selective*, i.e., only the relevant data must be materialized, since, in general, it consists of a small part of the whole data; and (iii) be *efficient*, i.e., must guarantee an efficient query processing by exploiting the database optimizers and evaluation engines.

We have recently seen research aiming to provide high level mapping languages for data migration purposes (Haas, 2005; Popa, 2002; Melnik, 2005). Meanwhile, the problem of query answering is addressed in (Funderburk, 2002; Fernández, 2002; Jiang, 2007; Krishnamurthy, 2003, Benham, 2003). The query answering approach in these works must handle mappings which are hard to manipulate, leading to less-efficient query processing. Moreover, they don't guarantee efficient query processing and the lack of mapping formalism in some works leads to complex query-like mappings management.

In this paper, we present a framework, called ***RelP***, for publishing and querying relational databases through XML views. In ***RelP***, users can

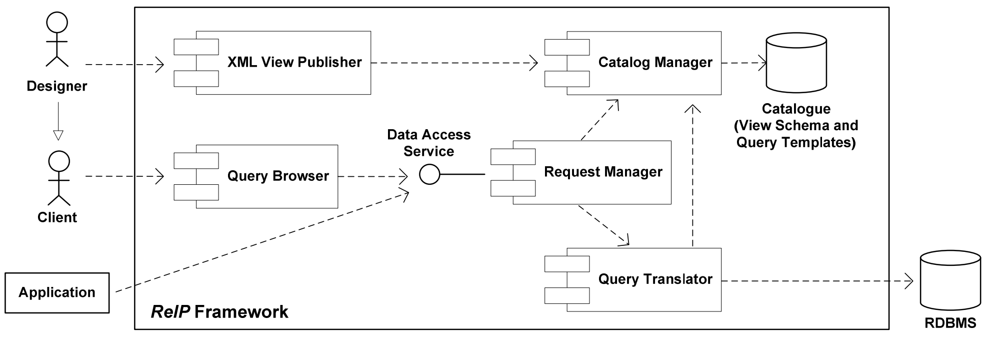


Figure 1: Components and interfaces required for ***RelP***.

define XQuery queries over an XML view schema. The query is translated into a single equivalent SQL/XML (Eisenberg, 2004) query defined over the relational schema. The choice for the SQL/XML standard relies in its flexibility and simplicity to build arbitrarily complex declarative queries that return XML data from relational data in the structure the user desires. These characteristics allow us to guarantee that a single SQL/XML query is obtained in the query answering process.

In our approach, a view is specified by a set of correspondence assertions (Popa, 2002; Vidal, 2006), which defines how to transform source states to view states. The views that we address are focused on schema-directed XML publishing (Bohannon, 2004). As such, the correspondence assertions induce schema mappings defined by the class of projection-selection-equijoin (PSE) SQL/XML queries, which support most types of data restructuring that are common in data exchange applications. We make a compromise in constraining the expressiveness of mappings so we can have a query answering algorithm that is much more efficient than those implemented by others XML publishing tools.

The main contribution of the paper is an algorithm that translates XML queries over a published XML view schema into a single SQL/XML query over the data source schema. The efficiency of our algorithm relies in two particulars: (i) the view mappings analysis is done at design time to generate a set of SQL/XML fragments (*Query Templates*), which are used in the query answering process; and (ii), the execution of SQL/XML queries takes advantage of traditional relational optimizers, which represents a considerable gain in performance (Liu, 2005). ***RelP*** also features a graphical interface to help in the creation of the XML view.

The rest of the paper proceeds as follows. Section 2 discusses the related works. Section 3 gives an overview of the main components of ***RelP***. Section 4 discusses the process for publishing an XML view with ***RelP***. Section 5 presents the query answering process in ***RelP***. Finally, Section 6 presents the conclusions and shows experimental results and corresponding analyses.

2 RELATED WORK

In the last years, several middlewares and frameworks have been proposed to treat the problem of XML publishing.

In XTables (Funderburk, 2002) and SilkRoute (Fernández, 2002), the XML view schema and mapping knowledge is specified by an XQuery query over a *canonical XML representation* of the relational schema. A query defined over the view schema is translated into equivalent SQL queries and the SQL results are tagged to produce the final XML document. Efficient query processing is not guaranteed, since for a single query more than one SQL queries can be generated. Moreover, the lack of mapping formalism leads to complex XQuery-based mappings generation and maintenance.

In Clio’s Three-Phase Framework (Jiang, 2007), a view is specified by a set of *tuple-generating dependencies* (tgds) (Haas, 2005). These mappings can be generated with the help of Clio tool (Haas, 2005). The framework (i), based on the view tgds, extracts the relevant data from the data source, (ii) generates XML fragments from the relevant data, and (iii) merges/groups the XML fragments to produce the final result. The framework is used in the transformation and transport of data between source and view schemas. Therefore, it is not possible to query the database through the view.

In XML Publisher (Vidal, 2004), the view is defined by its schema and a set of correspondence assertions between the view schema and the data source schema. Based on the view mappings, a *canonical object view* is created in the underlying DBMS. A query defined over the view schema is

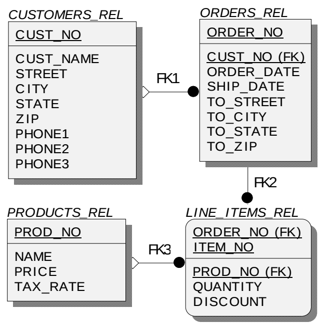


Figure 2: Relational Schema ORDERS_DB.

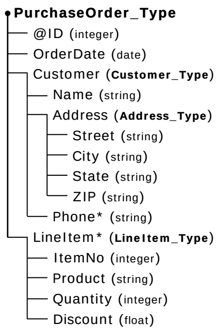


Figure 3: XML Type PurchaseOrder_Type.

first translated into a SQL:92 query over the canonical view schema, and then the DBMS view mechanism takes place to answer the query. The obligation to create an object view at the data source level makes the approach undesirably intrusive.

A significant amount of DBMSs, such as Oracle, DB2 and PostgreSQL, currently features publishing SQL data in XML format using the SQL/XML standard. Among them, only Oracle provides a SQL/XML view mechanism in which views can be queried using XQuery language.

In summary, on the one hand, we have middleware approaches that have complex mapping languages and do not guarantee efficiency in query processing. At the same time, we have some database systems that feature the SQL/XML standard, but a few number of them provides an SQL/XML view mechanism. Thus, the need for low-cost solutions that support the creation of XML views in a simple way to ensure easy access to data and efficient query processing motivates this work.

3 *RelP* ARCHITECTURE

A simplified view of the ***RelP*** architecture is shown in Figure 1. The *XML View Publisher* is an evolution of XVBA tool (Vidal, 2007). As XVBA, this component allows users to graphically define XML views, but, instead of generating the SQL/XML view definition, it generates the *query templates* of the view, which are used in the query translation. The view schema and its query templates are stored via the *Catalogue Manager* component.

The *Query Browser* component allows users to graphically define a query over the XML views. The *Query Translator* component translates an XML query posed in terms of an XML view schema into an SQL/XML query over the data source schema by using the query templates stored in the *Catalogue*.

The *Data Access Service* interface provides standard access for querying the XML views. It provides the following methods: (i) *getCapabilities*: requests the capabilities of the data access service (specifically, which XML views are serviced); (ii) *describeViewType*: requests the description of an XML view serviced by the data access service; (iii) *query*: requests the execution of an XQuery query over the schema of an XML view serviced by the data access service.

The *Request Manager* component simply redirects the requests to the appropriate components.

4 PUBLISHING AN XML VIEW

The publication of an XML view in ***RelP*** consists of two steps: (i) First, with the help of the *XML View Publisher*, the user defines the XML view schema and the correspondence assertions between the view schema and the relational schema, as explained in (Vidal, 2007). (ii) Then, the query templates are automatically generated based on the view assertions. The query templates of a view are composed of a set of SQL/XML fragments, each one responsible to generate an attribute/element of the view. The procedure that automatically generates the query templates can be found in (Lemos, 2010).

Consider the relational schema ORDERS_DB depicted in Figure 2. Suppose the XML view PurchaseOrder_XML, which is a set of *<PurchaseOrder>* elements, instances of type PurchaseOrder_Type (Figure 3), generated from the tuples of the relational scheme *ORDERS_REL*. The correspondence assertions of PurchaseOrder_XML can be found in (Vidal, 2006).

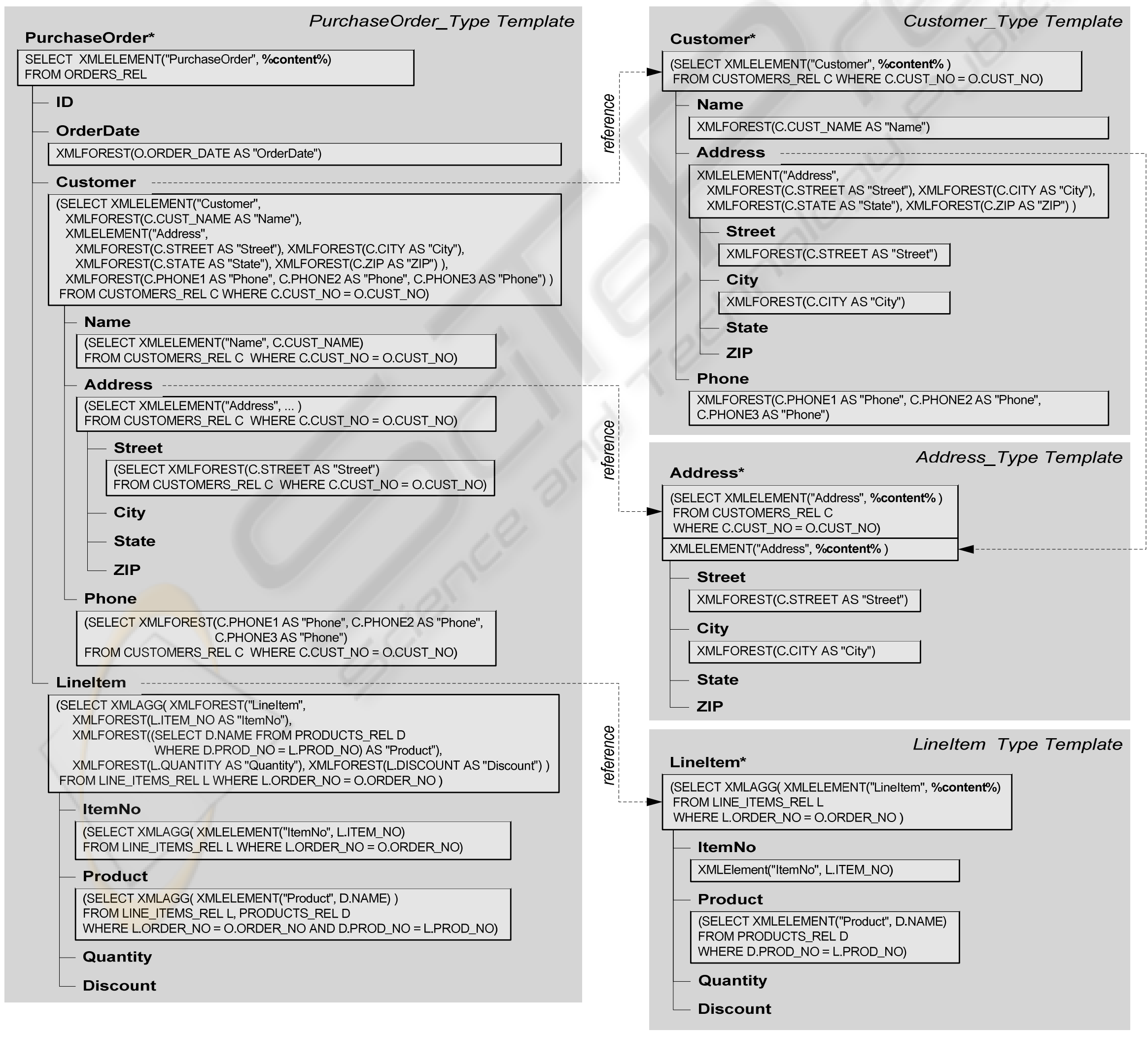
The query templates generated from the correspondence assertions of the view are shown in Figure 5. In the PurchaseOrder_Type template, for example, each attribute/element of the complex type PurchaseOrder_Type have one corresponding

```
INPUT: output node $\$node_{TP}$ of a tree pattern query; node $\$node_{QT}$ of a query template QT
OUTPUT: SQL/XML subquery Q

CASE 1: If $\$node_{TP}$ is not labelled with * THEN
Q := $\$node_{QT}$.query + GenFilter( $f$ ), where $f$ is the set of conditional expressions of $\$node_{TP}$;
CASE 2: If $\$node_{TP}$ is labelled with * THEN
LET $\$node'_{QT}$ := $\$node_{QT}$.reference;
Q := $\$node'_{QT}$.query + GenFilter( $f$ ), where $f$ is the set of conditional expressions of $\$node_{TP}$;
Qc := "";
FOR EACH output child $\$child_{TP}$ of $\$node_{TP}$, where $\delta$ is the path from $\$node_{TP}$ to $\$child_{TP}$ DO
LET $\delta_{QT}$ be the equivalent path of $\delta$ in QT;
Qc := Qc + $\$node'_{QT}.\delta_{QT}$.query;
END FOR
REPLACE %content% in Q by Qc;
RETURN Q;

Note: GenFilter( $f$ ) translates the conditional expressions in $f$ to the correspondent SQL filter clauses.
```

Figure 4: Procedure GenSubquery.



```
SELECT  XMLELEMENT("PurchaseOrder", %content%)
FROM ORDERS_REL
```

```
XMLFOREST(O.ORDER_DATE AS "OrderDate")
```

```
(SELECT XMLELEMENT("Customer",
  XMLFOREST(C.CUST_NAME AS "Name"),
  XMLELEMENT("Address",
    XMLFOREST(C.STREET AS "Street"), XMLFOREST(C.CITY AS "City"),
    XMLFOREST(C.STATE AS "State"), XMLFOREST(C.ZIP AS "ZIP") ),
  XMLFOREST(C.PHONE1 AS "Phone", C.PHONE2 AS "Phone", C.PHONE3 AS "Phone") )
 FROM CUSTOMERS_REL C WHERE C.CUST_NO = O.CUST_NO)
```

```
(SELECT XMLELEMENT("Name", C.CUST_NAME)
FROM CUSTOMERS_REL C  WHERE C.CUST_NO = O.CUST_NO)
```

```
(SELECT XMLELEMENT("Address", ... )
FROM CUSTOMERS_REL C  WHERE C.CUST_NO = O.CUST_NO)
```

```
(SELECT XMLFOREST(C.STREET AS "Street")
FROM CUSTOMERS_REL C  WHERE C.CUST_NO = O.CUST_NO)
```

```
(SELECT XMLFOREST(C.PHONE1 AS "Phone", C.PHONE2 AS "Phone",
                        C.PHONE3 AS "Phone")
FROM CUSTOMERS_REL C  WHERE C.CUST_NO = O.CUST_NO)
```

```
(SELECT XMLAGG( XMLFOREST("LineItem",
    XMLFOREST(L.ITEM_NO AS "ItemNo"),
    XMLFOREST((SELECT D.NAME FROM PRODUCTS_REL D
                    WHERE D.PROD_NO = L.PROD_NO) AS "Product"),
    XMLFOREST(L.QUANTITY AS "Quantity"), XMLFOREST(L.DISCOUNT AS "Discount") )
 FROM LINE_ITEMS_REL L WHERE L.ORDER_NO = O.ORDER_NO )
```

```
(SELECT XMLAGG( XMLELEMENT("ItemNo", L.ITEM_NO)
FROM LINE_ITEMS_REL L WHERE L.ORDER_NO = O.ORDER_NO)
```

```
(SELECT XMLAGG( XMLELEMENT("Product", D.NAME) )
FROM LINE_ITEMS_REL L, PRODUCTS_REL D
WHERE L.ORDER_NO = O.ORDER_NO AND D.PROD_NO = L.PROD_NO)
```

```
(SELECT XMLELEMENT("Customer", %content% )
 FROM CUSTOMERS_REL C WHERE C.CUST_NO = O.CUST_NO)
```

```
XMLFOREST(C.CUST_NAME AS "Name")
```

```
XMLELEMENT("Address",
  XMLFOREST(C.STREET AS "Street"), XMLFOREST(C.CITY AS "City"),
  XMLFOREST(C.STATE AS "State"), XMLFOREST(C.ZIP AS "ZIP") )
```

```
XMLFOREST(C.STREET AS "Street")
```

```
XMLFOREST(C.CITY AS "City")
```

```
XMLFOREST(C.PHONE1 AS "Phone", C.PHONE2 AS "Phone",
C.PHONE3 AS "Phone")
```

```
(SELECT XMLELEMENT("Address", %content% )
 FROM CUSTOMERS_REL C
 WHERE C.CUST_NO = O.CUST_NO)
```

```
XMLELEMENT("Address", %content% )
```

```
XMLFOREST(C.STREET AS "Street")
```

```
XMLFOREST(C.CITY AS "City")
```

```
(SELECT XMLAGG( XMLELEMENT("LineItem", %content%)
 FROM LINE_ITEMS_REL L
 WHERE L.ORDER_NO = O.ORDER_NO )
```

```
XMLElement("ItemNo", L.ITEM_NO)
```

```
(SELECT XMLELEMENT("Product", D.NAME)
FROM PRODUCTS_REL D
WHERE D.PROD_NO = L.PROD_NO)
```

Figure 5: Query Templates for PurchaseOrder_XML.

```
SELECT XMLELEMENT("PurchaseOrder",                                        from PurchaseOrder_Type Template, PurchaseOrder* node
    XMLFOREST(O.ORDER_DATE AS "OrderDate"),                               from PurchaseOrder_Type Template,  OrderDate node
    (SELECT XMLELEMENT("Customer",                                        from PurchaseOrder_Type Template, Customer node
        XMLFOREST(C.CUST_NAME AS "Name"),
        XMLELEMENT("Address",
            XMLFOREST(C.STREET AS "Street"), XMLFOREST(C.CITY AS "City"),
            XMLFOREST(C.STATE AS "State"), XMLFOREST(C.ZIP AS "ZIP") ),
        XMLFOREST(C.PHONE1 AS "Phone", C.PHONE2 AS "Phone", C.PHONE3 AS "Phone") )
     FROM CUSTOMERS_REL C WHERE C.CUST_NO = O.CUST_NO),
    (SELECT XMLAGG( XMLELEMENT("LineItem",                                from PurchaseOrder_Type Template, LineItem* node
        (SELECT XMLFOREST(D.NAME AS "Product")                            from LineItem_Type Template, Product node
         FROM PRODUCTS_REL D WHERE D.PROD_NO = L.PROD_NO),
        XMLFOREST(L.QUANTITY AS "Quantity") ) )                           from LineItem_Type Template, Quantity node
     FROM LINE_ITEMS_REL L  WHERE L.ORDER_NO = O.ORDER_NO AND L.QUANTITY < 20) )
FROM ORDERS_REL O, CUSTOMERS_REL C WHERE O.CUST_NO = C.CUST_NO AND C.CITY = 'Baltimore';
```

Figure 8: Translated SQL/XML query.

```
for $v1 in
view(PurchaseOrder_XML)/PurchaseOrder
where $v1/Customer/Address/City = "Baltimore"
return <PurchaseOrder>{
            $v1/OrderDate, $v1/Customer,
            for $v2 in $v1/LineItem
            where $v2/Quantity < 20
            return <LineItem>{
                        $v2/Product, $v2/Quantity
                  }</Line Item>
      }</PurchaseOrder>
```

Figure 6: XQuery Q_{X1}.

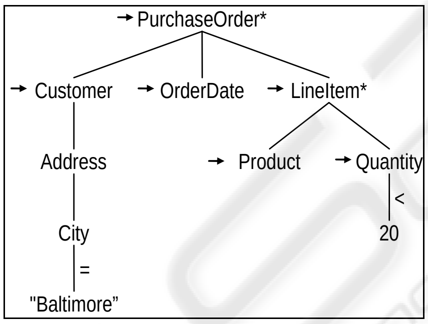


Figure 7: Tree pattern query T_{X1}.

SQL/XML subquery that specifies how it is computed from the data source. The remaining templates contain the fragments of other complex types used in the view type definition.

5 ANSWERING XML QUERIES IN *RELP*

The user can query an XML view through the *Data Access Service* interface using XQuery language. The user can graphically formulate the query using the *Query Browser* component.

The *Query Translator* component translates an XML Query Q_X posed in terms of an XML view schema into a SQL/XML query Q_S over the data source schema. Q_S is a correct translation for Q_X, which means that the result of executing Q_X over the XML view state is equivalent to the result of executing Q_S over the database state, in a given time (Lemos, 2010).

The *Query Translator* component accepts any XQuery queries that can be expressed as a tree pattern query (TPQ). In our approach, a tree pattern query is a labelled tree whose nodes are labelled by element/attribute names or data values. Nodes pointed by arrows are called *output nodes*, and indicates that the correspondent attribute/element is part of the query result. Element nodes having its content restructured are also labelled by the symbol *. Filters are selection expressions modelled labelling the edge with a Boolean operator. Filters are applied to the closest unbounded ancestor output node.

For instance, consider the XQuery Q_{X1} presented in Figure 6. The query asks for the orders of customers in Baltimore, and for each order, it asks for the items having quantity greater than 20. The correspondent TPQ T_{X1} is shown in Figure 7. In TPQ T_{X1}, the filter "City = Baltimore" is applied to PurchaseOrder node and "Quantity > 20" is applied to LineItem node.

The Query Translator algorithm translates an XQuery query into a single correspondent SQL/XML query over the source relations. The translation is done by matching the TPQ against the query templates of the view. The SQL/XML fragments of the matched nodes are composed to generate the final query.

The procedure GenSubquery shown in Figure 4 generates the SQL/XML subquery that computes the XML element represented by an output node. For example, consider the XQuery Q_{X1} of Figure 6. The steps of the query translation are presented below:

(1) Generation of the TPQ T_{X1} (see Figure 7).
(2) Matching of the output node PurchaseOrder* of T_{X1} with the node PurchaseOrder* of the PurchaseOrder_Type template. The conditional expression Customer/Address/City = "Baltimore" is translated into a correspondent SQL clause, which is added to the WHERE clause of the SQL/XML fragment of PurchaseOrder* node.
(3) Matching of the output nodes Customer and OrderDate with the nodes Customer and OrderDate of PurchaseOrder_Type template, respectively.
(4) Matching of the output node LineItem* with the node LineItem* of LinteItem_Type template. The conditional expression Quantity > "20" is translated into the correspondent SQL clause and added to the WHERE clause of the fragment of LineItem* node.
(5) The placeholder %content% in the SQL/XML fragment of PurchaseOrder* node is replaced by the SQL/XML fragments of Customer, OrderDate and LineItem*.
(6) Finally, the nodes Product and Quantity of T_{X1} are matched with the nodes Product and Quantity of LineItem_Type template, respectively. The placeholder %content% in the SQL/XML fragment of LineItem* node is replaced by the SQL/XML fragments of Product and Quantity.

Figure 8 shows, in details, the SQL/XML translation for the XQuery Q_{X1} shown in Figure 6. Figure 8 indicates the template and path used to generate each subquery.

6 PERFORMANCE ANALYSIS AND CONCLUSIONS

In this paper, we presented ***RelP***, a framework for publishing and querying relational databases through XML views. We first showed how to specify a view with the help of correspondence assertion. Next, we presented an algorithm that translates XML queries over a published XML view schema into a single SQL/XML query over the data source schema.

We evaluate the performance of our algorithm with respect to the query complexity and time response. We compared our implementation with the approach of using Oracle's built-in XML view mechanism. The results in Figure 10 show that our approach can be significantly faster, if not, we have similar performance (Lemos, 2010).

The main reason of this performance benefit comes from our query templates, which are generated at view creation time. It means that the query translator component doesn't have to deal with the complexity of the view mappings at execution time.

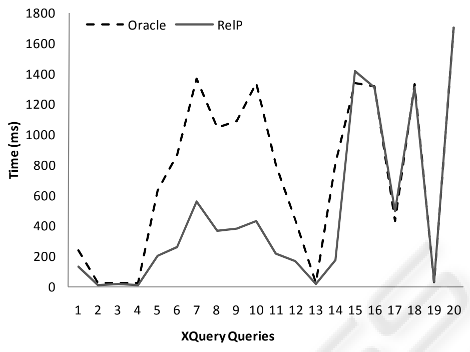


Figure 9: Response time vs. query.

REFERENCES

Benham, S E 2003, 'XML-Enabled Data Management Product Architecture and Technology', *XML Data Management, Native XML and XML-Enable Database Systems*, Addison Wesley.
Bohannon, P et al 2004, 'Incremental evaluation of schema-directed XML publishing', *SIGMOD'04*, pp. 503-514.
Eisenberg, A et al 2004, 'SQL:2003 has been published' *SIGMOD'04*, pp. 119-126.
Fernández, M et al 2002, 'SilkRoute: A framework for Publishing relational data in XML', *TODS'02*, pp.438-493.
Funderburk, J E et al 2002, 'XTABLES: Bridging relational technology and XML', *IBM Systems Journal*, vol.41, n. 4.
Haas, L M et al 2005, 'Clio Grows Up: From Research Prototype to Industrial Tool', *SIGMOD'05*, pp. 805-810.
Jiang, H et al 2007, 'Mapping-driven XML Transformation', *WWW'07*, pp. 1063-1072.
Krishnamurthy, R et al 2003, 'XML-to-SQL Query Translation Literature: The State of the Art and Open Problems', *XSym'03*, pp. 1-18.
Lemos, F C et al 2010, 'Using SQL/XML for Efficiently Translating Queries over XML View of Relational Data', *Technical Report*, http://lia.ufc.br/~arida
Liu, Z H et al 2005, 'Native XQuery Processing in Oracle XMLDB', *SIGMOD International Conference on Management of Data*, pp. 828-833.
Melnik, S et al 2005, 'Supporting Executable Mappings in Model Management', *SIGMOD'05*, pp. 167–178.
Popa, L et al 2002, 'Translating Web Data', *VLDB'02*, pp. 598–609.
Vidal, V M P & Lemos, FC 2007, 'XVBA: A Tool for Semi-Automatic Generation of SQL/XML Views', *IV Demo Session*, SBBD'07, pp. 57-62.
Vidal, V M P et al 2006, 'Automatic Generation of SQL/XML Views', *SBBD'06*, pp. 221–235.
Vidal, V M P et al 2004, 'XML Publisher: Um Framework para Publicação de Dados Armazenados em Banco de Dados Relacional ou Objeto Relacional como XML', *1st Demo Session, SBBD'04*, pp. 07-12.