# Continuous Queries within an Architecture for Querying XML-Represented Moving Objects

Thomas Brinkhoff and Jürgen Weitkämper

Institute of Applied Photogrammetry and Geoinformatics (IAPG)
Fachhochschule Oldenburg/Ostfriesland/Wilhelmshaven (University of Applied Sciences)
Ofener Str. 16/19, D-26121 Oldenburg, Germany
{Thomas.Brinkhoff,Weitkaemper}@fh-oldenburg.de

**Abstract.** The development of spatiotemporal database systems is primarily motivated by applications tracking and presenting mobile objects. Another important trend is the visualization and processing of spatial data using XML-based representations. Such representations will be required by Internet applications as well as by location-based mobile applications. In this paper, an architecture for supporting queries on XML-represented moving objects is presented. An important requirement of applications using such an architecture is to be kept informed about new, relocated, or removed objects fulfilling a given query condition. Consequently, the spatiotemporal database system must trigger its clients by transmitting the required information about the relevant updates. Such queries are called *continuous queries*. For processing continuous queries, we have to reduce the volume and frequency of transmissions to the clients. In order to achieve this objective, parameters are defined, which model technical restrictions as well as the interest of a client in a distinct update operation. However, delaying or even not transmitting update operations to a client may decrease the quality of the query result. Therefore, measures for the quality of a query result are required.

## 1. Introduction

The research activities in the field of *spatial database systems* have been shifted to the investigation of *spatiotemporal database systems*. Projects like "Chorochronos" [21] and a sequence of workshops (e.g. [10], [24], [25]) are indicators of this trend. The main topics of interest are the structure of time and space, data models and query languages, the efficient processing of queries by using adequate storage structures and database architectures, and the design of graphical user interfaces for spatiotemporal data. The investigation of spatiotemporal database systems is especially motivated by applications, which require to track and to visualize *moving objects*. Many of these applications originate from the field of *traffic telematics*, which combines techniques from the areas of telecommunication and computer science in order to establish traffic information and assistance services. Such applications ([33], [1]) require the management of moving objects, e.g., of vehicles of a fleet.

Another trend is the visualization and processing of spatial data via the Internet using XML. Current *geographical information systems (GIS)* or systems just announced (e.g. [9], [23]) (will) support the XML-based *Geography Markup Language (GML)* [15]. The next step will be the support of *mobile applications*. Now, the (intermediate) standard GPRS (General Radio Packet Service) is being launched in Europe. By introducing the new mobile telephone standard UMTS (Universal Radio Packet Service), this trend will be dramatically enforced. One common property of mobile applications is that they refer to locations and, therefore, require the transmission of spatial or spatiotemporal data. The appearance of mobile applications has also an impact on the devices used for presenting data: In addition to personal computers and workstations, personal digital assistants (PDAs) or mobile telephones are used as Internet clients. However, the computing power of such devices is rather restricted compared to traditional computers. In addition, the speed and throughput of wireless networks are limited and are subject to large variations.

The combination of these trends requires a suitable architecture for the XML-based representation and visualization of moving spatial objects. This architecture consists of the chain beginning at the client, which is responsible for the visualization of the spatial objects and for the user interaction, over suitable middleware up to the spatiotemporal database system. A special task of such an architecture is to support clients differing in the technology used for presenting and transmitting data in order to support traditional Internet applications as well as location-based mobile applications.

Applications tracking and presenting mobile objects require to be kept informed about new, relocated, or removed objects fulfilling a given query condition. Consequently, the spatiotemporal database system must trigger its clients by transmitting the necessary information about such update operations. The query, which causes this process, is called *continuous query* [28]. The result set of a continuous query is not only influenced by the update operations occurring in the database and by the given query condition, but – especially for mobile applications – also by the capability of the client to process the result. Typically, technical restrictions limit the computing power of the clients, the maximum resolution of spatial distances the client is able to distinguish, and the maximum speed and throughput of the connection between the database system and the client. Therefore, it is not advisable to transmit the complete result of a continuous query. Instead, a reasonable *filtering* must be performed. However, delaying or even not transmitting update operations to a client may decrease the quality of the query result. Therefore, indicators for the quality of the result of a continuous query are required.

The paper starts with a description of the architecture for querying XML-represented moving objects. The components of the architecture are discussed; a special attention is given to the integration of representations like GML and SVG (Scalable Vector Graphics). In section 3, a short definition of the data model describing the spatiotemporal objects is given. After a classification of update operations occurring in a spatiotemporal database system, a formal definition of the result set of the continuous query is presented in section 3.4. In order to reduce the volume and frequency of transmissions to the clients, corresponding parameters are defined and integrated into the architecture in section 4. Measures for the quality of the results after applying such restrictions are presented in section 4.4. Section 5 gives an overview of related work. Finally, the paper concludes with a summary and an outlook on future work.

## 2. Architecture for Querying XML-represented Moving Objects

In this section, an architecture for querying XML-represented moving spatial objects is presented. The diagram in figure 1 gives an overview of the basic components and the dataflow within this architecture. The single components are described in the rest of this section.

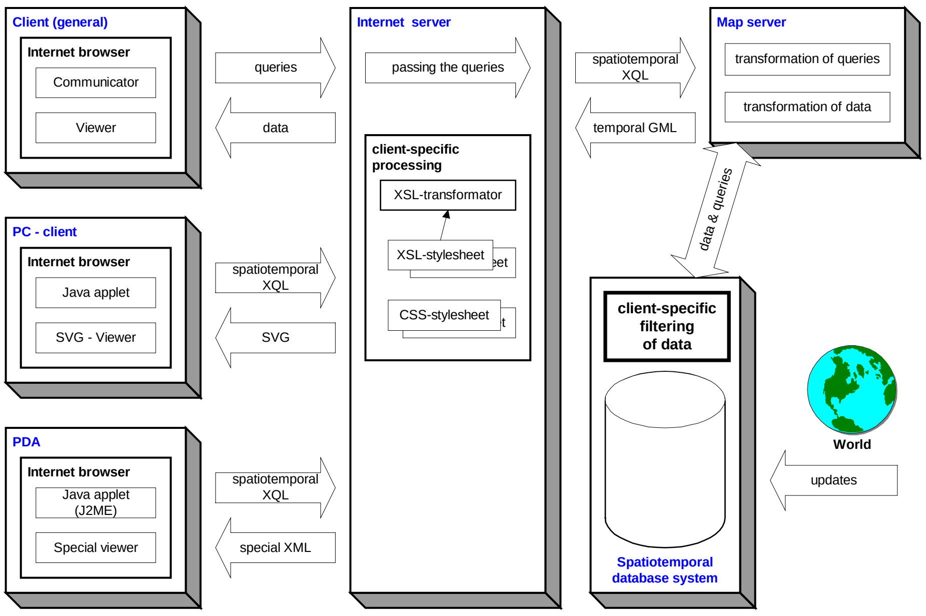

**Fig. 1:** An architecture for supporting an XML-based representation and visualization of moving spatial objects.

### 2.1 The Client

The main task of a client is the presentation of the data and the interaction with the user. From the point of view of a database system, the clients formulate the queries and take the query results. Another task of the client is the communication with the Internet server.

In the following, the visualization of moving spatial objects, typical queries, and the communication of the clients with the Internet server are discussed.

### 2.1.1 The Visualization

By using TV consoles, PDAs, or mobile telephones in addition to PCs and workstations as Internet clients, the properties of clients differ very much. This is caused by using different Internet browsers, operating systems, and hardware. These differences require using different techniques for presenting the data. Spatiotemporal data are primarily represented by vector data. However, there exists no general standard for representing vector data in the Internet. Typical solutions of this problem in the field of geographical information systems (GIS) are the following:

- The usage of raster maps, which are dynamically generated by the server site (see e.g. [5]). A restricted functionality of the client and a high data volume are the consequences.
- The second solution is using additional programs that extend the functionality of the Internet browser. Depending on the field, where the application is used, Active-X components, plug-ins, or applets can be found. Especially, occasional users of Internet applications presenting maps are deterred by such solutions [2].
- Comparable to the second solution is the use of proprietary data formats like Macromedia Flash [12], which have cartographic restrictions [14]. Furthermore, they are not standardized.

The goal is a data format for vector data, which is flexible as well as standardized. Therefore, an obvious solution is using an XML-based data representation. A suitable candidate is the graphical standard *SVG (Scalable Vector Graphics)*. Version 1.0 of SVG has currently the status of a candidate recommendation of the WWW consortium [32]. It has many features that allow representing and visualizing cartographic information [14]:

- SVG supports vector data as well as raster data. All essential geometric features are supported.
- SVG allows grouping objects, i.e., the definition of layers.
- Local coordinate systems are supported by transformations. Spatial objects can be clipped.
- An extensive and flexible formatting is supported by *cascading stylesheets (CSS2)*.
- Event handling can be embedded into a graphic by using scripts (JavaScript).

In order to use a graphical data format (and the corresponding viewers) for visualizing moving objects, update features with respect to the already visualized objects are required. Inserting new objects as well as removing and updating existing objects (especially with respect to their position but also with respect to their shape or other properties) are operations, which must be supported by the client (see also section 3.2). SVG allows such modifications by permitting a complete access to the corresponding *document object model (DOM)*. This object tree can be modified by embedded scripts (e.g. written in JavaScript). Figure 2 depicts an example where moving objects are visualized by an SVG viewer.

For standard Internet browsers it can be expected that they will soon support SVG without requiring additional plug-ins. However, for devices like PDAs or mobile telephones, this cannot be expected. Furthermore, the power of such "micro viewers" may be too limited for supporting moving objects. Therefore, the architecture must be flexible in order to support other data formats for such devices.

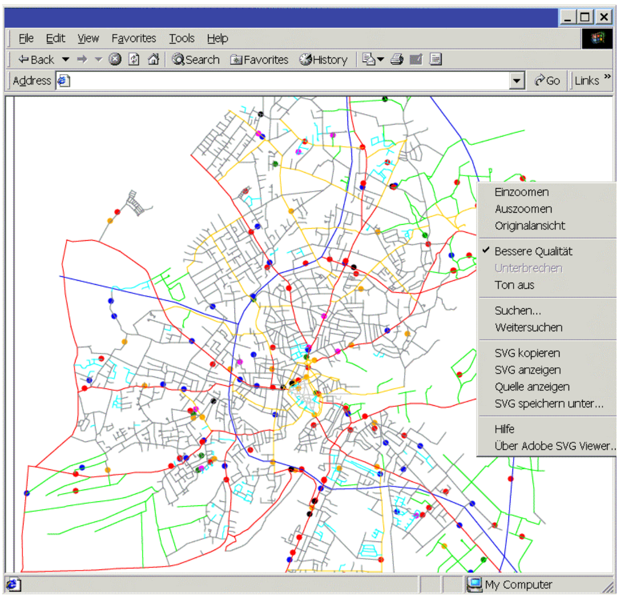

**Fig. 2:** Visualization of moving vehicles on the street map of the City of Oldenburg by Adobe's SVG viewer. The moving objects have been created by a data generator [3].

### 2.1.2 The Queries

In principle, we can distinguish the following types of queries:

- *Standard database queries* without any temporal or spatial query conditions.
- *Spatial queries* having one (or more) spatial query condition(s).
  This type of queries can be transformed into a set of basic spatial queries like window queries, point queries, and spatial joins. Well-known indexing and optimization techniques of spatial database systems can be used for performing this type of queries. For an overview see [7].
- A *spatiotemporal query* consists of temporal and of spatial query conditions.
  For optimizing such queries, a spatiotemporal database system is required. The result set of a spatiotemporal query typically consists of the objects existing at one distinct time stamp or at a period described by an interval of time stamps. Furthermore, the objects fulfill spatial conditions, e.g., they intersect a given query window. A spatiotemporal query may consist of additional query conditions without any spatial or temporal criteria. *Temporal queries* without any spatial query condition may also exist, but they are not typical of a spatiotemporal database system.

- For depicting moving objects, a variant of a spatiotemporal query is required: its temporal query condition consists of a time interval between the current time and the unlimited future. In this case, all current objects fulfilling the other query conditions are transmitted from the database system to the client. Then, a continuous update of the client is required: new, relocated or removed objects are detected by the database system and are - as soon as advisable or necessary – transmitted to the client. The treatment of such *continuous queries* is of special interest to this paper.

The formulation of spatiotemporal queries requires a foundation for representing the objects, their attributes and the query conditions. A fundamental work for the representation of spatiotemporal objects is the paper of Güting et al. [8] where spatiotemporal data types and operations are presented and embedded into a query language. An adaptation for a discrete data model can be found in [6]. For using such results in an XML-based environment, a corresponding XML notation is required. In figure 1, this is indicated by the notation „spatiotemporal XQL“.

### 2.1.3 The Communication with the Internet Server

The processing of the queries requires a communication with the Internet server. Typically, the transmission of data from the server to the client is initiated by the client. However, the processing of continuous queries requires server-initiated data transmissions. This is an untypical operation and cannot be found in comparable server architectures. A client does not know when and how many objects are changed in the spatiotemporal database. On the abstract level that means that the roles of the server and of the client have changed.

For updating clients typically server-push technology is used [13]. Such a pushing is based on a stable TCP connection and allows an update by transmitting complete HTML pages. One disadvantage of this technique is the high number of active connections to the server. For applications depicting spatiotemporal objects, transmitting complete pages is unsuitable because generally only a small part of the depicted objects has been modified between two updates. For example, in a map where moving vehicles or traffic jams are depicted, only some of these objects may change but not the rest or the background like the street map. A complete transmission would be a waste of time; especially for mobile applications, this solution cannot be applied.

A solution of this problem is to start a server thread at the client site. This can be done by a small applet, which registers the client at the Internet server. This mechanism allows an identification of the client after an HTTP request is finished. This identification, e.g. the IP address of the client and an assigned port number, are stored in an administrative database. In figure 1, the server thread at the client site is indicated by the “communicator” within the box of the Internet browser.

### 2.2 The Internet Server

The Internet server is a standard HTTP server handling the page requests of the clients. Additionally, it passes the queries of the clients to the map server and transforms the query results into the client-specific data representations considering particular style formats.

#### 2.2.1 Processing of the Query Results

As mentioned before, different types of clients may require different data formats. We assume that these representations are described by XML. On the other hand, the other components of the architecture should not handle several data formats in order to minimize their complexity. Instead, a uniform representation (also described by XML) is reasonable to use. For the representation of spatial objects, the OpenGIS consortium (OGC) proposed an XML-based specification, the *Geography Markup Language GML* [15]. In contrast to SVG, which is a format for displaying vector data, GML allows characterizing the properties of geographic features. The GML definition must be extended for representing properties, which are specific for moving spatial objects. This extension is called *temporal GML* in the following; it is the data format used for the data provided by the map server.

For converting temporal GML into another XML representation, the usage of XSL transformations is suitable. The transformation part of *XSL (Extensible Style Language) (XSLT)* [31] converts an XML document into another XML document by using XSL stylesheets. For supporting different data formats, it is only necessary that the Internet server provides suitable XSL stylesheets. An example for an XSL stylesheet converting GML into SVG is depicted in figure 3. The integration of the XSL transformers into the Internet server may be done by Java servlets, Java Server Pages (JSP), or by Active Server Pages (ASP).

In principle, it is possible to perform the XSL transformation on the client site, e.g., by the Internet browser. (For example, Microsoft's Internet Explorer shows quite reasonable results after updating Windows on the MSXML version 3.0.) However, for devices of low performance as well as for devices receiving data by (slow) wireless connections, a transformation at the server side is preferable. Additionally, by using different CSS stylesheets, a client-specific or even a user-specific formatting of the data can be achieved.

In comparison to binary data, XML documents considerably increase the required data volume. However, this problem is mostly compensated by high compression rates. Preliminary results show that a document describing a large set of spatiotemporal objects will be compressed by a factor of about 20 if a GML representation and ZIP compression are used. For SVG, a factor of about 7.5, and for a binary format, a factor of about 2.5 have been achieved.

### 2.3 The Map Server

The main task of a map server is to coordinate the co-operation of the database system(s) and the Internet server. In commercial solutions for managing spatial data, (one of) the database system(s) is often replaced by a GIS. The map server passes the queries to the relevant database system(s) and processes the query results. A spatiotemporal database system manages the data according to spatial and to temporal aspects. Therefore, it cannot be assumed that the data in the database is (hierarchically) represented by XML or that the database system can be queried by an XML-based query language. The map server transforms the queries into the database-specific query language and converts the results into temporal GML.

GML fragment describing a segment of a street (according to GML version 1.0)

```
<Feature typeName="StreetEdge3">
   <name>734407488</name>
   <property typeName="streetname" type="string">Heideweg</property>
   <geometricProperty typeName="location">
      <LineString><coordinates>8961,8839 8894,8710</coordinates></LineString>
   </geometricProperty>
</Feature>
```

Corresponding SVG fragment

```
<polyline class="c3" style="stroke-width:25;" points="8961,8839 8894,8710">
</polyline>
```

Part of the XSL stylesheet that transforms the GML fragment into a corresponding SVG fragment

```
<xsl:template match="featureMember [starts-with(@typeName,'StreetEdge')]">
  <xsl:element name="polyline">
    <xsl:attribute name="class">
      c<xsl:value-of select="substring-after(@typeName,'StreetEdge')"/>
    </xsl:attribute>
    <xsl:attribute name="style">
      stroke-width:<xsl:value-of select="$revscale"/>;
    </xsl:attribute>
    <xsl:attribute name="points">
      <xsl:value-of select="Feature/geometricProperty/LineString/coordinates"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```

**Fig. 3:** Example for transforming GML into SVG by using XSL transformations.

## 2.4 The Spatiotemporal Database System

The final component of the architecture depicted in figure 1 is the spatiotemporal database system. It stores the non-moving spatial objects as well as the moving objects. Our architecture supports especially querying spatial and spatiotemporal data; these queries are processed by the spatiotemporal database system.

The spatiotemporal database system serves one or more clients requesting data from the database. Changes of the database are performed by processes running outside of the described architecture (indicated by the symbol “World” in figure 1). Database triggers are used for initiating the update of the clients.

According to Sellis [21], the main topics of interest in the area of spatiotemporal databases are related to the structure of time and space, to data models and query languages, to the efficient processing of queries by using adequate storage structures and databases architectures, and to the design of graphical user interfaces for spatiotemporal data. An important aspect of the implementation of spatiotemporal database systems has been the development of suitable storage structures and indexing techniques for optimizing the processing of spatiotemporal queries; examples for such publications are [29], [30], [11], and [17].

In the rest of this paper, we will concentrate the discussion on the continuous query.

## 3. Continuous Queries

After a short definition of the model used in this paper for describing spatiotemporal objects, a classification of the update operations and the definition of the result set of a continuous query are presented in this section.

### 3.1 The Model of Spatiotemporal Objects

The following discussion assumes an architecture consisting of a spatiotemporal database system, which stores the positions and other attributes of moving objects. In a temporal database, *valid time* and *transaction time* are distinguished. The valid time describes the time when the data were valid or are valid in the modeled reality. The transaction time is the time, when the data were committed in the database system.

In the following, we assume that a moving object *obj* having an object identifier *obj.id* is described by a sequence of records $obj_i$ consisting of a spatial location $obj_i.loc$ (short: $loc_i$), a time stamp $obj_i.time$ (short: $time_i$) giving the beginning of the valid time, a time stamp $obj_i.trTime$ (short: $trTime_i$) giving the transaction time, and non-spatiotemporal attributes $obj_i.attr$ (short: $attr_i$) ($i \in \mathbb{N}$). A *time stamp t* is represented by a natural number ($t \in \mathbb{N}$). If the records $obj_i$ und $obj_{i+1}$ exist, the valid time of record $obj_i$ corresponds to the time interval $[time_i , time_{i+1})$. If no record $obj_{i+1}$ exists for a record $obj_i$, the record $obj_i$ is the current state of the corresponding moving object from the point of view of the database system. Furthermore, a final record $obj_i$ may exist with $i > j$ for all records $obj_j$ of this spatiotemporal object. A final record $obj_i$ indicates the end of the lifetime of the object, e.g., by an empty location. In order to simplify the discussion, we assume that $time_i \leq trTime_i$ holds.

This form of representing spatiotemporal objects is only one option; it can be replaced by another model because of conceptual reasons [8] or because of implementation purposes [6].

#### 3.1.1 Describing Spatiotemporal Objects in GML

The building blocks of GML are so-called *simple features*, which "describe the geography of entities of the real world" [15]. A feature in GML (version 1.0) may consist of a name, a description, a bounding box, several (alphanumeric) properties, and several geometric properties. For realizing the above data model in GML, we have to represent those attributes by related elements. The object identifier can be mapped to the element `name` and the locations can be described by the geometric properties of GML. The attributes *time* and *trTime* are of special interest. One possibility is to represent them by (alphanumeric) properties. A property element is defined as follows in GML:

```
<!ELEMENT property (#PCDATA)>
<!ATTLIST property
   typeName   CDATA #REQUIRED
   type ( boolean | integer | real | string ) "string" >
```

That means we can define the types of properties without changing or extending the DTD (document type definition) of GML (version 1.0).

### 3.2 Types of Update Operations

At one time stamp, one or more objects may be updated. We can distinguish three types of updates concerning a spatiotemporal object *obj*:

1. The insertion of a new object:
   In this case, no record having the same object identifier *obj.id* existed before.
2. The modification of an existing object:
   In the general case, any attribute of the object may be changed, i.e. at least one record having the object identifier *obj.id* and an older time stamp is already stored in the database. In this paper, we are especially interested in relocated objects. Using the model described in section 3.1, a modification requires the insertion of a new record into the database.
3. The deletion of an existing object:
   In this case, at least one record having the object identifier *obj.id* with an older time stamp is already stored in the database. The end of the lifetime of the object is indicated by inserting a corresponding record into the database.

In principle, each of these update operations must notify all concerned clients in order to update them.

### 3.3 Classification of Update Operations

A simple solution for performing a continuous query is to inform each client about every update. However, this is a very inadequate and expensive solution. Especially for the case where the client is an Internet client, such a solution should not be considered.

Determining the interest of a client in an update operation requires the evaluation of the object representation at two consecutive time stamps. This leads to the following classification of updates. An overview of this classification is given in figure 4.

**Class 1: updates of high interest**

This class of updates represents operations, which are of high interest to a client, because they represent essential changes. The class consists of the following update operations:

- (I1) Insertion of a new object fulfilling the query condition of the client.
- (D1) Deletion of an object, which has fulfilled the query condition of the client at the previous time stamp.
- (I2) Modification of an existing object which fulfills the query condition of the client now but which has not fulfilled the query condition at the previous time stamp.
- (D2) Modification of an existing object which does not fulfill the query condition of the client now but which has fulfilled the query condition at the previous time stamp.

With respect to the client, the two operations (I1) and (I2) are insert operations and the two other operations (D1) and (D2) are delete operations.

**Class 2: updates of no interest**

This class consists of the updates, which are of no interest for a client:

- Insertion of a new object not fulfilling the query condition of the client.
- Deletion of an object, which has not fulfilled the query condition of the client at the previous time stamp.
- Modification of an existing object which neither fulfills the query condition of the client now nor at the previous time stamp.

A client should not be informed about updates of class 2. Therefore, a component must exist, which eliminates these operations with respect to distinct clients.

**Class 3: updates of medium interest**

There exists one operation not belonging to the two classes described above:

- (M1) Modification of an existing object, which fulfills the query at the previous time stamp as well as now.

With respect to a client, operations of this class are (the only) operations modifying an existing object. In general, the number of these updates considerably exceeds the number of operations of high interest. A client is of course interested to be informed about these modifications. However, if in the sequence of modifications of low relevance some are skipped, this will be acceptable for a client. Therefore, the operations (M1) are called *updates of medium interest* in the following.

| fulfills query? | | operations | | |
|---|---|---|---|---|
| previous record | current record | insertion | deletion | modification |
| no | no | - | - | - |
| no | yes | I1 | - | I2 |
| yes | no | - | D1 | D2 |
| yes | yes | I1 | D1 | M1 |

**Fig. 4:** Types of update operations.

### 3.4 Definition of the Result of a Continuous Query

Using the model of spatiotemporal objects of section 3.1 and the classification described above, we can define the *result* of a continuous query $q$ with a query condition $c$ for a client. In the following, $ts$ denotes the database time when the query is made, $ts+n$ the time when the execution of the query is stopped, and $res_k$ is the result of the query at database time $ts+k$:

(1) $res_0 = \{ (obj_i, \text{I1}) \mid obj_i \text{ fulfills } c \wedge time_i \leq ts \wedge (\forall\ time_j \text{ with } j \neq i \Rightarrow time_j < time_i)\}$

$res_k = \{ (obj_i, \text{I1}) \mid obj_i \text{ fulfills } c \wedge trTime_i = ts+k \wedge (i = 0) \} \cup$

$\{ (obj_i, \text{I2}) \mid obj_i \text{ fulfills } c \wedge \neg(obj_{i-1} \text{ fulfills } c) \wedge trTime_i = ts+k \} \cup$

$\{ (obj_i, \text{D1}) \mid obj_i \text{ indicates end of lifetime} \wedge obj_{i-1} \text{ fulfills } c \wedge trTime_i = ts+k \} \cup$

$\{ (obj_i, \text{D2}) \mid \neg(obj_i \text{ fulfills } c) \wedge obj_{i-1} \text{ fulfills } c \wedge trTime_i = ts+k \} \cup$

$\{ (obj_i, \text{M1}) \mid obj_i \text{ fulfills } c \wedge obj_{i-1} \text{ fulfills } c \wedge trTime_i = ts+k \}$ for $k \in \{1, \ldots, n\}$

The set $res_0$ corresponds to the initial result of the continuous query. The results $res_k$ ($k$>0) contain the incremental updates and should be transmitted to a client at database time $ts+k$. However, this is an unrealistic assumption. Therefore, this definition is only a set target that cannot be achieved. In section 5, we will use this definition for defining a measure of quality. Suitable parameters for restricting the results of a continuous query are presented in the next section.

## 4. Filtering the Results of a Continuous Query

Technical restrictions limit the computing power of the clients, the maximum resolution of spatial distances the client is able to distinguish, and the maximum speed and throughput of the connection between the database system and the client. Therefore, it not advisable to transmit each result set $res_k$ of a continuous query. Instead, a reasonable filtering must be performed.

In order to regulate this filtering, parameters are required for controlling the process dependent of the (type of) client. In this section, parameters restricting the time and the space dimension are presented. Then, a measure of interest is discussed, which allows indicating the interest of a client in an update operation. Finally, a component for filtering the query results is integrated into our architecture.

### 4.1 Restricting Parameters

The general demand of a client is to be informed about any interesting update as soon as possible. However, as mentioned before technical restrictions may prevent to fulfill such a request. For continuous queries, restrictions with respect to the time and to the spatial dimension are of special interest.

### 4.1.1 Time-oriented Parameters

A client is generally connected to the database system by a network. Using the Internet or a wireless communication, the data volume, which can be transmitted, is restricted. A frequent update of a client using a slow connection and / or a connection of restricted capacity is unnecessary. Quite the reverse, too fast or too voluminous data transmissions lead to data jams preventing a contemporary update of the client. Another aspect is the computing power of the client. Thus, the number of updates to be sent is restricted. On the other hand, a minimum data volume may exist which should be sent in order to use the capacity of the data connection and/or of the client.

The following three parameters describe these restrictions:

- In order to simplify the considerations it is assumed that all operations require the same transmission volume and the same computing power by the client. Then, the parameter *maxOps* describes the number of update operations, which can be sent to a client per a single data transmission at maximum. Another parameter *minOps* denotes the minimum number of operations reasonable be sent. It holds: $minOps \leq maxOps$.
- Furthermore, in order to support, e.g., wireless data connections, we assume that there exists a minimum time period *minPeriod* between two transmissions to a client. If update operations were sent to the client at database time *t*, the next operations should not be sent before *t+minPeriod*.

### 4.1.2 Space-oriented Parameters

Applications depicting spatial data have in general a maximum *resolution*, i.e. there exists (in the two-dimensional case) a minimum distance which is distinguishable or of relevance. If the movement of an object is smaller than the minimum x-distance $minDist_x$ and the minimum y-distance $minDist_y$, it is unnecessary to inform the client about this movement. Using these two distances, we can define the function *isRelevant* as follows:

$$(2)\quad isRelevant\ (obj_{prev}, obj_{curr}) := \begin{array}{l} x\text{-}distance\ (loc_{curr}, loc_{prev}) \geq minDist_x \ \vee \\ y\text{-}distance\ (loc_{curr}, loc_{prev}) \geq minDist_y \end{array}$$

All the parameters described above differ for distinct (types of) clients.

## 4.2 Measure of Interest

In the former section, we defined parameters in order to filter the operations to be sent to the client. Consequently, we must select a subset of operations to be transmitted from the set of all update operations of interest. For performing this filtering, we need a selection criterion. In section 3.3, we have already classified the operations in two relevant subsets: class 1 consisting of operations of high interest and class 3 of operations of medium interest.

For class 1, we can formulate the following requirement: A client should be informed about the update operations of class 1 as soon as possible. If not all update operations of class 1 can be sent to a client, the transmission should start with the operations having the oldest time stamps (with respect to the valid time). The other operations of this class must be buffered by the database system and are transmitted (if possible) by the next transmission(s).

Typically, the number of updates of class 3 considerably exceeds the number of operations of high interest. Therefore, it is necessary to consider the update operations of class 3 in more detail. The interest of a client in such an update depends on different factors.

In order to describe the interest of a client in an update operation, we need a *measure of interest*. The interest depends especially on the last description the client has received. This object description is called $obj_{last}$ in the following. The current description of the object is called $obj_{curr}$. The measure of interest $intr(obj_{last},obj_{curr})$ can use the attributes *time*, *loc*, and *attr* representing the valid time, the location, and the non-spatiotemporal attributes of the object description, respectively. The measure of interest may change in an arbitrary manner between two succeeding update operations.

For example, a measure of interest may be computed as follows (*dist* denotes the Euclidean distance):

$$(3) \quad intr(obj_{last},obj_{curr}) := (time_{curr} - time_{last}) + w_{loc} * dist\,(loc_{curr}, loc_{last})$$

A suitable factor $w_{loc}$ is required for equalizing the influence of time and space.

### 4.3 Integration of a Component for Filtering Objects

Computing the measure of interest requires an efficient computation of the last object descriptions the client has received for deciding, which update operations of class 3 are of relevance to a client. If the relevance exceeds a given threshold, the object representation will be sent. If the number of records exceeds the maximum allowed number, some operations have to be buffered..

The component responsible for determining the objects to be transmitted and for buffering the candidates, which may be sent at the next transmission(s), is the *filtering component* of our architecture. In figure 1, it is depicted as a part of the spatiotemporal database system.

## 5. Measure of Quality

A measure of quality can be used for two purposes:

1. For determining the quality of a filtering algorithm. Then, an assessment is possible which of two or more algorithms shows the best behavior in the same situation.
2. For controlling the behavior of an algorithm running in the filter component. If the quality becomes too low, for example, the parameters presented in section 4.1 may be changed.

## 5.1 Definitions of Restricted Result Sets

For the definition of a measure of quality, two further result sets of a continuous query are defined. A sequence $res'_i$ will be called a *complete result* of a continuous query $q$ if the following conditions hold:

(4) $res'_0 \subseteq res_0$

$res'_k \subseteq \{\, (obj_i, \mathrm{I1}) \notin RES'(k) \mid obj_i \text{ fulfills } c \wedge ts \leq trTime_i \leq ts{+}k \wedge (\mathrm{i} = 0) \,\} \cup res_0 \cup$

$\{\, (obj_i, \mathrm{I2}) \notin RES'(k) \mid obj_i \text{ fulfills } c \wedge \neg(obj_{i-1} \text{ fulfills } c) \wedge ts \leq trTime_i \leq ts{+}k \,\} \cup$

$\{\, (obj_i, \mathrm{D1}) \notin RES'(k) \mid obj_i \text{ indicates eol} \wedge obj_{i-1} \text{ fulfills } c \wedge ts \leq trTime_i \leq ts{+}k \,\} \cup$

$\{\, (obj_i, \mathrm{D2}) \notin RES'(k) \mid \neg(obj_i \text{ fulfills } c) \wedge obj_{i-1} \text{ fulfills } c \wedge ts \leq trTime_i \leq ts{+}k \,\} \cup$

$\{\, (obj_i, \mathrm{M1}) \notin RES'(k) \mid obj_i \text{ fulfills } c \wedge obj_{i-1} \text{ fulfills } c \wedge ts \leq trTime_i \leq ts{+}k \,\}$

for $k \in \{1, \ldots, n'\}$ with $n' \geq n$

with $\bigcup_{m=0}^{n} res_m = \bigcup_{m=0}^{n'} res'_m$ and

$(obj_i,\mathrm{op_i}) \in res'_k \Rightarrow (obj_j,\mathrm{op_j}) \notin RES'(n'{+}1) \vee (obj_j,\mathrm{op_j}) \in RES'(k)$ for $j < i$

where $RES'(k)$ is defined by $RES'(k) := \bigcup_{m=0}^{k-1} res'_m$

Again, the set $res'_0$ corresponds to the initial result of a continuous query and the results $res_k$ ($k$>0) contain the incremental updates, which are transmitted to the client at database time $ts+k$. The union of all subsets $res_k$ of definition (1) is identical to the union of all subsets $res'_k$ of definition (4). Furthermore, the order of operations ($obj_i$,op) is not altered for a spatiotemporal object *obj*.

A sequence $res''_i$ will be called a *consistent partial result* of a continuous query $q$ if the following conditions hold ($res'_k$ is defined in equation (4)):

(5) $res''_k \subseteq' res'_k$ for $k \in \{0, \ldots, n'\}$

with $(obj_i,\mathrm{op_i}) \in res''_k \Rightarrow (obj_i,\mathrm{op_i}) \in res'_k \vee$

$(\mathrm{op_i} \in \{\mathrm{I1,I2}\} \wedge (obj_i,\mathrm{M1}) \in res'_k \wedge (obj_j,\mathrm{op_i}) \in res'_j \text{ with } j<i) \vee$

$(\mathrm{op_i} = \mathrm{M1} \wedge (obj_i,\mathrm{op'_i}) \in res'_k \wedge (obj_j,\mathrm{op_j}) \in res'_j \text{ with } j<i,\ \mathrm{op'_i} \in \{\mathrm{I1,I2}\},$

$\mathrm{op_j} \in \{\mathrm{D1,D2}\})$

and $(obj_i,\mathrm{op_i}) \in res''_k \Rightarrow (obj_j,\mathrm{op_j}) \notin RES''(n'{+}1) \vee (obj_j,\mathrm{op_j}) \in RES''(k)$ for $j < i$

and $(obj_i,\mathrm{op_i} \in \{\mathrm{D1,D2,M1}\}) \in res''_k \Rightarrow \exists\, (obj_j,\mathrm{op_j} \in \{\mathrm{I1,I2}\}) \in RES''(k)$ with $j<i$

and $(obj_i,\mathrm{op_i} \in \{\mathrm{I1,I2}\}) \in RES''(n'{+}1) \Rightarrow$

$\exists\, (obj_j,\mathrm{op_j} \in \{\mathrm{D1,D2}\}) \in RES''(n'{+}1) \vee obj_j \text{ fulfills } c \text{ at } ts{+}k$ with $j > i$

where $RES''(k)$ is defined by $RES''(k) := \bigcup_{m=0}^{k-1} res''_m$

In the result set of (5) some operations may miss – in the extreme case all subsets are empty. However, it is guaranteed that for each modification or deletion a previous insert operation exists and that for each object existing in the result set also a delete operation is included if the object has been deleted in the database before *ts+n'*. The operator $\subseteq$' is used instead of $\subseteq$ for supporting the situations that an insert operation and a succeeding modification are combined to one insert operation or that a pair of succeeding delete and insert operations is combined to one modification. These cases are covered by the first implication of (5).

## 5.2 Measure of Quality

As mentioned in section 3.4, the result of a continuous query can be used for defining a measure of quality. Comparing the real result with that result set, some update operations may be sent to the client after some delay. In this case, we can define the *average delay* as follows:

$$\text{(6)}\quad \text{avDelay} := \left( \sum_{k'=0}^{n'} \sum_{(obj_i, op) \in res'_{k'}} k' - k \; with (obj_i, op) \in res'_k \right) / \left| \bigcup_{k'=0}^{n'} res'_{k'} \right|$$

In the consistent partial result, some operations (*obj*$_i$,op$_i$) may miss. The larger the temporal and the spatial distance of (*obj*$_i$,op$_i$) to its “neighbored” operations (*obj*$_{i-1}$,op$_{i-1}$) and (*obj*$_{i+1}$,op$_{i+1}$), the worse is omitting such an operation. Therefore, the temporal and the spatial distances are squared in the following definition of the average *omitting degree (omitDeg)*, which is the sum of a time component *(timeOD)* and a distance component *(distOD)*.

$$\text{(7)}\quad timeOD := \sum_{k=0}^{n} \sum_{(obj_i, op) \in res_k \, with (obj_i, op) \notin RES''} \frac{(time_i - time_{i-1})^2 + (time_i - time_{i+1})^2}{4 \cdot timeSum}$$

$$distOD := \sum_{k=0}^{n} \sum_{(obj_i, op) \in res_k \, with (obj_i, op) \notin RES''} \frac{dist(loc_i, loc_{i-1})^2 + dist(loc_i, loc_{i+1})^2}{4 \cdot distSum}$$

$$omitDeg := timeOD + distOD$$

with
$$timeSum := \sum_{k=0}^{n} \sum_{(obj_i, op) \in res_k \, with (obj_i, op) \in RES} (time_i - time_{i+1})^2$$

$$distSum := \sum_{k=0}^{n} \sum_{(obj_i, op) \in res_k \, with (obj_i, op) \in RES} (dist(loc_i, loc_{i+1}))^2$$

$$RES := \bigcup_{k=0}^{n} res_k \text{ and } RES'' := \bigcup_{k=0}^{n'} res''_k$$

*dist* denotes the Euclidean distance between two locations. The divisors *timeSum* and *distSum* are used for normalizing the distances. If no neighbored operation exists, the distance to it will be defined as 0. If an object is completely missing in *RES”*, all corresponding operations are aggregated in *omitDeg*. The complete result of a continuous query has the omitting degree 0. An empty result set would have an omitting degree of 1.

## 6. Related Work

Terry et al. [28] presented the notion of "continuous queries" in the field of database systems. They proposed an incremental evaluation approach in order to avoid repetitive computations. In the NiagaraCQ [4], also performance issues of the database system have been the target. In order to support millions of queries, similar query conditions are grouped, which allows sharing common computations and reducing the I/O cost.

The presented work is also related to the field of spatiotemporal database systems. In general, they have been discussed already in section 2.4. This section refers to more specific papers with respect to continuous queries.

Wolfson et al. [33] address in their paper the problem to determine when the location of a moving object in database should be updated: a *dead-reckoning update policy* dictates that there is a deviation for which an object (e.g. a car with a GPS receiver) should send a location or speed update to the database. In some sense, this problem is complementary to continuous queries. In [27], three types of queries (instantaneous queries, continuous queries and persistent queries) are distinguished for the case of having dynamic spatial attributes, i.e. the attribute changes continuously as a function of time, without being explicitly updated.

Because the transmission of updates is triggered by the database system, the area of *active database systems* must be considered. Paton and Díaz give a good overview of this topic in [16]; they close their article with the remark that further research is required (among others) on the integration of active behavior with temporal facilities. With respect to this topic, the paper of Sistla and Wolfson [26] presents two languages for specifying temporal triggers; one of them handles temporal logic of future events. However, special mechanisms for filtering events are not discussed in this publication. The focus of Ramamritham et al. in their paper on integrating temporal, real-time, and active databases [18] is on recovery and transaction issues.

Besides, the work of Rosenblum and Wolf [20] should be mentioned. In their framework for Internet-scale event observation and notification, they demand a filter policy within their observation model, however, without giving any details.

For designing and implementing algorithms used by the filtering component, attention should be given to the work on incremental maintenance of materialized views where similar problems occur, see e.g. [19] and [22].

## 7. Conclusions

In this paper, an important query in the field of spatiotemporal database systems – the continuous query – has been investigated. The work was motivated by applications tracking and presenting mobile objects in an Internet environment where different types of clients including mobile devices are used. For such applications querying moving spatial objects, an architecture has been proposed, which supports an XML-based data representation. This architecture has been the base for discussing continuous queries.

After a classification of the update operations in a spatiotemporal database system, a formal definition of the result set of continuous queries has been presented. In order to reduce the volume and frequency of transmissions to the clients, parameters have been defined in order to model technical restrictions as well as the interest of a client in a distinct update operation. A component for performing such a filtering has been integrated into our architecture. However, delaying and not transmitting update operations to a client mean to decrease the quality of the query result. Therefore, two quality measures have been presented.

The next step will be the development on efficient algorithms for performing the filtering. A special attention will be given to maintain the quality of the query results. A further requirement to such algorithms concerns their scalability for supporting large number of clients.

The definition of continuous queries and the XML-based representation of spatiotemporal objects were based on a quite simple model of spatiotemporal objects. Therefore, future work should cover a definition using a more expressive data model. Especially, the support of motion described by a motion vector instead of a constant object position must be investigated.

Another aspect is the behavior of the restricting parameters. The resolution of a client may be changed by performing a zoom operation and the parameters *minOps*, *maxOps* and *minPeriod* may be affected by the traffic of other users of the network connection or by a changed capacity of the connection. For example, using the new mobile telephone standard UMTS, the maximum speed of a connection will depend on the distance of the mobile telephone to the next base station. Therefore, filter algorithms are required, which observe varying parameters.

## 8. References

1. Brinkhoff T.: “Requirements of Traffic Telematics to Spatial Databases”. *Proceedings 6th International Symposium on Large Spatial Databases*, 1999, Hong Kong, China, in: Lecture Notes in Computer Science, Vol.1651, Springer, 365-369.
2. Brinkhoff T.: “The Impacts of Map-Oriented Internet Applications on Internet Clients, Map Servers and Spatial Database Systems”, *Proceedings 9th International Symposium on Spatial Data Handling*, Beijing, China, 2000.
3. Brinkhoff T.: “Generating Network-Based Moving Objects”, *Proceedings 12th International Conference on Scientific and Statistical Database Management,* Berlin, Germany, 2000, 253-255.
4. Chen J., DeWitt D.J., Tian F., Wang Y.: „NiagaraCQ: A Scalable Continuous Query System for Internet Databases“, *Proceedings ACM SIGMOD International Conference on Management of Data,* Dallas, TX, 2000, 379-390.
5. ESRI Inc.: "ArcView Internet Map Server", Nov. 1999, `http://www.esri.com/software/arcview/extensions/imsext.html`
6. Forlizzi L., Güting R.H., Nardelli E., Schneider M.: “A Data Model and Data Structures for Moving Objects Databases”, *Proceedings ACM SIGMOD International Conference on Management of Data,* Dallas, TX, 2000, 319-330.

7. Gaede V., Günther O.: “Multidimensional Access Methods”. *ACM Computing Surveys*, Vol. 30, No. 2, 1998, 170-231.
8. Güting R.H., Böhlen M.H., Erwig M., Jensen C.S, Lorentzos N.A., Schneider M., Vazirgiannis M.: “A Foundation for Representing and Querying Moving Objects”, *ACM Transactions on Database Systems*, Vol. 25, No.1, 1-42, 2000.
9. Intergraph Corp.: „Intergraph Announces GeoMedia 4.0 Product Suite for Enterprisewide GIS”, Nov. 2000, `http://www.intergraph.com/gis/newsroom/press00/gm40_rlsf.asp`
10. Seminar Integrating Spatial and Temporal Databases, Schloss Dagstuhl, Wadern, Germany, November 22-27, 1998.
11. Kollios G., Gunopulos D., Tsotras V.J.: "On Indexing Mobile Objects", *Proceedings ACM Symposium on Principles of Database Systems*, Philadelphia, PA, 1999, 261-272.
12. Macromedia Inc.: “Macromedia Flash“, 2000, `http://www.macromedia.com/software/flash/`
13. Netscape: „An Exploration of Dynamic Documents”, `http://home.netscape.com/assist/net_sites/pushpull.html`
14. Neumann A., Winter A.: „Kartographie im Internet auf Vektorbasis, mit Hilfe von SVG nun möglich“, *carto:net*, Version 2.0, 09/2000, `http://www.carto.net/papers/svg/index.html`
15. OpenGIS Consortium: “Geography Markup Language (GML)”, `http://www.opengis.org/techno/specs.htm`
16. Paton N.W, Díaz O.: “Active Database Systems”, *ACM Computing Surveys*, Vol. 31, No. 1, 1999, 63-103.
17. Pfoser D., Jensen C.S., Theodoridis Y.: „Novel Approaches to the Indexing of Moving Object Trajectories“, *Proceedings 26th International Conference on Very Large Databases*, Cairo, Egypt, 2000, 395-406.
18. Ramamritham K., Sivasankaran R., Stankovic J.A., Towsley D.F., Ming Xiong: “Integrating Temporal, Real-Time, and Active Databases”, *ACM SIGMOD Record*, Vol. 25, No.1, 1996, 8-12.
19. Roussopoulos N.: „An Incremental Access Method for ViewCache : Concept, Algorithms, and Cost Analysis ”, *ACM Transactions on Database Systems*, Vol. 16, No. 3, Sept. 1991, 535-563.
20. Rosenblum D.S., Wolf A.L.: “A Design Framework for Internet-Scale Event Observation and Notification”, *Proceedings 6th European Conference on Software Engineering* held jointly with the 5th ACM SIGSOFT Symposium on Software Engineering, 1997, 344-360.
21. Sellis T.: “Research Issues in Spatio-temporal Database Systems”. *Proceedings 6th International Symposium on Large Spatial Databases*, Hong Kong, China, 1999. In: Lecture Notes in Computer Science, Vol.1651, Springer, 5-11.
22. Staudt M., Jarke M.: „Incremental Maintenance of Externally Materialized Views“,*Proceedings 22nd International Conference on Very Large Databases*, Bombay, India, 1996, 75-86.
23. Smallworld: “Smallworld Internet Application Server: Product Overview”, Nov. 2000, `http://www.gesmallworld.com/english/products/internet/SIAS.pdf`
24. Workshop on Spatio-Temporal Database Management, Edinburgh, Scotland, September 10-11, 1999.
25. Workshop on Spatio-Temporal Data Models and Languages, Florence, Italy, August 30-31, 1999.
26. Sistla A.P., Wolfson O.: “Temporal Triggers in Active Databases”. *IEEE Transactions on Knowledge and Data Engineering*, Vol.7, No. 3, 1995, 471-486.
27. Sistla A.P., Wolfson O., Chamberlain S., Dao S.: "Modeling and Querying Moving Objects", *Proceedings 13th International Conference on Data Engineering*, Birmingham, UK, 1997, 422-432.

28. Terry D., Goldberg D., Nichols D., Oki B.: „Continuous Queries over Append-Only Databases“, *Proceedings ACM SIGMOD International Conference on Management of Data*, San Diego, CA, 1992, 321-330.
29. Theodoridis Y., Sellis T., Papadopoulos A.N., Manolopoulos Y.: "Specifications for Efficient Indexing in Spatiotemporal Databases", *Proceedings 10th International Conference on Scientific and Statistical Database Management*, Capri, Italy, 1998, 123-132.
30. Vazirgiannis M., Theodoridis Y., Sellis T.: “Spatio-Temporal Composition and Indexing for Large Multimedia Applications”, *ACM Multimedia Systems*, Vol. 6, No. 5, 1998.
31. World Wide Web Consortium: "EXtensible Markup Language, W3C Recommendation", Febr. 1998, `http://www.w3.org/TR/REC-xml`
32. World Wide Web Consortium: "Scalable Vector Graphics (SVG) 1.0 Specification, W3C Candidate Recommendation", Aug. 2000, `http://www.w3.org/TR/2000/CR-SVG-20000802/`
33. Wolfson O., Sistla A.P., Chamberlain S., Yesha Y.: "Updating and Querying Databases that Track Mobile Units", *Distributed and Parallel Databases*, Vol. 7, No. 3, 1999, 257-287.