Translating WFS Query to SQL/XML Query

Vânia Vidal, Fernando Lemos, Fábio Feitosa

Departamento de Computação – Universidade Federal do Ceará (UFC)
Fortaleza – CE – Brazil

{vvidal, fernandocl, fabiofbf}@lia.ufc.br

Abstract. *The purpose of the WFS specification, proposed by the the OpenGIS Consortium (OGC), is to describe the manipulation operations over geospatial data using GML. Web servers providing WFS service are called WFS Servers. WFS servers publish GML views of geographic features stored in data sources such that the user can query and update data sources through a user defined feature type schema.*

In this work, we study the problem of answering WFS queries through a feature type schema, when the data is stored in a relational database. A feature type is specified by the feature type schema and a set of correspondence assertions. The feature type's correspondence assertions formally specify relationships between the feature type schema and the relational database schema. We define the semantic for WFS query answering and present an algorithm that translate a WFS query defined over a feature type schema into a SQL/XML query defined over the relational database schema.

1. Introduction

The mission of the OpenGIS Consortium (OGC) [8] is to promote the development and use of advanced open system standards and techniques in the geoprocessing area and related information technologies. Two important OGC's initiatives are: the Geography Markup Language (GML) [9] and the Web Feature Service (WFS) [11]. The purpose of the WFS specification is to describe the manipulation operations over geospatial data using GML. Their objective is to provide queries, updates and exchange of geospatial data as geographic features instances encoded in GML. According to OGC, a geographic feature is an "abstraction of a real world phenomenon associated with a location relative to the Earth". It is possible to describe the feature form and localization through its geometric attributes, remaining the other attributes to represent its non-geographic properties. Given that, WFS servers publish GML views of geographic features stored in data sources such that the user can query and update data sources through the feature type schema

According to WFS specification [11], the requests supported by a WFS Server must be as follow: 1. *DescribeFeatureType*: Retrieves the XML [20] schema of the feature types.

2. *GetFeature*: Retrieves feature instances of a feature type. It is possible to apply selection filters on the *GetFeature* request. Those filters are specified by OGC in [7] and enclose logical, arithmetical and space conditions. 3. *LockFeature*: Allows blocking a feature type while it is being updated. It is optional. 4. *Transaction*: Allows updating feature types. It is optional. 5. *GetCapabilities*: Retrieves a list of feature types serviced by the WFS Server and what operations are supported on each feature type.

In this work, we study the problem of answering WFS queries through a feature type schema, when the data is stored in a relational database. In this work, a feature type is specified by the feature type schema and a set of correspondence assertions [13,14,18]. The feature type's correspondence assertions formally specify relationships [6,14] between the feature type schema and the relational database schema. We show that, based on the feature type correspondence assertions, one can efficiently translate WFS query to SQL/XML query.

SQL/XML [16] is an ANSI and ISO standard that provides support for using XML in the context of an SQL database system. The SQL/XML standard is being developed under the auspices of INCITS Technical Committee H2 as a new part (Part 14) of the SQL standard and is aligned with SQL:2003 [2]. SQL/XML is simple, highly intuitive and, in some development scenarios, is an ideal way of returning relational data. Moreover, we can apply XSL transformation on the XML result.

To the best of our knowledge, there is no work on translating WFS query to SQL/XML query. There is a vast amount of work on XML-to-SQL translation [3,5,15]. Since WFS query is more restricted than XML queries, the approach proposed here is simpler and more efficient. The Deegree WFS [1], which is an open source implementation of the WFS Specification, in case where the feature has complex properties, it reformulates a WFS query into several SQL queries, each of which will compute the value of a complex property. So, our solution, which translates to only one SQL/XML query, is much more efficient than Deegree's approach.

The main contributions of the paper are.

- We propose the use of correspondence assertions as the formalism to specify the mapping between a XML schema and a relational database schema.
- We formally specify the conditions for a set of correspondence assertions to fully specify a feature type in terms of relational database and, if so, we show that the mappings defined by the feature type correspondence assertions can be expressed as an SQL/XML query.
- We propose an algorithm that, based on the feature type’s correspondence assertions, generates an SQL/XML query that constructs <*featureMember*> elements from the

corresponding tuples of relational database. We note that other mapping formalisms are either ambiguous or require the user to declare complex logical mappings [4,23].

- We define the semantic for WFS query answering and present an algorithm that translate, based on the feature type correspondence assertions, a WFS query defined over a feature type schema into a SQL/XML query defined over the relational database. Moreover, we formally justify that SQL/XML query generated by the algorithm is a correct translation.

The paper is organized as follows. Section 2 presents our mapping formalism. Section 3 discusses how to specify a Feature Type using correspondence assertions. Section 4 presents the semantic for WFS query answering and present an algorithm that translates a WFS query to a SQL/XML query. Finally, Section 5 contains the conclusions.

2. Correspondence Assertions

In this section, let $R_1,...,R_n$ be relation schemes of a relational schema S. Let $\mathrm{R}_1,...,\mathrm{R}_n$ be relations over $R_1,...,R_n$, respectively.

Definition 2.1: Let fk be a foreign key of R_1 that references R_2. Assume that fk is of the form $R_1[a_1,...,a_m] \subseteq R_2[b_1,...,b_m]$.

(i) fk is a *link* from R_1 to R_2, and the inverse of fk, denoted fk^{-1}, is a *link* from R_2 to R_1.

(ii) Let r_1 be a tuple of R_1. Then $r_1/fk = \{ r_2 \mid r_2 \in \mathrm{R}_2 \text{ and } r_1.a_i = r_2.b_i, 1 \leq i \leq m \}$;

(iii) Let r_2 be tuple of R_2. Then $r_2/fk^{-1} = \{ r_1 \mid r_1 \in \mathrm{R}_1 \text{ and } r_1.a_i = r_2.b_i, 1 \leq i \leq m \}$. □

Definition 2.2: Let ℓ be a link from R_1 to R_2, and let r_1 and r_2 be tuples of R_1 and R_2, respectively. We say that:

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

(ii) ℓ is *single occurrence* iff a tuple of R_1 can reference at most one tuple of R_2 through ℓ. Otherwise, ℓ is *multiple occurrence*. □

Definition 2.3: Let $\ell_1, ..., \ell_{n-1}$ be links. Assume that ℓ_i is a foreign key of the form $R_i[a_1^{\ell_i},...,a_{m_i}^{\ell_i}] \subseteq R_{i+1}[b_1^{\ell_i},...,b_{m_i}^{\ell_i}]$ or the inverse of a foreign key of the form $R_{i+1}[b_1^{\ell_i},...,b_{m_i}^{\ell_i}] \subseteq R_i[a_1^{\ell_i},...,a_{m_i}^{\ell_i}]$, for $1 \leq i \leq n-1$.

(i) $\varphi = \ell_1. ... .\ell_{n-1}$ is a *path* from R_1 to R_n, and the tuples of R_1 *reference tuples of* R_n *through* φ.

(ii) φ is *single occurrence* iff ℓ_i is single occurrence, for $1 \leq i \leq n-1$. Otherwise, φ is *multiple occurrence*.

(iii) Let r_1 be a tuple of R_1. Then, $r_1/\varphi = \{ r_n \mid (\exists r_2 \in \mathrm{R}_2) ... (\exists r_n \in \mathrm{R}_n) (r_i.a_k^{\ell_i} = r_{i+1}.b_k^{\ell_i}$, for $1 \leq k \leq m_i$ and $1 \leq i \leq n-1)\}$. □

Definition 2.4: Let $a_1,...,a_m$ be attributes of R_1 and let r_1 be a tuple of R_1.

(i) $r_1/ a_1 = \{ v \mid v = r.a_1$ and $v \neq NULL \}$.

(ii) $r_1/\{a_1,...,a_m\} = \{ v \mid v = r.\ a_i$ where $1 \leq i \leq m$, and $v \neq NULL \}$.

(iii) $r_1/ NULL = \{ r_1 \}$. □

Definition 2.5: Let φ be a *path* from R_1 to R_n, $a_1,...,a_m$ be attributes of R_n. Let r_1 be a tuple of R_1.

(i) $r_1/\varphi.\ a_1 = \{ v \mid \exists r_n \in r_1/\varphi$ and $v \in r_n/ a_1\}$.

(ii) $r_1/\varphi.\{a_1,...,a_m\} = \{ v \mid \exists r_n \in r_1/\varphi$ and $v \in r_n/\{a_1,...,a_m\} \}$. □

We say that a XML schema type T is *restricted* iff T is a complex type defined using the ComplexType and Sequence constructors only. 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 2.6: A *correspondence assertion (CA)* is an expression of the form $[T/c] \equiv [R/exp]$ where c is an element of T, with type T_c, and *exp* is such that:

(i) If c is single occurrence and T_c is a simple type, then *exp* has one of the following forms:
 - a, where a is an attribute of *R* whose type is compatible with T_c.
 - φ.a, where φ is a path from *R* to *R*' such that φ has single occurrence, and a is an attribute of *R'* whose type is compatible with T_c.

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

(iii) If c is single occurrence and T_c is a complex type, then *exp* has one of the following forms:
 - φ, where φ is a path from *R* to *R'* such that φ is single occurrence;
 - NULL .

(iv) If c has multiple occurrence and T_c is a complex type then *exp* has the following form:
 - φ, where φ is a path from *R* to *R'* such that φ is multiple occurrence. □

Definition 2.7: 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 property c of T, there is a single CA of the form $[T/c] \equiv [R/exp]$ in $\mathcal{A}$, called *the CA for* c *in $\mathcal{A}$.*

(ii) For each assertion in $\mathcal{A}$ of the form [T/c] ≡ [R/φ], where c is of complex type T_c and φ is a path from R to R', then $\mathcal{A}$ fully specifies T_c in terms of R'.

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

Definition 2.8: 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 element of type T, which is a GML *abstractGeometry* type, and S_2 be a set of geometric objects database. Let g be a function that converts a geometric object to a GML fragment [12]. We say that $S_1 \equiv_{\mathcal{A}} S_2$ iff, $\$t \in S_1$ iff $\exists o \in S_2$ and $\$t = g(o)$.

(ii) Let S_1 be a set of element of a XML Schema simple type T. Let S_2 be a set of SQL scalar data types. Let f be the function that maps an SQL value to instances of T. We say that $S_1 \equiv_{\mathcal{A}} S_2$ iff $\$t \in S_1$ iff there is $v \in S_2$ and \$t/text() = $f(v)$.

(iii) Let S_1 be a set of element of a XML Schema complex type T, but not a GML *abstractGeometry* type. Let S_2 be a set of tuples of R. We say that $S_1 \equiv_{\mathcal{A}} S_2$ iff $\$t \in S_1$ iff there is $r \in S_2$ and \$t/node() $\equiv_{\mathcal{A}}$ 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 c of T such that [T/c] ≡ [R/exp] is the CA for c in $\mathcal{A}$ (which exists by assumption on $\mathcal{A}$), then \$t/c $\equiv_{\mathcal{A}}$ r/exp. If \$t $\equiv_{\mathcal{A}}$ r, we say that \$t is *semantically equivalent to* r *as specified by* $\mathcal{A}$. □

3. Specifying Feature Type

In this section, let S be a relational schema, R be a relation scheme of S. Let σ_S be an instance of S and R be a relation over R.

Definition 3.1: A feature type F over S is a triple F = < T, R, $\mathcal{A}$ > where T is the XML type for feature instances, R is the name of the master relation (table) of F which contains the geometric attribute, and $\mathcal{A}$ is the set of path correspondence assertions which fully specifies T in terms of R. □

In the rest of this Section let F be a feature type as in Definition 3.1.

Definition 3.2: The *extension* of the feature type F on σ_S is an XML document σ_F, where the root element contains a sequence of <F> elements of type T, such that each <F> element matches a tuple of R. More formally,

Document("σ_F")/F = { \$f | \$f is an instance of T and $\exists r \in$ R such that \$f $\equiv_{\mathcal{A}}$ r}. □

Definition 3.3: The extension of F can be computed by the SQL/XML query given by:

σ_F = SELECT XMLELEMENT("Extension_of_F ", XMLAGG(
XMLELEMENT("F", τ[R→T](r))
)) FROM R r.

$\tau[R \rightarrow T]$ is a constructor function such that given a tuple r of R, $\tau[R \rightarrow T](r)$ constructs an instance \$f of type T such that \$f $\equiv_{\mathcal{A}}$ r. □

Figure 3.1 presents the algorithm GenConstructor that receives as input a XML Schema type T, a relation scheme *R*, a set of correspondence assertions $\mathcal{A}$ such that $\mathcal{A}$ fully specifies T in terms of *R* and an alias *r* for the relation scheme *R* and generates the SQL/XML sub-query $\tau[R \rightarrow T]$. The proof of correctness of GenConstructor algorithm can be found in [19].

Example 3.1: Suppose the relational schema in Figure 3.3. Consider, for example, the feature type F_Station where *Station_Rel* is the Master Table, and the type $T_{Station}$, shown in Figure 3.4, is the feature instance type. Figure 3.5 shows the correspondence assertions of F_Station, which fully specifies $T_{Station}$ in terms of *Station_Rel*. These assertions are generated by: (1) matching the elements of $T_{Station}$ with attributes or paths of *Station_Rel*; and (2) recursively descending into sub-elements of $T_{Station}$ to define their correspondence. The extension of the feature type F_Station is defined by the SQL/XML query shown in Figure 3.6. The query returns an XML document where the root element contains a sequence of <F_Station> elements of type $T_{Station}$, such that each <F_Station> element matches a tuple of *Station_Rel*. Figure 3.8 shows the extension of F_Station on the database state in Figure 3.7.

Input: a XML Type T, a relation scheme *R*, a set of correspondence assertions $\mathcal{A}$ such that $\mathcal{A}$ fully specifies T in terms of *R* and an alias *r* for relation scheme *R*.
Output: Function $\tau[R \rightarrow T]$ such that, given a tuple r of an instance of *R*, $\tau[R \rightarrow T](r)$ constructs an instance \$t of T where \$t $\equiv_{\mathcal{A}}$ r.
Let I := 1;
Let Q[1..m] an array of string;
For each property p_i of T with 1<i<m, where [T/ p_i] ≡ [*R*/ exp] is the CA for p_i in $\mathcal{A}$ do
    Q[i] = GenSQL/XML(p_i, exp, *r*);
    i := i + 1;
end for;
return Q[1] + "," … "," + Q[m]

Figure 3.1 – Algorithm GenConstructor

Input: a property p of type T_p, an expression *exp* and an alias *r* for relation scheme *R*.
Output: a SQL/XML sub-query Q such that, given a tuple **r** of R, an instance of *R*, Q returns a set $\mathcal{S}$ of <p> elements of type T_p such that $\mathcal{S} \equiv$ **r**/*exp*.

In case of

Case 1: If p is single occurrence, T_p is an atomic type and *exp* = a, then
  Q = "XMLFOREST(*r*.a AS \"p\")";

Case 2: If p is single occurrence, T_p is an atomic type and *exp* = φ.a, then
  Q = "XMLFOREST((SELECT r_n.a FROM Joinφ(*r*)) AS \"p\")"

Case 3: If p is multiple occurrence, T_p is an atomic type and *exp* = {a_1,...,a_n}, then
  Q = "XMLCONCAT(XMLFOREST(*r*.a_1 AS \"p\")," + …+ "XMLFOREST(\"p\", *r*.a_n AS \"p\"))"

Case 4: If p is multiple occurrence, T_p is an atomic type and *exp* = φ/ {a_1,...,a_n}, then
  Q = "XMLCONCAT((SELECT XMLFOREST(r_n.a_1 AS \"p\", … , r_n.a_n AS \"p\") FROM Joinφ(*r*)))"

Case 5: If p is multiple occurrence, T_p is an atomic type and *exp* = φ/ a, then
  Q = "(SELECT XMLAGG(XMLFOREST(r_n.a AS \"p\") FROM Joinφ(*r*))"

Case 6: If p is single occurrence, T_p is a complex type and *exp* = φ, then
  Q = "XMLELEMENT(\"p\", (SELECT XMLCONCAT(" +
    + GenConstructor(T_p, R_n, $\mathcal{A}$, r_n) + ") FROM Joinφ(*r*)))"

Case 7: If p is multiple occurrence, T_p is a complex type and *exp* = φ, then
  Q = "(SELECT XMLAGG(XMLELEMENT(AS \"p\", " +
    + GenConstructor(T_p, R_n, $\mathcal{A}$, r_n)+")) FROM Joinφ(*r*))"

Case 8: If p is single occurrence, T_p is a complex type and *exp* = NULL, then
  Q = "XMLELEMENT(\"p\", " + GenConstructor(T_p, *R*, $\mathcal{A}$, *r*) + ")"

Case 9: If p is single occurrence, T_p is a geometric type and *exp* = a, then
  Q = "XMLFOREST(SDO_UTIL.TO_GMLGEOMETRY(*r*.a) AS \"p\")"

end case;
return Q ;

NOTE: In the Algorithm we have that:

(i) φ is a path of the form $\ell_1. \ldots . \ell_{n-1}$ as defined in Definition 3.3. Thus, given a tuple **r** of R, we have:
  r.φ = SELECT r_n FROM R_1 r_1,..., R_n r_n
    WHERE $r.a_k^{\ell_1} = r_1.b_k^{\ell_1}$, $1 \leq k \leq m_1$ AND $r_{i-1}.a_k^{\ell_i} = r_i.b_k^{\ell_i}$, $1 \leq k \leq m_i$, $2 \leq i \leq n$.

(ii) Joinφ(*r*) is defined by the following SQL fragments:
  R_1 r_1,..., R_n r_n WHERE $r.a_k^{\ell_1} = r_1.b_k^{\ell_1}$, $1 \leq k \leq m_1$, AND $r_{i-1}.a_k^{\ell_i} = r_i.b_k^{\ell_i}$, $1 \leq k \leq m_i$, $2 \leq i \leq n$.
  Such that, given a tuple **r** of R, then **r**.φ = SELECT r_n FROM Joinφ(*r*).

(iii) The SQL/XML function:
- XMLElement() constructs XML elements;
- XMLForest() constructs a sequence of XML elements, dropping eventual null values;
- XMLConcat() concatenates XML elements; and
- XMLAgg() aggregates XML elements.

Figure 3.2 – Algorithm GenSQL/XML

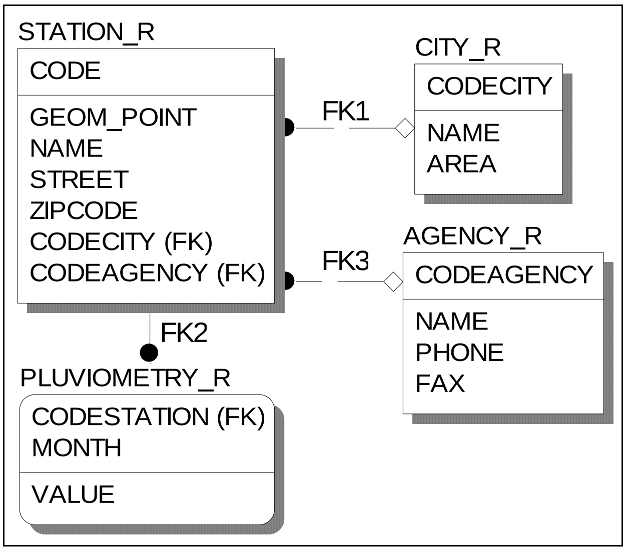

Figure 3.3 – Relational Schema DB_Station

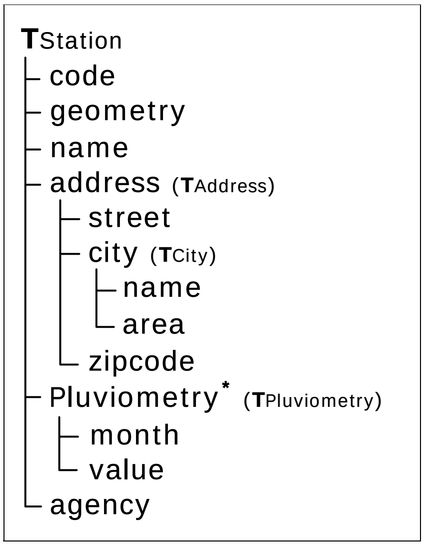

Figure 3.4 – XML type TStation

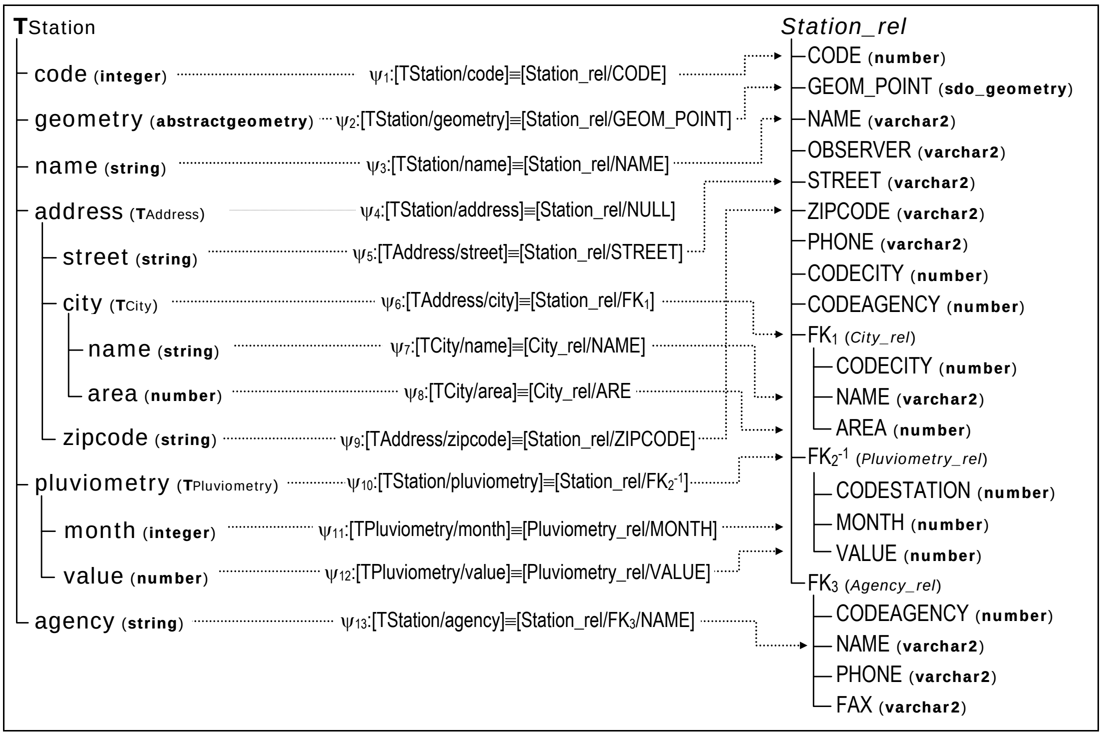

Figure 3.5 - Correspondence Assertions of F_Station

```
SELECT XMLELEMENT( "Extension_of_F_Station", XMLAGG(
    XMLELEMENT( "F_Station",
        XMLFOREST(S.CODE AS "code"), ..........................................from ψ1*
        XMLFOREST(SDO_UTIL.TO_GMLGEOMETRY(S.GEOM_POINT) AS "geometry"),........from ψ2
        XMLFOREST(S.NAME AS "name"), ..........................................from ψ3
        XMLELEMENT( "address", ................................................from ψ4
            XMLFOREST(S.STREET AS "street"), ..................................from ψ5
            XMLELEMENT( "city", ...............................................from ψ6
                (SELECT XMLCONCAT(
                        XMLFOREST(C.NAME AS "name"), ..........................from ψ7
                        XMLFOREST(C.AREA AS "area")) ..........................from ψ8
                 FROM City_rel C WHERE C.CODECITY = S.CODECITY) ),
            XMLFOREST(S.ZIPCODE AS "zipcode") ),...............................from ψ9
        (SELECT XMLAGG(XMLELEMENT( "pluviometry", .............................from ψ10
                        XMLFOREST(PL.MONTH AS "month"),........................from ψ11
                        XMLFOREST(PL.VALUE AS "value") ) ) ....................from ψ12
         FROM Pluviometry_rel PL WHERE S.CODE = PL.CODESTATION),
        XMLFOREST( (SELECT A.NAME FROM Agency_rel A
                    WHERE A.CODEAGENCY = S.CODEAGENCY) AS "agency") ...........from ψ13
    )
) )
FROM Station_Rel S;
*ψi is the assertion based on the algorithm generated the sub-query
```

Figure 3.6 – SQL/XML Definition for extension of F_Station

Station_rel

CODE	NAME	STREET	ZIPCODE	GEOM_POINT	CODE AGENCY	CODECITY
164	Serragem	R. Prinicpal s/n	62755000	-4.45,-38.5	1	2309458
165	Arisco	Sitio Penha	62755000	-4.65,-38.55	2	2309458
481	Arruda	R. São Francisco,606	62113000	-3.85,-40.66	1	2312908

Agency_rel

CODEAGENCY	NAME	PHONE	FAX
1	FUNCEME	31011088	31011093
2	SUDENE	34339031	34339045

Pluviometry_rel

CODESTATION	MONTH	VALUE
164	01	87.8
164	02	171.6
165	01	50.4
481	03	150

City_rel

CODECITY	NAME	AREA
2309458	OCARA	1450
2312908	SOBRAL	19820

Figure 3.7 – An instance of DB_Station

```xml
<Extension_of_F_Station>
<F_Station>
  <code>164</code>
  <geometry>
    <gml:Point>
      <gml:coordinates cs=","
        decimal="." ts="">
          -4.45,-38.5
      </gml:coordinates>
    </gml:Point>
  </geometry>
  <name>Serragem</name>
  <address>
    <street>R. Principal s/n</street>
    <city>
      <name>OCARA</name>
      <area>1450</area>
    </city>
    <zipcode>62755000</zipcode>
  </address>
  <pluviometry>
    <month>1</month>
    <value>87.8</value>
  </pluviometry>
  <pluviometry>
    <month>2</month>
    <value>171.6</value>
  </pluviometry>
  <agency>FUNCEME</agency>
</F_Station>
<F_Station>
  <code>165</code>
  <gml:Point>
    <gml:coordinates cs=","
      decimal="." ts="">
        -4.65,-38.55
    </gml:coordinates>
  </gml:Point>
  <name>Arisco</name>
  <address>
    <street>Sitio Penha</street>
    <city>
      <name>OCARA</name>
      <area>1450</area>
    </city>
    <zipcode>62755000</zipcode>
  </address>
  <pluviometry>
    <month>1</month>
    <value>50.4</value>
  </pluviometry>
  <agency>SUDENE</agency>
</F_Station>
<F_Station>
  <code>481</code>
  <gml:Point>
    <gml:coordinates cs=","
      decimal="." ts="">
      -3.85,-40.66</gml:coordinates>
  </gml:Point>
  <name>Arruda</name>
  <address>
    <street>R. Sao Francisco, 606</street>
    <city>
      <name>SOBRAL</name>
      <area>19820</area>
    </city>
    <zipcode>62113000</zipcode>
  </address>
  <pluviometry>
    <month>3</month>
    <value>150</value>
  </pluviometry>
  <agency>FUNCEME</agency>
</F_Station>
</Extension_of_F_Station>
```

Figure 3.8 – Extension of F_Station

4. Translating WFS query to SQL/XML query

The WFS *GetFeature* operation allows retrieval of features from a web feature service. A *GetFeature* request is processed by a WFS Server and an XML document containing the result set is returned to the client. In this work, a WFS *GetFeature* request is encoded in an XML document with a root element whose name is "GetFeature" and type is "wfs:GetFeatureType" [11]. A *<GetFeature>* element contains one or more *<Query>* elements, each of which in turn contains the description of a query. A *<Query>* element contained in a *GetFeature* request delivers feature instances of a given feature type, where each feature instance matches a tuple of the Master Table. The results of all queries contained in a *GetFeature* request are concatenated to produce the result set. A *<Query>* element contains:

(i) A mandatory attribute *typeName* which is used to indicate the name of a feature type to be queried.

(ii) A sequence of zero or more *<wfs:PropertyName>* elements which specifies what properties to retrieve. The value of each *<wfs:PropertyName>* element is an XPath [22] expressions that references a property or sub-property of the relevant feature type.

(iii) An optional *<Filter>* element which is used to define spatial and non-spatial constraints on a query. Filter specifications shall be encoded as described in the OGC Filter Encoding Implementation Specification [7].

Figure 4.1 shows an example of a WFS query over feature type F_Station. In the rest of this section, consider $F = < T, R, \mathcal{A} >$ be a feature type over S as defined in Section 3. Let σ_F be the extension of F on the current instance of S.

Definition 4.1: Let Q_W be a WFS Query over F. The canonical XQuery [21] Q_x for Q_W is defined as follows:

(i) If Q_W has no *<wfs:PropertyName>* elements then

```
Qx = FOR $f IN document("σF")/F
      WHERE $f satisfies the filter of QW
      RETURN <gml:featureMember> $f </gml:featureMember>
```

(ii) If Q_W has n *<wfs:PropertyName>* elements as shown in Figure 4.2. Then the canonical XQuery Q_x for Q_W is the one shown in Figure 4.3. □

Definition 4.2: Let Q_W be a WFS Query over F and let Q_x be the canonical XQuery for Q_W. We define the result of Q_W to be the result of evaluating Q_X. Notice that Q_X is evaluated on σ_F. Intuitively, this is what a user would see if we are to materialize the extension of the feature type. □

```
<wfs:Query typeName="F_Station">
   <wfs:PropertyName>name</wfs:PropertyName>
   <wfs:PropertyName>address/city</wfs:PropertyName>
   <wfs:PropertyName>pluviometry</wfs:PropertyName>
   <wfs:PropertyName>geometry</wfs:PropertyName>
  <ogc:Filter>
    <ogc:And>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>agency</ogc:PropertyName>
        <ogc:Literal>FUNCEME</ogc:Literal>
      </ogc:PropertyIsEqualTo>
      <ogc:BBox>
        <ogc:PropertyName>geometry</ogc:PropertyName>
        <gml:Envelope>
          <gml:lowercorner>-5.2  -42.5</gml:lowercorner>
          <gml:upperCorner>-2.5  -38.7</gml:upperCorner>
        </gml:Envelope>
      </ogc:BBox>
    </ogc:And>
  </ogc:Filter>
</wfs:Query>
```

Figure 4.1 – WFS Query Q_{W1}

```
<wfs:Query typeName="F">
  <wfs:PropertyName>Path1</wfs:PropertyName>
  <wfs:PropertyName>Path2</wfs:PropertyName>
   …
  <wfs:PropertyName>Pathn</wfs:PropertyName>
 <ogc:Filter>
</wfs:Query>
```

Figure 4.2 – WFS Query Q_W

```
FOR $f in document("σF")/F
WHERE $f satisfies the filter of QW
RETURN <gml:featureMember>{
            <F> {
               $f/path1,
               $f/path2,
               …
               $f/pathn
            }</F>
         }<gml:featureMember>
```

Figure 4.3 – Canonical XQuery Q_X for Q_W

```
FOR $f in document("σF")/Station
WHERE $f satisfies the filter of QW1
RETURN <gml:featureMember>{
            <F_Station> {
               $f/name,
               $f/address/city,
               $f/pluviometry,
               $f/geometry
            }</F_Station>
         }<gml:featureMember>
```

Figure 4.4 – Canonical XQuery Q_{X1} for Q_{W1}

```
<gml:FeatureMember>
  <F_Station>
    <name>Serragem</name>
    <city> <name>OCARA</name>
           <area>1450</area> </city>
    <pluviometry> <month>1</month>
       <value>87.8</value> </pluviometry>
    <pluviometry> <month>2</month>
       <value>171.6</value> </pluviometry>
    <geometry>
      <gml:Point>
        <gml:coordinates cs="," decimal="." ts="">
          -4.45,-38.5</gml:coordinates> </gml:Point>
    </geometry>
  </F_Station>
</gml:FeatureMember>

<gml:FeatureMember>
  <F_Station>
    <name>Arruda</name>
    <city> <name>SOBRAL</name>
           <area>19820</area> </city>
    <pluviometry> <month>3</month>
       <value>150</value> </pluviometry>
    <geometry>
      <gml:Point>
        <gml:coordinates cs="," decimal="." ts="">
          -3.85,-40.66</gml:coordinates> </gml:Point>
    </geometry>
  </F_Station>
</gml:FeatureMember>
```

Figure 4.5 – XML fragment resulting from Q_{W1}

In our approach, the extension of a feature type is virtual, computed by an SQL/XML query (see Definition 3.3). Therefore, Q_W should be translated to an SQL/XML query defined over the database schema as follows.

Definition 4.3: Let Q_W be a WFS Query over feature type F, and Q_x be the canonical XQuery for Q_W. Let Q_S be a SQL/XML query over S which returns a set of *<gml:featureMember>* elements. We say that Q_S *is a correct translation for* Q_W iff for any instance σ_s of S if σ_F is the extension of F on σ_s, $\mathcal{S}_1$ is the set of *<gml:featureMember>* elements resulting from evaluating Q_S on σ_s, and $\mathcal{S}_2$ is the set of *<gml:featureMember>* elements resulting from evaluating Q_X on σ_F, then $\mathcal{S}_1 = \mathcal{S}_2$. □

```
SELECT XMLELEMENT("gml:FeatureMember",
      XMLELEMENT("Station",
            XMLFOREST(S.NAME AS "name"),  -------------------------------------- ( TranslatePath(name) )
            XMLELEMENT( "city",                                        ( TranslatePath(address/ city) )
                  (SELECT XMLCONCAT(
                        XMLFOREST(C.NAME AS "name"),
                        XMLFOREST(C.AREA AS "area") )
                   FROM City_rel C WHERE C.CODECITY = S.CODECITY)),
             (SELECT XMLAGG(XMLELEMENT("pluviometry",                ( TranslatePath(pluviometry) )
                          XMLFOREST(PL.MONTH AS "month"),
                          XMLFOREST(PL.VALUE AS "value") ) )
              FROM Pluviometry_rel PL
              WHERE S.CODE = PL.CODESTATION),
            XMLFOREST(                                                  ( TranslatePath(geometry) )
                  SDO_UTIL.TO_GMLGEOMETRY(S.GEOM_POINT)
            AS "geometry") ) )
FROM Station_rel S, Agency_rel A
WHERE  S.CODEAGENCY = A.CODEAGENCY AND A.NAME = 'FUNCEME' AND
           mdsys.sdo_relate( S.GEOM_POINT, mdsys.sdo_geometry(2003, NULL, NULL,
                                                   mdsys.sdo_elem_info_array(1, 1003, 3),
                                                   mdsys.sdo_ordinate_array(-5.2, -42.5, –2.5, -38.7)),
                             'mask=ANYINTERACT querytype=WINDOW') = 'TRUE';
```

Figure 4.6 – SQL/XML Query Q_{S1}

Consider, for example, the WFS query Q_{W1} shown in Figure 4.1. The canonical XQuery for Q_{X1} is shown in Figure 4.4. The result of Q_{W1} is defined by the result of evaluating Q_{X1}. Suppose $\sigma_{DB\_Station}$, the instance of DB_Station shown in Figure 3.7, and $\sigma_{F\_Station}$ the corresponding extension of F_Station shown in Figure 3.8. Evaluating Q_{X1} on $\sigma_{F\_Station}$ we obtain the XML fragment shown in Figure 4.5, which is the result for Q_{W1}. The same result can be obtained by evaluating the query Q_{S1} in Figure 4.6 over $\sigma_{DB\_Station}$, as Q_{S1} is a correct translation for Q_{w1}.

The Algorithm TranslateWFSQuery shown in Figure 4.7 receives as input a WFS query Q_W and generates the SQL/XML query Q_S such that Q_S is a correct translation for Q_w. The Algorithm uses the functions TranslatePath and TranslateFilter defined in following.

Definition 4.4: Let $\delta_F = p_1 /.../ p_n$ be a path of T. TranslatePath(δ_F) returns an SQL/XML sub-query $\mathcal{Q}$, that computes the value of path δ_F. More formally, for any instance \$t of T if \$t $\equiv_{\mathcal{A}}$ r, where r is a tuple of R then $\mathcal{Q}$(r) returns a set $\mathcal{S}$ of <p_n> elements where $\mathcal{S}$ = \$t/ $p_1 /.../ p_n$. Note that $\mathcal{Q}$ has a reference for a tuple r of R. □

Figure 4.6 shows the SQL/XML sub-query that computes the value of each path expression of the WFS query Q_{W1} shown in Figure 4.1. Note that each sub-query references a tuple s of Station_rel.

```
Input: WFS query QW.
Output: SQL/XML query QS

Let <P, L> := TraslateFilter(f), where f is the filter of QW;
In case of:
1. If QW has no <wfs:PropertyName> elements then
   Qs =  "SELECT XMLELEMENT( \" gml:featureMember \",
                    XMLELEMENT( "F", τ[R→T](r) )
          ") ) FROM R r, L WHERE P";
2. If QW has n (n>0) <wfs:PropertyName> elements then
   Let Prop[ i ] := TranslatePath(Pathi), where Pathi is the value of i-th <wfs:PropertyName> element, 1≤i≤n;
   Qs =  "SELECT XMLELEMENT( \" gml:featureMember \",
              XMLELEMENT(\" F \", "
                  + Prop[ 1 ] + ", " + Prop[ 2 ] + ", " ... + Prop[ n ] +
          ") ) FROM R r, " + L + WHERE P";
End case
Return Qs;
```

Figure 4.7 – Algorithm TranslateWFSQuery

In our approach, we can generate, at feature type definition time, TranslatePath(δ_F) for each path δ_F of T. TranslatePath(δ_F) is automatically generated based on the CA of the proprieties in δ_F as follows: Let δ_F = p_1 /.../ p_n be a path of T where [T/ p_1] ≡ [*R*/ δ_1], [Tp_i/ p_{i+1}] ≡ [*R*p_i/ δ_{i+1}], are the CA of p_i in $\mathcal{A}$, for 1≤i≤n-2 and [Tp_{n-1}/ p_n] ≡ [*R*p_i/ *exp*] is the CA for p_n in $\mathcal{A}$ (δ_i can be null). Let $\mathcal{Q}$ = GenSQL/XML(p_n, δ_1. .... .δ_{n-1}.*exp*, r) (see Function GenSQL/XML in Figure 3.2). Then, TranslatePath (δ_F) = $\mathcal{Q}$. Theorem 4.1 below shows that $\mathcal{Q}$ is a correct translation for δ_F (satisfies Definition 4.4).

Theorem 4.1: Let δ_F be a path of T as in Definition 4.4, and let $\mathcal{Q}$ = GenSQL/XML(p_n, δ_1. ... .δ_{n-1}.*exp*, r). Let r be a tuple of R and \$t be an instance of T such that \$t $\equiv_{\mathcal{A}}$ r. Let $\mathcal{S}$ be the set of <p_n> elements returned from $\mathcal{Q}$(r). So we have that $\mathcal{S}$ = \$t/ p_1 /.../ p_n. □

Proof: From Algorithm GenSQL/XML in Figure 3.2, we have that $\mathcal{S} \equiv_{\mathcal{A}}$ r/δ_1. ... .δ_{n-1}.*exp*. Since \$t $\equiv_{\mathcal{A}}$ r, from definition 2.8 and the CA of p_i in $\mathcal{A}$, 1≤i≤n-2, we can show that \$t/ p_1 /.../ p_n $\equiv_{\mathcal{A}}$ r /δ_1. ... .δ_{n-1}.*exp*. Therefore, $\mathcal{S}$ = \$t/ p_1 /.../ p_n .

Definition 4.5: Let *f* be the filter element of a WFS Query. TranslateFilter(*f*) returns a tuple <$\mathcal{P}$,$\mathcal{L}$>, where $\mathcal{P}$ is an SQL conditional (boolean) expression and $\mathcal{L}$ is a list of relations names required to process the conditions in $\mathcal{P}$, such that for any instance \$t of T if \$t $\equiv_{\mathcal{A}}$ r, where r is a tuple of R then \$t satisfies *f* iff r satisfies $\mathcal{P}$. □

Due to space limitation, the function TranslateFilter is not discussed here. The semantics of this function is well specified in [7,10].

Theorem 4.2 below shows that Algorithm TranslateWFSQuery in Figure 4.7 correctly translates a WFS query.

Theorem 4.2. Let Q_W be a WFS Query over feature type F, and Q_x be the canonical XQuery for Q_W. Suppose p_i is the property referenced by $Path_i$ on Q_W, $1 \le i \le n$. Let Q_S be a SQL/XML query over S generated by the algorithm TranslateWFSQuery. Let σ_s be an instance of S, and σ_F be the extension of F on σ_S. Let $\mathcal{S}_2$ be the set of <*gml:featureMember*> elements resulting from evaluating Q_S on σ_s, and $\mathcal{S}_1$ be the set of <*gml:featureMember*> elements resulting from evaluating Q_X on σ_F. So, $\mathcal{S}_1 = \mathcal{S}_2$

Proof: Suppose: (i) Q_W has n (n>0) <*wfs:PropertyName*> elements, where $Path_i$ is the value of i-th <*wfs:PropertyName*> element, $1 \le i \le n$; (ii) p_i is the property referenced by $Path_i$, $1 \le i \le n$; (iii) $\$f_1 \in \mathcal{S}_1$.

($\rightarrow$) We first prove that $\mathcal{S}_1 \supset \mathcal{S}_2$.

(1) From Q_X we have that $\$f_1$ has properties p_1, ..., p_n and exists $\$f \in \sigma_F$, where $\$f$ satisfy the filter condition f of Q_W, and $\$f_1/p_i = \$f/path_i$, $1 \le i \le n$.

(2) From Definition 3.2 and Definition 4.4, exists $r \in R$ such that $r \equiv_{\mathcal{A}} \$f$.

(3) From Definition 4.5, r satisfy $\mathcal{P}$ where $<\mathcal{P}, \mathcal{L}>$ = TranslateFilter(f).

(4) From (2) and (3) and Q_S, we have that exists $\$f_2 \in \mathcal{S}_2$, where $\$f_2$ has properties p_1, ..., p_n and $\$f_2/p_i$ = TranslatePath($Path_i$)(r).

(5) From (2) and Definition 4.3, TranslatePath($Path_i$)(r)= $\$f/Path_i$, for $1 \le i \le n$.

From (1), (4) and (5), we have that $\$f_2/p_i = \$f_1/p_i$, $1 \le i \le n$. Thus, $\$f_2 = \$f_1$ and therefore $\mathcal{S}_1 \supset \mathcal{S}_2$.

($\leftarrow$) The proof that $\mathcal{S}_2 \supset \mathcal{S}_1$ is similar to the above.

The proof for the case where Q_W has no <*wfs:PropertyName*> elements follows from Theorem 3.1. ■

5. Conclusions

We argued in this paper that we may fully specify a feature type in terms of the relational database by using correspondence assertions, in the sense that the assertions define a mapping from tuples of the relational schema to instances of the feature type. We defined the semantics of WFS query answering, and presented an algorithm that translate, based on the feature type's correspondence assertions, a WFS query defined over a feature type schema into a SQL/XML query defined over the relational database. Moreover, we showed that the TranslateWFSQuery Algorithm correctly translates a WFS query.

We are currently working on the development of GML Publisher [17], a framework for publishing geographic data stored in relational database as GML. The publication of a feature type in GML Publisher consists of three steps: (1) The user defines the XML schema of feature type instance. (2) The correspondence assertions of the feature type are generated by matching the feature type XML schema and the relational database schema. (3) Based on the feature type correspondence assertions, GML Publisher generates the SQL/XML query that computes the extension of the feature type, and the SQL/XML sub-query that computes the value of each feature type path expression.

Acknowledgments

This work is financed by CNPq and project *GML Publisher: Um Framework para Publicação de Feições Geográficas como GML* (process no. 478571/2004-6).

References

1. Deegree. http://deegree.sourceforge.net (visited on September 26th, 2005).
2. Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E. and Zemke, F. (2004) *SQL:2003 has been published*. In: SIGMOD Record, v.33, p.119-126.
3. Fernández, M., Kadiyska, Y., Suciu, D., Morishima, A., and Tan, W. C. (2002) *SilkRoute: A framework for publishing relational data in XML*. In: TODS, v.27, n.4, p.438–493.
4. Hernández, M. A., Miller, R. J., Haas, L. M. (2001) *Clio: A Semi-Automatic Tool For Schema Mapping*. In: SIGMOD Conference.
5. Krishnamurthy, R., Kaushik, R., and Naughton, J. F. (2003) *XML-SQL Query Translation Literature: The State of the Art and Open Problems*. In: XSym.
6. Madhavan, J., Bernstein, P., Rahm, E. (2001) *Generic Schema Matching with Cupid*. In: 27th Proceedings of the International Conference on Very Large Databases (VLDB), p.49–58.
7. OpenGIS Consortium. Filter Encoding Implementation Specification: Version 1.1. http://www.opengis.org (visited on September 26th, 2005).
8. OpenGIS Consortium. http://www.opengis.org (visited on September 26th, 2005).
9. OpenGIS Consortium. *Schema for Geography Markup Language (GML)*. Version 2.0. http://www.opengis.org (visited on September 26th, 2005).
10. OpenGIS Consortium. *Simple Features Implementation Specification for SQL*. Version 1.1. http://www.opengis.org (visited on September 26th, 2005).
11. OpenGIS Consortium. *Web Feature Service (WFS) Implementation Specification*. Version 1.0. http://www.opengis.org (visited on September 26th, 2005).
12. Oracle Corporation. Disponível em: http://technet.oracle.com. (Acessado em 24 de abril 2005).
13. Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A., Fagin, R. (2002) *Translating Web Data*. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), p.598–609.

14. Rahm, E., Bernstein, P.A. (2001) *A Survey of Approaches to Automatic Schema Matching*. In: VLDB Journal, v.10, n.4, p.334–350.

15. Shanmugasundaram, J., Kiernan, J., Shekita, E., Fan, C., and Funderburk, J. (2001) *Querying XML views of relational data*. In VLDB.

16. SQL/XML. http://www.sqlx.org/ (visited on September 26th, 2005).

17. Vidal, V. M. P., Feitosa, F. B. (2004) *GML Publisher: Um Framework para Publicação de Feições Geográficas como GML*. In: III Workshop of Theses and Dissertations on Databases, 19th Brazilian Symposium on Databases. Brasília, Brazil.

18. Vidal, V.M.P., Boas, R. V. (2002) *A Top-Down Approach for XML Schema Matching*. In: Proceedings of the 17th Brazilian Symposium on Databases. Gramado, Brazil.

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

20. World-Wide Web Consortium. *XML Schema Part 0*: Primer Second Edition. http://www.w3.org/TR/xmlschema-0/ (visited on September 26th, 2005).

21. World-Wide Web Consortium. *XQuery: An XML Query Language*. Version 1.0. http://www.w3.org/TR/xquery/ (visited on September 26th, 2005).

22. World-Wide Web Consortium: *XML Path Language (XPath)*. Version 1.0. http://www.w3.org/TR/xpath (visited on September 26th, 2005).

23. Yu, C., Popa, L. (2004) Constraint-Based XML Query Rewriting For Data Integration. In: SIGMOD, p.371–382.