OCEANUS: A Spatio-Temporal Data Stream System Prototype

Zdravko Galić
Faculty of Electrical Engineering and Computing
Department of Applied Computing
University of Zagreb
Unska 3, 1000 Zagreb, Croatia
zdravko.galic@fer.hr

Krešimir Križanović
Faculty of Electrical Engineering and Computing
Department of Applied Computing
University of Zagreb
Unska 3, 1000 Zagreb, Croatia
kresimir.krizanovic@fer.hr

Emir Mešković
Faculty of Electrical Engineering
University of Tuzla
Franjevačka 2, 75000 Tuzla, BiH
emir.meskovic@untz.ba

Mirta Baranović
Faculty of Electrical Engineering and Computing
Department of Applied Computing
University of Zagreb
Unska 3, 1000 Zagreb, Croatia
mirta.baranovic@fer.hr

ABSTRACT

Recent advances in wireless communication, miniaturization of spatially enabled devices and global navigation satellite systems (GNSS) services have resulted in a large number of novel application domains. Applications in these novel domains (moving objects tracking, sensor networks, fleet management, real-time intelligent transportation systems, etc.) process huge volume of continuous streaming data, i.e. data that is produced incrementally over time, rather than being available in full before processing. Data stream management systems (DSMS) have been developed to manage continuous data streams. Usually based on relational paradigm, they have rudimentary support for spatial data. Recent research efforts in data stream management systems focus mainly on processing continuous queries over traditional data streams, and only a few papers addressed spatio-temporal continuous queries. In this paper we present OCEANUS, an ongoing effort to extend TelegraphCQ DSMS with spatial support providing a platform for spatio-temporal streaming applications.

Categories and Subject Descriptors

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

ACM SIGSPATIAL IWGS'12, November 6, 2012. Redondo Beach, CA, USA

General Terms

Design, Management, Languages

Keywords

Data stream management, Continuous queries, Spatio-temporal data streams, Moving objects

1. INTRODUCTION

Several research projects provided prototype systems that manage large and possibly unbounded data streams and continuous queries as well as common relations and one-time queries. Recent research efforts in data stream management systems focus mainly on processing continuous queries over traditional data streams, based on simple relational paradigm. While spatio-temporal DSMS are still in a research phase, there are several commercial relational data streaming systems and even some standard DBMS [2] offer data stream support. These DSMSs allow continuously adaptive query processing and support handling of a large number of continuous queries posed over large and varying data streams.

However, spatial and temporal properties of both data streams and continuous queries are disregarded and most of current DSMSs offer only rudimentary support for spatio-temporal data. Hence, a new approach in which objects' movements are considered as continuous, time varying and possibly unbounded data streams has been introduced.

In a DSMS, data arrives in form of concurrent and continuous data streams. Queries on these data streams are typically continuous monitoring queries, involving both data streams and persistent relations, and emitting streaming data in real time as results. Data streams have several significant characteristics different from those of traditional relations [10]:

(i) They are sequences of records, ordered by arrival time

or by another ordered attribute such as generation time (which is likely to be correlated with, but not equivalent to, the arrival time). These records (or tuples) arrive in the system over a period of time instead of being available *a priori*.

(ii) They are produced by a variety of external sources, meaning that a DSMS has no control over the arrival order or the rate at which data arrives.

(iii) They are produced continually and, therefore, have unbounded or unknown length. Thus, a DSMS may not know if or when the stream "ends".

Continuous queries are queries over data streams and standard relations that continuously produce results as new tuples appear in the input streams of the query. At some point they are registered in the system and start producing results, and they last until they are cancelled. They can be used to extract significant parts from a large stream of data, to periodically output aggregate information or to raise an alarm if the system steps out of normal working parameters.

In this paper we are concerned with streams that contain data about objects whose position and/or extent changes over time. To efficiently manage spatio-temporal data streams a DSMS that supports spatio-temporal data types and operations is needed.

2. RELATED WORK

Most of relevant research in the area of data streams was done within several projects each producing a prototype DSMS. Within STREAM project [3, 4], streams are transformed into relations using window operators and then queried using standard relational operators. Results can then be transformed back into streams. AURORA project [1] on MIT does not support a textual query language, using data flow diagrams to define queries instead. Stream Mill system [17, 5], developed at UCLA with the emphasis on data mining, enables the user to define custom aggregates which are then used to process streaming data. Telegraph project at UC Berkley [25] explores adaptive dataflow architecture which enables it to make scheduling decisions for each tuple.

Works [24] and [22] try to merge moving objects and data streams fields of research, viewing data streams in a standard way, as unbounded ordered relations.

Scalable on-line execution (SOLE) algorithm [21] is one of the first attempts to furnish query processors in data stream management systems with the required operators and algorithms to support a scalable execution of concurrent continuous spatio-temporal queries over spatio-temporal data streams.

Work presented in [14] defines a geo-stream as data stream carrying information about geometry or geometries changing over time. It expands upon data types and operations given in [7, 12, 13] by adding new data types called *stream* and *window*, also defining their abstract semantic and operations on them. This work resolves two important issues: definition of windows semantics through a data type based approach and design of streaming data types, operations and predicates. It presents a novel approach to handling streaming data by encapsulating it inside a single attribute in a relation. This is one of the first works that formally and successfully merges streams and moving objects, and our approach shares common ground with that work.

GeoInsight system [15] presents an integration of SQL spatial libraries into the continuous query pipeline of full-fleged commercial Microsoft StreamInsight engine to support the real-time processing of geo-stream data.

PLACE [22] views data streams as automatically changing relations incrementally calculating results and producing positive and negative updates of the result set. It supports *predicate* windows that use predicates to determine when each tuple expires from the system, and allows the user to construct complex queries out of simpler operators such as *inside* or *kNN*. The PLACE continuous query processor extends the processing of continuous sliding window queries beyond time-based and tuple-count windows - objects are qualified to be part of the window once they satisfy a certain query predicate.

Our work may be most similar to that of [24, 23] who also use open-source TelegraphCQ stream engine, but with PostgreSQL built-in spatial data types and operators and, therefore, with rather limited (and non-standard) spatio-temporal support.

3. THE TYPE SYSTEM

There are three kinds of different temporal predicates in spatio-temporal queries. Accordingly, the queries can be classified into three classes: historical, current and future query [19]. In this section we define a system of data types on abstract level supporting historical and current queries, using many-sorted algebra and second-order signature, building upon work in [7, 12, 14].

Modelling on abstract level allows us to make definitions in terms of infinite sets, without fixing any finite representations of these sets. In spatio-temporal context, a moving point is a continuous curve in 3D (2D + time) or 4D (3D + time) space, i.e. mapping from an infinite time domain into an infinite 2D (or 3D) space domain. Although the abstract level is the conceptual model we are interested in, it should be noted that it is not directly implementable and that additional step of fixing a concrete, finite representation is needed.

A many-sorted algebra consists of sets and functions. A *signature* is a pair of sets $(\mathcal{S}, \Omega)$, the elements of which are called *sorts* and *operations* respectively. Each operation consists of a (k+2)-tuple

$$n : s_1 \times \cdots \times s_k \rightarrow s, \text{ with } s_1, \times \ldots s_k, s \in S \text{ and } k \geq 0.$$

Operation names (*operators*) are denoted by n; $s_1, \ldots, s_k, s \in S$ are sorts. In the case $k = 0$ the operation is called a *constant of sort s*. Informally, a sort denotes a name of a type, and an operation denotes a function [18].

A second-order signature consist of two coupled many-sorted signatures. The first signature defines a type system. In this context sorts are called *kinds* and denote collections of types and operators are *type constructors*. This signature generates a set of *terms*, which are exactly the *types*. Second signature defines operations over terms of the first signature.

Definitions of *tuple*, *stuple*, and *srelation* type constructors, in table 1, already use some extensions to the basic concepts. The first is that a type constructor may use as sorts not only kinds but also types. The second extension makes it possible to define operators taking a variable number of operands. The notation s^+ denotes a list of one or more

operands of sort s. The third extension is that if $s_1, ..., s_n$ are sorts, then $(s_1 \times ... \times s_n)$ is also a sort [11].

The concept of temporal types requires additional consideration, and will be described briefly. To model values of a spatio-temporal type σ that changes over time, we introduce the notion of temporal function which is an element of type

$\tau(\sigma) : time \nrightarrow \sigma$

Temporal functions are the basis for an algebraic model of spatio-temporal data types, where σ is assigned one of the spatial data types *point, multiPoint, lineString, multiLineString* or *polygon*, resulting in *movingPoint* as values of type $\tau(point)$, *movingMultiPoint* as values of type $\tau(multiPoint)$ and *movingPolygon* as values of type $\tau(polygon)$.

Table 1 shows a signature defining data types that a spatio-temporal DSMS should support. Abstract semantics of BASE, SPATIAL, TIME, RANGE and TEMPORAL sorts has been precisely defined in [7, 11, 12, 13]. When applied to a compatible simple data type, type constructors produce a new data type e.g. *range(int)*, *moving(lineString)*, etc. TEMPORAL constructor can be applied to BASE and SPATIAL data types.

IDENTIFIER, TUPLE, and RELATION sorts, as well as *id*, *tuple* and *relation* constructors, are cornerstones of (object-) relational algebra. That subset of sorts, i.e. subset of algebra, is a basis for building spatio-temporal DBMS in extensible DBMS environment, such as SECONDO [6], with the goal to support modelling and querying *history* of movement.

Sorts STUPLE and STREAM are used to model data stream analogous to relations, sort WINDOW captures the concept of sliding windows, and sort IRELATION is used to apply windows to data streams.

3.1 Abstract semantic

Data stream represents a potentially unbounded sequence of elements generated at a data source. Data stream element consists of a tuple with schema Σ and a timestamp representing actual time when the tuple is generated. A single data stream may contain elements from a number of different data sources if their tuples have the same schema Σ.

For stream element arrivals we assume a continuous and linear time domain $\mathcal{T}$, which represents *valid time*. A *time instant* τ is any value from $\mathcal{T}$, i.e. $\tau \in \mathcal{T}$.

Definition Data stream $\mathcal{S}$ is a possibly infinite sequence of elements $\langle s, \tau \rangle$ where s is a tuple belonging to the schema (type) of $\mathcal{S}$.

Each data stream element is mapped to exactly one timestamp that represents its unique time reference. If a data source moves around in space, i.e. represents a spatio-temporal object, then a spatial stamp is also attached to the elements of corresponding data stream. It represents location and/or geometry of a data source when element is generated. Timestamp and spatial stamp together uniquely determine spatio-temporal status of an object and are explicitly included in data stream schema.

Definition Spatio-temporal data stream $\mathcal{S}_{\mathcal{ST}}$ is a mapping $\mathcal{S}_{\mathcal{G}} : \mathcal{T} \times \mathcal{G} \rightarrow 2^{\mathcal{R}}$ from time domain and (geo)spatial domain to the powerset of the set $\mathcal{R}$ tuples with schema Σ. In addition to an attribute $\mathcal{A}_{\mathcal{T}}$, another attribute $\mathcal{A}_{\mathcal{G}}$ acts as the geospatial reference for the represented entity and obtains its values exclusively from the (geo)spatial domain $\mathcal{G}$ [24].

Because streams are unbounded, when a stream is used in an SQL-like query, the system must be told how and when to consider the data in the stream. In most existing DSMS this is usually done using a *window* specification derived from SQL-99 to map from streams to relations. Stream-to-relation mapping is based on the concept of a *sliding window* over a stream, i.e. a window that at any point in time contains a historical snapshot of a finite but unbounded subset of the stream. In what follows, we focus on *time-based sliding windows* only, extending the approach for stream-to-relation and relation-to-stream operators presented in [4, 14] and our previous work [16, 20]. Table 2 shows abstract semantics of time-based sliding windows and corresponding SQL-like notation, which will later be used for the purpose of language embedding.

A data type is defined my specifying its name and its set of possible values called *carrier set* or *domain*. Carrier set of a type t is denoted by D_t. Carrier set for the type constructor *now* is the value of the current instant of time, and the carrier set of the type constructor *past*

$past : instant \times now \rightarrow range(instant)$

is a closed time interval, having *now* on the right side.

Data type $streaming(\sigma)$, similar to $moving(\sigma)$, can be considered as a partial function from *instant* to σ.

$streaming(\sigma) : instant \nrightarrow \sigma$

In that context, we can talk about values of a streaming data type at a given time. Both data types $streaming(\sigma)$ and σ can have value $undefined$.

4. SPATIO-TEMPORAL OPERATORS

To make potentially unbounded data streams easier to manage, sliding window operators based on SQL-99 expanded by spatial windows introduced in [22] and [24] are used. Window operators enable the user to view only a part of a data stream. Tuple based sliding windows are not appropriate in the context of spatio-temporal streams, and we limit our discussion to time-based windows only. For time-based and spatial windows a sliding effect can be achieved by specifying a sliding step. In that case, a new window state and query results are calculated each sliding step (e.g every 5 minutes or 50 meters).

After applying window *now* to data type $streaming(\sigma)$, result is of type σ. After applying time interval window of size $tsize$ to data type $streaming(\sigma)$, result is of type $streaming(\sigma)$, where it retains its values for time in $(now - tsize, now]$, and has value $\perp$ otherwise.

At some situations it might be convenient for a window over a data stream to refer to a time interval in the past. For example, if the current data in the stream needs to be compared to the data from yesterday. To specify this time-shifted window, a starting point and window size need to be specified.

4.1 Temporal lifting

Operations on $BASE, SPATIAL, TIME$ and $RANGE$ data types are applied to $TEMPORAL$ data types using

Table 1: Type system as a signature

Type constructor	Signature	Target sort
int, real,		
string, bool		→ BASE
point, multiPoint, lineString		
multiLineString, polygon		→ SPATIAL
instant		→ TIME
range	BASE ∪ TIME	→ RANGE
moving, intime	BASE ∪ SPATIAL	→ TEMPORAL
id		→ IDENTIFIER
tuple	$(id \times (\text{BASE} \cup \text{SPATIAL} \cup \text{TIME} \cup \text{RANGE} \cup \text{TEMPORAL}))^+$	→ TUPLE
relation	TUPLE	→ RELATION
stuple	$((id \times (\text{BASE} \cup \text{SPATIAL} \cup \text{TIME}))^+ \times \text{TIME})$	→ STUPLE
stream	STUPLE	→ STREAM
now, unbounded,		
past		→ WINDOW
srelation	STREAM × WINDOW	→ IRELATION
rstream	IRELATION	→ STREAM
streaming	BASE ∪ SPATIAL	→ TEMPORAL

Table 2: Window abstract semantic and CSQL notation

WINDOW	Abstract semantic	CSQL notation
now	$D_{now} = now$	<now>
past	$D_{past} = \{X \subseteq D_{instant} \mid \forall x \in X(x \leq now) \wedge \forall y \in D_{instant}(x < y \leq now \Rightarrow y \in X)\}$	<past *time_interval*>
unbounded	$D_{unbounded} = (\infty, now]$	<unbounded>

temporal lifting. Temporal lifting is the key concept for achieving consistency of operations on nontemporal and temporal types. The idea is to allow argument sorts and target sort of a signature (the arguments and the result of the corresponding operation) to be of temporal type. Formally, for each operation with signature

$o : s_1 \times \cdots \times s_n \to s$

its corresponding temporally lifted version is defined by:

$\Uparrow o : \tau(s_1) \times \cdots \times \tau(s_n) \to \tau(s)$

Consistency of operation on moving and non-moving data types is achieved by first defining all operations on non-moving data types, and afterwards systematically extending them to corresponding moving data types changing the resulting data type into a moving data type as well. The same principle can be applied to streaming data types.

Consider binary topological predicate *inside* for points and polygons, with signature:
$inside : point \times polygon \to boolean$

Applying lifting to streaming data types results in following three signatures:
$inside : streaming(point) \times polygon \to streaming(boolean)$
$inside : point \times streaming(polygon) \to streaming(booelan)$
$inside : streaming(point) \times streaming(polygon) \to streaming(boolean)$

It is worthy to note that while the process of lifting produces signatures for all possible operations on streaming and moving data types, it does not define how to implement each of those lifted operations.

4.2 Streaming data operations

In the context of spatio-temporal data streams both streaming and moving data types attempt to model the same real world phenomena: a unit of data changing over time. However, there are some differences. Moving data types do not change without explicit instructions from the user while streaming data types are changed by an outside source. While moving data types can be defined continuously, streaming data types always have a stepwise sliced representation: one value holds until next one arrives. Since both sets of data types refer to same objects in the real world, most of operations defined for moving data types can be applied to streaming data types as well.

All operations applicable to $BASE$ and $SPATIAL$ data types can also be applied to streaming data types through lifting. Several groups of operations defined specifically for moving data types classified into **projection to domain and range** and **interaction with domain and range** are defined in [12]. Here we present a subset of them, appropriate for streaming data types:

$deftime : streaming(\sigma) \to range(instant)$
$trajectory : streaming(point) \to lineString$
$streaming(multiPoint) \to multiLineString$
$atinstant : streaming(\sigma) \times instant \to intime(\sigma)$
$atperiods : streaming(\sigma) \times range(instant) \to streaming(\sigma)$
$at : streaming(\sigma) \times \sigma \to streaming(\sigma)$
$streaming(\sigma) \times range(\sigma) \to streaming(\sigma)$
$present : streaming(\sigma) \times instant \to bool$
$streaming(\sigma) \times range(instant) \to bool$

Since streaming and moving data types are so similar and capture the same real world phenomena, it is reasonable to include a function to convert streaming data types to moving data types. If it was allowed to use such a function inside

a continuous query, a result would be a moving data type that changes with time in effect a $streaming(moving(\sigma))$. To avoid unnecessary complexity such a function cannot be used within continuous queries. Its intended purpose is to store historical streaming data as moving object data types in a more efficient manner and to facilitate the use of indexes available for moving data types.

Function *atperiods* also takes a streaming data type at input and produces a moving data type at output.

5. IMPLEMENTATION

A discrete model, as a finite instantiation of an abstract model is needed for implementation. The discrete model uses so-called sliced representation: temporal changes of a value along the time dimension are decomposed into fragment intervals called slices and temporal changes for moving points can be represented within each slice by simple linear function.

Abstract type system supports standard data types (*int, real, bool, string*), data type to represent time (*instant*), spatial types (*point, multiPoint, lineString, multiLineString, polygon*) [8], temporal data types obtained by a type constructors *moving* and *streaming*, type that represents a set of intervals obtained by a type constructor *range* and a type whose values are pairs consisting of an instant of time and a value of the respective argument type obtained by type constructor *intime*.

Each of the abstract model data types can be directly implemented in terms of corresponding types of a programming language, except for temporal types. In a discrete model, *moving* and *streaming* constructors are replaced by a type constructors that enable so-called sliced representation of temporal types. Type constructor *mapping* represents a set of unit types whose values are pairs consisting of a time interval and a simple function describing temporal development of a value during that time interval.

Spatial stamp and timestamp represent two separate attribute values - usually implemented in the context of spatio-temporal databases with related, but separate spatial data type and data type that represents time. Together they represent a unique *spatio-temporal reference* of a data stream element and we propose its implementation in DSMS with composite temporal data type whose values are pairs of instant in time and location in space. This data type is derived using the type constructor *intime*, presented in table 1. Instant component implicitly used in type constructor *intime* represents valid time.

Definition Spatio-temporal reference of a spatio-temporal data stream element is obtained by applying type constructor *intime* to the appropriate spatial data type, i.e. *intime* (σ).

Carrier set for data type $intime(\sigma)$, at discrete level, is
$D_{intime(\sigma)} := D_{instant} \times D_{\sigma}$

5.1 User-defined aggregate functions

A set of aggregate functions available to users of a data stream system is usually limited to five standard functions: *min, max, count, sum* and *avg*. However, many data stream systems allow the user to extend the set of available aggregate functions by defining user-defined aggregate functions (UDAF).

To create a UDAF, the user must implement at least three functions:

Init This function is used to initialize any variables needed for the computation later on. Intuitively, it is similar to a constructor.

Accumulate This function is called once for each value aggregated. Generally, this function will "add" the value to the running total computed thus far.

Terminate This function is used to end the calculation and return the final value of the aggregate function. It may involve some calculations on variables which were defined for use with the aggregate function.

Similar to a continuous query, UDAF is considered *blocking* [10] if it requires its entire input before it can return a result (blocking UDAFs always have terminate operation). Such blocking aggregates can only be applied over data streams using a window operator and are termed *windowed aggregates*, while UDAFs invoked without a window operator are termed *base aggregates*. UDAFs in which terminate operation is not defined are called non-blocking and can be applied to data streams freely.

The proposed temporal data type for representing a spatio-temporal reference of a data stream element enables implementation of sliced representation for spatio-temporal objects. Spatio-temporal object is formed of unit types each consisting of a time interval and a value describing simple function on the unit time interval. For example, for two successive spatio-temporal data stream elements with spatio-temporal references consisting of successive point locations and time instants, it is possible to define the unit type that describes movement of the corresponding point object as a linear function of time.

Complete sliced representation presents a moving object as a set of temporal units whose time intervals are disjoint, and if adjacent their values are different. Assembling unit types into a moving object based on spatio-temporal references of data stream elements can be achieved using UDAF. Aggregation can be defined as a state that is modified with every input data. In terms of spatio-temporal data streams, such state represents a moving object in a form of described sliced representation (*mapping(upoint)* data type), while spatio-temporal reference of data stream element represents new input data (*intime(point)* data type). Function *iterate*, that is used in UDAF, performs creation of unit type and adds it to the moving object. Since this function computes a new state (moving object) from the old one and the value of spatio-temporal reference of a new data stream element, UDAF for extracting moving object out of data stream doesn't need a terminate function and is thus considered non-blocking. Moving object is not defined before the first input value has arrived, and initial value for the internal state is NULL. This concept can be implemented using following PostgreSQL-like syntax:

```
CREATE AGGREGATE toMoving (intime(point))
(
       SFUNC = iterate,
       STYPE = mapping(upoint),
       INITCOND = NULL
);
```

Definition Sliced representation of an spatio-temporal object is obtained by applying an aggregate function over spatio-temporal references of data stream elements.

In combination with different window operators, i.e. stream-to-relation operators [4], proposed UDAFs produce spatio-temporal objects as intermediate results of a continuous spatial query. DSMSs that support not only data streams but persistent relations as well, can be extended with ability to store intermediate results of continuous queries into relations. With this approach, movement history of objects extracted out of spatio-temporal data streams can easily be persisntenly stored. A powerful set of previously unavailable continuous queries can be achieved by joining relations containing history of movements with data streams carrying information about current movements.

5.2 Continuous SQL: CSQL

Approach proposed in previous chapter is implemented by user defined aggregate function named `toMoving`, that takes a spatio-temporal reference of a data stream element *intime(point)* as its only argument. Aggregate function `toMoving` produces moving object of the appropriate temporal data type.

Given concept is illustrated through CSQL, an SQL-like query language that supports continuous queries over data streams. Examples in this paper are based on the following spatio-temporal data stream schema:

```
carStream (
    id:         INTEGER,
    driver:     SMALLINT,
    geom:       point,
    speed:      REAL,
    tcqtime:    TIMESTAMP TIMESTAMPCOLUMN
)
```

Attribute *tcqtime* is a default *timestamp* column. It specifies the creation time of a tuple, which is assumed to be monotonically increasing.

Following two examples illustrate spatio-temporal continuous queries and their visualization (Fig. 1) enabled within our prototype:

***Q1**: Continuously report all cars, their speed and location, within a particular city district.*

```
SELECT id, speed, geom
FROM   carStream <NOW>, cityDistrict
WHERE  cityDistrict.name = 'Crnomerec'
AND    ST_Within(carStream.geom, cityDistrict.geom)
```

***Q2**: Find all cars that have travelled more than 50 km during last hour.*

```
SELECT id, val(final(toMoving(intime(geom, tcqtime))))
FROM   carStream <PAST '1 hour'>
GROUP  BY id
HAVING ST_Length(trajectory(toMoving(intime(geom,tcqtime)
       ))) > 50000
```

6. CONCLUSIONS

Proposed approach for spatio-temporal data stream management is based on extending relational-based DSMS with a full-fledged set of spatio-temporal data types and associated operations. Implementation of abstract data model at discrete level, including spatio-temporal predicates and operations, is based on the concept of extracting spatio-temporal objects out of data streams using UDAFs.

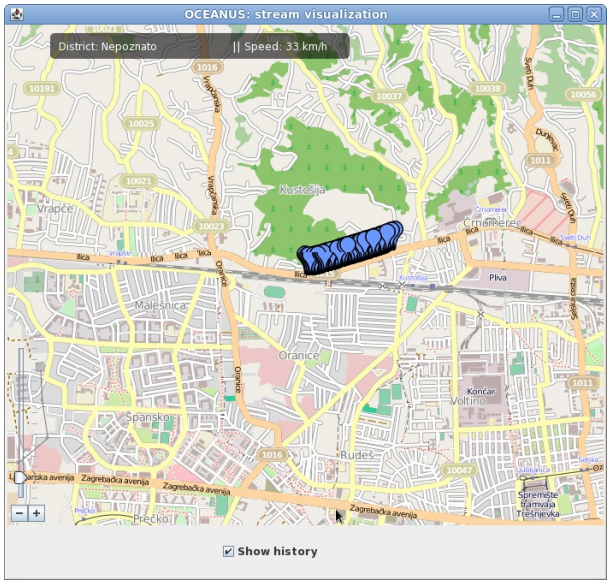

Figure 1: User interface prototype (with OpenStreetMap background)

Our approach is based on extending TelegraphCQ prototype with PostGIS, enabling continuously adaptive query processing with full spatial support and taking advantage of open source processing software and extensible features. Besides that, it supports rich windowing schemes over portions of data streams and user-defined aggregate functions over sliding windows. These features enable implementation of the proposed concept and extension of the continuous query language, thus allowing posing continuous spatio-temporal queries.

An important class of queries over data streams can be only answered by using predicate-window query model [9], that limits the focus of a continuous query to the stream tuples that qualify a certain predicate.

Beside predicate-windows, the focus of our future work will be on a full prototype implementation based on proposed spatio-temporal type system, storing transient results of continuous queries into spatio-temporal DBMS, and developing sophisticated graphical user interface.

7. ACKNOWLEDGMENTS

This work was supported by the *Geospatial Sensors and Moving Objects Databases* and *Semantic Integration of Heterogeneous Data Sources* research projects, funded by the Ministry of Science under grant numbers 036-0361983-2020 and 036-0361983-2012.

The authors would like to thank the anonymous reviewers for their valuable insights, suggestions and constructive recommendation for improving this paper.

8. REFERENCES

[1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Aurora: a new model and architecture for data stream management. *The International Journal on Very Large Databases*, 12(2):120–139, 2003.

[2] M. H. Ali, B. Chandramouli, J. Goldstein, and R. Schindlauer. The extensibility framework in Microsoft StreamInsight. In *ICDE*, pages 1242–1253, 2011.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1):19–26, 2003.

[4] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The International Journal on Very Large Databases*, 15(2):121–142, 2006.

[5] Y. Bai, H. Thakkar, H. Wang, C. Luo, and C. Zaniolo. A data stream language and system designed for power and extensibility. In P. S. Yu, V. J. Tsotras, E. A. Fox, and B. Liu, editors, *CIKM*, pages 337–346. ACM, 2006.

[6] V. T. de Almeida, R. H. Güting, and T. Behr. Querying moving objects in SECONDO. In *Mobile Data Management*, pages 47–51. IEEE Computer Society, 2006.

[7] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis. Abstract and discrete modeling of spatio-temporal data types. In *ACM-GIS*, pages 131–136, 1998.

[8] Z. Galić. *Geospatial Databases*. Golden Marketing - Tehnička knjiga, 2006. [in Croatian].

[9] T. M. Ghanem, W. G. Aref, and A. K. Elmagarmid. Exploiting predicate-window semantics over data streams. *SIGMOD Record*, 35(1):3–8, 2006.

[10] L. Golab and M. T. Özsu. *Data Stream Management*. Synthesis Lectures on Data Management. Morgan Claypool Publishers, 2010.

[11] R. H. Güting. Second-order signature: A tool for specifying data models, query processing, and optimization. In P. Buneman and S. Jajodia, editors, *SIGMOD Conference*, pages 277–286. ACM Press, 1993.

[12] 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 Transactions on Database Systems*, 25(1):1–42, 2000.

[13] R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.

[14] Y. Huang and C. Zhang. New data types and operations to support geo-streams. In T. J. Cova, H. J. Miller, K. Beard, A. U. Frank, and M. F. Goodchild, editors, *GIScience*, volume 5266 of *Lecture Notes in Computer Science*, pages 106–118. Springer, 2008.

[15] S. J. Kazemitabar, U. Demiryurek, M. H. Ali, A. Akdogan, and C. Shahabi. Geospatial stream query processing using Microsoft SQL Server StreamInsight. *PVLDB*, 3(2):1537–1540, 2010.

[16] K. Križanović, Z. Galić, and M. Baranović. Data types and operations for spatio-temporal data streams. In *Proceedings of the 12th International Conference on Mobile data Management, Vol. II*, pages 11–14, 2011.

[17] Y.-N. Law, H. Wang, and C. Zaniolo. Query languages and data models for database sequences and data streams. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 492–503. Morgan Kaufmann, 2004.

[18] J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of Abstract Data Types*. Wiley/Teubner computing series, 1996.

[19] X. Meng and J. Chen. *Moving Objects Management: Models, Techniques and Applications*. Tsinghua University Press and Springer-Verlag, 2010.

[20] E. Mešković, Z. Galić, and M. Baranović. Managing moving objects in spatio-temporal data streams. In *Proceedings of the 12th International Conference on Mobile data Management, Vol. II*, pages 15–18, 2011.

[21] M. F. Mokbel and W. G. Aref. SOLE: scalable on-line execution of continuous queries on spatio-temporal data streams. *The International Journal on Very Large Databases*, 17(5):971–995, 2008.

[22] M. F. Mokbel, X. Xiong, M. A. Hammad, and W. G. Aref. Continuous query processing of spatio-temporal data streams in PLACE. *GeoInformatica*, 9(4):343–365, 2005.

[23] K. Patroumpas, E. Kefallinou, and T. K. Sellis. Monitoring continuous queries over streaming locations. In W. G. Aref, M. F. Mokbel, and M. Schneider, editors, *GIS*, page 81. ACM, 2008.

[24] K. Patroumpas and T. K. Sellis. Managing trajectories of moving objects as data streams. In J. Sander and M. A. Nascimento, editors, *STDBM*, pages 41–48, 2004.

[25] TelegraphCQ. http://telegraph.cs.berkeley.edu/telegraphcq/v2.1/.