Automatic Generation of SQL/XML Views

Vânia Maria Ponte Vidal[1], Marco Antonio Casanova[2], Fernando Cordeiro Lemos[1]

[1]Department of Computing – UFC
Fortaleza – CE – Brazil

[2]Department of Informatics – PUC-Rio
Rio de Janeiro – RJ – Brazil

vvidal@lia.ufc.br, casanova@inf.puc-rio.br, fernandocl@lia.ufc.br

Abstract. *This paper proposes an approach to generate XML views of relational data, using SQL/XML. The paper first specify the conditions for a set of correspondence assertions to fully specify the view in terms of the relational schema and, if so, we show that the mappings defined by the view correspondence assertions can be expressed as SQL/XML view definition. This paper focuses on an algorithm that automatically generates the SQL/XML query from the view correspondence assertions.*

1. Introduction

XML has emerged as the standard information exchange format for Internet-based business applications. However, since most business data is currently stored in relational database systems, the problem of publishing relational data in XML format has special significance. A general and flexible way to publish relational data in XML format is to create XML views of the underlying relational data.

The exported view may be either virtual or materialized. Materialized views improve query performance and data availability, but they must be updated to reflect changes to the base source [12]. In the case of virtual views, the data still persists in relational databases, while applications may access the data in XML format through the XML view [1]. Exporting virtual XML views of relational data raises the problems of defining the XML view and evaluating an XML query posed over the view. The XML query is translated into SQL by composing it with the view definition.

The publication of relational data through a virtual XML view has been addressed, for example, in XPeranto [3] and SilkRoute [6]. In both works, the XML view is defined as an XQuery over the canonical XML view that represents the database tables and their attributes. This query specifies the view schema and the mapping knowledge, describing how the schema is related to the canonical view. The evaluation of an XML query over the view is performed using a middleware on top of relational database. The middleware translates the XML query into equivalent SQL queries. Then, the SQL results are tagged to produce the resulting XML document. In these systems, efficient query processing is not guarantee.

In the *DB2 XMLExtender* [2] and in *SQL Server* [11], the mapping knowledge is stored within *annotated schemas* [11]. In both cases, the mapping definition is very complex. Moreover, SQL Server provides the FOR XML clause to provide modes to transform query results into XML. The mapping knowledge is defined at access time and not stored in any way, which violates the mapping transparency.

With the introduction of the XML datatype [1] and the SQL/XML standard [4] as part of SQL:2003 [4], users may resort to the SQL/XML publishing functions to create virtual XML views over base relational schemas. Oracle [1] was the first DBMS to support, with its XML DB module [1], the creation of XML Views as SQL/XML queries over the relational data. The advantages of this approach rely on the use of a standard to publish relational data and on the capacity to process the SQL/XML publish functions within the SQL statements, which represents a gain in performance [9]. Thus, XML Query rewrite can be performed inside the DBMS [8], as opposed to non-integrate mid-tier solution.

However, creating SQL/XML view definitions demands advanced knowledge of SQL/XML and is time consuming. Moreover, users will have to redefine the XML view whenever the base relational schema changes. Therefore, tools that facilitate the task of XML view creation and the maintenance should be developed.

We propose in this paper an approach where the SQL/XML view definition is derived from view correspondence assertions, which specify relationships between the view schema and the relational schema. In the case of materialized views, as we shown in [12], all rules required to maintain the view can be automatic generated based on the view correspondence assertions.

This paper has three major contributions. First, we propose the use of correspondence assertions [10][12] for specifying the mapping between an XML view schema and a base relational schema. We formally specify the conditions under which a set of correspondence assertions fully specifies the XML view in terms of the relational source and, if so, we show that the mappings defined by the view correspondence assertions can be expressed as an SQL/XML query view definition. Second, we propose an algorithm that, based on the view correspondence assertions, generates the SQL/XML query that constructs the XML view elements from the relational tuples. Third, we propose the XMLView-By-Assertions (XVBA) tool that facilitates the task of XML view creation and maintenance. We note that the mapping formalisms used by other schema mapping tools are either ambiguous [7] or require the user to declare complex logical mapping [14].

This article is organized as follows. Section 2 discusses XML Views and the SQL/XML standard. Section 3 presents our mapping formalism. Section 4 discusses how to specify XML view using correspondence assertions. Section 5 presents the algorithm that automatically generates the SQL/XML view definition from the correspondence assertions. Finally, Section 6 presents the conclusions.

2. XML Views

With the introduction of the XML datatype and the SQL/XML standard, users may create a view of XML type instances over relational tables using SQL/XML publishing functions [1], such as XMLElement(), XMLConcat(), etc.

Consider, for example, the relational schema ORDERS_DB and the XML type PurchaseOrder_Type, whose graphical representations are shown in Figure 1 and 2 respectively. To generate instances of PurchaseOrder_Type from ORDERS_DB, we create the SQL/XML view PurchaseOrder_XML shown in Figure 3. As illustrated in Figures 4 and 5, for each tuple in table ORDERS_REL, the XML view uses the SQL/XML standard publishing functions to construct an instance of the XMLType PurchaseOrder_Type.

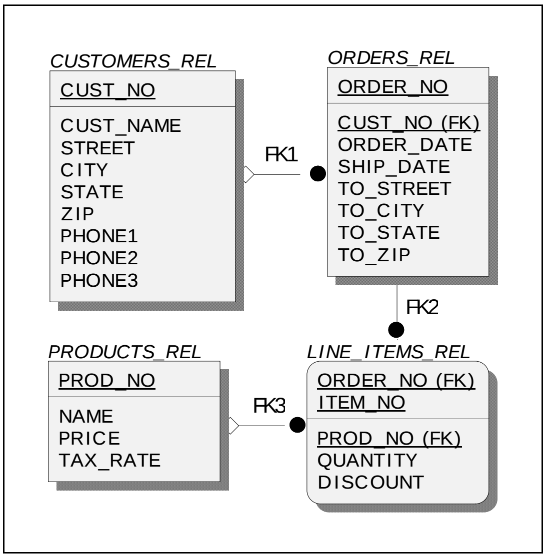

Figure 1 – Relational Schema
ORDERS_DB

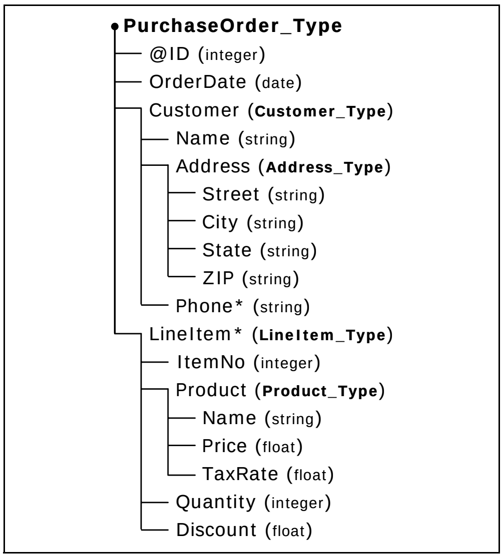

Figure 2 – XML Type
PurchaseOrder_Type

```
CREATE OR REPLACE VIEW PurchaseOrder_XML OF XMLTYPE
XMLSCHEMA "PurchaseOrder.xsd" ELEMENT "PurchaseOrder"
AS SELECT XMLELEMENT("PurchaseOrder",
  XMLATTRIBUTES(O.ORDER_NO AS "ID"), ...................from Ψ1     τ[ORDERS_REL→ PurchaseOrder_Type]
  XMLFOREST(O.ORDER_DATE AS "OrderDate"),..........from Ψ2
  (SELECT XMLELEMENT("Customer", ..........................from Ψ3
    XMLFOREST(C.CUST_NAME AS "Name"), ...........from Ψ4     τ[CUSTOMERS_REL→ Customer_Type]
    XMLELEMENT("Address", .......................................from Ψ5
      XMLFOREST(C.STREET AS "Street"), ...............from Ψ6     τ[CUSTOMERS_REL→ Address_Type]
      XMLFOREST(C.CITY AS "City"), ........................from Ψ7
      XMLFOREST(C.STATE AS "State"), ...................from Ψ8
      XMLFOREST(C.ZIP AS "ZIP") ), .........................from Ψ9
    XMLFOREST(C.PHONE1 AS "Phone"),   .............from Ψ10
    XMLFOREST(C.PHONE2 AS "Phone"),
    XMLFOREST(C.PHONE3 AS "Phone") )
   FROM CUSTOMERS_REL C
   WHERE C.CUST_NO = O.CUST_NO),
  (SELECT XMLAGG( XMLELEMENT("LineItem", ............from Ψ11
    XMLFOREST(L.ITEM_NO AS "ItemNo"), ................from Ψ12     τ[LINE_ITEMS_REL→ LineItem_Type]
    (SELECT XMLELEMENT("Product", .......................from Ψ13
      XMLFOREST(D.NAME AS "Name"), ...................from Ψ14     τ[PRODUCTS_REL→ Product_Type]
      XMLFOREST(D.PRICE AS "Price"), ...................from Ψ15
      XMLFOREST(D.TAX_RATE AS "TaxRate") ) .....from Ψ16
    FROM PRODUCTS_REL D
    WHERE D.PROD_NO = L.PROD_NO),
    XMLFOREST(L.QUANTITY AS "Quantity"), ............from Ψ17
    XMLFOREST(L.DISCOUNT AS "Discount") ) ) .......from Ψ18
   FROM LINE_ITEMS_REL L
   WHERE L.ORDER_NO = O.ORDER_NO ) )
FROM ORDERS_REL O;
```

Figure 3 – PurchaseOrder_XML View

CUSTOMERS_REL

CUST_NO	CUST_NAME	STREET	CITY	STATE	ZIP	PHONE1	PHONE2	PHONE3
193	Bryan Huston	8 Automation Ln	Albany	NY	12205	+91 11 012 4813	+91 11 083 4813	+91 33 012 4827
195	Cary Stockwell	400 E Joppa Rd	Baltimore	MD	21286	+91 11 012 4835	NULL	NULL

PRODUCTS_REL

PROD_NO	NAME	PRICE	TAX_RATE
2638	HD 10GB 5400	125.50	0.01
1721	PC Bag - L/S	256.28	0.005
1761	Mouse +WP/CL	32.89	0.0

LINE_ITEMS_REL

ORDER_NO	ITEM_NO	PROD_NO	QUANTITY	DISCOUNT
405	1	2638	35	0.07
407	1	1721	15	0.05
408	1	1721	30	0.05
407	2	1761	60	0.10

ORDERS_REL

ORDER_NO	CUST_NO	ORDER_DATE	SHIP_DATE	TO_STREET	TO_CITY	TO_STATE	TO_ZIP
405	193	01/07/05	05/07/05	8 Automation Ln	Albany	NY	12205
407	195	29/06/05	01/07/05	400 E Joppa Rd	Baltimore	MD	21286
408	195	24/05/04	25/05/04	23985 Bedford Rd N	Battle Creek	MI	49017

Figure 4 – An instance of ORDERS_DB

```
<PurchaseOrder ID="405">
  <OrderDate>2005-07-01</OrderDate>
  <Customer>
    <Name>Bryan Huston</Name>
    <Address>
      <Street>8 Automation Ln</Street>
      <City>Albany</City>
      <State>NY</State>
      <ZIP>12205</ZIP>
    </Address>
    <Phone>+91 11 012 4813</Phone>
    <Phone>+91 11 083 4813</Phone>
    <Phone>+91 33 012 4827</Phone>
  </Customer>
  <LineItem>
    <ItemNo>1</ItemNo>
    <Product>
      <Name>HD 10GB 5400</Name>
      <Price>125,5</Price>
      <TaxRate>0.01</TaxRate>
    </Product>
    <Quantity>35</Quantity>
    <Discount>0.01</Discount>
  </LineItem>
</PurchaseOrder>

<PurchaseOrder ID="407">
  <OrderDate>2005-06-29</OrderDate>
  <Customer>
    <Name>Cary Stockwell</Name>
    <Address>
      <Street>400 E Joppa Rd</Street>
      <City>Baltimore</City>
      <State>MD</State>
      <ZIP>21286</ZIP>
    </Address>
    <Phone>+91 11 012 4835</Phone>
  </Customer>
  <LineItem>
    <ItemNo>1</ItemNo>
    <Product>
      <Name>PC Bag - L/S</Name>
      <Price>256,28</Price>
      <TaxRate>0.05</TaxRate>
    </Product>
    <Quantity>15</Quantity>
    <Discount>0.05</Discount>
  </LineItem>
  <LineItem>
    <ItemNo>2</ItemNo>
    <Product>
      <Name>Mouse +WP/CL</Name>
      <Price>32,89</Price>
      <TaxRate>0</TaxRate>
    </Product>
    <Quantity>60</Quantity>
    <Discount>0.1</Discount>
  </LineItem>
</PurchaseOrder>

<PurchaseOrder ID="408">
  <OrderDate>2004-05-24</OrderDate>
  <Customer>
    <Name>Cary Stockwell</Name>
    <Address>
      <Street>400 E Joppa Rd</Street>
      <City>Baltimore</City>
      <State>MD</State>
      <ZIP>21286</ZIP>
    </Address>
    <Phone>+91 11 012 4835</Phone>
    <Phone>+91 11 083 4835</Phone>
  </Customer>
  <LineItem>
    <ItemNo>1</ItemNo>
    <Product>
      <Name>PC Bag - L/S</Name>
      <Price>256,28</Price>
      <TaxRate>0.005</TaxRate>
    </Product>
    <Quantity>30</Quantity>
    <Discount>0.05</Discount>
  </LineItem>
</PurchaseOrder>
```

Figure 5 – An instance of PurchaseOrder_XML **view**

In more detail, consider the instance (or database state) of ORDERS_DB shown in Figure 4. The corresponding instance of PurchaseOrder_XML is shown in Figure 5. This view instance contains a sequence of <PurchaseOrder> elements of type PurchaseOrder_Type, which are the primary elements of the view. Each <PurchaseOrder> element is constructed from a tuple of the ORDERS_REL table by using the SQL/XML publishing function XMLElement(). Function XMLElement() takes as arguments an element name, an optional collection of attributes, and zero or more additional arguments that make up the element content.

The sub-elements and attributes of a <PurchaseOrder> element are constructed by using SQL/XML sub-queries. For example,

- attribute ID is constructed using the subquery in line 4. Function XMLAttributes() produces, from its arguments, the attributes of its owner XMLElement() function. These arguments are value expressions to be evaluated, with optional aliases. The datatype of an attribute value expression cannot be an object type or a collection. If an attribute value expression evaluates to NULL, then no corresponding attribute is created.
- sub-element <Date> is constructed using the subquery in line 5. Function XMLForest() produces a forest of XML elements from its arguments, which are expressions to be evaluated, with optional aliases. If an expression evaluates to NULL, then no corresponding element is created.
- sub-element <LineItem> is constructed by the subquery in lines 18 to 29. Function XMLAgg() is an aggregate function that produces a forest of XML elements from a collection of XML elements where NULL arguments are dropped from the result. In lines 18 to 29, we have that, for each tuple in ORDERS_REL table, the relevant tuples of the LINE_ITEMS_REL table are retrieved and converted into a sequence of <LineItem> elements.

3. Basic Definitions

In this section, let $R, R_1,...,R_n$ be relation schemes of a relational schema S. Let R, $R_1,...,R_n$ be relations over $R, R_1,...,R_n$, respectively.

Definition 1: Let fk be a foreign key of R_1 that references R_2. Then, we say that:

i) fk is a *link* from R_1 to R_2.

ii) fk^{-1}, the *inverse* of a fk, is a *link* from R_2 to R_1. □

Definition 2:

i) Let ℓ be a link from R_1 to R_2 of the form $R_1[a_1,...,a_m] \subseteq R_2[b_1,...,b_m]$. Let r_1 be a tuple of R_1. Then, $r_1/\ell = \{ r_2 \in R_2 \mid r_1.a_i = r_2.b_i$, for $1 \le i \le m \}$.

ii) Let ℓ be a link from R_2 to R_1 of the form $R_1[a_1,...,a_m] \subseteq R_2[b_1,...,b_m]$. Let r_2 be tuple of R_2. Then, $r_2/\ell = \{ r_1 \in R_1 \mid r_1.a_i = r_2.b_i$, for $1 \le i \le m \}$. □

Definition 3: Let ℓ be a link from R_1 to R_2, and r_1 and r_2 be tuples of R_1 and R_2, respectively. Then, we say that:

i) r_1 *references* r_2 *through* ℓ iff $r_2 \in r_1/\ell$.

ii) ℓ *has single occurrence* iff a tuple of R_1 can reference at most one tuple of R_2 through ℓ; otherwise, ℓ *has multiple occurrence*. □

Definition 4: Let $\ell_1,\ldots,\ell_n$ be links. Assume that:

i) ℓ_1 is a foreign key of R of the form $R[a_1^{\ell 1},\ldots,a_{m1}^{\ell 1}] \subseteq R_1[b_1^{\ell 1},\ldots,b_{m1}^{\ell 1}]$ or the inverse of a foreign key of R_1 of the form $R_1[b_1^{\ell 1},\ldots, b_{m1}^{\ell 1}] \subseteq R[a_1^{\ell 1},\ldots,a_{m1}^{\ell 1}]$

ii) ℓ_i is a foreign key of R_{i-1} of the form $R_{i-1}[a_1^{\ell i},\ldots, a_{mi}^{\ell i}] \subseteq R_i[b_1^{\ell i},\ldots,b_{mi}^{\ell i}]$ or the inverse of a foreign key of R_i of the form $R_i[b_1^{\ell i},\ldots, b_{mi}^{\ell i}] \subseteq R_{i-1}[a_1^{\ell i}, \ldots, a_{mi}^{\ell i}]$, for $2 \le i \le n$.

Then, we say that:

i) $\varphi = \ell_1. \ldots .\ell_n$ is a *referential path* from R to R_n.

ii) the tuples of R *reference tuples of* R_n *through* φ.

iii) φ *has single occurrence* iff ℓ_i *has single occurrence*, for $1 \le i \le n-1$; otherwise, φ *has multiple occurrence*. □

Definition 5: Let $\varphi = \ell_1. \ldots .\ell_n$ be a *referential path* from R to R_n. Let r be a tuple of R. Then,

$$r / \varphi = \{ r_n \in R_n \mid (\exists r_1 \in R_1)\ldots (\exists r_{n-1} \in R_{n-1}) (r.a_k^{\ell 1} = r_1.b_k^{\ell 1}, \text{ for } 1 \le k \le m_1) \text{ and } (r_{i-1}.a_k^{\ell i} = r_i.b_k^{\ell i}, \text{ for } 1 \le k \le m_i \text{ and } 2 \le i \le n) \}.$$ □

Definition 6: A *path* of R is an expression of one of following forms:

i) NULL

ii) a, where a is an attribute of R.

iii) $\{a_1,\ldots,a_n\}$, where $a_1,\ldots,a_n$ are attributes of R.

iv) φ.a, where φ is a referential path from R to R' and a is an attribute of R'.

v) $\varphi.\{a_1,\ldots,a_n\}$, where φ is a referential path from R to R' and $a_1,\ldots,a_n$ are attributes of R'. □

Definition 7: Let r be a tuple of R.

i) $r / NULL = \{ r \}$.

ii) $r / a = \{ v \mid v = r.a \text{ and } v \ne NULL \}$, where a is an attribute of R.

iii) $r / \{a_1,\ldots,a_m\} = \{ v \mid v = r.a_i \text{ with } 1 \le i \le m \text{ and } v \ne NULL \}$, where $a_1,\ldots,a_n$ are attributes of R.

iv) $r / \varphi.a = \{ v \mid \exists r' \in r / \varphi \text{ and } v \in r'/a \}$, where φ is a referential path from R to R', a is an attribute of R' and r' is a tuple of R'.

v) $r / \varphi.\{a_1,\ldots,a_m\} = \{ v \mid \exists r' \in r / \varphi \text{ and } v \in r' / \{a_1,\ldots,a_m\} \}$, where φ is a referential path from R to R', $a_1,\ldots,a_n$ are attributes of R' and r' is a tuple of R'. □

We say that an XML Schema complex type T is *restricted* iff T is defined using the *complexType* and *sequence* constructors only, and the type of its attributes is an XML simple type.

In the rest of this section, let T be a restricted XML Schema complex type, and let R and R' be relation schemes of a relational schema S.

Definition 8: A *correspondence assertion (CA)* is an expression of the form $[T/e] \equiv [R/\delta]$ where e is an element or an attribute of T, with type T_e, and δ is a path of R such that:

i) If e is an attribute or a single occurrence element and T_e is a simple type, then δ has one of the following forms:
   - a, where a is an attribute of R whose type is compatible with T_e;
   - φ.a, where φ is a referential path from R to R' such that φ has single occurrence, and a is an attribute of R' whose type is compatible with T_e.

ii) If e is a multiple occurrence element and T_e is an simple type, then δ has one of the following forms:
- φ.a, where φ is a referential path from *R* to *R'* such that φ has multiple occurrence and a is an attribute of *R'*, whose type is compatible with T_e;
- $\{a_1,...,a_n\}$, where $a_1,...,a_n$ are attributes of *R* such that the type of a_i is compatible with T_e, for $1 \le i \le n$;
- $\varphi.\{a_1,...,a_n\}$, where φ is a referential path from *R* to *R'* such that φ has single occurrence, and $a_1,...,a_n$ are attributes of *R'* such that the type of a_i is compatible with T_e, for $1 \le i \le n$.

iii) If e is a single occurrence element and T_e is a complex type, then δ has one of the following forms:
- φ, where φ is a referential path from *R* to *R'* such that φ has single occurrence;
- NULL

iv) If e is a multiple occurrence element and T_e is a complex type, then δ is a path from *R* to *R'* such that δ has multiple occurrence. □

Definition 9: Let $\mathcal{A}$ be a set of correspondence assertions. We say that $\mathcal{A}$ *fully specifies* T *in terms of R* iff

i) For each element or attribute e of T, there is a single CA of the form [T/e] ≡ [*R*/δ] in $\mathcal{A}$, called *the CA for* e *in* $\mathcal{A}$.

ii) For each assertion in $\mathcal{A}$ of the form [T/e] ≡ [*R*/δ], where e is an element of complex type T_e and δ is a referential path from *R* to *R'*, then $\mathcal{A}$ fully specifies T_e in terms of *R'*.

iii) For each assertion in $\mathcal{A}$ of the form [T/e] ≡ [*R*/NULL], where e is an element of complex type T_e, then $\mathcal{A}$ fully specifies T_e in terms of *R*. □

Definition 10: Let $\mathcal{A}$ be a set of correspondence assertions such that $\mathcal{A}$ fully specifies T in terms of *R*. Let R be a relation over *R*.

i) Let S_1 be a set of elements that are instances of an XML simple type T. Let S_2 be a set of values of an SQL scalar data type. We say that $S_1 \equiv_{\mathcal{A}} S_2$ iff

$$\$t \in S_1 \text{ iff there is } v \in S_2 \text{ such that } \$t\ /\ text() = f(v)$$

where f is a function that maps an SQL value to an XML value [10].

ii) Let S_1 be a set of values of an XML simple type. Let S_2 be a set of values of an SQL scalar data type. We say that $S_1 \equiv_{\mathcal{A}} S_2$ iff

$$v_1 \in S_1 \text{ iff there is } v_2 \in S_2 \text{ such that } v_1 = f(v_2)$$

where f is a function that maps an SQL value to an XML value [10].

iii) Let S_1 be a set of elements of an XML Schema complex type T. Let S_2 be a set of tuples of R. We say that $S_1 \equiv_{\mathcal{A}} S_2$ iff

$$\$t \in S_1 \text{ iff there is } r \in S_2 \text{ such that } \$t \equiv_{\mathcal{A}} r.$$

iv) Let r be a tuple of R and let \$t be an instance of T. We say that $\$t \equiv_{\mathcal{A}} r$ iff, for each element e of T such that [T/e] ≡ [*R*/δ] is the CA for e in $\mathcal{A}$ (which exists by assumption on $\mathcal{A}$), then $\$t/e \equiv_{\mathcal{A}} r/\delta$, and, for each attribute a of T such that [T/a] ≡ [*R*/δ] is the CA for a in $\mathcal{A}$ (which exists by assumption on $\mathcal{A}$), then $DATA(\$t/@a) \equiv_{\mathcal{A}} r/\delta$.

If $\$t \equiv_{\mathcal{A}} r$, we say that \$t is *semantically equivalent to* r *as specified by* $\mathcal{A}$. □

4. Specifying XML Views

We propose to specify an XML view with the help of a set of correspondence assertions [12], which axiomatically specify how the XML view elements are synthesized from tuples of the base source. Let S be the base relational schema. An *XML view*, or simply, a *view* over S is a quadruple V=< e, T, *R*, $\mathcal{A}$>, where:

(i) e is the name of the primary element of the view;

(ii) T is the XML type of element e;

(iii) *R* is a relation scheme or a relational view scheme of S;

(iv) $\mathcal{A}$ is a set of path correspondence assertions that fully specifies T in terms of *R*.

We say that the pair <e,T> is the *view schema* of V and that *R* is the *pivot relation scheme* of the view.

Consider, for example, the view PurchaseOrder_XML, whose primary element <*PurchaseOrder*> has type PurchaseOrder_Type, and whose pivot relation scheme is *ORDERS_REL*. Figure 3 shows an SQL/XML specification of PurchaseOrder_XML and Figure 2 depicts a graphical representation of PurchaseOrder_Type. Figure 6 shows the correspondence assertions of PurchaseOrder_XML, which fully specifies PurchaseOrder_Type in terms of *ORDERS_REL*.

We developed a tool, called XML View-By-Assertions (XVBA), to support the definition of view correspondence assertions. XVBA features a simple graphical interface which allows the user to navigate to related tables. The process starts with the user

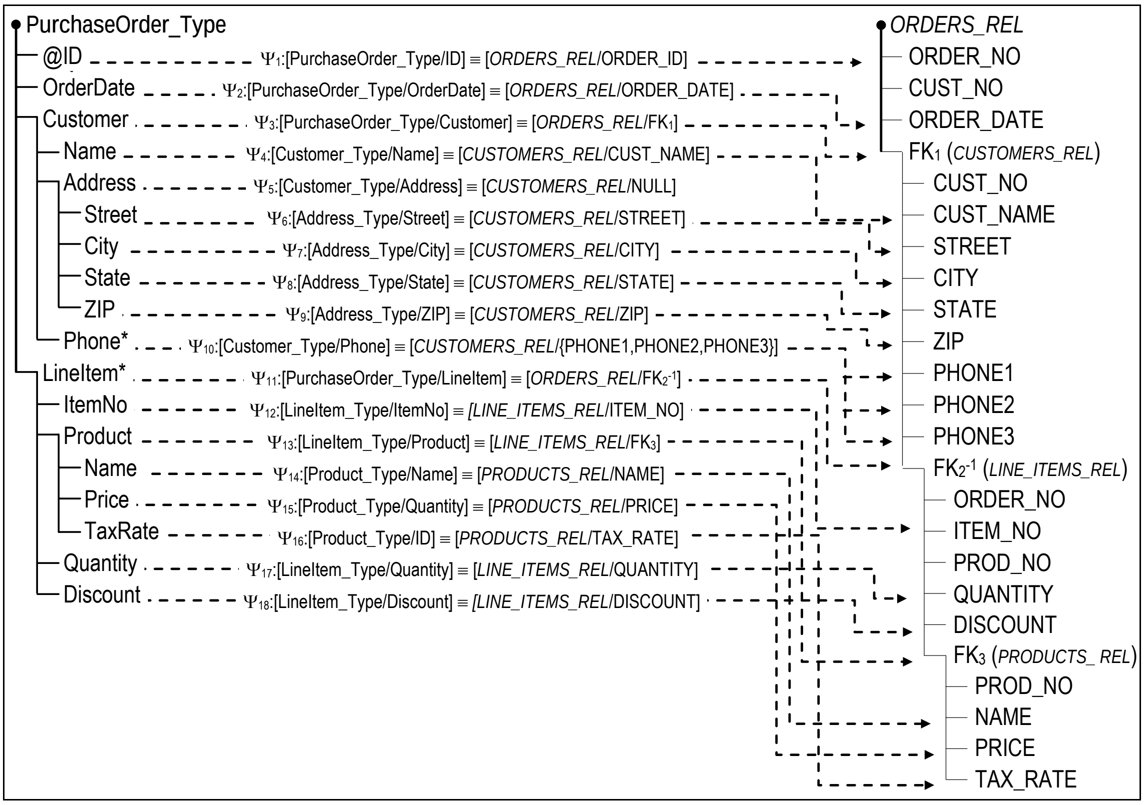

Figure 6 – Correspondence Assertions of PurchaseOrder_XML **view**

loading a source and view schemas into XVBA. The user can then graphically connect elements or attributes of the XML type with attributes or paths of the pivot relation.

The correspondence assertions of PurchaseOrder_XML are generated by: (1) matching the elements and attributes of PurchaseOrder_Type with attributes or paths of *ORDERS_REL*; and (2) recursively descending into sub-elements of PurchaseOrder_Type to define their correspondence assertion. For example, to define the assertion of the element LineItem (ψ_{11}:[PurchaseOrder_Type/LineItems] $\equiv$ [*ORDERS_REL*/FK_2^{-1}]), the user selects the element LineItem on the view schema and the inverse foreign key FK_2^{-1} on the database schema.

5. Mapping Assertions to SQL/XML

Let S be a relational schema and V=< e, T, *R*, $\mathcal{A}$ > be a XML view over S. In this section, we show that the view correspondence assertions in $\mathcal{A}$ define a mapping that can be correctly translated to an SQL/XML query view definition.

Given an instance σ_S of S, let $\sigma_S(R)$ denote the relation that σ_S associates with *R*. Moreover, given an element e, the *extended content* of e is the list of attributes and child elements of e. The correspondence assertions in $\mathcal{A}$ define a functional mapping, denoted DEF_V, from instances of the source schema S to instances of the view schema. Given an instance σ_S of S, the value of V on σ_S is given by:

$DEF_V(\sigma_S)$ = { \$t | \$t is an <e> element of type T and $\exists r \in \sigma_S(R)$ such that \$t $\equiv_{\mathcal{A}}$ r }

The SQL/XML definition of V is given by:

```
CREATE VIEW V OF XMLTYPE
AS SELECT XMLELEMENT( "e", τ[R→T][r] )
   FROM R r
```

where $\tau[R \rightarrow T][r]$ is a sequence of SQL/XML sub-queries, one for each element or attribute of T. Given a tuple r of $\sigma_S(R)$, $\sigma_S(\tau[R \rightarrow T][r])(r)$ denotes the result of evaluating the SQL/XML sub-queries in the instance σ_S, with *r* replaced by r. We will prove that, given an instance \$t of T whose extended content is constructed from $\sigma_S(\tau[R \rightarrow T][r])(r)$, then \$t $\equiv_{\mathcal{A}}$ r.

Figure 7 presents the algorithm GenConstructor that generates the constructor function $\tau[R \rightarrow T][r]$. Figure 8 presents the algorithm GenSQL/XMLSubquery, where φ is a path of the form $\ell_1. \ldots .\ell_n$, as defined in Definition 6, and $Join\varphi(r)$ is defined by the following SQL fragment:

$R_1\ r_1, \ldots, R_n\ r_n$
WHERE $r.a_1^{\ell 1} = r_1.b_1^{\ell 1}$ AND … AND $r.a_{m1}^{\ell 1} = r_1.b_{m1}^{\ell 1}$
AND $r_1.a_1^{\ell 2} = r_2.b_1^{\ell 2}$ AND … AND $r_1.a_{m2}^{\ell 2} = r_2.b_{m2}^{\ell 2}$
…
AND $r_{n-1}.a_1^{\ell n} = r_n.b_1^{\ell n}$ AND … AND $r_1.a_{mn}^{\ell n} = r_2.b_{mn}^{\ell n}$

Input: a XML Type T, a relation scheme R, a set of correspondence assertions $\mathcal{A}$ that fully specifies T in terms of R and an alias r for R.
Output: Function $\tau[R \rightarrow T][r]$.
Let τ be an string;
$\tau := \emptyset$;
If T has attributes then
  $\tau := \tau$ + "XMLAttributes("
  For each attribute a of type T where Ψ_a is the CA for a in $\mathcal{A}$ do
    $\tau := \tau$ + GenSQL/XMLSubquery($\mathcal{A}$, Ψ_a, r);
  end for;
  $\tau := \tau$ + ")"
End If;
For each element e of T where Ψ_e is the CA for e in $\mathcal{A}$ do
    $\tau := \tau$ + GenSQL/XMLSubquery($\mathcal{A}$, Ψ_e, r);
End for;
Return τ ;

Figure 7 – Algorithm GenConstructor

Input: a set of correspondence assertions $\mathcal{A}$ that fully specifies T in terms of R, the CA $[T/e] \equiv [R/\delta]$ in $\mathcal{A}$ where e is an element or attribute of type T_e, and an alias r for R
Output: a SQL/XML sub-query
Let Q be an string;
In case of
  Case 1: If e is a single occurrence element, T_e is a simple type and δ = a, then
    Q := "XMLFOREST(r.a AS \"e\")";
  Case 2: If e is a single occurrence element, T_e is a simple type and $\delta = \varphi$.a, then
    Q := "XMLFOREST((SELECT r_n.a FROM Joinφ(r)) AS \"e\")";
  Case 3: If e is a multiple occurrence element, T_e is a simple type and $\delta = \{a_1,...,a_n\}$, then
    Q := "XMLCONCAT(XMLFOREST(r.a_1 AS \"e\")," + …+ "XMLFOREST(\"e\", r.a_n AS \"e\"))";
  Case 4: If e is a multiple occurrence element, T_e is a simple type and $\delta = \varphi/ \{a_1,...,a_n\}$, then
    Q := "XMLCONCAT((SELECT XMLFOREST(r_n.a_1 AS \"e\", … , r_n.a_n AS \"e\") FROM Joinφ(r)))";
  Case 5: If e is a multiple occurrence element, T_e is a simple type and $\delta = \varphi/$ a, then
    Q := "(SELECT XMLAGG(XMLFOREST(r_n.a AS \"e\") FROM Joinφ(r))";
  Case 6: If e is a single occurrence element, T_e is a complex type and $\delta = \varphi$, then
    Q := "(SELECT XMLELEMENT(\"e\", " + GenConstructor(T_e, R_n, $\mathcal{A}$, r_n) + ") FROM Joinφ(r)) "
  Case 7: If e is a multiple occurrence element, T_e is a complex type and $\delta = \varphi$, then
    Q := "(SELECT XMLAGG(XMLELEMENT(AS \"e\",
      + GenConstructor(T_e, R_n, $\mathcal{A}$, r_n) + ")) FROM Joinφ(r))";
  Case 8: If e is a single occurrence element, T_e is a complex type and δ = NULL, then
    Q := "XMLELEMENT(\"e\", " + GenConstructor(T_e, R, $\mathcal{A}$, r) + ")";
  Case 9: If e is an attribute, T_e is a simple type and δ = a, then
    Q := " r.a AS \"e\" ";
  Case 10: If e is an attribute, T_e is an simple type and $\delta = \varphi$.a, then
    Q := " (SELECT r_n.a FROM Joinφ(r)) AS \"e\" ";
End case;
return Q ;

Figure 8 – Algorithm GenSQL/XMLSubquery

The correctness of these algorithms follows from the propositions and theorem below. In what follows, let T be a XML Schema type, R be a relation scheme, $\mathcal{A}$ be a set of correspondence assertions that fully specifies T in terms of R, and r be an alias for R.

Proposition 1: Let Ψ be the CA $[T/e] \equiv [R/\delta]$ for element e of type T_e in $\mathcal{A}$. Let GenSQL/XMLSubquery ($\mathcal{A}$, Ψ, r) = $Q_e[r]$. Let r be a tuple of $\sigma_S(R)$ and $\mathcal{S}$ be the set of <e> elements resulting from evaluating $Q_e[r]$ in σ_S with r replaced by r. Then, we have that $\mathcal{S} \equiv_{\mathcal{A}}$ r/δ. □ (See [13] for the proof).

Proposition 2: Let Ψ be the CA $[T/a] \equiv [R/\delta]$ for attribute a of type T_a in $\mathcal{A}$. Let GenSQL/XMLSubquery ($\mathcal{A}$, Ψ, r) = $Q_a[r]$. Let r be a tuple of $\sigma_S(R)$ and v_a be the value resulting from evaluating $Q_a[r]$ in σ_S with r replaced by r. Then, we have that $v_a = f(v)$, where v is the only value in r/δ, and f is a function that maps SQL values to XML values [5]. □ (See [13] for the proof).

Theorem 1:

Let $a_1,...,a_k$ be the attributes of T and let $e_1,...,e_m$ be the elements of T.
Let GenConstructor(R, r, T, $\mathcal{A}$) = $\tau[R\to T][r]$
Let r be a tuple of $\sigma_S(R)$.
Let \$t be an <e> element of type T whose extended content is constructed from $\sigma_S(\tau[R\to T][r])(\mathrm{r})$.
Then, \$t $\equiv_{\mathcal{A}}$ r. □

Proof: Let $a_1,...,a_k$ be the attributes of T. Let Ψ_{ai} be the CA for a_i in $\mathcal{A}$ and T_{ai} be the type of a_i, for $1 \le i \le k$. Assume that Ψ_{ai} is of the form $[T/a_i] \equiv [R/\delta_{ai}]$. Let $e_1,...,e_m$ be the elements of T. Let Ψ_{ei} be the CA for e_i in $\mathcal{A}$ and T_{ei} be the type of e_i, for $1 \le i \le m$. Assume that Ψ_{ei} is of the form $[T/e_i] \equiv [R/\delta_{ei}]$. Let $\tau[R\to T][r]$ be the constructor function generated by GenConstructor. From the algorithm, we have that:

$\tau[R\to T][r]$ = XMLAttributes($Qa_1[r]$, ... $Qa_k[r]$), Qe_1 $[r]$, ... $Qe_m[r]$, where
$Q_{ai}[r]$ = GenSQL/XMLSubQuery($\mathcal{A}$,Ψ_{ai},r), for $1 \le i \le k$
$Q_{ei}[r]$ = GenSQL/XMLSubQuery($\mathcal{A}$,Ψ_{ei},r), for $1 \le i \le m$

Let r be a tuple of $\sigma_S(R)$. Let \$t be an <e> element of type T whose extended content is constructed from $\sigma_S(\tau[R\to T][r])(\mathrm{r})$. For $1 \le i \le k$, let v_{ai} be the value resulting from evaluating $Q_{ai}[r]$ in σ_S with r replaced by r. For $1 \le i \le m$, let $\mathcal{S}_{ei}$ be the set of <e_i> elements resulting from evaluating $Q_{ei}[r]$ in σ_S with r replaced by r. Therefore, \$t = <e a_1 ="v_{a1}" … a_k ="v_{ak}"> $\mathcal{S}_{e1}$ ... $\mathcal{S}_{em}$ </e>. From Proposition 1, we have $\mathcal{S}_{ei} \equiv_{\mathcal{A}}$ r/δ_{ei}, for $1 \le i \le m$. From Proposition 2, we have $v_{ai} = f(v)$, for $1 \le i \le k$, where v is the only value in r/δ_{ai}, and f is a function that maps SQL values to XML values. So, from Definition 10 (ii), we have that DATA(\$t/@a_i) $\equiv_{\mathcal{A}}$ r/δ_{ai}, $1 \le i \le k$. Therefore, from Definition 10 (iv), we have that \$t $\equiv_{\mathcal{A}}$ r. □

In what follows, for simplicity, let $\tau[R\to T](\mathrm{r})$ denote the function $\sigma_S(\tau[R\to T][r])(\mathrm{r})$ that constructs the extended content of an instance \$t of T such that \$t $\equiv_{\mathcal{A}}$ r.

Consider, for example, the SQL/XML definition of PurchaseOrder_XML, shown in Figure 3. The constructor function τ[*ORDERS_REL*→PurchaseOrder_Type](O) (lines 3 to 29) constructs the extended content of an instance of PurchaseOrder_Type from a tuple of ORDERS_REL. The constructor function contains four sub-queries, one for each element or attribute of PurchaseOrder_Type. In GenSQL/XMLSubquery, each subquery is generated from the CA of the corresponding element or attribute. Figure 3 shows the assertion that generates each SQL/XML subquery of τ[*ORDERS_REL*→PurchaseOrder_Type](O).

We will show that τ[*ORDERS_REL*→PurchaseOrder_Type](O) constructs the extended content of an instance $P of PurchaseOrder_Type, for each tuple O of an instance of *ORDERS_REL,* such that $P is semantically equivalent to O, as specified by the assertions of PurchaseOrder_XML.

Let ORDERS_DB be an instance of the relational schema ORDERS_DB. Let CUSTOMER_REL, ORDERS_REL, PRODUCTS_REL and LINE_ITEMS_REL be the instances that ORDERS_DB associates with the relation schemes *CUSTOMER_REL*, *ORDERS_REL*, *PRODUCTS_REL* and *LINE_ITEMS_REL* of ORDERS_DB, respectively.

Let O be a tuple of ORDERS_REL and let $P be an instance of PurchaseOrder_Type whose extended content is constructed with τ[*ORDERS_REL*→ PurchaseOrder_Type](O). From Definition 10 and from Ψ_1, Ψ_2, Ψ_3 and Ψ_{11}, we have that $P $\equiv_{\mathcal{A}}$ O iff:

(1) DATA($P / @ID) $\equiv_{\mathcal{A}}$ O / ORDER_NO,
(2) $P / OrderDate $\equiv_{\mathcal{A}}$ O / ORDER_DATE,
(3) $P / Customer $\equiv_{\mathcal{A}}$ O / FK_1, and
(4) $P / LineItem $\equiv_{\mathcal{A}}$ O / FK_2^{-1}.

Proof of (1): From line 4 of Figure 3, we have that:
DATA($P / @ID) = { v | $v = f$(O.ORDER_NO) and $v \neq$ NULL }.
From Definition 7 (ii), we have that
O / ORDER_NO = { D | D = O.ORDER_NO and O.ORDER_NO $\neq$ NULL }.
So, from Definition 10 (ii), we have that: DATA($P / @ID) $\equiv_{\mathcal{A}}$ O / ORDER_NO. □

Proof of (2): From line 5 of Figure 3, we have that:
$P / OrderDate = { $D | $D=<OrderDate>$f$(O.ORDER_DATE) </OrderDate> and O.ORDER_DATE $\neq$ NULL }.
From Definition 7 (ii), we have that:
O / ORDER_DATE = { D | D = O.ORDER_DATE and O.ORDER_DATE $\neq$ NULL }.
So, from Definition 10 (i), we have that: $P / OrderDate $\equiv_{\mathcal{A}}$ O / ORDER_DATE. □

Proof of (3): From lines 6-17 of Figure 3, we have that:
$P / Customer = { $C | $\exists$ C $\in$ CUSTOMER_REL such that C.CUST_NO = O.CUST_NO and the extended content of $C is constructed from τ[*CUSTOMERS_REL*→Customer_Type](C)}.
In following, we show that, given a tuple C $\in$ CUSTOMER_REL and an element $C whose extended content is constructed from τ[*CUSTOMERS_REL*→Customer_Type](C), then $C $\equiv_{\mathcal{A}}$ C. From Ψ_4, Ψ_5, e Ψ_{10}, we have that $C $\equiv_{\mathcal{A}}$ C iff:

(3.1) $C / Name $\equiv_{\mathcal{A}}$ C / CUST_NAME,
(3.2) $C / Address $\equiv_{\mathcal{A}}$ C / NULL, and
(3.3) $C / Phone $\equiv_{\mathcal{A}}$ C / {PHONE1, PHONE2, PHONE3}.

Proof of (3.1): The proof follows from line 7 of Figure 3 and is similar to the proof of (2).

Proof of (3.2): From lines 8-12 of Figure 3, we have that:

$C / Address = { $A } where the extended content of $A is constructed from

τ[*CUSTOMERS_REL*$\rightarrow$Address_Type](C).

In following, we show that, if the extended content of $A is constructed from τ[*CUSTOMERS_REL*$\rightarrow$Address_Type](C) then $A $\equiv_{\mathcal{A}}$ C. From Ψ_6, Ψ_7, Ψ_8 and Ψ_9, we have that $A $\equiv_{\mathcal{A}}$ C iff

(3.2.1) $A / Street $\equiv_{\mathcal{A}}$ C / STREET,

(3.2.2) $A / City $\equiv_{\mathcal{A}}$ C / CITY,

(3.2.3) $A / State $\equiv_{\mathcal{A}}$ C / STATE, and

(3.2.4) $A / ZIP $\equiv_{\mathcal{A}}$ C / ZIP.

The proofs of (3.2.1), (3.2.2), (3.2.3) and (3.2.4) are similar to the proof of (2) and follow from lines 9, 10, 11 and 12 of Figure 3, respectively. So, from (3.2.1), (3.2.2), (3.2.3) and (3.2.4), we have that $C / Address = {$A} where $A $\equiv_{\mathcal{A}}$ C.

From Definition 7 (i), we have that C / NULL = {C}. Therefore, from Definition 10 (iii), we have that $C / Address $\equiv_{\mathcal{A}}$ C / NULL. □

Proof of (3.3): From lines 13-15 of Figure 3, we have that

$C / Phone = S1 $\cup$ S2 $\cup$ S3 where:

S1 = { $H | $H = <Phone> f(C.PHONE1) </Phone> and H $\neq$NULL },

S2 = { $H | $H = <Phone> f(C.PHONE2) </Phone> and H $\neq$NULL }, and

S3 = { $H | $H = <Phone> f(C.PHONE3) </Phone> and H $\neq$NULL }.

From Definition 7 (iii) we have that:

C / {PHONE1, PHONE2, PHONE3} = { H | (H = C.PHONE1 or H = C.PHONE2 or H = C.PHONE3) and H $\neq$ NULL }.

Therefore, from Definition 10 (ii), we have that:

$C / Phone $\equiv_{\mathcal{A}}$ C / {PHONE1, PHONE2, PHONE3}.

So, from (3.1), (3.2) and (3.3), we have that:

$P / Customer = { $C | $\exists$ C $\in$ CUSTOMER_REL such that C.CUST_NO = O.CUST_NO and $C $\equiv_{\mathcal{A}}$ C }.

From Definition 2 (i), we have that:

O / FK_1 = { C | $\exists$ C $\in$ CUSTOMER_REL such that C.CUST_NO = O.CUST_NO }.

Therefore, from Definition 10 (iii), we have that: $P / Customer $\equiv_{\mathcal{A}}$ O / FK_1. □

Proof of (4)*:* From lines 18-29 of Figure 3, we have that:

$P / LineItem = { $L | $\exists$ L $\in$ INTEMS_REL such that L.ORDER_NO = O.ORDER_NO and the extended content of $L is constructed from τ[*LINE_ITEMS_REL*$\rightarrow$LineItem_Type](L) }.

In following, we show that, given a tuple L $\in$ LINE_ITEMS_REL and an element $L whose extended content is constructed from τ[*LINE_ITEMS_REL*$\rightarrow$LineItem_Type](L), then $L $\equiv_{\mathcal{A}}$ L. From Ψ_{12}, Ψ_{13}, Ψ_{17} and Ψ_{18}, we have that $L $\equiv_{\mathcal{A}}$ L iff:

(4.1) $L / ItemNo $\equiv_{\mathcal{A}}$ L / ITEM_NO,

(4.2) $L / Product $\equiv_{\mathcal{A}}$ L / FK_3,

(4.3) $L / Quantity $\equiv_{\mathcal{A}}$ L / QUANTITY, and

(4.4) $L / Discount $\equiv_{\mathcal{A}}$ L / DISCOUNT.

The proofs of (4.1), (4.3) and (4.4) are similar to the proof of (2) and follow from lines 19, 26 and 27 of Figure 3, respectively.

Proof of (4.2): From lines 19-25, we have that

$L / Product = { $D | ∃D ∈ PRODUCTS_REL such that D.PROD_NO = L.PROD_NO and the extended content of $D is constructed from τ[*PRODUCTS_REL*→ Product_Type](D) }.

In following, we show that, given a tuple D ∈ PRODUCTS_REL and an element $D whose extended content is constructed from τ[*PRODUCTS_REL*→ Product_Type](D), then $D $\equiv_{\mathcal{A}}$ D. From Ψ_{14}, Ψ_{15} and Ψ_{16}, we have that $D $\equiv_{\mathcal{A}}$ D iff:

(4.2.1) $D / Name $\equiv_{\mathcal{A}}$ D / NAME,

(4.2.2) $D / Price $\equiv_{\mathcal{A}}$ D / PRICE, and

(4.2.3) $D / TaxRate $\equiv_{\mathcal{A}}$ D / TAX_RATE.

The proofs of (4.2.1), (4.2.2) and (4.2.3) are similar to the proof of (2) and follow from lines 21, 22 and 23 of Figure 3, respectively. So, from (4.2.1), (4.2.2) and (4.2.3), we have that:

$L / Product = { $D | ∃D ∈ PRODUCTS_REL such that D.PROD_NO = L.PROD_NO and and $D $\equiv_{\mathcal{A}}$ D }.

From Definition 2 (i), we have that:

L / FK_3 = { D | ∃ D ∈ PRODUCTS_REL such that D.PROD_NO = L.PROD_NO }.

Therefore, from Definition 10 (iii), we have that: $L / Product $\equiv_{\mathcal{A}}$ L / FK_3.

So, from (4.1), (4.2), (4.3) and (4.4), we have that:

$P / LineItem = { $L | ∃ L ∈ INTEMS_REL such that L.ORDER_NO = O.ORDER_NO and and $L $\equiv_{\mathcal{A}}$ L }.

From Definition 2 (ii), we have that:

O / FK_2^{-1} = { L | ∃ L ∈ ITEMS_REL such that L.ORDER_NO = O.ORDER_NO }.

Therefore, from Definition 10 (iii), we have that: $P / LineItem $\equiv_{\mathcal{A}}$ O / FK_2^{-1}. □

6. Conclusions

We argued in this paper that we may fully specify an XML view in terms of a relational schema using view correspondence assertions, in the sense that the assertions define a mapping from instances of the relational schema to instances of the XML view schema.

We presented an algorithm that generates, based on the view correspondence assertions, the SQL/XML view definition. Moreover, we showed that the SQL/XML query generated by the algorithm correctly represents the mapping defined by the view correspondence assertions.

As future work, we envision a number of extensions to our mapping formalism to express broader types of view schema. We are working in generalizing our mapping formalism and algorithm to deal with XML views of object-relational data.

References

[1] Adams, D., *Oracle® XML DB 10g Release 2 Developer's Guide*. http://www.oracle.com/technology/documentation/database10gR2.html, 2005.

[2] Benham, S.E., *IBM XML-Enabled Data Management Product Architecture and Technology*. In: XML Data Management, Native XML and XML-Enable Database Systems, Chaudhri, A.B., Rashid, A., and Zicari, R. (eds.), Addison Wesley, 2003.

[3] Carey, M. J., Kiernan, J., Shanmugasundaram, J., Shekita, E. J., Subramanian, S. N., *XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents*. In: Proceedings of the 26th International Conference on Very Large Data Bases, Pages 646–648, 2000.

[4] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E. and Zemke, F., *SQL:2003 has been published*. In: ACM SIGMOD Record, Volume 33, Issue 1 (March 2004), COLUMN: Standards, Pages 119–126, 2004.

[5] Eisenberg, A., Melton, J., *SQL/XML and the SQLX Informal Group of Companies*. ACM SIGMOD Record, Volume 30, Issue 3 (September 2001), COLUMN: Standards, 2001.

[6] Fernández, M., Kadiyska, Y., Suciu, D., Morishima, A., Tan, W., *SilkRoute: A framework for publishing relational data in XML*. In: ACM Transactions on Database Systems (TODS), Volume 27, Issue 4 (December 2002), Pages 438–493, 2002.

[7] Hernandez, M.A., Miller, R.J., Haas, L., Yan, L., Ho, C.T.H., Tian, X., *Clio: A semi-automatic tool for schema mapping*. In: International Conference on Management of Data, Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Page 607, 2001.

[8] Krishnaprasad, M., Liu, Z. H., Manikutty, A., Warner, J. W., Arora, V., Kotsolovos, S., *Query Rewrite for XML in Oracle XML DB*. In: Proceedings of the 30th International Conference on Very Large Data Bases, Pages 1122-1133, 2004.

[9] Liu, Z. H., Krishnaprasad, M., Arora, V., *Native XQuery Processing in Oracle XMLDB*. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Pages 828 - 833, 2005.

[10] Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A., Fagin, R., *Translating Web Data*. In: Proceedings of the International Conference on Very Large Data Bases, Pages 598–609, 2002.

[11] Rys, M., *XML Support in Microsoft SQL Server 2000*. In: XML Data Management, Native XML and XML-Enable Database Systems, Chaudhri, A.B., Rashid, A., and Zicari, R. (eds.), Addison Wesley, 2003.

[12] Vidal, V. M. P., Araujo, V. S., Casanova, M. A., *Towards Automatic Generation of Rules for the Incremental Maintenance of XML Views over Relational Data*. In: WISE 2005, Pages 189-202, November, 2005.

[13] Vidal, V.M.P., Casanova, M.A., Lemos, F.C. *Automatic Generation of XML Views of Relational Data*. In: Technical Report (http://lia.ufc.br/~arida). Universidade Federal do Ceará, November, 2005.

[14] Yu, C., Popa, L., *Constraint-Based XML Query Rewriting For Data Integration*. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of data, SESSION: Research sessions: Data Integration, Pages 371–382, 2004.