Scalable Management of Trajectories and Context Model Descriptions

Ralph Lange

Institute of Parallel and Distributed Systems (IPVS)
Universität Stuttgart
Universitätsstraße 38, 70569 Stuttgart*

Abstract. The ongoing proliferation of sensing technologies constitutes a huge potential for context-aware computing. It allows selecting relevant information about our physical environment from different sources and providers all over the globe. A fundamental challenge is how to provide efficient access to these immense amounts of distributed dynamic context information – particularly due to the mobility of devices and other entities. To enable such access to current and past position information about moving objects, we propose a family of protocols (CDR, GRTS) for efficiently tracking a moving object's trajectory at some remote database in real-time as well as a distributed indexing scheme (DTI) for optimized access to trajectory data that is partitioned in space to multiple database servers. For discovering context information that is relevant for the situation of an application, we propose a powerful formalism for describing context models in a concise manner and a tailored multidimensional data structure (SDC-Tree) for retrieving relevant context models out of potentially millions of descriptions.

1 Introduction

Context-awareness refers to the idea that applications and systems adapt to their context of use including location, noise, nearby devices and user habits amongst others. From the beginnings in the 1990s, billions of sensors (cameras, satellites, smartphones, RFID readers, ...) have been deployed all over the globe [4]. The immense amounts of data acquired by these sensors allow creating comprehensive models of our physical environment, such as detailed roadmaps with real-time traffic data and 3D building models.

The availability of such models constitutes a huge potential for context-aware applications and systems. They are no longer bound to predefined sensors but can dynamically select relevant context information from different providers. In future, millions of context models are expected to be available from different providers [16].

*The work described in this paper was performed at the Universität Stuttgart and partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627. The author thanks Prof. Dr. Kurt Rothermel and Dr. Frank Dürr for their great support and the many fruitful discussions. He is now with TRUMPF Werkzeugmaschinen GmbH + Co. KG.

The sharing of context information poses many challenges [8]. In the last years, a number of context management frameworks and systems have been proposed for mobile devices or buildings with networked devices and sensors (e.g. [15, 3]). Large-scale or global scenarios, however, require *distributed* approaches for context management and raise a number of new questions. A fundamental challenge is how to provide efficient access to the immense amounts of distributed, dynamic context information.

In the first instance, a scalable approach to discover the context models that are relevant for the situation of an application is needed [8]. This comprises two subproblems:

1. *Formally describing context models:* To decide whether a context model is relevant for the situation of an application, an adequate formalism for describing context models and for matching these descriptions against compatible queries is needed.

2. *Indexing of context model descriptions:* To discover the relevant context models out of potentially millions of descriptions, an appropriate index structure is required.

Information about the positions and movements of mobile objects – i.e. about their trajectories – is of particular importance for context-aware computing, since it may serve as index to other context information. Therefore, moving objects databases (MODs) [6] are used in context management systems to track and manage trajectory information about mobile objects. The need to manage large numbers of trajectories in real-time raises two subproblems beyond existing solutions:

3. *Efficient real-time trajectory tracking:* Many positioning systems (e.g. GPS) are based on sensors that are attached to the moving objects and thus require transmitting the position data over a wireless network to the MOD. To minimize storage consumption and communication cost, efficient tracking protocols are needed that allow trading these costs off against data accuracy.

4. *Distributed indexing of space-partitioned trajectories:* Managing a large number of trajectories may require partitioning the data to multiple database servers. A promising scheme is spatial partitioning as it supports scalable processing of queries about single trajectories as well as of queries about all trajectory sections at a certain location. The former class of queries, however, requires a distributed index structure over the partitions of each trajectory.

Following the general layering of context management systems – from the sensors up to the applications – we propose solutions for the third and fourth subproblems in the Sections 2 and 3, before we tackle the first two subproblems in Section 4. Finally, the paper is concluded in Section 5 with a summary.

2 Efficient Real-Time Trajectory Tracking

Usually, a moving objects trajectory is represented as a polyline in time and space, where the vertices are timestamped positions acquired by a positioning system [6]. Many of these systems (including GPS) are based on sensors that are attached to the moving objects. Transmitting and storing every sensed position, however, causes high communication costs and generally consumes too much storage capacity. Therefore, a tracking protocol is needed that allows trading these costs off against the accuracy of the trajectory information known to the MOD.

A simple approach to reduce the storage consumption is to transmit all sensed positions to the MOD and to reduce the amounts of data at the server-side using line simplification

algorithms such as the Douglas-Peucker algorithm [5]. This approach, however, does not tackle the problem of communication cost.

For tracking the current position of a moving object in real-time, various works (e.g., [19, 12]) proposed the use of dead-reckoning: With this technique, the update messages by the moving object include a prediction function for the future movement of the object. The object does not need to send another update as long as its locally sensed position does not impend to deviate from the prediction by more than some accuracy bound ϵ. The most simple but nevertheless efficient variant is linear dead reckoning (LDR), using a linear prediction given by a timestamped position and a velocity vector.

Dead reckoning reduces the amount of update messages significantly. Yet, it does not generate a connected polyline. Trajcevski et al. show in [17] that LDR with $\epsilon' = \epsilon/2$ allows for trajectory tracking with accuracy bound ϵ, which is very conservative and leaves room for significant improvement.

We propose Connection-Preserving Dead Reckoning (CDR) and Generic Remote Real-Time Trajectory Simplification (GRTS), which compose a family of trajectory tracking protocols. Both allow trading the communication cost and storage consumption off against a tolerated deviation ϵ and incorporate sensor inaccuracies and transmission delays.

2.1 Connection-Preserving Dead Reckoning

CDR is based on LDR but includes a second update condition: It causes an update not only if the object impends to deviate from the prediction by more than ϵ but also if the current sensed position cannot be used as vertex for a simplified polyline. The latter case occurs if the line segment between the position given in the last update message and the current position (possibly) deviates by more than ϵ from the actual movement in-between. In this case, CDR transmits a new update message including the previously sensed position (which fulfilled both update conditions) and a new velocity prediction. For evaluating the second update condition, CDR keeps all sensed positions after the last update message in the *sensing history* $\mathbb{S}$.

It can be shown, that many positions may be removed from $\mathbb{S}$ even before the next update without affecting CDR's second update condition. This optimization reduces the size of $\mathbb{S}$ by 49% on average, cf. [11].

Despite this optimization, the size of $\mathbb{S}$ is generally unbounded and thus the computing time per sensing operation. For this reason, we propose the variant CDR_m, which limits the size of $\mathbb{S}$ to a predefined parameter m. CDR_m maintains an additional floating-point variable $d_\mathbb{S}$ providing aggregated information about all sensed positions that could not be stored in $\mathbb{S}$. The second update condition is split into two subconditions for $\mathbb{S}$ and $d_\mathbb{S}$, correspondingly.

Extensive simulations with real GPS traces show that CDR_m with $m = 500$ reach the same reduction rates as CDR, while limiting the computing time to 0.03 ms on a 3 GHz PC processor.

2.2 Generic Remote Real-Time Trajectory Simplification

GRTS separates the simplification of the past trajectory as far as possible from tracking of the current position using LDR. Therefore, it can be realized with any line simplification algorithm.

GRTS likewise maintains a sensing history $\mathbb{S}$. If the moving object impends to deviate from the prediction by more than ϵ, GRTS applies the simplification algorithm to $\mathbb{S}$ and adds the vertices of the resulting simplified trajectory to the new update message. Figure 1 shows the corresponding pseudo code. The vertices in the update message may replace vertices of a previous update, i.e. the simplification computed by the moving object may replace an existing section of the simplified trajectory stored by the MOD – unlike CDR, which always adds one vertex per update message.

```
1. [...]                                  ▷ Initialization and first update.
2. while report movement do
3.   s_R ← sense position
4.   S ← S || (s_R)                       ▷ Append s_R to sensing history.
5.   if LDR causes update then
6.     U ← line simplification with bound ε on S
7.     U ← U \ (first(U))                 ▷ Belongs to stable part.
8.     v_V ← compute new predicted velocity ...
9.     send update (|V \ U|, U \ V, v_V) to MOD
10.    V ← U                              ▷ Variable part.
11.    U ← ()                             ▷ Clear vertices for update.
12.    if V and S should be reduced then
13.      V' ← some prefix of V for stable part ...
14.      S ← (s ∈ S | s.t ≥ last(V').t)
15.      V ← V \ V'                       ▷ Set new variable part.
16.    end if
17.  end if
18. end while
19. [...]                                 ▷ Final simplification and update.
```

Figure 1: Basic GRTS algorithm.

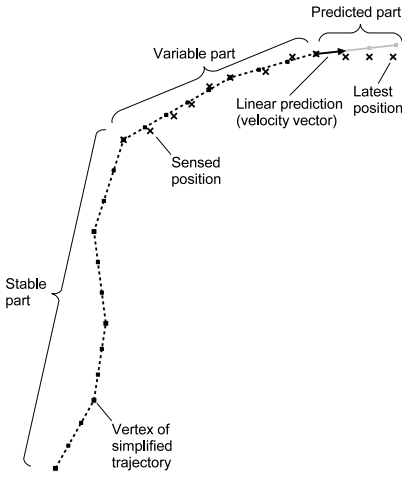

Figure 2: Three parts of the simplified trajectory by GRTS.

GRTS implicitly divides the simplified trajectory into three parts as illustrated in Figure 2: The *stable part* is stored by the MOD only. The *variable part* is known to the MOD and the moving object. The *predicted part* is given by the prediction function of the last update message. $\mathbb{S}$ comprises the sensed positions (denoted by small crosses) of the variable and predicted part.

Two basic variants GRTS_k and GRTS_m are differentiated: GRTS_k limits the number of vertices of the variable part to k, but not the size of $\mathbb{S}$. Therefore, it is of theoretical interest only.

In contrast, GRTS_m limits $|\mathbb{S}|$ to a predefined parameter m. If $|\mathbb{S}|$ reaches m, the simplification algorithm is applied to $\mathbb{S}$. The vertices of the simplification are stored for the next update message and the sensed positions that are spanned by the first line segment of the simplification are removed from $\mathbb{S}$. Hence, GRTS_m generates a new vertex for the simplified trajectory every m sensing operations. To alleviate this undesirable effect, we propose an advanced variant GRTS_mc, using a compression technique on $\mathbb{S}$. The fundamental idea of this compression technique is to keep the results of a simplification caused by m in $\mathbb{S}$, to be able to revise and improve this simplification later on. As explained in [11], this technique also allows to directly incorporate varying sensor inaccuracies.

2.3 Evaluation

We realized GRTS_k, GRTS_m, and GRTS_mc with the optimal line simplification algorithm by Imai and Iri [7] and a simple online heuristic which has been proposed in various works (e.g., [13]). We abbreviate the former realizations by $\text{GRTS}_\text{k}^\text{Opt}$, $\text{GRTS}_\text{m}^\text{Opt}$, and $\text{GRTS}_\text{mc}^\text{Opt}$ and the latter ones by $\text{GRTS}_\text{k}^\text{Sec}$, $\text{GRTS}_\text{m}^\text{Sec}$, and $\text{GRTS}_\text{mc}^\text{Sec}$.

In simulations with more than 330 hours of real GPS traces, we compared the GRTS realizations to the CDR variants, to the existing trajectory tracking approach based on LDR with $\epsilon' = \epsilon/2$ (LDRH) as well as to the optimal line simplification algorithm (Ref^Opt) by Imai and Iri and the Douglas-Peucker algorithm (Ref^DP).

Figure 3 shows the reduction rate – i.e. the number of sensed positions divided by the number of vertices of the simplified trajectory – depending on ϵ. The reduction rate of $\text{CDR}_\text{m}$with

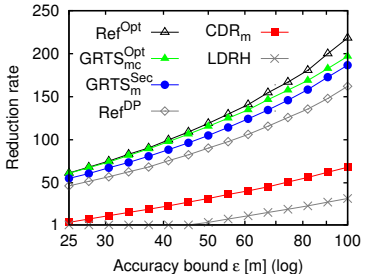

Figure 3: Comparison of reduction rates.

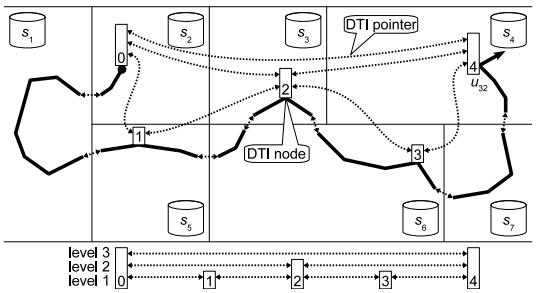

Figure 4: Skip list and DTI.

$m = 500$ (or CDR) is at least twice the reduction rate of LDRH. All GRTS realizations (with $m = 500$) again outperform the CDR variants by at least factor 2.7. This confirms the importance of separating tracking of the current position from simplification of the past trajectory.

Moreover, the GRTS realizations reach the best possible reduction by at least 85% and always outperform Ref^{DP}, which is a surprising result given the fact that Ref^{DP} is performed offline on the entire GPS traces.

We further analyzed the communication costs and the computational costs, also by means of a prototypical implementation for Android-based smartphones [11]. From these experiences we draw the following conclusions for the selection of a tracking protocol:

1. $\text{GRTS}^{\text{Sec}}_{\text{m}}$ affords high reduction performance at comparatively low computational costs and is easy to implement. Therefore, we suppose that it meets the requirements of most use cases.

2. $\text{GRTS}^{\text{Opt}}_{\text{mc}}$ should be used if reduction has maximum priority and the moving objects have sufficient computational power.

3. CDR_{m} should be selected if communication costs have maximum priority, since it minimizes the amounts of transmitted data.

3 Distributed Indexing of Space-Partitioned Trajectories

Queries about trajectories, or about context information that is associated with trajectories, can be classified into *coordinate-based queries* and *trajectory-based queries* [6]. Queries of the former class refer to all trajectories satisfying a certain spatial relationship to a specific region or point – such as a range query, returning all objects in a given region during a given time interval. Queries of the latter class refer to the trajectory of a single moving object.

In case that the number of trajectories managed by a MOD exceeds the capacity of a single database server, we can similarly distinguish two schemes for partitioning trajectory data to multiple servers:

1. *Object-based partitioning:* Each trajectory is stored entirely on a single server.

2. *Spatial partitioning:* Each server is associated with a certain geographic *service region* and stores all trajectory sections inside this region.

A critical issue with query processing in such a MOD is routing a given query to the servers that store the queried data. Our analysis in [9] shows that spatial partitioning is superior to object-based partitioning in terms of query routing: For a coordinate-based query, the set of servers that store relevant data is given directly by the queried region and the mapping from space to servers. An algorithm for efficient distributed processing of range queries has been proposed in [18]. For a trajectory-based query the set of relevant servers is not given directly – but it is limited to few neighboring servers due to the functional dependency between space and time defined by the queried trajectory. Therefore, we propose a distributed index for efficient routing of trajectory-based queries in space-partitioned MODs.

3.1 Distributed Trajectory Index

We distinguish three phases for processing a trajectory-based query: In the first phase, the query is routed to an arbitrary server storing a section of the queried trajectory. One may think of different solutions such as dedicated directory service, a distributed hash table or an anycast-like discovery protocol. In the second phase, the query is routed along the trajectory from server to server until the server is reached that stores the start (or end) of the queried trajectory section. In the third phase the query is being processed and further routed along the queried trajectory section if necessary.

If a server stores two (or more) sections of the queried trajectory, the second phase may be shortened by skipping the section in-between. The idea of the Distributed Trajectory Index (DTI) is to introduce explicit shortcuts to temporal distant sections stored by other servers. Such a DTI pointer is simply a copy of the position data it refers to. The pointer can be used to route a query directly to the server that stores this position as illustrated in Figure 4. Thus, a DTI realizes a temporal routing overlay on top of the geographic routing network between the servers.

The DTI scheme employs the idea of maintaining multiple pointers spanning different distances from skip lists [14] as illustrated at the bottom of Figure 4: It periodically creates a DTI node with increasing sequence number at the current end of the trajectory and then creates DTI pointers to previous DTI nodes (and vice-versa) in style of a perfect, bidirectional skip list.

3.2 DTI with Summaries

DTI solely accelerates the second phase of query processing. For aggregate queries, our extended scheme DTI+S furthermore increases the routing performance in the third phase. It augments each DTI pointer with a *summary*, referring to the trajectory section cut short by that pointer. Summaries contain aggregated information about that section such as the length or the minimum bounding rectangle. In the third phase, the DTI+S algorithm decides based on the query type and the partial results whether one or more summaries can be used to speed up the processing. If so, it selects the most beneficial summary and routes the query along the corresponding DTI pointer.

3.3 Evaluation

We evaluated DTI and DTI+S by simulating a space-partitioned MOD consisting of 1000 servers over one year. The number of objects (and thus trajectories) is irrelevant since a DTI is maintained independently for each trajectory. The servers together cover a rectangular area of $4500\,\mathrm{km} \times 2000\,\mathrm{km}$ ($\approx$ Continental U.S.). After the simulation of eight months, 10^7 random queries have been posed about different section of the trajectories.

The results show that DTI reduces the average time in the second phase by 69% compared to routing without DTI. DTI+S may even reduce the overall time for processing aggregate queries by more than 95%. See [9] for details.

4 Describing and Indexing of Context Models

Regarding the discovery of context models, a distributed global-scale context management system can be considered as a heterogeneous information systems (HIS) consisting of a large number of information sources. Most existing approach for source discovery in HIS utilize the schema mappings between the sources or to some shared schema to exclude those sources from processing a certain query or task that do not provide any information about the queried relations or classes (e.g. [2, 1]). However, this approach does not scale to a global context management system, where thousands of providers of context models may use the same classes and relations to provide information about different clippings/locations – e.g., 3D models of buildings all over the world – and applications are interested in one or few models only.

Therefore, a dedicated formalism that utilizes explicit descriptions of the sources' contents is required. Different approaches are imaginable, ranging from simple keyword lists to logic-based formalisms using constraints. We observe that those two extremes also imply different semantics for matching a source description against a discovery query: The former implies *positive semantics* in view of the fact that a description matches a given query only if both share one or more explicitly stated keywords. Logic-based formalisms, on the contrary, generally imply *negative semantics* as they consider a description to match a query unless the converse can be proven by the corresponding constraints.

In the following, we propose an extended logic-based description formalism for *ontology-based* HIS – which applies to context management systems in particular – that allows to trade off between positive and negative matching semantics.

4.1 Description Formalism

Every source (or context model) is described by one or more *defined classes*, i.e. by a class of the shared ontology followed by constraints such as

$$D_1 = \langle \textsf{BuildingPart} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{\text{“British Museum”}\}\rangle\rangle \ ,$$

to describe a building model of the British Museum. Such a *source class* should be 'minimal' in terms of the number of constraints, i.e. alternative constraints should be used for additional source classes. For example, D_1 does not require a second constraint on the location, as the name {“British Museum”} unambiguously refers to the famous museum in London. Instead, the location can be used for a second source class

$$D_2 = \langle \textsf{BuildingPart} : \textsf{location} \in \{44 \text{ Gt Russell St, London, UK}\}\rangle \ .$$

This differentiation allows to discover the building model either by the name or the location.

A query Q, in contrast, consists of exactly one defined class containing all constraints known to the application or user.

To evaluate a query Q against a source description $\{D_1, \ldots, D_n\}$ we define two predicates. The *query matching predicate* $\rightsquigarrow_{\text{Q}}$ forms a necessary condition for matching. It is $D_i \rightsquigarrow_{\text{Q}} Q$ iff every constraint given in D_i is overlapped by a constraints given in Q. The query dismatching predicate $/\!\!/_{\text{Q}}$ forms a sufficient condition for dismatching. It is $D_j /\!\!/_{\text{Q}} Q$ iff D_j and Q have constraints on the same attribute or relation that do not overlap. By adding *pseudo constraints* $a \in *$ or $r \in *$ to Q, the query issuer can trade the influence by $\rightsquigarrow_{\text{Q}}$ off against the influence by $/\!\!/_{\text{Q}}$.

4.2 Source Description Class Tree

To realize a scalable discovery service for a distributed context management system or general HIS, we further propose the Source Description Class Tree (SDC-Tree), which is tailored to the above formalism.

Every node of the tree represents a extended defined class N_i named *node class*. Every node class subsumes the node classes of the child nodes by the so-called index subsumption predicate $\succeq_{\mathrm{I}}$, which implies $\rightsquigarrow_{\mathrm{Q}}$. The node classes thus compose a partial order, reflecting the tree. The node class of the root is $\langle C_\top, \textsc{True} : \rangle$, where $C_\top$ denotes the top class of the ontology.

A defined class D_i of a source description is passed down the tree to those leaf nodes whose node classes match D_i by the index matching predicate $\rightsquigarrow_{\mathrm{I}}$, implying $\rightsquigarrow_{\mathrm{Q}}$. Thus D_i is stored at every leaf with node class N_i, where $N_i \rightsquigarrow_{\mathrm{I}} D_i$. Similarly, a query Q is passed down the tree, but using $\rightsquigarrow_{\mathrm{Q}}$ instead of $\rightsquigarrow_{\mathrm{I}}$. If a node class N_i does not match Q, the corresponding subtree is pruned.

Two defined classes may differ in three aspects: (1.) In terms of the ontology class, (2.) regarding the existence of constraints on a certain attribute or relation, and (3.) in the value ranges of the constraints. The node classes allow to distinguish source classes by all three aspects. Correspondingly, there exist three ways to split a leaf node and its node class into two or more child nodes and node classes.

In [10], we propose the Generic Split Algorithm (GSAlg) to split a leaf node automatically once the number of defined classes at a leaf node exceeds a certain threshold. In this case, GSAlg rates all possible splits and chooses the best one for the present set of defined classes. Details on the performance of the SDC-Tree using GSAlg are also documented in [10].

5 Summary

In future, context-awareness will be a key characteristic of most (mobile) applications and services. Driven by the advances of sensing technologies, millions of context-models for different aspects and clippings of the physical world can be expected.

Sharing these models by a wide variety of applications poses a number of challenges. The first fundamental problem is how to provide efficient access to such immense amounts of distributed dynamic context information [8]. For discovering context models that are relevant for the situation of an application, we proposed a powerful formalism allowing describing context models concisely by multiple defined classes as well as the Source Description Class Tree (SDC-Tree) for indexing such descriptions. To provide efficient access to context information about moving objects, we further proposed a family of protocols (CDR, GRTS) for efficient real-time trajectory tracking and the Distributed Trajectory Index (DTI) to optimize query processing in space-partitioned moving objects databases.

References

[1] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. *ACM SIGMOD Record*, 3(3):53–58, Sept. 2003.

[2] S. Bergamaschi, S. Castano, and M. Vincini. Semantic Integration of Semistructured and Structured Data Sources. *ACM SIGMOD Record*, 28(1):54–59, Mar. 1999.

[3] H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich, and D. Chakraborty. Intelligent Agents Meet the Semantic Web in Smart Spaces. *IEEE Internet Computing*, 8(6):69–79, Nov. 2004.

[4] C.-H. Chen-Ritzo, C. Harrison, J. Paraszczak, and F. Parr. Instrumenting the planet. *IBM Journal of Research and Development*, 53(3):1:1–1:16, May 2009.

[5] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, Dec. 1973.

[6] R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

[7] M. Iri and H. Imai. *Computational Morphology*, chapter Polygonal Approximations of a Curve – Formulations and Algorithms, pages 71–86. North-Holland Publishing Company, 1988.

[8] R. Lange. *Scalable Management of Trajectories and Context Model Descriptions*. PhD thesis, Universität Stuttgart, Germany, Nov. 2010.

[9] R. Lange, F. Dürr, and K. Rothermel. Scalable Processing of Trajectory-Based Queries in Space-Partitioned Moving Objects Databases. In *Proc. of 16th ACM GIS*, Irvine, CA, USA, Nov. 2008.

[10] R. Lange, F. Dürr, and K. Rothermel. Indexing Source Descriptions based on Defined Classes. In *Proc. of 14th IDEAS*, Montreal, QC, Canada, Aug. 2010.

[11] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *The VLDB Journal*, 20(5):671–694, Oct. 2011.

[12] A. Leonhardi and K. Rothermel. A Comparison of Protocols for Updating Location Information. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 4(4):355–367, Oct. 2001.

[13] N. Meratnia and R. A. de By. Spatiotemporal Compression Techniques for Moving Point Objects. In *Proc. of 9th EDBT*, Heraklion, Crete, Mar. 2004.

[14] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[15] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, Oct. 2002.

[16] K. Rothermel, D. Dudkowski, F. Dürr, M. Bauer, and C. Becker. Ubiquitous Computing – More than Computing Anytime Anyplace? In *Proc. of 49th Photogrammetric Week*, Stuttgart, Germany, Sept. 2003.

[17] G. Trajcevski, H. Cao, P. Scheuermann, O. Wolfson, and D. Vaccaro. Online Data Reduction and the Quality of History in Moving Objects Databases. In *Proc. of 5th MobiDE*, Chicago, IL, USA, June 2006.

[18] G. Trajcevski, H. Ding, P. Scheuermann, and I. F. Cruz. BORA: Routing and Aggregation for Distributed Processing of Spatio-Temporal Range Queries. In *Proc. of 8th MDM*, Mannheim, Germany, May 2007.

[19] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7(3):257–287, July 1999.