A Web-Enabled Extension of a Spatio-Temporal DBMS

Markus Innerebner Michael Böhlen Igor Timko
Free University of Bozen-Bolzano, Italy
{innerebner, boehlen, timko}@inf.unibz.it

ABSTRACT

Many database applications deal with spatio-temporal phenomena, and during the last decade a lot of research targeted location-based services, moving objects, traffic jam preventions, meteorology, etc. In strong contrast, there exist only very few proposals for an implementation of a spatio-temporal database system let alone a web-based spatio-temporal information system.

This paper describes the design and implementation of a web-based spatio-temporal information system. The system uses Secondo as spatio-temporal DBMS for handling moving objects and MapServer as an OGC-compliant rendering engine for static spatial data. We describe the architecture of the system and compare our system with a standalone application. The paper investigates in detail issues that arise in the context of the web. First, we describe an implementation of a lightweight client that takes advantage of the functionality offered by Secondo and MapServer. Second, we describe how moving objects can be represented in GML. We discuss possible GML representations, propose an extension of GML that uses 3D segments (2D location + time) to represent moving objects, and present experiments that compare the solutions.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

Keywords

Spatio-temporal Web Applications, GML, OGC

1. INTRODUCTION

During the last decade spatio-temporal phenomena have become an important research area. Many real world applications, including location-based services, traffic jam prevention, and weather forecasts, focus on moving objects. For example location-based services target moving physical persons and strive to provide them with information according to their interests and current location. In the area of traffic jam detection and prevention trajectories of moving vehicles are collected to identify traffic jam areas. For weather forecasts data of moving weather fronts has to be investigated to come up with accurate forecasts for the future.

Despite the ubiquity of spatio-temporal phenomena, there exist only few complete spatio-temporal DBMSs (STDBMSs) offering spatio-temporal data types and operators. All of them use proprietary representations of moving objects, which hampers their deployment in web-based information systems. We describe the design and implementation of a web-based spatio-temporal information system that takes advantage of the functionality of existing systems. It uses Secondo, a spatio-temporal DBMS, for handling moving objects and MapServer, an OGC-compliant rendering engine, for handling static spatial data.

The technical contributions of the paper are as follows:

- We propose a modular system architecture that can be used in combination with web applications. Specifically, we describe a service oriented extension of Secondo with standard services to store, transport and process spatio-temporal information. A web user or service can combine, use and manage these services to work with spatial and spatio-temporal information (moving points).
- We describe a web interface that is implemented as a lightweight client. The rendering of spatial layers is delegated to the server-side rendering engine and only moving points are rendered on the client.
- We show how moving points can be represented with the dynamic features of GML. First, we discuss a GML representation where moving objects are modeled as ordered sequences of snapshots. Second, we extend GML to moving points using the approach from Lema et al. [19]. The trajectory of a moving point is decomposed into units where each unit represents a 3D segment (2D location + time).
- We present empirical results that demonstrate that for moving objects with frequent gaps in their definition time (i.e., a gap after every or every second segment) the new representation is more suitable, whereas for moving objects with rare gaps (i.e., with more than two segments after one gap) the representation with multiple ordered sequences of snapshots performs well.

The rest of the paper is organized as follows. Section 2 describes spatio-temporal systems, geographical rendering engines, standards in the spatial area and other related work in this area. In Section 3 we explain how moving points can be represented in GML. Section 4 describes the architecture of our system and explains how spatio-temporal queries are processed. Section 5 compares two GML moving point representations. Section 6 concludes the paper and points to future research directions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACMGIS '07, November 7-9, 2007, Seattle, WA
Copyright 2007 ACM ISBN 978-1-59593-914-2/07/11 ...$5.00.

2. RELATED WORK

Only very few database systems exist that support moving objects. DOMINO [29] is a layer built on top of a DBMS to support moving object databases. It uses ArcView as an application server to handle spatial data and ArcIMS as a rendering engine. Spatio-temporal data is stored in Informix. DOMINO focuses on present and future trajectory data. CONCERT, Secondo and DEDALE are representatives of extensible systems [3] that offer support for application-specific modules. CONCERT [24] provides flexible index support and its abstract object storage types allow to implement spatio-temporal data types. The main contribution of DEDALE [11], another spatio-temporal system, is the handling of interpolated data. Secondo [14] features user-defined algebras, and it provides a powerful spatio-temporal algebra that supports moving objects. A stand-alone client application contains several viewers to visualize the various data types. The most relevant viewer can visualize spatial and spatio-temporal objects. Queries are formulated as text strings (SQL or an algebraic format, Second Order Signature) and the result is sent back as a nested list, a very compact format to describe structural data. The client parses the result and produces graphical objects that use and extend the Swing library of Java.

The Open Geospatial Consortium (OGC) is an organization that provides standards for geo-spatial data and location-based services. Many GIS systems nowadays follow the OGC standards [7]. OGC proposed a vendor-neutral Geographic Markup Language (GML) [8], a XML-based encoding for modeling, storing, and transporting geographic information. GML includes spatial and non-spatial properties of geographic features. A feature is a tuple with alphanumeric attributes associated with a geometry (point, line, polygon, etc.)

An OGC compliant system has to provide operations to retrieve geographic features in form of a service oriented architecture. This means the system has to offer services to operate with spatial data. These services are accessible over the HTTP protocol or via method invocation.

Two different services must be developed to deal with geographic information. First, to request a graphical visualization of geographic data (map), an OGC compliant system must provide a specific service, called WMS (Web Map Service) [1], with the following functionality: list what kind of maps are deliverable (GetCapabilities) and produce a map of a specified layer (GetMap). Second, to query, insert, delete, or update geographic features, an OGC compliant system must offer a service called WFS (Web Feature Service) [32]. This service defines interfaces for data access and manipulation on geographic features, using HTTP as the distributed computing platform. Via these interfaces, a web service can combine, use and manage geodata from different sources. The service must have the following functionality: list all queryable layers, the available operations to be invoked (query, insert, update, delete, etc.), and the predicates to be applied (GetCapabilities). It must be possible to request features and receive as result a valid XML document that contains the content (GetFeature). In such a feature element, spatial properties are expressed with GML. When requesting geographic maps or features, we usually want as result a subset of the database objects. OGC provides a XML encoding for filtered expressions, called Filter Encoding Implementation Specification (FEIS) [31]. It provides a set of logical, comparison, spatial, and some temporal operators and predicates.

For the spatio-temporal domain, OGC provides limited support in terms of dynamic features. Specifically, a moving point can be modeled with the dynamic features TimeSlice and MovingObject-Status. Both of them represent a moving point as a sequence of its states (snapshots at specific time instants). The TimeSlice representation is described in detail in Section 3. Currently FEIS does not provide spatio-temporal operators or filters with dynamic features.

The increase of Web-GIS application during the last five years has fueled the development of rendering engines that produce maps. The most well-known systems are ArcIMS [9], MapServer [30] and GeoServer [5]. The main task of a rendering engine is to facilitate the cooperation between geographic sources (stored in DBMS or in shape files) and the application (in our case the web client) [4, 30]. MapServer as well as GeoServer are capable to interpret GML. They can be configured to be accessible via the WMS and WFS services. Common Web-GIS client implementations are MapBender and MapBuilder. The latter is a powerful, standards compliant client that is written in Java Script. MapBuilder visualizes a map either by fetching images with WMS requests or by rendering images on the client. The client can only render basic geometry types of GML (point, line, polygon). For moving objects, no rendering exists.

During the last years significant research efforts have focused on proposing advanced techniques for handling moving objects (e.g., indexing of moving objects [18, 21–23], continuous nearest neighbor queries for moving objects [2, 16, 17], aggregation of moving objects [21, 27, 28]). These results focused on selected aspect and were not presented in the context of a working system.

As for modeling moving objects, Güting et al. [13, 19] propose to model and query moving points as a sequence of units. Each unit represents the movement of a point along a straight line. This approach is implemented in Secondo. In this paper, we propose a GML representation of moving objects based on the Güting's approach. An extension of the Güting's approach to network-constrained objects is presented in [15]. Speicys et al. [26] propose another data model for network-constrained objects that supports NN queries. Sistla et al. [25] describe a model based on dynamic attributes that supports queries about current and near future positions of moving objects. This approach is implemented in DOMINO. Grumbach et al. [12] present a constraint-based model for moving objects. This model is implemented in DEDALE.

3. MOVING POINTS IN GML

Results of queries about moving objects may contain a lot of data. This data must be sent from the server to the client for visualization. For an efficient rendering, we propose to send this data in one block (i.e., in one GML file). This raises the issue of modeling moving objects in GML. In this section, we propose two GML representations of moving points (moving point objects).

For illustrating the representations, we use the example moving point from Figure 1. The object is the city bus number 10A of Bozen-Bolzano, Italy. On *June 1, 2007, at 10:00:00 AM*, the bus is at position with coordinates (*680206.67, 5151256.16*). Between *10:00:00 AM and 10:01:33 AM*, the bus moves linearly to position (*680286.01, 5151314.74*). Then, the bus makes a turn and again moves linearly until it reaches position (*680362.90, 5151363.56*), and so on. Thus, we have a linear approximation of the bus's movement. The bus's positions are specified in the UTM coordinate reference system.

The first representation, termed the GML TimeSlice representation, is an existing representation [8]. It is based on the idea that a moving point is a sequence of its states. A state is a pair of time and position and is expressed with the TimeSlice element. Hence, a moving point is captured by a sequence of TimeSlice elements. In order to capture discontinuities in movements (i.e., gaps in definition time of the moving point, which is common in query results), each sequence of TimeSlice elements that captures one continuous "piece" of movement is put into one history element.

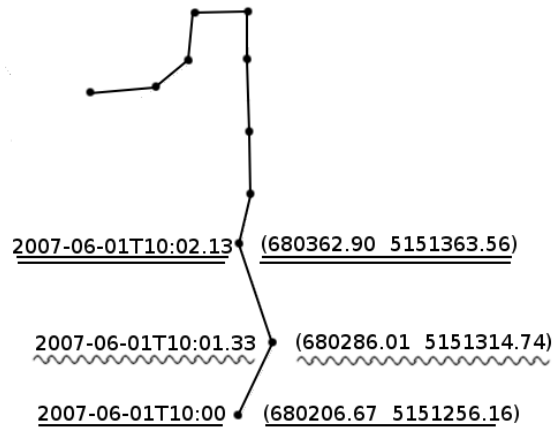

Figure 1: Moving Point: Bus Number 10A in Bozen-Bolzano

Example 3.1 illustrates a GML document that represents the moving point from Figure 1 in the GML TimeSlice format. The movement of the bus between positions (*680206.67, 5151256.16*) and (*680286.01, 5151314.74*) is represented as two *TimeSlice* elements, one between lines 7 and 18 and the other one between lines 19 and 30. Because the movement of the bus is continuous, i.e., without gaps in the definition time, the document contains only one *history* element, between lines 6 and 44. The corresponding states in Example 3.1 and in Figure 1 are indicated by the same style of underlining (e.g., the first state is underlined by a single line).

EXAMPLE 3.1. *GML TimeSlice Representation*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <bz10m:Buses xmlns="http://www.opengis.net/gml" xmlns:bz10m="http://
       www.inf.unibz.it/dis/bz10m">
3   <bz10m:id>1</bz10m:id>
4   <bz10m:busLine>Linea10a</bz10m:busLine>
5   <bz10m:trip gml:id="93fd2ebb-1cd6-457f-8fb9-45b634e0e907">
6    <history>
7     <TimeSlice gml:id="0282cb38-8350-45fb-94bd-61dd3cf80de3">
8      <validTime>
9       <TimeInstant>
10       <timePosition>2007-06-01T10:00</timePosition>
11      </TimeInstant>
12     </validTime>
13     <location>
14      <Point srsName="urn:EPSG:geographicCRS:4258">
15       <pos>680206.67 5151256.16</pos>
16      </Point>
17     </location>
18    </TimeSlice>
19    <TimeSlice id="4a231b33-11ee-4110-9f5f-9dc8675f5f1b">
20     <validTime>
21      <TimeInstant>
22       <timePosition>2007-06-01T10:01.33</timePosition>
23      </TimeInstant>
24     </validTime>
25     <location>
26      <Point srsName="urn:EPSG:geographicCRS:4258">
27       <pos>680286.01 5151314.74</pos>
28      </Point>
29     </location>
30    </TimeSlice>
31    <TimeSlice id="4a231b33-11ee-4110-9f5f-9dc8675f6f16">
32     <validTime>
33      <TimeInstant>
34       <timePosition>2007-06-01T10:02.13</timePosition>
35      </TimeInstant>
36     </validTime>
37     <location>
38      <Point srsName="urn:EPSG:geographicCRS:4258">
39       <pos>680362.90 5151363.56</pos>
40      </Point>
41     </location>
42    </TimeSlice>
43    <!-- other TimeSlice elements -->
44   </history>
45  </bz10m:trip>
46 </bz10m:Buses>
```

In Secondo, a moving point is represented as a sequence of *units*. A unit is a pair of states, which represents movement between the two states (see [19] for details). Thus, the GML TimeSlice representation and Secondo's representation of moving points do not match. For this reason, it is necessary to convert query results from Secondo's representation to the GML TimeSlice representation. This conversion must take into account that moving points are allowed to have gaps in their definition time. In the Secondo representation, gaps are modeled *implicitly*, by not having a unit for the corresponding time interval. In contrast, in the GML TimeSlice representation, gaps are modeled *explicitly* by having a *history* element for each period of continuous movement. In case of a trajectory with a large number of gaps this will increase the size of the document. During the conversion to TimeSlice elements gaps need to be identified: this is done for each unit with a verification whether the time of its end state is equal to the time of the start state of the next unit.

In order to directly support Secondo's mpoint type, we propose a new GML representation, termed GML MovingObjectUnit representation, that maps each Secondo unit into one GML element. This simplifies the conversion between Secondo's representation and GML. For the GML MovingObjectUnit representation, we reuse existing GML elements along with two new elements: (1) *MovingObjectUnit* that represents a unit and (2) *units* that represents a sequence of moving object units. In this representation, discontinuities in movements are represented by having no moving object units for the time periods when moving points are not defined. Thus, a complete moving point is captured by one *units* element. Since a moving object unit models the movement from one point to another point, we restrict the occurrences of points in the LineString type to exactly two *position* elements.

Example 3.2 presents a GML document that represents the moving point from Figure 1 in the GML MovingObjectUnit representation. The movement of the bus between positions *(680206.67, 5151256.16*) and (*680286.01, 5151314.74*) is represented as the MovingObjectUnit element between lines 7 and 22. The complete movement of the bus is captured by one *units* element, between lines 6 and 40. The corresponding states in Example 3.2, Example 3.1, and in Figure 1 are indicated by the same style of underlining (e.g., the very first state is underlined by a single line).

EXAMPLE 3.2. *GML MovingObjectUnit Representation*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <bz10m:Buses xmlns="http://www.opengis.net/gml" xmlns:bz10m="http://
       www.inf.unibz.it/dis/bz10m">
3   <bz10m:id>1</bz10m:id>
4   <bz10m:busLine>Linea10a</bz10m:busLine>
5   <bz10m:trip id="c33c1bf1-7730-4757-8f61-56c347f776af">
6    <units>
7     <MovingObjectUnit>
8      <validTime>
9       <TimePeriod>
10       <beginPosition>2007-06-01T10:00</beginPosition>
11       <endPosition>2007-06-01T10:01.33</endPosition>
12      </TimePeriod>
13      <includeBegin>true</includeBegin>
14      <includeEnd>false</includeEnd>
15     </validTime>
16     <location>
17      <LineString srsName="urn:EPSG:geographicCRS:4258">
18       <pos>680206.67 5151256.16</pos>
19       <pos>680286.01 5151314.74</pos>
20      </LineString>
21     </location>
22    </MovingObjectUnit>
23    <MovingObjectUnit>
24     <validTime>
25      <TimePeriod>
26       <beginPosition>2007-06-01T10:01.33</beginPosition>
27       <endPosition>2007-06-01T10:02.13</endPosition>
28      </TimePeriod>
29      <includeBegin>true</includeBegin>
30      <includeEnd>false</includeEnd>
31     </validTime>
32     <location>
33      <LineString srsName="urn:EPSG:geographicCRS:4258">
34       <pos>680286.01 5151314.74</pos>
35       <pos>680362.90 5151363.56</pos>
36      </LineString>
37     </location>
38    </MovingObjectUnit>
```

```
39      <!-- other MovingObjectUnit elements. Even in case of gaps -->
40    </units>
41   </bz10m:trip>
42  </bz10m:Buses>
```

Cox et al. [8] propose another GML representation of moving points, which we term the GML MovingObjectStatus representation. This representation, like the GML TimeSlice representation, represents a moving point as a sequence of its states. Since the two representations are similar we do not consider the GML MovingObjectStatus representation in this paper.

Section 5 presents the experimental comparison between the GML TimeSlice and GML MovingObjectUnit representation.

4. SYSTEM ARCHITECTURE AND IMPLEMENTATION

Our goal is to implement a web-based spatio-temporal information system with a lightweight client. Our strategy is to select a spatio-temporal database system, which implements a considerable amount of spatio-temporal functionality, and to integrate it with a web-based, lightweight client. For the implementation (data exchange format and the available service methods), we conform to the OGC standards as much as possible. In this section, we present the architecture of our system.

4.1 Architecture

Figure 2 illustrates the architecture of our system. The three system components are (from left to right) Spatio-Temporal (ST) Server, the Light Graphical User Interface (Light-GUI), and the rendering engine. The general interaction between the components is as follows: after having received the user query, the Light-GUI decomposes the query into two components: spatio-temporal (i.e., moving points) and spatial (i.e., a background map). It sends the spatio-temporal and spatial component to the ST Server and the rendering engine, respectively. It receives responses and visualizes the query result. In the following, we discuss each system component in detail.

The ST Server provides the Light-GUI with the spatial and spatio-temporal data and meta data (i.e., features of spatial and spatio-temporal layers). The ST Server uses Secondo as a DBMS, because it provides many data types and operators related to moving objects and, in particular, to moving points [19].

To offer an OGC compliant system we have to provide WFS services to query spatio-temporal data and we have to represent the the result in GML format. Secondo accepts SQL queries and returns the results in the nested list format. Hence a Converter converts WFS requests into SQL expressions and transforms the nested list into GML. For this Secondo's date format needs to be converted into the corresponding date format pattern provided from XML-schema and each moving point needs to specify what coordinate reference system is used. In addition, we handle repeating data intelligently, which helps to reduce the GML document size considerably. Specifically, we employ reification [33]. This means that we assign a unique ID to each distinct location mentioned in the nested list. Thanks to this, the resulting XML document contains the full specification (i.e., XML element) for each distinct location only once. For each repetition of the distinct location, the document contains the XLink attribute that points to the full specification. As usual after the conversion to XML, the size of the data increases considerably. We follow the standard approach and let the web server compress XML documents, which yields high compression rate [10].

The Light-GUI is implemented in a web browser. We do not simply migrate Secondo's GUI, written as a Java standalone program, into a Java applet, but create our user interface from scratch using an existing web mapping client framework. There are several reasons for this decision. First, we want to offer a web-based application that runs within web browsers without additional extension (i.e., JRE need to be installed to make Java applets running in a browser). Second, we want of offer a lightweight client that does not have to take care of the memory expensive rendering process of spatial data. Finally, we want to provide the user with predefined settings to facilitate the formulation of spatio-temporal queries. We follow the approach of GIS systems and want to allow web users to create spatio-temporal queries in a graphical manner.

The Light-GUI contains MapBuilder [6] as a mapping client. MapBuilder is lightweight (written entirely in Java Script), open-source, and standard-compliant. The client is capable to render the basic geometry GML types such as point, line, and region, which is enough for a Web-GIS system. Through the integration of AJAX [20], the data transfer between client and server (WMS/WFS) is done asynchronously, which avoids latencies when loading images. Its implementation is based on the Model-View-Controller concept, which provides a clear separation of presentation and application content.

Note that for visualizing spatio-temporal data, the Light-GUI does not rely on a WMS server that generates static images continuously. This would yield an unacceptable system load. Instead, it obtains the spatio-temporal data from the ST Server. It interprets a GML document (WFS response) that contains moving points either in the GML TimeSlices or in the GML MovingObjectUnit representation. The moving point is visualized by advancing through its states in discrete time steps. At each step, we advance to the next state and make it visible on the screen, while the last visible state is made invisible. The states given explicitly in the GML document are always displayed. We control the precision of the visualization by following a simple heuristic: if the temporal and/or spatial distance between two consequent states exceeds a certain threshold, then the intermediate state obtained by a linear interpolation is additionally displayed.

The rendering engine provides the Light-GUI with results of spatial user queries in the form of lightweight images. At the moment, we use UMN-MapServer [30], but it is easily possible to use any other OGC-compliant rendering engine.

4.2 Query Specification

Standard WEB-GIS applications provide a configurable system: in a configuration file, the maintainer of the application specifies the functionality to be available for the web user, the available layers in the map, etc. Our Light-GUI follows the exact same approach. For setting up a working system, the maintainer of the web site, has to provide three different files. The first file, the main configuration file, specifies the available GIS functionality (zooming, panning, buffering, feature querying, etc.) that is already implemented on the client and other useful widgets to be offered to the user. By enabling them they can be used without additional implementation effort. The second file, the HTML page, specifies the layout of the application (i.e., where to position the map or where to set the action commands). The third file, called context file, specifies the displayed data: that includes the name of layers and features. It also contains the style of layer rendering and the coordinate reference system (EPSG code). In contrast to features layers are rendered on the server side.

EXAMPLE 4.1. *Layer Specification in MapBuilder*

```
<Layer queryable="1" hidden="0">
 <Server service="OGC:WMS" version="1.1" title="OGC:WMS">
  <OnlineResource xlink:type="simple"
```

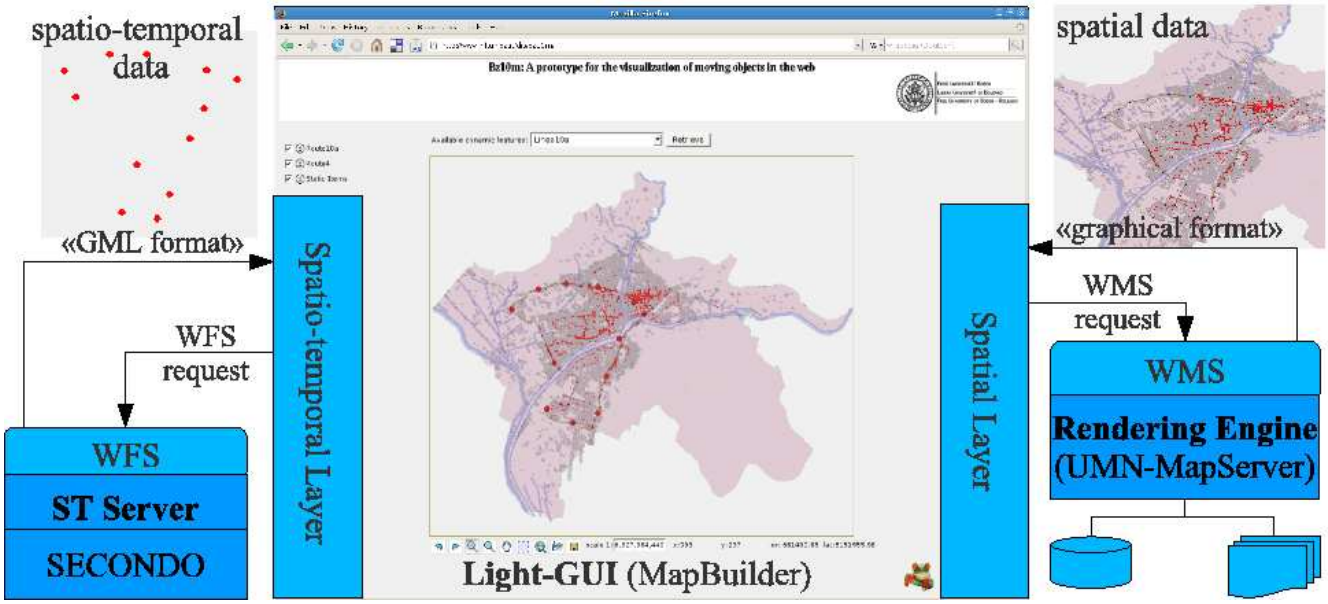

Figure 2: Architecture of the System

```
    xlink:href="http://www.inf.unibz.it/dis/maps/cgi-bin/mapserv" />
  </Server>
  <Name>Districts</Name>
  <Title>Districts</Title>
  <SRS>EPSG:4258</SRS>
  <FormatList>
   <Format current="1">image/png</Format>
  </FormatList>
  <StyleList>
   <Style current="1">
    <Name>default</Name>
    <Title>default</Title>
   </Style>
  </StyleList>
</Layer>
```

Example 4.1 shows how a layer element is specified in the context file. The layer *Districts* fetched from the host *inf.unibz.it/dis* will be displayed within the map of our client.

Spatial Query Specification The client offers predefined spatial queries implemented in widgets. For example if a user wants to execute a zoom query, he specifies in the map his area of interest with a bounding box. The client collects these values (BBOX) and the input values from the HTML document (LAYERS), adds additional parameters (WIDTH, HEIGHT, SERVICE, VERSION) from the other configuration files and sends the following WMS request to the rendering engine:

EXAMPLE 4.2. *WMS GET request*

```
http://www.inf.unibz.it/dis/maps/cgi-bin/mapserv?
SERVICE=WMS&
VERSION=1.1.1&
REQUEST=GetMap&
FORMAT=image/png&
STYLES&
SRS=EPSG:4258&
WIDTH=400&
HEIGHT=400&
LAYERS=Districts&
BBOX=674100,5145567,686620,5155835
```

The WMS compliant rendering engine interprets this request, processes the query, and sends the produced map back to the requestor. This map will be updated on the client.

Spatio-Temporal Query Specification Spatio-temporal information is roughly queried like spatial information. The two differences are WFS requests to the spatio-temporal server and the client side rendering of GML objects. Spatio-temporal queries in comparison to spatial queries are much more complex, because in addition to the spatial aspect, the temporal aspect must be considered.

Therefore our goal is to facilitate to web users the formulation of spatio-temporal queries, using a graphical approach. A real world scenario might be a tourist that wants to get the following information from the system (see Example 4.3)

EXAMPLE 4.3. *Example of a spatio-temporal query*

Show all buses that are inside the district "Industriezone" of Bozen-Bolzano, on June 1, 2007, between 10:00 and 10:30 AM.

In order to answer the query in Example 4.3 we have to formulate a spatio-temporal query. One possibility is to formulate it in Secondo's algebra (see Example 4.4).

EXAMPLE 4.4. *Example Query in Relational Algebra*

$$\Pi_{Buses.id,\, Buses.busLine,\, Buses.dynamicGeometry}\ (\sigma_{Districts.name="Industriezone" \wedge Buses.dynamicGeometry.between("2007-06-01T10:00" and "2007-06-01T10:30") \wedge Buses.dynamicGeometry.passes(Districts.geometry)}\ (Buses \times Districts))$$

Districts is a relation with the non-spatial attribute *"name"* of type *"string"* and the spatial attribute *"geometry"* of type "region". *Buses* is a relation with the spatio-temporal attribute "dynamicGeometry" of type "mpoint". The predicate *between* checks whether a given moving point is defined during a given time interval. The predicate *passes* is used as a join condition. It tests if that moving point ever gets inside a region.

Instead of an algebraic expression we can use Secondo's SQL-like query language (see Example 4.5). The optimizer takes the SQL query and produces an efficient algebraic expression. In the example below *ti* represents a time interval.

EXAMPLE 4.5. *Query in Secondo's SQL syntax*

```
select [ B:id,B:busLine,B:dynamicGeometry ]
from   [ Districts as D, Buses as B ]
where  [ B:dynamicGeometry present ti,
         D:name = "Industriezone",
         B:dynamicGeometry passes D:geometry ]
```

We follow the approach used on OGC compliant systems to query spatial features. This means that we use OGC's FEIS to specify filter criteria on the features to be queried. As already mentioned in Section 2, FEIS does not support any spatio-temporal operators and predicates. Hence, we extend FEIS with new spatio-temporal operators and predicates. Our main objective is not to maximize the expressiveness, but to seamlessly extend the filtering encoding to support spatio-temporal predicates.

Example 4.6 shows how an SQL expression can be mapped into a WFS-GetFeature request. The join condition is implicitly specified in the attribute *"typeName"* within the element *wfs:Query* that contains the two features (Buses and Districts) to be joined in combination with selection predicates. The predicates *ogc:PropertyIsEqual* and *ogc:Between*, already defined in FEIS, are reused. In order to maintain as much as possible the existing OGC operators we replace Secondo's predicate *present* with *ogc:Between* and *'='* with *ogc:PropertyIsEqual*. The spatio-temporal predicate *ogc:Passes* is a new operator that corresponds to operator *passes* in Secondo.

EXAMPLE 4.6. *WFS POST request*

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs" xmlns:ogc="http://www.
      opengis.net/ogc"
 xmlns="http://www.inf.unibz.it/dis/bz10m" version="1.0.0" service="WFS"
       maxFeatures="100">
 <wfs:Query typeName="Buses=B, Districts=D">
 <wfs:PropertyName>B.id</wfs:PropertyName>
 <wfs:PropertyName>B.busLine</wfs:PropertyName>
 <wfs:PropertyName>B.dynamicGeometry</wfs:PropertyName>
  <ogc:Filter>
   <ogc:And>
    <ogc:PropertyIsEqualTo>
     <ogc:PropertyName>D.name</ogc:PropertyName>
     <ogc:Literal>Industriezone</ogc:Literal>
    </ogc:PropertyIsEqualTo>
    <ogc:Between>
     <ogc:PropertyName>B.dynamicGeometry</ogc:PropertyName>
     <ogc:LowerBoundary>
      <ogc:Literal>2007-06-01T10:00</ogc:Literal>
     </ogc:LowerBoundary>
     <ogc:UpperBoundary>
      <ogc:Literal>2007-06-01T10:30</ogc:Literal>
     </ogc:UpperBoundary>
    </ogc:Between>
    <ogc:Passes>
     <ogc:PropertyName>B.dynamicGeometry</ogc:PropertyName>
     <ogc:PropertyName>D.geometry</ogc:PropertyName>
    </ogc:Passes>
   </ogc:And>
  </ogc:Filter>
 </wfs:Query>
</wfs:GetFeature>
```

As mentioned before, we want to facilitate the formulation of spatio-temporal queries. Therefore the web user should be able to create the WFS request from Example 4.6 in a graphical manner. In order to create this request, the user selects the dynamic feature *Buses* from a combo box, specifies the *time period* from a calendar, selects the spatial layer *district* again from a combo box, and types in a text field the district name. In addition, he selects the spatio-temporal predicate that is applied. The client parses the HTML document and converts (with an XSLT stylesheet) the specified input values into a corresponding WFS request with some filter expression. The request is sent to the spatio-temporal system. The system converts the request into the Secondo understandable query language and sends it to the database. The converter converts the result again into GML and sends the XML document as a WFS response back to the client. Now the client interprets this document and renders the moving point.

Figure 3 shows a snapshot captured on *June 1, 2007, at 10:01 AM*. Hence, the image not only shows buses that are inside the specified district at this moment, but also those buses that will pass the district in the future within the specified time interval.

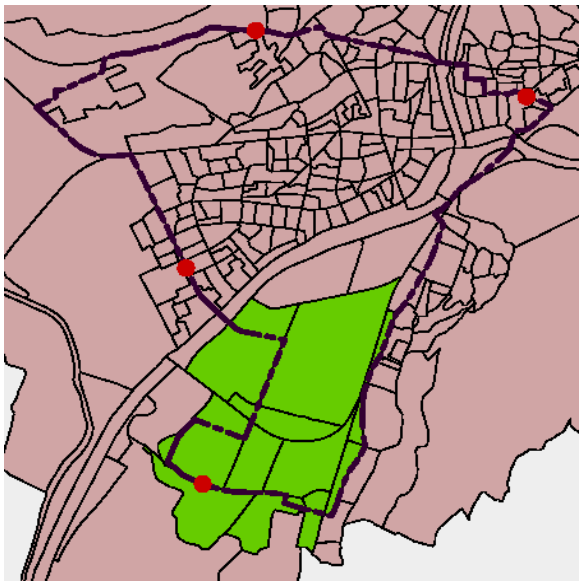

Figure 3: Result of the Spatio-Temporal Query

At the moment in the HTML document there are specified this necessary input values, that the user selects to formulate a spatio-temporal WFS request. In the future it should be possible to specify this parameters in a configuration file, where the maintainer of the project specifies, what kind of spatio-temporal queries are available for the user.

4.3 Our System vs. Secondo

This section provides a more detailed comparison between our system and Secondo.

In a web setting a bottleneck of Secondo is the client-side rendering. In a typical case, many map layers must be visualized (e.g., buildings, streets, bus stops, traffic lights, etc.) Each layer must be converted into a graphical object. Therefore, a lot of data must be retrieved and processed. With our system, the rendering engine produces a compact image file. It is possible to further improve performance by obtaining two independent spatial layers from two different rendering engines.

In addition, as mentioned in Section 4.2, in order to obtain the result of spatio-temporal queries on our system (e.g., the query from Example 4.4 visualized as in Figure 3), only a small amount of standard configurations are needed (e.g., specifying the layers). With Secondo, the user must go through a sequence of steps to do the configuration. Specifically, first, the user needs to load the spatial layer Districts and specify its layout (colors, border style, etc.). Second, the user needs to load the second layer that describes the bus route. Finally, the user must type the query expression from Example 4.5 to obtain the trajectories of the queried spatio-temporal layer. After these steps, the user may store the layout configuration in a file and save the launched commands in a session to reuse them at later time. Thus, with Secondo, the user has more flexibility but must make a bigger initial effort.

With our system, the user can easily use standard features of GIS systems, such as obtaining an integrated overview map for better navigation, legends for layers, and other spatial utilities (e.g., scalebar, history buttons, etc.).

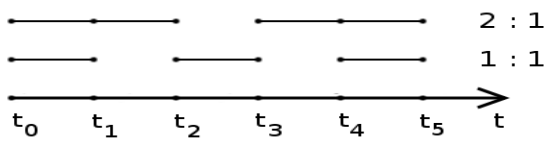

Figure 4: Unit-Gap Proportion

Because of the open, service oriented architecture of our ST Server, other systems may also use the services provided by the server. The data exchange formats used by our system (e.g., our GML representations of moving points, etc.) follow the XML standard and is easily handled by most systems.

A drawback of our system is the duplicate storage of some spatial data. For example, in order to process the query from Example 4.4, the *Districts* relation must be stored both in the rendering engine and in Secondo. Secondo uses the relation for processing the spatio-temporal join, while the rendering engine uses the relation for rendering. The problem arises because the rendering engine cannot interpret spatial geometries stored in Secondo databases.

As mentioned above Secondo's client offers more flexibilty in the formulation of queries. Queries are expressed in Secondo's SQL-like query language or in Second Order Signature. The latter language is very powerful, not limited to geographic data, and offers in addition to the query operators also constructs to define objects at runtime. For advanced users this powerful functionality is very useful.

A possible advantage of Secondo's heavy client is that the object structure of the data is present. This might be used for editing spatial and spatio-temporal data. In our client the spatial information are only available in a rendered image. Even if the data are fetched via WFS we do not have the entire object structure on the client side. Therefore Secondo's client is better suited for editing spatial and spatio-temporal data.

5. EXPERIMENTS

This section presents experimental evaluation of the GML representations of moving points from Section 3.

For our experiments, we use real-world data on city bus number 10A of Bozen-Bolzano, Italy. We have 10 data sets. Each data set contains 120 moving points. Each moving point represents one complete route of a physical bus (i.e., its movement from the start stop to the end stop). Thus, the same physical bus is represented by as many moving points as many times it traverses its complete route. For every data set, the total lifespan of all the moving points covers one day (i.e., 6 AM – 9PM). The difference of the data sets is as follows: the moving points from different data sets have different frequency of gaps in their definition time. Specifically, there is one data set without gaps. Then, there is one data set with a gap instead of every n^{th} unit, termed a data set with (n-1):1 unit-gap proportion, for $n = 2, \ldots, 10$. Figure 4 illustrates the idea of the data sets with 2:1 and 1:1 unit gap proportion. The data set with no gaps contains 22080 units. Figure 5 compares the size of GML documents generated in response to the query that asks for a complete data set. We can see that for the low values of n (i.e., frequent gaps) the GML MovingObjectUnit representation is preferable, while for the high values of n (i.e., rare gaps) the GML TimeSlice representation is preferable. The reason for this is as follows: the more frequent the gaps, the less units per moving point, while the number of states

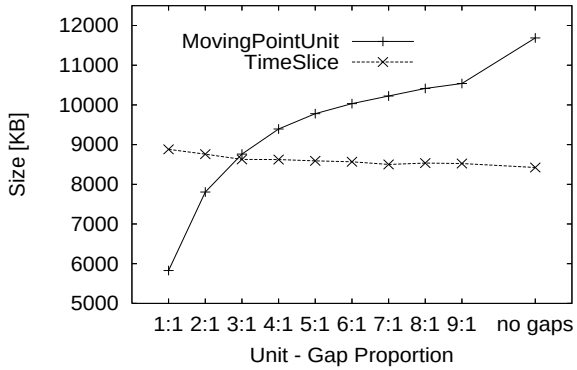

Figure 5: Size Comparison between TimeSlice and MovingObjectUnit

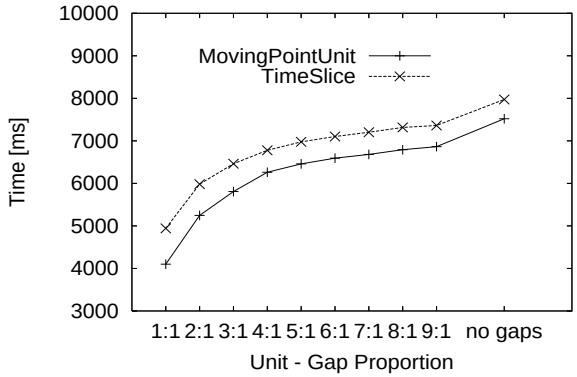

Figure 6: Time Comparison between TimeSlice and MovingObjectUnit

remains the same. Recall that the GML MovingObjectUnit representation captures units, while the GML TimeSlice representation captures states. In other words, with rare gaps, in the MovingObjectUnit representation, for many units, the end point of each unit will be the same as the start point of its successor. Therefore, many points are stored twice. At the same time, with frequent gaps, each TimeSlice element is hold in a separate parent element (history), while there is always one parent element (units) for all the units. Note that the GML TimeSlice representation does not depend on the unit-gap proportion.

Figure 6 compares the time needed for generation of the GML documents in response to the query that asks for a complete data set. For each data set, 100 identical queries were issued. The figure presents the average time per one query. We can see that the GML MovingObjectUnit representation is always faster than the GML TimeSlice representation. This is because in order to convert moving point unit from the nested list format to the GML TimeSlice representation, we need to check whether there is a gap and for each gap we need to create a new parent element. In both the figures, all the curves (except the TimeSlice curve in Figure 5) are increasing, because the less gaps the moving point has, the larger its nested list is.

6. CONCLUSION AND FUTURE WORK

Spatio-temporal databases are an active area of research. Many research challenges have been solved during the last years and the first systems are being deployed. At the same time web applications are getting more and more important and are often as widespread as standalone applications. Service oriented architectures are the standard approach to make services available over the Internet. In this paper we show how to integrate current state of the art systems for spatio-temporal and geographic data to build web-based information systems for moving objects. We show how to represent moving points in a standard XML-conform format and how to extend GML with a type for trajectories with gaps.

In order to query spatio-temporal data we assumed the spatio-temporal operators from Secondo and focused on moving points. In the future, we will integrate other spatio-temporal data types (e.g., moving regions) and model them with GML. To reduce the latency when querying spatio-temporal information we need novel solutions that decrease the size of the document sent to the client. A promising direction is to send data according to the zoom factor. Also there is a need for query processing techniques that take into account that data is managed by different systems.

7. REFERENCES

[1] J. Beaujardiere. Ogc web map service interface. Technical report, Open GIS Consortium, 2004.

[2] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest and reverse nearest neighbor queries for moving objects. *VLDB J.*, 15(3):229–249, 2006.

[3] M. Breunig, C. Türker, M. H. Böhlen, S. Dieker, R. H. Güting, C. S. Jensen, L. Relly, P. Rigaux, H.-J. Schek, and M. Scholl. Architectures and implementations of spatio-temporal database management systems. In *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, pages 263–318, 2003.

[4] T. Brinkhoff and J. Weitkämper. Continuous queries within an architecture for querying xml-represented moving objects. In *SSTD*, pages 136–154, 2001.

[5] G. Community. Geoserver, an open source server that connects your information to the geospatial web.

[6] M. Community. Mapbuilder, a web mapping client.

[7] O. G. Consortium. Ogc implementors.

[8] S. Cox, P. Daisey, R. Lake, C. Portele, and A. Whiteside. Geographic information- geographic markup language (gml) v 3.1 - iso/tc 211/wg 4/pt 19136. Technical report, Open GIS Consortium, 2004.

[9] R. East, R. K. Goyal, A. Haddad, A. Konovalov, A. Rosso, M. Tait, and J. Theodore. The architecture of arcims, a distributed internet map server. In *SSTD*, pages 387–403, 2001.

[10] A. Foundation. Apache module mod_deflate.

[11] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. The dedale prototype. In *Constraint Databases*, pages 365–382, 2000.

[12] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-temporal data handling with constraints. *GeoInformatica*, 5(1):95–115, 2001.

[13] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and quering moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.

[14] R. H. Güting, V. T. de Almeida, D. Ansorge, T. Behr, Z. Ding, T. Höse, F. Hoffmann, M. Spiekermann, and U. Telle. Secondo: An extensible dbms platform for research prototyping and teaching. In *ICDE*, pages 1115–1116, 2005.

[15] R. H. Güting, V. T. de Almeida, and Z. Ding. Modeling and querying moving objects in networks. *VLDB J.*, 15(2):165–190, 2006.

[16] X. Huang, C. S. Jensen, and S. Saltenis. Multiple k nearest neighbor query processing in spatial network databases. In *ADBIS*, pages 266–281, 2006.

[17] C. S. Jensen, J. Kolar, T. B. Pedersen, and I. Timko. Nearest neighbor queries in road networks. In *GIS*, pages 1–8, 2003.

[18] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient b+-tree based indexing of moving objects. In *VLDB*, pages 768–779, 2004.

[19] J. A. C. Lema, L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. Algorithms for moving objects databases. *Comput. J.*, 46(6):680–712, 2003.

[20] S. Olson. *Ajax on Java*. O'Really, 2007.

[21] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang. Indexing spatio-temporal data warehouses. In *ICDE*, pages 166–175, 2002.

[22] M. Pelanis, S. Saltenis, and C. S. Jensen. Indexing the past, present, and anticipated future positions of moving objects. *ACM Trans. Database Syst.*, 31(1):255–298, 2006.

[23] D. Pfoser and C. S. Jensen. Indexing of network constrained moving objects. In *GIS*, pages 25–32, 2003.

[24] L. Relly, A. Kuckelberg, and H.-J. Schek. A framework of a generic index for spatio-temporal data in concert. In *Spatio-Temporal Database Management*, pages 135–151, 1999.

[25] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *ICDE*, pages 422–432, 1997.

[26] L. Speicys, C. S. Jensen, and A. Kligys. Computational data modeling for network-constrained moving objects. In *GIS*, pages 118–125, 2003.

[27] J. Sun, D. Papadias, Y. Tao, and B. Liu. Querying about the past, the present, and the future in spatio-temporal databases. In *ICDE*, pages 202–213, 2004.

[28] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-temporal aggregation using sketches. In *ICDE*, pages 214–226, 2004.

[29] G. Trajcevski, O. Wolfson, H. Cao, H. Lin, F. Zhang, and N. Rishe. Managing uncertain trajectories of moving objects with domino. In *ICEIS*, pages 218–225, 2002.

[30] R. R. Vatsavai, S. Shekhar, T. E. Burk, and S. Lime. Umn-mapserver: A high-performance, interoperable, and open source web mapping and geo-spatial analysis system. In *GIScience*, pages 400–417, 2006.

[31] P. Vretanos. Ogc filter encoding implementation specification. Technical report, Open GIS Consortium, 2005.

[32] P. Vretanos. Ogc web feature service implementation. Technical report, Open GIS Consortium, 2005.

[33] Wikipedia. Reification.