A Spatio-Temporal Query Language for a data model based on XML.

J. E. Córcoles¹, J. García-Consuegra², J. Peralta³ and E. Navarro²
Universidad de Castilla-La Mancha
¹Sección de Tecnología de la Información . Instituto de Desarrollo Regional,
²Departamento de Informática . Escuela Politécnica Superior de Albacete (EPSA).
³Sección de Teledetección y SIG. Instituto de Desarrollo Regional.

Abstract
The recent emergence of eXtensible Markup Language (XML) as a new standard for data representation and exchange on the World-Wide Web has drawn considerable attention in the Geographic Information Systems (GIS) world. OCG (OpenGIS Consortium) is contributing with a XML specification to the representation of geographic information (called GML - Geographical Markup Language). GML allows the exchange of geographical information in the Web. In addition, using XML to represent this type of information can benefit the interoperability between different architectures of GIS. In this case, we need a query language to retrieve and analyse information from this data structure.

In this paper, a query language for a spatio-temporal model based on XML is shown (STQ). The model uses a subset of the spatio-temporal model called STML that has been developed in our research group. The query language has a familiar select-from-where syntax and is based on SQL (Structured Query Language). STQ includes a set of spatial operators (overlap, cross, etc), and set of temporal operators (previous, next, etc). In addition, STQ includes traditional operators (=, >, <,...) for non-spatial information. The result of a query is another XML document. Since integration of XML in Web environments is easily achieved, STQ may be easily used to query geographical information on the Web.

Introduction
XML [Bray, T. et al] is the metalanguage defined by the World Wide Web Consortium (W3C) that can be used to describe a broad range of hierarchical mark up languages. It is a set of rules, guidelines and conventions for describing structured data in a plain text. Using a text format instead of a binary format allows the programmer or end user to look at or utilize the data without relying on the program that produced it. However, the primary producer and consumer of XML data is the computer program and not the end-users.

Like HTML, XML makes use of tags and attributes. Tags are words bracketed by the ‘<’ and ‘>’ characters and attributes are strings of the form "name="value"" that are inside of tags. While HTML specifies what each tag and attribute means, as well as their presentation attributes in a browser, XML uses tags only to delimit pieces of data and leaves the interpretations of the data to the application using it. In other words, XML define only the structure of the document and does not define any of the presentation semantic of that document.

XML-based documents can be used in a wide variety of applications including vertical markets, e-commerce, business-to-business communications, and enterprise applications messaging. An interesting application of XML language in GIS is the use such exchange language of geographic information. In the last few years, a large group of different file formats has appeared on the market. File format like the shapefile by Esri (.shp) are well-known in the world of Geographic Information System. However, this great diversity of file
formats makes the exchange of information between different commercial applications very difficult. The solution to the problem of these heterogeneous systems is the use of standard languages such as XML to represent geographic information with a unique format.

Nowadays, there are not very many data models based on XML. In March of 2000, OpenGIS have proposed a format to represent geographical information with XML. This format is called GML (Geography Markup Language) [OpenGIS]. This Language is a XML encoded for the transport and storage of geographic information, including both the geometry and properties of geographic features. The mechanisms and syntax that GML uses to encode geographic information in XML are defined in the specification of OpenGIS. Furthermore, some new extensions have been added to the GML model, such as temporal operators.

The conversion of different formats to a unique XML format is very important in the exchange of information on the Web. Nowadays, Web applications are on the increase and users need formats to exchange information as well as language to query this information on Web applications. The Internet browser has become the main tool for users. For this reason, users who wish to make their queries in the browser can see the results in the same browser. Middleware applications to convert different formats are not sought by users. XML and Query Language [Maier, D] over XML documents permit this.

In this paper, a query language for a spatio-temporal model based on XML is shown. This query language is called STQ (Spatio-Temporal Query language). It allows us to obtain information stored in XML format and distributed in several XML documents. The query language has a familiar select-from-where syntax and is based on SQL (Structured Query Language). STQ includes a set of spatial operators (overlap, cross, etc), and set of temporal operators (before, after, etc). In addition, STQ includes traditional operators (=, >, <,...) for non-spatial information. In the first version the query language shown in this paper is applied over information stored in GML format.

To understand the main innovative features in this work, the state of the art in query languages over XML is shown in the next section. The syntax of this language is described in section 3. Important features and differences with other query languages are also shown. The query process and one example of execution are shown in the Section 4. Section 5 shows conclusions and feature works.

**Query Languages over XML**

The first aim of this section is to show the “state of the art” of different query languages over XML to obtain the main features of each query language. The purpose of this comparison is to show the main innovations offered by STQ.

To obtain a common pattern in the comparative, the definition of a group of features that all query languages over XML must support is necessary. These features has been obtained from [Bonifati,A. and Ceri, S.,] and [Quass, D.]. In addition, the possibility of a language supporting spatial and/or temporal operators is defined. This feature is important because it is only supported by STQ language.
Necessary Features in Query Languages over XML

Clean Semantics: Any language for XML must be able to express simple queries in a simple way. A good XML query language should be usable by novice web-users, not just database experts. One possibility for semantic clarity is to base the query language upon the well-known select-from-where statement of SQL.

Path Expressions: Since XML elements are stored hierarchically, the query languages should support “path expressions”, which allow the writer easy access to nested elements.

Ability to return an XML document: The standard behaviour of most query languages is to return a set of elements records. For an XML language the returned value should be an XML document. One possibility for returning an XML document is to allow the query return value to be embedded within XML markup tags.

Ability to query and return XML tags and attributes: An XML data element contains data, tags and, optionally, attributes. It is imperative that an XML query language be able to query the element tags and attributes as well as the data. Tags and attributes should be accessible for referencing in any part of the query (the Select, From and Where clause).

Intelligence type coercion: Since both textual and numeric data are represented as strings in XML, the query languages should be intelligent enough in comparison operations to determine whether a string comparison is intended or if coercion is required.

Handles unexpected data: DTD is a flexible structure. Therefore, it is critical that an XML query language “do the right things” as far as possible in the face of unexpected data. For example, as in the case of a query writer expecting only a single occurrence of the phone element for each employee, when employee elements have multiple phone elements.

Ability to allow queries when the DTD is not fully known: It may often be the case with XML that the query writer understands a part of the DTD [Bray, T. et al], but not the whole DTD. The query language needs to support wildcards in the expressions to allow the query writer to “skip past” parts of the document structure of which he or she is not aware.

Returns unnamed attributes: In restructuring information it is often useful to express things like “return all child elements”, or “return all child elements except this one”. An XML query language should support queries that return elements even when their tags are unknown.

Preserves Order: SGML documents have an implicit order, as do XML documents. It is important for an XML query language to be able to optionally guarantee that the order of returned results is the same as in the original document. An extension to the order by clause could be used for this purpose.

Allows Spatial and Temporal operators: The existence of spatial and temporal operators is necessary in the current information systems. Most database management systems implement these operators in their versions.

Language Comparison

Three well-known languages upon XML and the features that have been implemented in its specifications are shown in the Table 1. These languages are XQL (XML Query Language) [Robie., J.], XML-QL [Deutsch, A., et al] and Lorel [Abiteboul, et al.]. Notice that there is no language that supports both spatial and temporal operators. That is the innovation of the query language upon XML shown in this paper. Spatial and temporal operators for geographic information are implemented. In this first version information must be represented in a specific format (GML). However, in later versions this language will support other formats in XML.

<table>
<thead>
<tr>
<th>Feature</th>
<th>XQL</th>
<th>XML-QL</th>
<th>LOREL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean Semantics</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Feature</td>
<td>STQ</td>
<td>SQL</td>
<td>OQL</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Path Expressions</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Ability to return an XML document</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Ability to query a return XML tags and attributes</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Intelligence type coercion</td>
<td>Partial</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Handles unexpected data</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Allows queries when the DTD is not fully known</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Returns unnamed attributes</td>
<td>YES</td>
<td>YES</td>
<td>-</td>
</tr>
<tr>
<td>Preserves Order</td>
<td>Partial</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Allows Spatial and Temporal operators</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Table 1: Comparative between languages over XML

Syntax of STQ

In this section, the syntax and semantics of the query language STQ is described. The syntax of STQ is based on SQL. The Select-From-Where statement is widespread in query languages since it allows for a quick learning of the language. It is necessary to highlight that in this language it is not intended to do the implementation of these features of SQL. Only important features to run simple and powerful queries are developed. On the other hand, many features of the query languages upon XML are not considered in traditional query languages (SQL, OQL, etc.), although they are necessary in this kind of language (i.e. path expressions).

A study of the features of query languages in XML is applied below to STQ. These features are explained in the previous section. Some features are developed in STQ completely, others partially and others are not yet developed. To show an example of the STQ syntax, one DTD in GML is used (appendix 1). Simple information about a city is represent by this DTD. In this model a city is made up of alphanumeric attributes such as Name, Description, etc, and Cadastral Parcels. The Cadastral Parcels have alphanumeric attributes such as CadastreId, street, number, etc and Extendof which is the polygon representing a Parcel.

Clean Semantics: All languages based in select-from-where statements allow powerful and well-structured queries to be written. For example, to obtain the Identity Cadastral (<cadastreId>) of all Cadastral Parcels of Albacete city, the next query may be used.

```
Select C.modelMember.Parcel.cadastreId
From [http://www.uclm.es//prove.xml].CityModel as C
Where C.name Like 'Albacete'
```

Path Expressions: The implementation of path expressions is mandatory in query languages over XML. To represent the path expressions in STQ the dot notation is used. This notation is implemented in the most Oriented objects programming languages. In these languages, the dot notation is used to access methods and attributes of an object. One example of path expression is showed in the last query (Block.cadastreId).

```
Select C.modelMember.Parcel.extendOf
From [http://www.uclm.es//prove.xml].CityModel as C
Where C.name Like 'Albacete'
```

Ability to return an XML document: For compatibility with other tools, the result set of a STQ query is another XML document. The structure of the new document is defined by the tags and attributes that have been selected in the Select clause. For example, if the user wants to obtain the Cadastral Parcel (<extendOf>) of Albacete city (<CityModel><Name>) with the number (<parcel><number>) equal to 103 then he or she should make the following query:

```
Select C. ModelMemeber.Parcel.extendOf
From [http://www.uclm.es//prove.xml].CityModel as C
Where C.name Like 'Albacete' and Parcel.number = 103
```
The XML document return for the query is:

```xml
<extendOf>
  <Polygon>
    <outerBoundaryIs>
      <LinearRing>
        <coordinates>
          0.0,0.0 40.0,0.0 40.0,10.0 0.0,10.0
        </coordinates>
      </LinearRing>
    </outerBoundaryIs>
  </Polygon>
</extendOf>
```

**Ability to query and return XML tags and attributes:** With this feature a high level of flexibility between the users and the information structure is achieved. STQ allows some attributes and tags to be partially defined in the query. In STQ the symbol ‘%’ is used as a wildcard in the partial definitions. How STQ allows for this feature is shown in the next example: Find all Cadastral Parcels that are in a street with the name “Capitán Grant”. Since the user does not know the exact name of the tag `street`, the query may be

```sql
Select M.Parcel.extendOf
From [http://www.uclm.es/probe.xml].CityMode.modelMember as M
Where M.(stree%) Like 'Capitan Grant'
```

This query finds all Cadastral Parcels whose child’s tag name begins with the string ‘stree’. The depth of the tag ‘stree%’ may be any in `CityMode.modelMemberl as M`.

**Intelligence type coercion:** The automatic coercion between types is not yet supported by STQ. However, some automatic casts are possible like (String-Integer).

**Handles unexpected data:** To implement this feature, in STQ the operator ‘Like’ and ‘=’ are handled in the same way as an existential quantifier. Thus, if a tag has several values then STQ selects the first to carry out the condition.

**Ability to allow queries when the DTD is not fully known:** In order to make it unnecessary for the user to know the DTD of a document, in STQ it is possible to use tags and attributes that may be used as optional extras. The syntax of STQ shows the optional tags and attributes in brackets each one of them separated by the symbol ‘|’. This symbol ‘|’ means “optimality between deferent tags or attributes. In this way, the user indicates that he or she wants to find values stored in elements with a different syntax but with the same semantic. For example, to return all cities that have in the name or description the string ‘Alba’. The following query may be used.

```sql
Select C
From [http://www.uclm.es/probe.xml].CityModel as C
Where C.(name | description) Like 'Albacete'
```

**Returns unnamed attributes:** STQ allows us to establish in the Select clause whether it returns only tags or tags with all these children. To achieve this the symbol ‘^’ is used. This symbol means that a determinate tag or attribute is not included in the result set. For example, if the
user wants to return all elements of a city except the temporal features (<modelDate>) and (<modelMember>) spatial features, the following query should be written:

\[
\text{Select C.(} ^* \text{modelDate}, ^* \text{modelMember)} \\
\text{From [http://www.uclm.es/prove.xml].CityModel as C}
\]

**Preserves Order:** In the current version of STQ data order is not preserved. Furthermore, the order in which each piece of data is stored is not respected. Thus, the result set return for a query has an unpredictable order.

**Allows Spatial and Temporal operators:** A broad set of topological operators has been implemented in STQ. Most of them are defined by the 9-Intersection model [Egenhofer, M., Clementini. E., et al.] [Egenhofer. M. and Franzosa. R]. According to this model, each object \( p \) is represented in \( \mathbb{R}^2 \) space as a point set which has an interior, a boundary and an exterior. The topological relation between any two objects \( p \) and \( q \) is described by the nine intersections of \( p \)'s interior, boundary and exterior, with the interior, boundary and exterior of \( q \) (based on the concepts of point-set topology). Out of 512 different relations that can be distinguished by the model, only the following eight are meaningful for polygon objects: disjoint, meet, equal, overlap, contains (and the converse relation inside) and covers (and the converse covered_by). These operators are applied over polygons, but some of them, can be applied over polylines too. On the other hand, there are other operators that define the topological relations between polygons and polylines. Some of these operators that have been included in STQ specifications are: within, near, crossing, along, finish. Operators like (crossing, finish, near, equal) have the same semantics if they are only applied over polylines.

Other spatial operators have been included in STQ specification. These operators are: Length, surface, perimeter and buffering. The function of Length, surface, and perimeter is obvious, while buffering is defined as a buffering zone around an object.

Besides spatial operators, temporal operators have been included in SQT (Date Is, After and Before).

If a date is provided in the query (i.e. Date is 26/5/2000) some checks should be done:

\[
\begin{align*}
\text{if date in [vt] then} & \\
\text{read record} & \\
\text{elseif date in [tt] then} & \\
\text{the entity/attribute/property does not exist or is not true} & \\
\text{else} & \\
\text{there is no available data in the system for that date} & \\
\text{end}
\end{align*}
\]

Where [tt] stands for the transaction time range and [vt] stands for valid time range. The semantics of these time dimensions are well documented ([Pavlopoulos et al] [Tryfona N et al]).

The **From** clause of a query always has the same structure. A piece of grammar is shown below.

\[
\text{select_statement ::= .} \\
\text{.} \\
\text{FROM table_commalist: f1}
\]
All the elements included in the From clause must be preceded by the URL address (addresshttp) of the document in which these elements are stored. This option allows various XML documents to be used in the same query.

**Query Processing.**

The query process for a STQ query has two well-defined phases. The first phase is a query decomposition to transform the STQ query in an Operator Tree. This tree is the graphical representation of a relational algebra query. The second phase is to run the algebra relational operator in the order established in the tree. This section is divided into two sub-phases: the execution (select in relational algebra) of alphanumeric operators and the execution of spatial operators. The first subsection is made using XQL, and the second sub-phase using a spatial structure called R-tree. Each phase is described below.

**Query Decomposition**

A query in STQ is a relational calculus query. The query decomposition is the first phase of a query processing that transforms a relational calculus query into relational algebra query. This phase is carried out after the lexical and syntactical correction of STQ query has been done. When this phase is successfully completed the output query is semantically correct and good in the sense of that redundant work is avoided. The successive steps of query decomposition are: (1) normalization, (2) analysis, (3) redundancy elimination and (4) rewriting [Tamer Özsu, M et al.].

1. Normalization: The input query may be arbitrarily complex, depending on the facilities provided by the language. The normalization process transforms the query into a normalized form to facilitate further processing. With this kind of language, the most important transformation is that of a query qualification (Where clause), which may be an arbitrarily complex, quantifier-free predicate, preceded by all necessary quantifiers (∀ or ∃). There are two possible normal forms for the predicates, one giving precedence to the AND (\(\land\)) and the other to the OR (\(\lor\)). In SQT the *conjunctive normal* (conjunction of disjunctions) form is used being more practical since query qualifications typically include more AND than OR predicates.

2. Analysis: Query analysis enables rejection of normalized queries for which further processing is either impossible or unnecessary. The main reason for a rejection is that the query is semantically incorrect. When this case is detected, the query is simply returned to the user with an explanation. Otherwise, query processing is continued. The technique used to detect semantic mistakes is called a *query graph* or *connection graph* [Ullman, J. D.]. The query graph is useful to determine the semantic correctness of a conjunctive multivariable query without negation. Such a query is semantically incorrect if its query graph is not connected.

3. The normalization process may contain redundant predicates. A naive evaluation of a qualification with redundancy can well lead to duplicated work. Such redundancy, and thus redundant work, may be eliminated by simplifying the qualification with well-known idempotency rules as (p \(\land\) p \(\iff\) p, p \(\lor\) p \(\iff\) p, etc.).
4. The last step of query decomposition rewrites the query in relational algebra. This is typically carried out in the two following steps: Straightforward transformation of the query from relational calculus into relational algebra, and restructuring of the relational algebra query to improve performance. For the sake of clarity the relational algebra query is represented graphically by an operator tree. An operator tree is a tree in which a leaf node is an element of a XML document, and a non-leaf node is an intermediate relation produced by a relational algebra operators. The sequence of operations is directed from the leaves to the root, which represents the result query. By applying transformation rules, many different trees may be found equivalent to the one produced by a traditional method [Smith J. M. and Chang P. Y.]. These rules allow for an optimal process query, which is the main target.

Operator Execution

When the operator tree is generated, the next step is the execution of the relational operators that make up the tree. Two kinds of processes may be distinguished. The first process is to execute alphanumeric operators like ( name = ‘Capitan’ or length > 3). These sentences are executed with XQL. This process has facilities that could serve as a basis for an XML query language. XQL is a notation for selecting and filtering the elements and text of XML documents. XQL can be considered a natural extension to the XSL pattern syntax; it is designed with the goal of being syntactically very simple and compact, with a reduced expressive capacity.

To execute the spatial query (spatial operators) another method different from XQL is used. This method is based on the use of a spatial index such as R-Tree [Guttman. A.] [Papadias, D. et al]. It is a spatial data structure based on MBRs (Minimal Boundary Box) [Samet, H]. The R-Tree is a height-balanced tree which consists of intermediate and leaf nodes (R-trees are direct extensions of B-Trees in k-dimensions). The MBR of the actual objects is assumed to be stored in the leaf nodes of the tree. Intermediate nodes are built by grouping rectangles at the lower level and an intermediate node is associated with some rectangle which encloses all rectangles that correspond to lower level nodes.

The Rtree is filled in each query, with the information indicated in the From clause. When the tree of an XML document is generated (using the XML parser) [DOM], the Rtree is generated at the same time. The MBR of each spatial element may be obtained from a tag <boundedby> or it may be generated starting from the spatial object stored in the <extendOf> tag. Each element of Rtree has an identifier with the node of the XML tree in which the spatial coordinates are stored. In this way, there is a direct relation between each node of the Rtree and each node in the XML tree that represents spatial coordinates.

Example of Query Processing

In this section, we show how the previous phases are combined to execute one query over XML is described. The example query intends to obtain all Parcel Identifiers (<modelMember><cadastreId>) that have a number (<modelMember><number>) greater than 100, and whose surface is contained in the zone with name ‘Carretas’.

The SQT query that represents the last query may be:

```
Select P.Cadastreid 
From [http://www.uclm.es//prove.xml].CityModel.modelMember.Parcel as P, 
[http://www.uclm.es//prove.xml].CityModel.modelMember.District as D
```
Where \( P\.number > 100 \)

and \( D\.name \) like ‘Carretas’

and \( P\.extendOf\.Contain.(D\.extendOf) \)

In the \textit{Where} clause there are two alphanumeric operators (>, Like) and one spatial operator (\textit{Contains}). The joint of these elements is represented in the spatial operator \textit{Contains}. After the decomposition process, this query generates the operator tree shown in figure 1.

![Figure 1: simplified operator tree](image)

The sequence of operations is directed from the leaves to the root. For this reason the first step is to execute the alphanumeric operators (\( D\.name \) like ‘Carretas’ and \( P\.number > 100 \)). The order of execution of these operators is not relevant because it is necessary for these operators to be executed upon the total information. The execution of these operators is applied by the XQL command. In this example, a direct conversion from STQ to XQL may be applied, but in other queries higher level operators are necessary. This is so, because there are some STQ facilities that may not be represented in XQL, for example the use of the wildcard in tags or attributes. As shown above, the same abstract level is applied to execute temporal operators.

Once the result set of these operators is obtained, the spatial operator is applied. To achieve this two phases are necessary. In the first phase all spatial elements (\( P\.extendOf \)) contained in each MBR of the operator parameter (\( D\.extendOf \)) are obtained. To this end, the Rtree is queried with operators such as contains, overlap, etc. These operators search for a spatial structure whose MBR is contained in or overlaps another MBR. The result set obtained by the Rtree is a set of identifiers of the nodes in the XML document that have their MBR contained in or overlapping the parameter MBR. This process is repeated to all elements included in the operator parameter. In the second phase, the operators have to apply over elements returned by the first phase. This is a difficult task, even more so if the elements have several coordinates. However, when this phase is tried there is a minimum number of affected elements.

Once all spatial operators are applied, the result set is the totality of elements that have carried out all the conditions. The last step is to make a projection to obtain the elements indicated in the select clause. The result is:
Conclusions and Future work

In this paper, a XML query language called STQ is shown. The main feature of this language is that it includes spatial and temporal operators in its specifications. This feature is not included in the most widely-known XML query languages. To carry out queries with spatial and temporal operators, the existence of the XML document that stores spatial and temporal data models is needed (one example of a model based on XML is GML by OpenGML). In order to make these languages more standard, all tests of this language are applied over a GML data model.

On the other hand, STQ includes in its specifications many features that are obligatory in this kind of languages. However STQ does not provide the same flexibility and power as other more tried languages (Lorel, XQL, etc). All the studies on STQ have been analyzed to carry out spatial and temporal queries efficiently since this kind of language is of great importance in the immediate future of GIS.

In future work, there are some features that need to be included. For instance, features such as coercion types that include spatial types or order by tags. Furthermore, similar features to the Dataguides of Lorel, will be included to make a single user interface that the user may make use of Web queries. This language could well be the basic element to create a global search engine of spatial information on the Web.

References

Egenhofer. M., Clementini. E. and di Felice. P., Topological Relations between Regions with
Tamer Özsu, M., Valduriez, P., Principles of distributed database systems. Prentice Hall. 1999
Appendix 1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT CityModel (
   description?, name?, boundedBy,
   modelDate,
   modelMember*) >
<!ELEMENT modelDate (validTime, transactionTime) >
<!ELEMENT validTime (#PCDATA) >
<!ELEMENT transactionTime (#PCDATA) >
<!ELEMENT modelMember ( Parcel | District ) >
<!ELEMENT Parcel (cadastreId, number?, street?, boundedBy?
   modelDate, extentOf, location) >
<!ELEMENT District (districtId, districtName, boundedBy?
   modelDate, extentOf) >
<!ELEMENT cadastreId (#PCDATA) >
<!ELEMENT street (#PCDATA) >
<!ELEMENT number (#PCDATA) >
<!ELEMENT districtId (#PCDATA) >
<!ELEMENT districtName (#PCDATA) >

<CityModel>
   <description>
      Digital model of Albacete City
   </description>
   <boundedBy>
      <Box srsName="EPSG:4326">
         <coordinates>
            <coordinates>
            </coordinates>
         </coordinates>
      </Box>
   </boundedBy>
   <modelDate>
      <validTime>
         1/1/1995..31/12/1997
      </validTime>
      <transactionTime>
         1/1/1995..31/12/1999
      </transactionTime>
   </modelDate>
   <modelMember>
      <Parcel>
         <cadastreId>3050</cadastreId>
         <number>103</number>
         <street>Rosario</street>
         <modelDate>
            <validTime>
               1/1/1996..31/12/1997
            </validTime>
            <transactionTime>
               1/1/1995..31/12/1997
            </transactionTime>
         </modelDate>
         <extentOf>
            <Polygon>
               <outerBoundaryIs>
                  <LinearRing>
                     <coordinates>
                        0.0,0.0 40.0,0.0 40.0,10.0 0.0,10.0
                     </coordinates>
                  </LinearRing>
               </outerBoundaryIs>
            </Polygon>
         </extentOf>
      </Parcel>
   </modelMember>
</CityModel>
```
<coordinates>
</coordinates>
</LinearRing>
</outerBoundaryIs>
</Polygon>
</extentOf>
<location>
<Point>
<coordinates> 20.0,5.0 </coordinates>
</Point>
</location>
</Parcel>
</modelMember>
<modelMember>
.Parcel>
<cadastreId>
3008
</cadastreId>
<number>
101
</number>
</street>
Rosario
</street>
<modelDate>
<validTime>
1/1/1995..31/12/1997
</validTime>
</transactionTime>
1/1/1995..31/12/1999
</transactionTime>
</modelDate>
<extentOf>
<Polygon>
<outerBoundaryIs>
<LinearRing>
<coordinates>
40.0,0.0 50.0,0.0 50.0,10.0
40.0,10.0
</coordinates>
</LinearRing>
</outerBoundaryIs>
</Polygon>
</extentOf>
<location>
<Point>
<coordinates> 45.0,5.0 </coordinates>
</Point>
</location>
</Parcel>
</modelMember>
<modelMember>
.District>
<districtId>
105
</districtId>
<districtName>
Carretas
</districtName>
<modelDate>
<validTime>
1/1/1995..31/12/1999
</validTime>
</transactionTime>
1/1/1995..31/12/1999
</transactionTime>
</modelDate>
<extentOf>
<Polygon>
<outerBoundaryIs>
<LinearRing>
<coordinates>
0.0,0.0 90.0,0.0 90.0,60.0 0.0,60.0
</coordinates>
</LinearRing>
</outerBoundaryIs>
</Polygon>
</extentOf>
</District>
</modelMember>
</CityModel>