Towards A Streams-Based Framework for Defining Location-Based Queries

Huang, Xuegang; Jensen, Christian S.

Published in:
Proceedings of the Second International Workshop on Spatio-Temporal Database Management

Publication date:
2004

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Huang, X., & Jensen, C. S. (2004). Towards A Streams-Based Framework for Defining Location-Based Queries. In J. Sander, & M. A. Nascimento (Eds.), *Proceedings of the Second International Workshop on Spatio-Temporal Database Management* (pp. 73-80) http://www.informatik.uni-trier.de/~ley/db/conf/stdbm/stdbm2004.html

Towards A Streams-Based Framework for Defining Location-Based Queries

Xuegang Huang Christian S. Jensen

Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, DK-9220, Aalborg, Denmark
{xghuang,csj}@cs.aau.dk

Abstract

An infrastructure is emerging that supports the delivery of on-line, location-enabled services to mobile users. Such services involve novel database queries, and the database research community is quite active in proposing techniques for the efficient processing of such queries. In parallel to this, the management of data streams has become an active area of research.

While most research in mobile services concerns performance issues, this paper aims to establish a formal framework for defining the semantics of queries encountered in mobile services, most notably the so-called continuous queries that are particularly relevant in this context. Rather than inventing an entirely new framework, the paper proposes a framework that builds on concepts from data streams and temporal databases. Definitions of example queries demonstrates how the framework enables clear formulation of query semantics and the comparison of queries. The paper also proposes a categorization of location-based queries.

Keywords: Location-based service, data stream, continuous query, skyline query, range query, nearest-neighbor query.

1 Introduction

The emergence of mobile services, including mobile commerce, is characterized by convergences among new technologies, applications, and services. Notably, the ability to identify the exact geographical location of a mobile user at any time opens to range of new, innovative services, which are commonly referred to as location-based services (LBSs) or location-enabled services.

Proceedings of the Second Workshop on Spatio-Temporal Database Management (STDBM'04),
Toronto, Canada, August 30th, 2004.

In an LBS scenario, the service users are capable of continuous movement, and changing user locations are sampled and streamed to a processing unit, e.g., a central server. The notion of a data stream thus occurs naturally. Service requests result in queries being issued against the data streams and other, typically relational, data.

Conventional queries are one-time queries, i.e., queries that are simply issued against the state of the database as of the time of issue, upon which they, at a single point in time, return a result. In our scenario, so-called continuous queries are also natural. Such queries are "active" (i.e., being re-evaluated) for a duration of time, and their results are kept up-to-date as the database changes during this time. As an example, an in-vehicle service may display the three nearest, reasonably priced hotels with rooms available along the route towards the vehicle's destination. The vehicle's location (a data stream) together with data about hotels (relational data) are continuously queried to provide the result (a data stream).

Significant results on the processing of location-based queries (LBQs) has already been reported. As LBQs are defined in different settings, no direct means are available for classifying and comparing these queries. As more and more work, considering more and more different kinds of queries, is reported, the need for comparison increases.

This paper presents a general framework within which the semantics of LBQs can be specified. This enables the definition of LBQs in a single framework, which in turn enables the comparison of queries. The framework is well defined—it is based on precise definitions of data structures and operations on these. The framework has the following characteristics.

- Streams as well as relations are accommodated.
- Because queries often involve ranked results, relations are defined to include order.
- Relational algebraic operators are extended to also apply to streams, by using mappings of streams to relations, and, optionally, mappings of relations to streams.

The result is an expressive yet semantically simple framework that may be extended with additional operators and mappings. To illustrate the extensibility, a new operator, the skyline operator, is introduced.

Rather than listing and defining all possible location-based queries, this paper represents several prominent ones, such as a range query, a nearest-neighbor query, and a location-based skyline query; and it discusses categorizations of LBQs.

The research area of stream data is quite active and has produced a number of interesting concepts in relation to the semantics of continuous queries. Specifically, significant research results have been reported on query processing for data streams (e.g., [3, 6, 22, 29]). Some works consider queries over data streams together with relations (e.g., [1, 18]), but only few works consider the formalization of queries over streams and relations.

Similarly, location-based query processing is an active area of research, and many interesting results have appeared. Much attention has been given to the indexing and query processing for moving objects. Numerous index structures and algorithms have been proposed for a variety of location-based queries (e.g., [4, 9, 12, 13, 14, 17, 19, 20, 23, 24, 26, 27, 28]), such as nearest neighbor queries, reverse neighbor queries, spatial range queries, distance joins, and closest-pair queries. A new type of query, the skyline query, has recently received attention [5, 8, 15, 21]. However, only little attention has been paid to query processing in relation to spatial data streams [16]. To the best of our knowledge, no formal frameworks have been proposed for the definition of location-based queries against relations and data streams.

Recently, Arasu et al. [1] have offered an interpretation of continuous queries over streams, by formalizing streams, relations, and mapping operators among them. We build on their general approach. To accommodate ordering as well as duplicates, we use list-based relations and a variant of the list-based relational algebra proposed by Slivinskas et al. [25]. To be able to express query semantics precisely, our approach also accommodates the notions of activation and deactivation times and reevaluation granularity.

The paper is outlined as follows. Section 2 defines the data structures underlying the framework and presents the application scenario. The next section completes the framework, by defining the operators that map between the different operators in the framework. Section 4 uses the framework to define different location-based queries and also discusses the categorization of location-based queries. The last section summarizes and offers directions for future research.

2 Data Structures and Application Scenario

2.1 Data Model Definition

Building on the relation concept defined by Slivinskas et al. [25], we define relations as lists to capture duplicates and ordering. We define schemas, tuples, and relation instances, then define the same concepts for streams.

Definition 2.1. A *relation schema* (Ω, Δ, dom) is a three-tuple where Ω is a finite set of attributes, Δ is a finite set of domains, and $dom : \Omega \to \Delta$ is a function that associates a domain with each attribute.

obj_id	obj_loc	obj_type
301	$(20, 35)$	police station
302	$(30, 80)$	hospital
303	$(65, 75)$	fi re department
304	$(80, 120)$	hospital
305	$(70, 80)$	police station

Figure 1: Relation r_{obj}

Relation r_{obj} in Figure 1 has schema (Ω, Δ, dom), where $\Omega = \{obj\_id, obj\_loc, obj\_type\}$, $\Delta = \{$number, location, string$\}$, and $dom = \{(obj\_id, \text{number}), (obj\_loc, \text{location}), (obj\_type, \text{string})\}$.

Definition 2.2. A *tuple* over *schema* $\mathcal{S} = (\Omega, \Delta, dom)$ is a function $t : \Omega \to \bigcup_{\delta \in \Delta} \delta$, such that for every attribute A of Ω, $t(A) \in dom(A)$. A *relation* over $\mathcal{S}$ is a finite sequence of tuples over $\mathcal{S}$.

The definition of a relation corresponds to the definition of a list or a sequence. A relation can thus contain duplicate tuples, and the ordering of tuples is significant. Relation r_{obj} from Figure 1 is the list $\langle t_1, t_2, t_3, t_4, t_5\}\rangle$, where, e.g., $t_1 = \{(obj\_id, 301), (obj\_loc, (20, 35)), (obj\_type,$ "police station"$)\}$.

Definition 2.3. A *stream schema* is a relation schema (Ω, Δ, dom), where Ω includes a special attribute T, Δ includes the time domain $\mathbb{T}$, and $dom(\mathrm{T}) = \mathbb{T}$.

We assume that domain $\mathbb{T}$ is totally ordered. While, for simplicity, we use the non-negative numbers as the time domain in the sequel, other domains may be used. For example, the real or natural numbers, the TIMESTAMP domain of the SQL standard, or one of the domains proposed by the temporal database community may be used.

Stream s_{usr} in Figure 2 has schema (Ω, Δ, dom), where $\Omega = \{usr\_id, usr\_v, usr\_loc, \mathrm{T}\}$, $\Delta = \{$number, velocity, location, $\mathbb{T}\}$, and $dom = \{\ (usr\_id, \text{number}), (usr\_v,$ velocity$), (usr_loc,$ location$), (\mathrm{T}, \mathbb{T})\}$.

usr_id	usr_v	usr_loc	T
$\cdots$	$\cdots$	$\cdots$	$\cdots$
1004	$(10, -15)$	$(90, 80)$	10
1002	$(-25, 25)$	$(200, 10)$	12
1003	$(-12, -11)$	$(60, 80)$	10
1004	$(0, 0)$	$(100, 60)$	12
1003	$(-10, 3)$	$(40, 58)$	12
1001	$(0, 1)$	$(16, 38)$	9
1004	$(20, 35)$	$(100, 60)$	15
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Figure 2: Stream s_{usr}

Definition 2.4. A *stream* is a possibly infinite multiset of tuples over *stream schema* $\mathcal{T}$.

For a stream tuple t_i, the time $\tau_i = t_i(\mathrm{T})$ indicates when the tuple became available in the stream. While a relation is ordered, we have chosen to not introduce an inherent order on streams. Streams come with the natural (partial) order implied by their time attribute.

Stream s_{usr} in Figure 2 is the possibly infinite multiset $s_{usr} = \{\!\!\{ \ldots, (1004, (10, -15), (90, 80), 10), \ldots, (1004, (20, 35), (100, 60), 15), \ldots \}\!\!\}$.

While a query is issued against an entire relation state, intuitively, a query issued at some time τ_q will only see either what has appeared in the stream so far, i.e., all tuples with timestamp less than or equal to τ_q, or what has appeared in the stream between some past time and τ_q. The latter may be assumed if what has appeared in the stream so far does not fit in the available memory.

2.2 Discussion

As we pointed out earlier, we use streams for modeling the locations of moving objects such as pedestrians, cars, and buses. We use relations for modeling aspects of an application domain that change discretely.

As we aim for a generic framework, we make no assumptions about the representations of the geographical locations and extents of objects that limit the applicability of the framework. However, to be specific, we assume that positions are simply points (x, y) in two-dimensional Euclidean space; in accord with this, a velocity vector is given by (v_x, v_y). We note that in some application scenarios, positions of objects are given in terms of road networks, using linear referencing [11]. The framework is also applicable in the context of this kind of positioning.

In the example we use throughout, stream s_{usr} in Figure 2 captures positions and velocities of moving users. Attribute usr_id records the ID of a user, and usr_v and usr_loc record the velocity and location of the user at the time instant recorded in attribute T. In a real-world application, multiple streams may well be present. For example, users moving by bicycle and by car may be captured by separate streams. For simplicity, we only use one stream.

Relation r_{usr} in Figure 3 captures discretely changing properties of the service users. As before, usr_id records the ID of a user; and attributes usr_name capture the first name of a user.

usr_id	usr_name
1001	Kate
1002	Bill
1003	Joan
1004	Tom

Figure 3: Relation r_{usr}

Finally, relation r_{obj} in Figure 1 records the points of interest that service users may query. Attribute obj_id captures the ID of a point of interest, obj_loc records its location, and obj_type records its type.

In real-world applications, additional attributes and relations may of course be used, beyond the ones introduced above.

In our scenario, the users of the services that issue the queries are moving, and the points of interests being queried are static. However, in other equally valid scenarios, a static user can query moving objects, e.g., a supermarket wants to know all the potential customers who are near the supermarket between 8:00 a.m. and 5:00 p.m. Also, a moving user may query other moving users—this may be typical of location-based games.

3 Mapping Operators

Queries are either one-time or continuous, they apply to relations and streams, and their results are either relations or streams.

Relations are well known, and the semantics of queries against relations are generally agreed upon. In contrast, what the appropriate semantics of queries against streams should be and how these should be defined are less obvious. Following Arasu et al. [1], we aim to maximally reuse

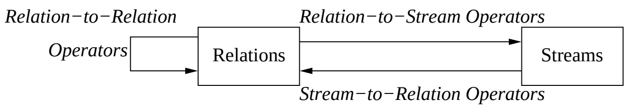

Figure 4: Mapping Operators

the relational setting in defining the semantics of queries against streams. We do this by introducing mapping operations between streams and relations, so that a query against a stream can be defined by mapping the stream to a relation, then applying a relational query, and then, optionally, mapping the result to a stream. This results in the framework of representations and operators outlined in Figure 4. Note that direct *stream-to-stream* operators are absent.

3.1 Relation-to-Relation Operators

3.1.1 Basic Algebra Operators

A relation-to-relation operator takes one or more relations $r_1, \cdots, r_n$ as arguments and produces a relation r as a result. As our relations are ordered, we use operators introduced by Slivinskas [25] as our relation-to-relation operators: selection (σ), projection (π), union-all ($\sqcup$), Cartesian Product ($\times$), difference ($\backslash$), duplicate elimination ($rdup$), aggregation (ξ), sorting ($sort$), and top (top).

These carry their standard meanings when applied to relations without order. As an example of how the operators are defined, consider selection σ. Based on the definitions in Section 2, we use $\mathcal{R}$ to be the set of all relations and let $r = \langle t_1, t_2, \ldots, t_n \rangle \in \mathcal{R}$. We let $p \in \mathcal{P}$, where (following standard practice) $\mathcal{P}$ is the set of all selection predicates (also termed propositional formulas, see, e.g., [2, pp. 13–14]) that take a tuple as argument and return True or False.

The selection operator $\sigma : [\mathcal{R} \times \mathcal{P} \rightarrow \mathcal{R}]$ is defined using λ-calculus rather than tuple relational calculus, to contend with the order. Being a parameter, argument p is expressed as a subscript, i.e., $\sigma_p(r)$.

$$\begin{aligned}\sigma \triangleq \lambda r, p.&(r = \perp) \rightarrow r,\\ &(tail(r) = \perp) \rightarrow (p(head(r)) \rightarrow head(r), \perp),\\ &\quad (p(head(r)) \rightarrow head(r), \perp) \mathbin{@} \sigma_p(tail(r))\end{aligned}$$

The arguments are given before the dot, and the definition is given after the dot. Thus, if r is empty (denoted as $\perp$), the operation returns it. Otherwise, if r contains only one tuple (the remaining part of the relation, $tail(r)$, is empty), we apply predicate p to the (first) tuple, $(head(r))$. If the predicate holds, the operation returns the tuple; otherwise, it returns an empty relation. If these conditions do not hold, the operation returns the first tuple or an empty relation (depending on the predicate), with the result of the operation applied to the remaining part of r appended (@). The common auxiliary functions $head$, $tail$, and @ are defined elsewhere (e.g., [25]). Since the objective is to obtain an expressive framework, the framework is kept open to the introduction of such auxiliary functions, although they may increase the conceptual complexity.

3.1.2 Skyline Operator

We proceed to demonstrate how a $skyline$ operator, which is of particular interest in location-based services, can be expressed in the framework.

To understand the operator, consider a set of points in l-dimensional space. One point p_1 dominates another point p_2 if p_1 is at least as good as p_2 in all dimensions and is better than p_2 in at least one dimension [5]. It is assumed that a total order exists on each dimension, and "better" in a dimension is defined as smaller than (alternatively, larger than) with respect to the dimension's total order. Next, we assume a relation r with attributes $\{a_1, \ldots, a_l, b_1, \ldots, b_m\}$ so that the sub-tuples corresponding to attributes $\{a_1, \ldots, a_l\}$ make up the l-dimensional points. The skyline operator then returns all tuples in r that are not dominated by any other tuple in r.

To be precise, we first define two auxiliary functions. Let $\mathcal{T}$ denote the set of all tuples of any schema. The first function is $Dmnt$: $[\mathcal{T} \times \mathcal{R} \times \Omega^l] \rightarrow \{\text{True}, \text{False}\}$, which returns True if there exists a tuple in the (second) relation argument that dominates the first argument tuple with respect to the argument attributes.

$$\begin{aligned}
&Dmnt \triangleq \lambda t, r, a_1, \ldots, a_l.(r = \bot) \rightarrow \text{False},\\
&\quad Eql(t, head(r), a_1, \ldots, a_l) \rightarrow Dmnt(t, tail(r), a_1, \ldots, a_l),\\
&\qquad Comp(t, head(r), a_1, \ldots, a_l) \rightarrow \text{True},\\
&\qquad\quad Dmnt(t, tail(r), a_1, \ldots, a_l)
\end{aligned}$$

Function Eql returns True if the two argument tuples are identical on all argument attributes $a_1, \ldots, a_l$. Function $Comp$ returns True if the second argument tuple is no worse than the first argument tuple on any of the argument attributes.

In the first line, if r is empty, the operation returns False. Otherwise, if the first argument tuple t is the same as the head of argument relation r on the argument attributes, the operation continues to consider the rest of r. Else, the third line checks if $head(r)$ is no worse than t. If so, t is dominated by $head(r)$, and the operation returns True. Otherwise, the operation proceeds with the rest of r.

Next, we define auxiliary function $Fltr$: $[\mathcal{R} \times \mathcal{R} \times \Omega^l] \rightarrow \mathcal{R}$. For two relations r_1 and r_2 having the same attributes $a_1, a_2, \ldots, a_l$, $Fltr$ collects all the tuples in r_1 that are not dominated by any tuple in r_2 with respect to attributes $a_1, a_2, \ldots, a_l$.

$$\begin{aligned}
&Fltr \triangleq \lambda r_1, r_2, a_1, \ldots, a_l.(r_1 = \bot) \rightarrow r_1,\\
&\qquad Dmnt(head(r_1), r_2, a_1, \ldots, a_l) \rightarrow\\
&\qquad\qquad Fltr(tail(r_1), r_2, a_1, \ldots, a_l),\\
&\qquad\quad head(r_1) \;@\; Fltr(tail(r_1), r_2, a_1, \ldots, a_l)
\end{aligned}$$

Here, if r_1 is empty, the operation returns it. Otherwise, if the head of r_1 is dominated by any tuples in r_2 on the argument attributes, the operation continues with the rest of r_1. Else, it returns the $head(r_1)$ with the result of the operation applied to $tail(r_1)$ appended.

The skyline operator $skyline : [\mathcal{R} \times \Omega^l] \rightarrow \mathcal{R}$ is defined next. Arguments Ω^l are parameters and are expressed as subscripts, i.e., $skyline_{a_1, \ldots, a_l}(r)$.

$$skyline \triangleq \lambda r, a_1, \ldots, a_l.(r = \bot) \rightarrow r, Fltr(r, r, a_1, \ldots, a_l)$$

3.2 Stream-to-Relation Operators

A *stream-to-relation* operator takes a stream as input and produces a relation. As relations are finite while streams can be infinite, windowing is commonly used to extract a relation from a stream [3]. We describe three types of sliding windows [1]: time-based, tuple-based, and partitioned. Other types of windows can be easily incorporated into the framework, as this does not affect other parts of the framework. The stream-to-relation operators map multisets into lists. We assume that each operator described next orders its result according to the time attribute T (tuples with the same time value may be in any order).

3.2.1 Time-Based Windows

A time-based sliding window operator $\mathcal{W}^a$, with absolute or now-relative time parameter τ_s, on a stream s returns all tuples $t \in s$ for which $\tau_s \leq t(\text{T}) \leq \tau_c$, where τ_c is the current time.

usr_id	usr_v	usr_loc
1001	$(0, 1)$	$(16, 38)$
1004	$(10, -15)$	$(90, 80)$
1003	$(-12, -11)$	$(60, 80)$
1002	$(-25, 25)$	$(200, 10)$
1004	$(0, 0)$	$(100, 60)$
1003	$(-10, 3)$	$(40, 58)$

Figure 5: Result of $\mathcal{W}^a_9(s_{usr})$ at Time 12

Note that $\mathcal{W}^a_{\tau_c}(s)$ consists of tuples that made their appearance in s at time τ_c, while $\mathcal{W}^a_0(s)$ consists of all tuples that appeared in the stream so far. For stream s_{usr} in Figure 2, suppose $\tau_c = 12$ and $\tau_s = 9$. The result of $\mathcal{W}^a_9(s_{usr})$ can be seen in Figure 5.

3.2.2 Tuple-Based Windows

A tuple-based sliding window operator $\mathcal{W}^b$, with positive integer parameter N, on a stream s returns the N most recent tuples in s, i.e., the tuples $t \in s$ for which $t(\text{T}) \leq \tau_c$ and such that no other tuples exist in S that have larger time

usr_id	usr_v	usr_loc
1003	$(-12, -11)$	$(60, 80)$
1002	$(-25, 25)$	$(200, 100)$
1004	$(0, 0)$	$(100, 60)$
1003	$(-10, 3)$	$(40, 58)$

Figure 6: Result of $\mathcal{W}^b_4(s_{usr})$ at Time 12

values (that do not exceed τ_c). If ties exist, tuples are chosen at random among the ties. Note also that fewer than N qualifying tuples may exist.

A tuple-based window is specified as $\mathcal{W}^b_N(s)$. Note that $\mathcal{W}^b_\infty(s) = \mathcal{W}^a_0(s)$. As an example, recall s_{usr} in Figure 2 and let $\tau_c = 12$. Then $\mathcal{W}^b_4(s_{usr})$ is given in Figure 6.

3.2.3 Partitioned Windows

A partitioned sliding window over stream s takes a positive integer N and a subset of s's attributes, $\{A_1, \ldots, A_m\}$, as parameters. This operation first partitions S into substreams based on the argument attributes, then computes a tuple-based sliding window of size N independently on each substream, and then returns the union of these windows.

usr_id	usr_v	usr_loc
1001	$(0, 1)$	$(16, 38)$
1002	$(-25, 25)$	$(200, 10)$
1004	$(0, 0)$	$(100, 60)$
1003	$(-10, 3)$	$(40, 58)$

Figure 7: Result of $\mathcal{W}_{1,usr\_id}(s_{usr})$

Using $\mathcal{W}$ as the operator name, the partitioned window can be expressed as $\mathcal{W}_{N,A_1,\ldots,A_m}(s)$. To exemplify, consider s_{usr} in Figure 2, let $\tau_c = 12$, $N = 1$, and let the set of attributes be $\{usr\_id\}$. Then the result of $\mathcal{W}_{1,usr\_id}(s_{usr})$ is given in Figure 7.

3.3 Relation-to-Stream Operators

A relation r may be subject to updates, so that its state varies across time. We use the notation $r(\tau)$ to refer to the state of r at time τ. With this definition, we can specify the two relation-to-stream operators Istream and Rstream (adapted from [1]). The operators $\sqcup$, $\times$, and $\backslash$ are the algebra operators defined in Section 3.1.1.

Istream ("Insert" stream) maps relation R into a stream S so that a tuple $t \in r(\tau)\backslash r(\tau - 1)$ is mapped to $(t, \tau) \in s$. Next, Rstream maps relation r into stream s by tagging each tuple in r with each time that it is present in r. Assuming that 0 is the earliest time instant, the operators are defined as follows.

$$\text{Istream}(r) = \sqcup_{\tau>0}((r(\tau)\backslash r(\tau-1)) \times \{\tau\}) \sqcup (r(0) \times \{0\})$$
$$\text{Rstream}(r) = \sqcup_{\tau\geq 0}(r(\tau) \times \{\tau\})$$

Assume that a moving tourist wants to continuously know the nearest hospitals. The result, which is subject to change as the tourist moves, may be returned as a stream produced using one of the windowing operators and a relation-to-stream operator. We discuss nearest-neighbor queries in the next section.

We have so far defined relational operators and mapping operators between streams and relations. As location-based queries involve operations on spatial data, spatial operators are intrinsic to such queries. We treat spatial operators as black boxes and simply assume a set of such operators. Specifically, we will use spatial operators proposed by the OpenGIS Consortium [7].

3.4 One-Time and Continuous Queries

A one-time query is a combination of stream-to-relation and relation-to-relation operators, while a continuous query is a possibly infinite numbers of one-time queries that are run repeatedly within a specified time interval according to a specified time granularity. The result of a continuous query can either be relations or streams. To generate a stream result, relation-to-stream operators are naturally employed by the continuous query; see Figure 8.

Let a one-time query be expressed as $LBQ_p(s, r)$, where s and r are argument streams and relations and p is the parameters of the query. Then a continuous query can be expressed as $CLBQ_p(s, r)[T_s, T_e, \mathcal{G}]$, where T_s and T_e are the start and end time of the continuous query and $\mathcal{G}$ is the time granularity of the query. A relation-to-stream operator may also be included in this expression to map the results into stream.

Since relations and the associated algebraic operators accommodate duplicates and order, any queries that can be expressed using traditional relational algebra can be presented in the framework; and the framework is open to new kinds of queries, as new algebraic and mapping operators can be added.

4 Location-Based Queries

The literature covers the processing of quite a few kinds of LBQs, including range, nearest-neighbor, and reverse nearest-neighbor queries, as well as closest pair queries, spatial joins, and spatial aggregate queries. Queries can concern past states of reality, or present and (anticipated) future states. Our focus will be on queries that concern the current state.

We proceed to demonstrate how the semantics of three queries can be specified: spatial range and nearest neighbor queries, and a new location-based skyline query. We end by discussing the categorization of location-based queries.

4.1 Spatial Range Query

Various kinds of spatial range queries are used commonly. A range query may be used for finding all moving objects within a circular region around a point of interest; and a continuous range query may be used for the monitoring of a region.

We assume a spatial range sr is given and define a range query (RQ) and continuous range query (CRQ) using a combination of stream-to-relation, relation-to-relation, and relation-to-stream operators.

To define the range query, we first obtain the most recent positions of all users. This is done by applying a partition window $\mathcal{W}_{1,usr\_id}(s_{usr})$ to stream s_{usr} and by applying a function CurLoc, using an extended projection. The function takes as argument a tuple $t \in s_{usr}$ that records the movement of an object, and it returns the location at time τ_c of the object.

$$r_s = \pi_{usr\_id,usr\_v,\text{CurLoc}(t)\text{ AS loc}}(\mathcal{W}_{1,usr\_id}(s_{usr}))$$
$$\text{CurLoc}(t) = t(usr\_loc) + t(usr\_v) \cdot (\tau_c - t(\text{T}))$$

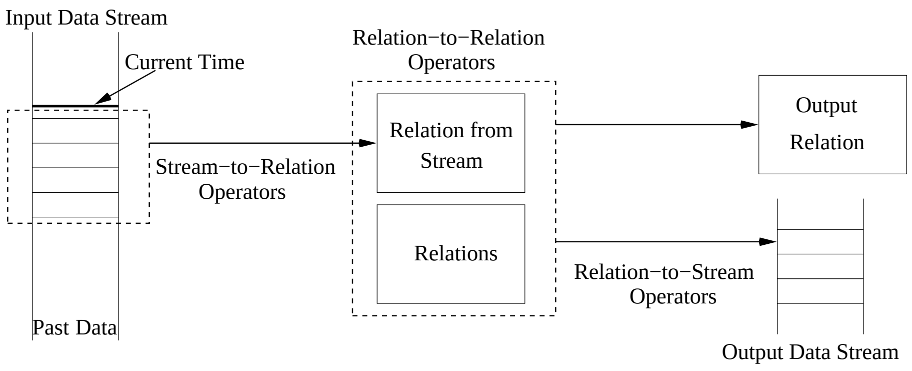

Figure 8: The Working of a Location-Based Query

Then a selection retrieves all users that are inside the spatial region. Operator within(sr, loc) returns true if loc is within spatial range sr. The one-time range query is then given as follows.

$$\mathrm{RQ}_{sr}(s_{usr}) = \sigma_{\text{within}(sr,\mathrm{loc})}(r_s)$$

Next, assume that the start and end times of the continuous range query are T_s and T_e, and that the time granularity is $\mathcal{G}$. By applying the Rstream operator, the definition of the continuous range query is given next.

$$\mathrm{CRQ}_{sr}(s_{usr})[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}] = \\ \mathsf{Rstream}(\sigma_{\text{within}(sr,\mathrm{loc})}(R_s))[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}]$$

To exemplify, let spatial range sr be a circle with radius 50 and center $(20, 35)$, and let $\mathrm{T}_s = 0$, $\mathrm{T}_e = 20$, and $\mathcal{G} = 1$. Then the result of the continuous range query against stream s_{usr} is given in Figure 9. Note that the result is a stream.

usr_id	usr_v	loc	T
...	...	...	...
1001	$(0, 1)$	$(16, 38)$	9
1001	$(0, 1)$	$(16, 39)$	10
1003	$(-12, -11)$	$(48, 69)$	11
1001	$(0, 1)$	$(16, 40)$	11
...	...	...	...

Figure 9: Result Stream of $\mathrm{CRQ}_{sr}(s_{usr})[0, 20, 1]$

According to the above definition, all the users' locations are calculated at each time instant. However, it may be that some users have not reported location data for several hours, rendering their location data useless. This suggests an alternative definition where only location data that has arrived within some time duration from the current time is used.

Using the same T_s, T_e, and $\mathcal{G}$ as above and assuming that we are only interested in location data that arrived since time τ_s, an alternative definition follows.

$$\mathrm{CRQ}_{sr,\tau_s}(s_{usr})[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}] = \\ \mathsf{Rstream}(\sigma_{\text{within}(sr,\mathrm{loc})}(r'_s))[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}]$$

In this definition, r'_s is r_s where s_{usr} is replaced by $\mathsf{Rstream}(\mathcal{W}^a_{\tau_s}(s_{usr}))$. Intuitively, this query may miss some users who are actually inside the spatial range, but have not reported their location for some time.

4.2 Nearest Neighbor Query

The k nearest neighbor query (kNNQ) is another basic LBQ. Example uses include locating the nearest hospitals or emergency vehicles. To formulate the query that finds the k nearest neighbors of an object m_id, we first define several auxiliary functions.

A partitioning window query $\mathcal{W}_{1,usr\_id}(s_{usr})$ first retrieves the most recent position data for each object from the stream. Then a selection with predicate $usr\_id = m\_id$ is applied to retrieve the position data for our object. Let r_l denote the relation resulting from this selection.

To compute the k objects nearest to m_id, we calculate, using a spatial operator "dist," the distance between m_id's current location and the locations of all other objects, which are stored in attribute *obj_loc* of r_{obj}. As the next step in computing the query, we apply a generalized projection to associate the distance to the user object with each other object:

$$r_s = \pi_{obj\_id, obj\_loc, obj\_type, \text{dis}}(r_{obj} \times r_l)$$

Here, "dis" denotes $\mathrm{dist}(obj\_loc, \mathrm{CurLoc}(t))$, function $\mathrm{CurLoc}(t)$ was defined earlier, and t denotes a tuple from the argument relation.

Then we apply the *sort* and *top* operators to r_s to express the query.

$$\mathrm{kNNQ}_{m\_id,k}(s_{usr}, r_{obj}) = top_k(sort_{\text{dis}}(r_s))$$

Let $r_p = \sigma_{obj\_type = \text{'police station'}}(r_{obj})$ contain all police stations and consider the query $\mathrm{kNNQ}_{1003,1}(s_{usr}, r_p)$ issued at time $\tau_c = 11$. The query finds the one police station nearest to user 1003. Using the definition, a window operator extracts all the most recent tuples for each user from stream s_{usr}. Then the tuple with $usr\_id = 1003$, $usr\_v = (-12, -11)$, $usr\_loc = (60, 80)$, and $\tau = 10$ is selected. The current location is approximated as $(60, 80) + (-12, -11) \cdot (11 - 10) = (48, 69)$. (We define the distance between points (x_1, y_1) and (x_2, y_2) as $|x_2 - x_1| + |y_2 - y_1|$). Among all objects in relation r_p, the object with $obj\_id = 304$ is selected.

If the user issues the same kind of query continuously while moving, a relation-to-stream operator may be used to map the result of each one-time query to a stream, which is expressed as follows:

$$\text{CkNNQ}_{m\_id,k}(s_{usr}, r_{obj})[\text{T}_s, \text{T}_e, \mathcal{G}] = \\ \text{Rstream}(\text{kNNQ}_{m\_id,k}(s_{usr}, r_{obj}))[\text{T}_s, \text{T}_e, \mathcal{G}]$$

The result of $\text{CkNNQ}_{1003,1}(s_{usr}, r_p)[0, 20, 1]$ is shown in Figure 10.

obj_id	obj_loc	obj_type	dis	τ
...	...	...	...	...
304	$(70, 80)$	police station	10	10
304	$(70, 80)$	police station	33	11
301	$(20, 35)$	police station	43	12
301	$(20, 35)$	police station	36	13
...	...	...	...	...

Figure 10: CkNNQ over Relation r_p

4.3 Location-Based Skyline Query

The query assumes the following scenario. A user drives along a pre-defined route towards a destination. The user wants to visit one or several points of interest enroute. The most attractive of the qualifying points of interest are those that are nearest to the user's current location and that result in the smallest detour. The detour is the extra distance traveled if the user visits the point of interest and then travels to the destination.

Let r_l, CurLoc and t be as defined earlier. We assume that spatial operator "dist" takes into account the user's route, and we denote the user's destination by $dest$. Then the detour fe can be expressed as follows.

$$fe(obj\_loc, t, dest) = \text{dist}(obj\_loc, \text{CurLoc}(t)) + \\ \text{dist}(obj\_loc, dest) - \text{dist}(\text{CurLoc}(t), dest)$$

Next, a (generalized) projection is applied to the Cartesian product of r_{obj} and r_l to get all the objects' distances and detours to the user:

$$r_s = \pi_{obj\_id, obj\_type, \text{dis}, \text{det}}(r_{obj} \times r_l)$$

Here, "dis" denotes $\text{dist}(obj\_loc, \text{CurLoc}(t))$ and "det" denotes $fe(obj\_loc, t, dest)$.

Finally, the skyline operation generates the result.

$$\text{SQ}_{m\_id,dest}(s_{usr}, r_{obj}) = skyline_{\text{dis},\text{det}}(r_s)$$

Following the scenario described above, let $\tau_c = 15$ and assume a user with $usr\_id = 1002$ wants to go to a hospital or a police station enroute to the destination, the location of which is $(10, 90)$. For simplicity, we use direct line segments as routes between two points. Using the calculation above, the current location of the user is $(200, 10) + (-25, 25) \cdot (15 - 12) = (125, 85)$. For all static objects with usr_type "police station" and "hospital," the distance and detour are listed in Figure 11. The skyline operator returns the last three tuples.

4.4 Towards a Categorization of Location-Based Queries

As it is obviously impossible to define all possible LBQs, we proceed to explore the space of possible LBQs by presenting several orthogonal categorizations of such queries.

obj_id	obj_type	dis	det
301	police station	155	100
302	hospital	100	10
304	hospital	80	60
305	police station	60	20

Figure 11: Intermediate Result

First, queries can be categorized based on whether they refer to data concerning the past, present, or future states of reality.

Second, queries can be categorized according to whether they are one-time or continuous queries. Continuous queries may be classified further, based on whether they are constant or time-parameterized. The latter occurs when a query refers to the (variable) current time. An example is a continuous query that retrieves all objects currently within a spatial range. A corresponding constant query might retrieve all objects that are (currently believed to be) within a spatial range at some fixed near future time. Constant continuous queries have been termed "persistent" in the literature.

Third, queries may be classified as being either "one-to-many" or "many-to-many." The former queries apply one predicate to many objects, returning one set, multiset, or list of objects. The latter conceptually repeatedly applies many different predicates to many objects, potentially retrieving many objects for each predicate.

A simple selection is thus an example of the former. The k nearest neighbor query in Section 4.2 retrieves (up to) k objects that are the nearest to some (i.e., "one") specified object; it is thus also a "one-to-many" query. In contrast, joins are "many-to-many" queries: The predicate involving one (left hand side) object is applied to many (right hand side) objects, and this repeated many times. The so-called "closest pair" query, which finds pairs of objects from two different groups that are closest, is also a "many-to-many."

Fourth, LBQs may be categorized based on whether they involve "topological," "directional," or "metric" predicates.

Fifth, based on the time at which a query is registered to the system, it can be "pre-defined," meaning that it is present before the streams it uses start, or it can be "ad hoc," meaning that it is registered after at least one of its streams has started.

5 Summary and Future Work

Substantial research has been reported on query processing in relation to mobile services, in particular location-enabled mobile services. Different techniques are applicable to different kinds of queries. Based on results from stream and temporal databases, this paper proposes a framework for capturing the semantics of the diverse kinds of queries that are relevant in this context. By enabling the definition of queries in a single framework, the paper's proposal enables the comparison of queries.

The framework consists of data types, relations and streams, as well as algebraic operations on relations and operations that map between streams and relations. The specific representations of spatial data and the associated operations on these are treated as black boxes, in order to enable applicability across different such representations and operations. The extensibility of the framework was exemplified by adding a skyline operator.

The use of the framework was illustrated by the definition of three location-based queries, a spatial range query, a nearest-neighbor query, and a location-based skyline query. Toy examples were given for illustrating these queries. Focus has been on the capture of the semantics of LBQs, and how to use one-time or continuous queries in actual location-based services is beyond the scope of the paper.

This paper represents initial work, and future work may be pursued in several directions. First, the framework may be enriched in various ways. One is to introduce explicit representations of the space within which the spatial objects are located and move, e.g., road networks. Second, while this paper has given one definition of each of three queries, it would be worthwhile to explore the different possible semantics that may be given to queries within the framework. Such a study may reveal whether or not desirable semantics can be specified in all cases. Third, more work on taxonomies for location-based queries is desirable. Interesting initial steps have been taken in this direction (e.g., [23]), but much more detail is desirable.

References

[1] A. Arasu, S. Babu, and J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. Technical Report. Department of Computer Science, Stanford University, 12 pages, 2002.

[2] P. Atzeni and V. De Antonellis. *Relational Database Theory.* Benjamin/Cummings, 1993.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proc. PODS,* pp. 1–16, 2002.

[4] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Šaltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *Proc. IDEAS,* pp. 44–53, 2002.

[5] S. Borzsonyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proc. ICDE,* pp. 421–430, 2001.

[6] D. Carney, U. Centintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring Streams: A New Class of Data Management Applications. In *Proc. VLDB,* pp. 215–226, 2002.

[7] E. Clementini and P. D. Felice. Spatial Operators. In *SIGMOD Record,* **29** (3), pp. 31–38, 2000.

[8] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *Proc. ICDE,* pp. 717–816, 2003.

[9] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *Proc. SIGMOD Conf.,* pp. 189–200, 2000.

[10] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. SIGMOD Conf.,* pp. 319–330, 2000.

[11] C. Hage, C. S. Jensen, T. B. Pedersen, L. Speičys, and I. Timko. Integrated Data Management for Mobile Services in the Real World. In *Proc. VLDB,* pp. 1019–1030, 2003.

[12] A. Hinze and A. Voisard. Location- and Time-Based Information Delivery in Tourism. In *Proc. SSTD,* pp. 489–507, 2003.

[13] S. Idreos and M. Koubarakis. P2P-DIET: Ad-hoc and Continuous Queries in Peer-to-Peer Networks Using Mobile Agents. In *Proc. SETN,* pp. 23–32, 2004.

[14] D. V. Kalashnikov, S. Prabhakar, W. G. Aref, and S. E. Hambrusch. Efficient Evaluation of Continuous Range Queries on Moving Objects. In *Proc. DEXA,* pp. 731–740, 2002.

[15] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proc. VLDB,* pp. 275–286, 2002.

[16] X. Liu, H. Ferhatosmanoglu. Efficient k-NN Search on Streaming Data Series. In *Proc. SSTD,* pp. 83–101, 2003.

[17] H. Mokhtar and J. Su. Universal Trajectory Queries for Moving Object Databases. In *Proc. MDM,* pp. 133–145, 2004.

[18] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously Adaptive Continuous Queries Over Streams. In *Proc. SIGMOD Conf.,* pp. 49–60, 2002.

[19] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *Proc. SIGMOD Conf.,* 2004.

[20] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group Nearest Neighbor Queries. In *Proc. ICDE,* pp. 301–312, 2004.

[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *Proc. SIGMOD Conf.,* pp. 467–478, 2003.

[22] L. Qiao, D. Agrawal, and A. E. Abbadi. Supporting Sliding Window Queries for Continuous Data Streams. In *Proc. SSDBM,* pp. 85–94, 2003.

[23] A. Y. Seydim, M. H. Dunham, and V. Kumar. Location Dependent Query Processing. In *Proc. MobiDE,* pp. 47–53, 2001.

[24] S. Šaltenis and C. S. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proc. ICDE,* pp. 463–472, 2002.

[25] G. Slivinskas, C. S. Jensen, and R. T. Snodgrass. Query Plans for Conventional and Temporal Queries Involving Duplicates and Ordering. In *Proc. ICDE,* pp. 547–558, 2000.

[26] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proc. SSTD,* pp. 79–96, 2001.

[27] S. Shekhar and J. S. Yoo. Processing In-Route Nearest Neighbor Queries: A Comparison of Alternative Approaches. In *Proc. ACMGIS,* pp. 9–16, 2003.

[28] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *Proc. VLDB,* pp. 287–298, 2002.

[29] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal and Spatio-Temporal Aggregations over Data Streams Using Multiple Time Granularities. In *INF. SYST.* **28** (1–2), pp. 61–84, 2003.