Semantic Optimisation for CEP

Olga Poppe
Supervisor: François Bry
Institute for Informatics
University of Munich, Germany
poppe@pms.ifi.lmu.de

ABSTRACT

Semantic optimisation (SO) of database queries, i.e. the use of metadata for query optimisation, is well investigated and has led to significant performance gains. SO for databases is, to a large extent, applicable to complex event processing (CEP) when regarding events as data and complex event specifications as queries.

However, simply transferring database techniques does not unleash the full potential of SO for CEP. In contrast to database systems where incoming ad hoc queries are evaluated against known and finite data which is available at once, CEP applications permanently evaluate a known set of standing queries against event data arriving on potentially infinite streams. Therefore, in CEP, queries are usually optimised (not the data like in database systems) and more complex and expensive algorithms may be used in static SO of event queries than of database queries.

In CEP, constraints are valid and queries are satisfiable during particular application states. Therefore, a new kind of metadata capable of formalising states, expressing complex state-related dependencies, and representing an arbitrary number of parallel processes is required. To this end, Instantiating Hierarchical Timed Automata (IHTA) are introduced. For formulating constraints in the context of states, the Event Stream Constraint Language ESCL is presented. ESCL constraints are short but expressive logic formulas capturing cardinality, causal, temporal and other dependencies between events and states. Queries are optimised using IHTA and ESCL constraints by the SO algorithm for CEP. The approach is illustrated by use cases.

Categories and Subject Descriptors

H.2.1 [**Information Systems**]: Database management—*Logical design*; H.2.4 [**Information Systems**]: Database management—*Systems*; G.1.6 [**Mathematics of Computing**]: Numerical Analysis—*Constrained optimisation*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and constraint programming*; F.1.1 [**Theory of Computation**]: Models of Computation—*Automata*

General Terms

Theory, Algorithms, Languages, Experimentation

Keywords

Event processing, Semantic query optimisation, Models of application-specific knowledge, Constraints, Automata, Modularisation

1. INTRODUCTION

Integrity constraints are formulas describing the domain of a database. They were initially developed to validate changes to a database, i.e., to ensure that the state of the database after its update remains consistent. In the middle of 1970's, researchers recognised that semantic information stored in databases as integrity constraints could be used for query optimisation. The idea of semantically transforming a query into a more efficient form using metadata (such as integrity constraints) is called *semantic query optimisation*. The result of the transformation is a query that may be *syntactically quite different* from the original one, but is guaranteed to be *semantically equivalent* to the original query, i.e., to return the same results as the original query for all databases satisfying the metadata.

The idea of semantic optimisation (SO) is applicable to CEP. Indeed, many event-based applications have rich semantics defined either by application rules like in the auction use case or by physical laws like in the sensor network use case. Both use cases are considered in Section 3. However, SO of event queries differs from classical SO due to the following peculiarities of CEP:

1) **Standing queries and moving data**

In databases, evaluation is *query-driven*. Large (but finite) data is permanently saved, available at once and known. Ad hoc queries are evaluated once against the whole data. Query answers are expected almost immediately. (Standing queries are rather typical for a data warehouse than for a database.) Because of these reasons, in databases, data is usually optimised. Static SO of queries must be efficient.

In CEP, evaluation is *data-driven*. Standing known and available queries are continuously evaluated against events arriving on streams. Streams are unbounded, potentially infinite, never available at once and their contents is often unknown. Query answers are computed in multiple (possibly arbitrarily many) evaluation steps. Each step takes place as soon as events become available. Ad hoc queries are not typical for CEP. They usually have to wait until relevant events arrive. They cannot be evaluated immediately since (some) past relevant events are probably already garbage collected. Because of these reasons, in CEP, queries are usually optimised. Their static SO may rely on more complex and expensive algorithms than static SO of database queries.

2) **Time-awareness**

In databases, tuples and integrity constraints are valid until update.[1] Query satisfiability can be determined at compile time.

In CEP, events occur, constraints are valid and queries are satisfiable during particular periods of time. Occurrence time of events and validity time of constraints is their inherent part. Validity time of constraints and time periods during which queries are satisfiable can be determined by application states, i.e. their exact duration is usually known at runtime. Therefore, a new kind of metadata is required for SO of CEP. This metadata must (1) formalise application states to allow for formulating CEP constraints and queries in the context of states to increase the expressiveness of constraints and queries, (2) capture complex temporal and causal relations between events and states as needed in many various CEP applications, and (3) represent an arbitrary number of parallel processes since in real life applications their number is not known beforehand. These requirements are illustrated by the use cases in Section 3. To cope with them, Instantiating Hierarchical Timed Automata are introduced in Section 4.

3) **Materialised views**

In databases, views are usually defined to keep queries short and readable. View maintenance is rather expensive because tuples of a view can be manually inserted, changed and deleted. Therefore, views are usually not materialised and, as a consequence, multi-query SO is limited.

In CEP, the relatively few results of all queries are usually materialised in order to delete relatively many base events and states. Derived events and states can be inserted, states can be changed but neither events nor states can be manually deleted (they are garbage collected). Hence, view maintenance in CEP is cheaper than in databases and views are usually materialised in CEP. As a consequence, multi-query SO plays a more important role in CEP than in databases.

Summarising, SO of event queries differs from SO of database queries as follows: (1) Static SO of event queries may rely on more complex and expensive algorithms than static SO of database queries. (2) A new kind of metadata is required for SO of CEP. (3) Multi-query SO plays a more important role in CEP than in database systems.

Each section of this paper corresponds to a chapter of the PhD work. They are related work (Section 2), use cases (Section 3), IHTA (Section 4), ESCL (Section 5), constraint-based subsumption (Section 6), the SO algorithm for CEP (Section 7), and finally future work (Section 8). Each section briefly describes overall ideas, done work and open issues.

2. RELATED WORK

Section 2.1 is devoted to the classification of the approaches on SO of CEP according to four criteria. With the help of these classifications, the approach presented in this article is motivated. Section 2.2 briefly describes (Hierarchical) Timed Automata upon which IHTA are based.

2.1 Approaches on SO of CEP

1) Regarding the way of expressing application semantics, the approaches are divided into two groups, namely SO by means of either *constraints* or *queries* themselves.

Constraints are formulated in DDL [22], DTD [37, 36], [27, 28], [16], [24], [23], [32], as regular expressions [19], [18], [21], [38], denial rules [17], or language independent formulas [37, 36], [16], [21], [8], [5], [39], [4, 33]. No approach uses a constraint language which is tailored to the peculiarities of CEP discussed in Section 1. Most approaches consider particular kind(s) of constraints to a limited extent. These kinds of constraints are conditions on event data [19], [18], [5], [38], cardinality constraints [37, 36], [16], [22], [5], [39], [4, 33], temporal [37, 36], [16], [19], [18], [21], [22], [8], [5], [38], [17] and causal dependencies [37, 36], [17]. Constraints are expressed only on events. They do not involve application states.

Not only constraints but also queries capture application semantics. Hence, they can be used for the SO of CEP as done in [7], [20], [26], [13], [12], [30], [3], [9, 31].

No approach describes the application workflow by automata and uses the semantics captured by them (e.g., temporal and causal dependencies, states) for SO of CEP.[2]

2) According to the time at which SO happens, the approaches are classified into *static*, *dynamic*, or *both*.

Most approaches are static [27, 28], [16], [32], [21], [26], [22], [30], [9, 31], [7], [20], [13]. They rely upon application semantics which is known at compile time and does not change at runtime.

There are some dynamic approaches. In [5], constraints are derived from the event stream. [38], [18] use constraints arriving on streams. [39], [12], [4, 33] rely on stream rate. In [17], [8], [24], constraints are known at compile time. They do not change at runtime but they are applied at runtime. [3] dynamically selects time intervals upon which query results are shared.

[37, 36], [23], [19] allow both static and dynamic SO of event queries.

3) Concerning language independency, there are numerous *specific* and few *general* approaches.

Some approaches are specific for XML data streams and XPath [24], [23] or XQuery [37, 36], [27, 28], [16], [32]. However the ideas of these approaches are independent from an event format or a query language. These approaches have poor event concept. An event is an XML document (or even a single tag) without occurrence time.

Other approaches are tailored to particular relational algebra operator(s) and therefore to particular kind(s) of queries. Much attention has been paid to join [19], [18], [8], [5], join combined with *count* [22], join combined with selection [13], [12], join or grouping [38], join, selection or projection [39], equi-join or aggregation [21], sliding-window aggregates [30], [3], event sequence or conjunction [17]. Join and aggregation are expensive operations requiring storage of relevant events and blocking of queries which cannot be evaluated as long as not all events are available.

Few approaches are general [7], [20], [26], [9, 31], [4, 33].

4) With respect to the implemented SO technique, there

[1] Disregarding soft constraints that are not necessarily always enforced.

[2] [17], [32], [36], [23] transfer queries into finite automata and evaluate the automata while events arrive.

are approaches aiming at:
– *Load distribution* [21], [26],
– *Storage minimisation* [27, 28], [5], [38], [12], [30], [9, 31],
– *Efficient data structures* [7], [20],
– *Computation sharing* [16], [13], [12], [30], [3],
– *Efficient join algorithms* [19], [18],
– *SO heuristics* such as join elimination [22] and detection of (temporary) unsatisfiable (sub) queries [37, 36], [24], [8], [38], [17],
– *Query plan rewriting* [39], [4, 33], and finally
– *Query compilation* [37, 36], [27, 28], [16], [24], [23], [32].

The SO approach presented in this article is tailored to data-driven, time-aware CEP relying on unbounded streams. It highlights the necessity of application states for the SO of CEP. States make queries and constraints more expressive, flexible, readable, and their evaluation more efficient. According to the above classifications this approach is characterised as follows. It relies on Instantiating Hierarchical Timed Automata (IHTA) and constraints expressed in the Event Stream Constraint Language ESCL. IHTA formalise states and capture involved temporal and causal constraints for an arbitrary number of parallel processes. ESCL constraints are short but expressive logic formulas completing IHTA with additional knowledge about events and states. The approach is static and general. It aims at query plan rewriting involving the implementation of SO heuristics, computation sharing, and storage minimisation.

2.2 Timed Automata

Timed Automata [2], [6], [1], [29] are (finite) automata the edges of which are labeled with events and temporal constraints on local clocks. Event data and event occurrence time are neglected. Timed Automata have been initially developed for stream verification, in particular, for the expression of constant bounds on the delays between events [2]. Later, Timed Automata have also been used for solving scheduling problems [1]. To the best of our knowledge, they have not been considered as metadata for SO.

In contrast to Timed Automata, Instantiating Hierarchical Timed Automata (IHTA) introduced in Section 4 allow access to event data and define temporal constraints on the occurrence time of matched events. IHTA are more expressive, flexible, and readable than Timed Automata.

It is difficult, if not impossible, to model real life systems of a certain size and complexity using flat automata. Most systems can be divided into relatively independent manageable processes. To this end, Hierarchical Timed Automata (HTA) [15] have been introduced. They support hierarchy of states and therefore modularisation. HTA allow for modelling a *fixed* number of processes running in parallel. However in real life applications their number is often unbounded. Therefore, IHTA adapt many ideas and notions of HTA to an arbitrary number of parallel processes. IHTA are more expressive than HTA.

3. USE CASES

As illustrative examples of the approach, two substantially different use cases are considered, namely an online auction and a sensor network. The difference is that an online auction use case has a rather involved workflow determining the structure of the stream. This workflow consists of multiple relatively independent processes (or workflows) running either sequentially or in parallel. This workflow reflects the application logic determined by the rules according to which an auction takes place. In a sensor network use case in contrast, there is no complex workflow (but there are states such as *normal*, *critical*, *emergency*). The application logic is determined by the physical laws.

Let us consider an online auction system allowing an arbitrary number of auctions to run in parallel. Each auction runs without time limit (but with a finite number of items to present), independently from the other auctions and according to a predefined workflow. Bidders register during the first 20 minutes in each auction. Afterwards, if at least two bidders are enrolled, at least one item is presented and the registered bidders can bid for it. Items are presented and possibly sold one after another in each auction. A bid for an item must be higher than a previous bid for the item. Two bids or hammer beats for an item are called *sequent* if they are not apart by another bid or hammer beat for the item. Two sequent bids for an item are at most 30 seconds apart. If a bid is followed by no other bid within 30 seconds, a hammer beat takes place. After a hammer beat there might be further bids. In this case the auction proceeds as described above. If there are no bids within 30 seconds after a hammer beat, then a further hammer beat takes place. After three sequent hammer beats, the item is sold at the price of the last bid to the bidder who issued this bid. Afterwards, the auction proceeds with another item. If there are no bids within 30 seconds after an item is presented, the item is not sold, and the auction proceeds with another item. When all items of an auction have been presented, the auction terminates.

This use case inspired from [38] is interesting because it has a rather involved workflow with multiple states (bidder enrolments, item offers, etc.) during which different events arrive and numerous diverse and complex dependencies between events hold (like between bids and hammer beats). The use case demonstrates that events must be treated differently depending on states in which they occur. For example, for each hammer beat event for an item it is essential to know how many sequent hammer beat events for the same item precede it. Only after the third sequent hammer beat event for an item no further bid events for the same item may follow. All queries asking for bid events can compute their answers and irrelevant bid events can be deleted. One way to express this knowledge is to count the number of sequent hammer beat events for an item every 90 seconds. Such constraints are rather unreadable and in many cases even not expressible since the time window is often unknown. The way that we propose is to formalise the application workflow as timed automata (Section 4) and to put queries and constraints in the context of states in which they are satisfiable or valid (Section 5). Besides, this use case shows the need for modelling an unbounded number of parallel processes like bidder enrolments or auctions.

Of course, not all CEP applications have comparably complex workflows and dependencies. That is why a sensor network use case will be considered in the future.

4. INSTANTIATING HIERARCHICAL TIMED AUTOMATA (IHTA)

IHTA consist of states and transitions between them. Consider Figure 1 depicting the workflow of an online auction modelled as (simplified) IHTA.

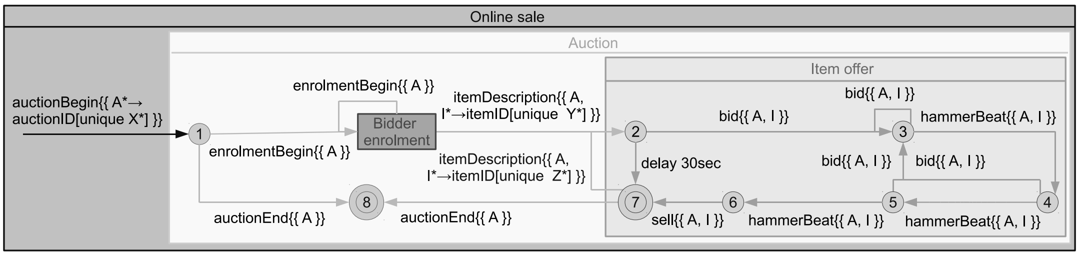

Figure 1: An online auction modelled as IHTA

Each state of IHTA is either atomic or non-atomic. *Non-atomic* states are IHTA themselves, e.g. the states *Bidder enrolment*, *Item offer*, *Auction*, and *Online sale* in Figure 1. All the other states are *atomic*. For the sake of readability, the bidder enrolment processes (which can be rather complex) are hidden within the state *Bidder enrolment*. Non-atomic states may contain other non-atomic states so that a hierarchy of states results (like in the considered use case).

Non-atomic states are modules allowing for readability, changeability, reuseability, and instantiation to model an arbitrary number of parallel processes as required in many applications. In the auction example, there is an arbitrary number parallel auctions with an arbitrary number of bidder enrolment processes running in parallel at the beginning of each auction. Every time a new auction or bidder enrolment begins, a new instance of the non-atomic state *Auction* or *Bidder enrolment* is created. Thus, an *arbitrary number* of instances of the same non-atomic state can exist *at the same time*, expressing that an arbitrary number of processes run in parallel. This feature of IHTA is new, not present in other formalisms [15], [25] allowing a *fixed* number of parallel processes to be modelled.

To represent the current state of each instance and thus the entire state of the application, automaton configuration is adapted from [15] to support an unlimited number of parallel processes. Automaton configuration is a dynamic structure containing the information about all running instances of non-atomic states, including the current state and the bindings of local variables of each instance.

The edges between states of IHTA are labeled by (1) incomplete and unordered event queries allowing access to data attributes and (2) temporal constraints on the occurrence time of events matched by the queries. (Event occurrence time is a time interval. Its bounds are saved as the values of time attributes of the respective tuple.) Both event queries and temporal constraints are expressed in ESCL (Section 5). Despite their expressiveness and flexibility, they are short and readable. These features of IHTA are new, not present in the other kinds of automata. Temporal constraints are omitted in Figure 1 for the sake of brevity.

Consider the transition between the states *Bidder enrolment* and 2 in Figure 1. Its incomplete and unordered query $q = itemDescription\{\{A, I^*\rightarrow itemID[unique\ Y^*]\}\}$ matches events of type *itemDescription* with attributes *auctionID* and *itemID* in arbitrary order since q is unordered (denoted by curly braces). Events matched by q may have other attributes besides *auctionID* and *itemID* since q is incomplete (denoted by double braces). *auctionID* and *itemID* are *explicit* attributes of q. Other attributes of events matched by q are *implicit* attributes of q. While matching, the variable Y^* is bound to the value of attribute *itemID* of matched events and the variable I^* to the whole term *itemID[unique Y*]*. The events matched by q must have the same value of the attribute *auctionID* as an event matched by the query $auctionBegin\{\{A^*\rightarrow auctionID[unique\ X^*]\}\}$ (consider Figure 1). A variable which is bound while matching is flagged with *, in contrast to a variable without * which is merely a place holder for its recent binding within an instance of the non-atomic state. The keyword *unique* means that its respective variable can be bound to a value only once.

IHTA are seen as a *specification* of the stream, i.e., no other events arrive during each state besides those described by IHTA. The automata capture the following kinds of constraints: (1) Cardinality constraints involving only explicit data attributes of the queries of IHTA. (2) Functional dependencies between time attributes. (3) Functional dependencies expressing equality of the values of explicit data attributes of the queries of IHTA. In some applications (like in the online auction) dozens of constraints can be *automatically derived* from IHTA. It is much easier, safer, and clearer for the user to specify IHTA instead of numerous complex constraints. Since the SO method for databases can be adapted to CEP (Section 7) and the method is based on constraints, constraints could be derived from IHTA and the adapted method could be applied to them. Alternatively an extra method working with IHTA directly could be developed. The choice depends on the efficiency of the evaluation methods and is subject of future work as well as the formal semantics of IHTA and their translation into rules deriving states to allow queries and constraints on states.

5. EVENT STREAM CONSTRAINT LANGUAGE (ESCL)

To keep IHTA readable, only a part of the application semantics is expressed by them. Only those attributes of event queries of IHTA are explicit which allow to relate each event of the stream to at least one instance of non-atomic states. Therefore the following kinds of *user-defined* constraints complete the IHTA: (1) Cardinality constrains and functional dependencies involving implicit data attributes of the queries of IHTA. (2) Functional dependencies expressing other relations as equality of the values of the explicit data attributes of the queries of IHTA. These constraints will be expressed in ESCL, the constraint language tailored to CEP. ESCL constraints are logic formulas capturing complex cardinality, temporal, causal and other dependencies between events and states. ESCL constraints are defined in the context of states which brings the following advantages: (1) The same events are treated differently depending on the state in which they occur. Due to states simple but expressive constraints are possible because the user does not have to manually describe all possibly numerous and involved event sequences of an arbitrary length leading to a state. (2) States make SO of event queries more efficient since only those constraints are relevant for a query which are valid during the state(s) the query is specified (or is satisfiable) within.

The grammar and the declarative semantics of ESCL defined by a Tarski-style model theory are given in [35]. The operational semantics of the language is the SO algorithm briefly described in Section 7. It is a subject of future work.

6. CONSTRAINT-BASED SUBSUMPTION

The SO algorithm for CEP relies on constraint-based subsumption.

Definition 1. Let C be a set of constraints. Let Q_1 and Q_2 be conjunctions of incomplete and unordered event or state queries and conditions on events or states matched by them. Q_1 *subsumes* Q_2 if all events and states matching Q_2 are also matched by Q_1 for all streams satisfying C.

We are aiming at decidable constraint-besed subsumption algorithm. Its formal definition and implementation are subjects of future work. This algorithm is different from the one used by the SO method for database queries [11] because of incomplete and unordered queries and consideration of a set of constraints. At the moment subsumption between two single incomplete and unordered queries is defined in [10]. The algorithm is in $O(n!^n)$ where n is the sum of sizes of two queries. A SO method with such a complexity is not advantageous. Whether the complexity of the subsumption algorithm can be improved will be investigated in the future.

7. SO ALGORITHM FOR CEP

The algorithm we are developing consists of two steps, namely *query compilation* and *query choice*. *Query compilation* is executed by the event residue method. For each initial query q the method extracts relevant portions of constraints, called residues, and uses them to generate a set of alternative semantically constrained queries. Each semantically constrained query *q'* is syntactically different from q but semantically equivalent to q (i.e. *q'* returns the same results as q for any streams satisfying the constraints) and can possibly be evaluated more efficiently than q. Whether a residue contributes to the optimisation of q is decided using SO heuristics. As the most overviews of SO techniques for database queries agree [11], [34], [14], there are the following six primary SO heuristics: result by contradiction, result by transformation, predicate elimination, predicate introduction, join elimination, and join introduction. SO of event queries can surely rely on these heuristics which have to be slightly adapted to cope with states. No additional heuristics are required for the SO of CEP. The event residue method is an adaption of the residue method for SO of database systems [11] to CEP.

During *query choice*, the evaluation costs of all alternative semantically constrained queries for q are determined using cost models, and one of them with the lowest (estimated) evaluation cost is processed instead of q.

The formal definition and implementation of the SO algorithm for CEP is a subject of future work.

8. FUTURE WORK

The following research directions will be investigated:
1) Further differences between SO of database systems and SO of CEP besides those described in Section 1.
2) Elaboration of the sensor network use case.
Formal definitions and implementation of (3) IHTA, (4) the constraint-based subsumption, and (5) the SO algorithm for CEP.
6) Dynamic SO of CEP, e.g. SO of queries at their runtime depending on current stream rate.
7) Query-based SO of CEP, i.e. SO of queries using queries themselves (consider the first classification in Section 2.1).
8) Constraint propagation.
As mentioned in Section 1, in databases, views are usually not materialised. Therefore, they are reduced to their base relations during SO and constraints are defined on base relations [11]. In CEP in contrast, views are usually materialised. Hence, constraints have to be defined for derived events and states as well. In order to reduce the number of manually defined constraints, some of them can be automatically propagated from base events and states to the derived events and states. For example, functional dependencies and cardinality constraints can be propagated. Some constraints can be derived from queries. This is a practically necessary and interesting future research direction. We are not aware of existing approaches to this topic.
9) Two-stage query answering, namely query answering using constraints and query answering using data. Consider the query asking for all bidders participating in an auction and being registered for this auction. Instead returning a long list of all registered bidders, the query could be first answered using the constraint stating that all bidders participating in an auction are registered for it. And then, if required by the user, the list of registered bidders can be returned. This more qualitative and informative query answering is subject of future work.

9. ACKNOWLEDGMENTS

We kindly thank S. Brodt, S. Giessl, S. Hausmann, and T. Dang for many suggestions for improvement of this work. This research is founded by the German Research Foundation (Deutsche Forschungsgemeinschaft) within the project “QONCEPT – Semantic Query Optimisation in CEP Technologies” under reference number BR 2355/1-1.

10. REFERENCES

[1] Y. Abdeddaim, E. Asarin, and O. Maler. Scheduling with Timed Automata. Elsevier Science, 2006.

[2] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. Int. Conf. on Automata, Languages and Programming*, 1990.

[3] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. In *Proc. Int. Conf. on Very Large Database*, 2004.

[4] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2000.

[5] S. Babu, U. Srivastava, and J. Widom. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Transactions on Database Systems*, 29(3), 2004.

[6] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2004.

[7] I. Botan, G. Alonso, P. M. Fischer, D. Kossmann, and N. Tatbul. Flexible and scalable storage management for data-intensive stream processing. In *Proc. Int. Conf. on Extending Database Technology*, 2009.

[8] F. Bry and M. Eckert. Temporal order optimizations of incremental joins for composite event detection. In *Proc. Int. Conf. on Distributed Event-Based Systems*. ACM, 2007.

[9] F. Bry and M. Eckert. On static determination of temporal relevance for incremental evaluation of complex event queries. In *Proc. Int. Conf. on Distributed Event-Based Systems*, 2008.

[10] F. Bry, T. Furche, and B. Linse. Simulation subsumption or Déjà vu on the Web. In *Proc. Int. Conf. on Web Reasoning and Rule Systems*, 2008.

[11] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. volume 15, pages 162–207. ACM, 1990.

[12] J. Chen, D. J. DeWitt, and J. F. Naughton. Design and evaluation of alternative selection placement strategies in optimizing continuous queries. In *Proc. Int. Conf. on Data Engeneering*, pages 345–356, 2002.

[13] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for Internet databases. *SIGMOD Rec.*, 29(2), 2000.

[14] Q. Cheng, J. Gryz, F. Koo, T. Y. C. Leung, L. Liu, X. Qian, and K. B. Schiefer. Implementation of two semantic query optimization techniques in DB2 universal database. In *VLDB*, pages 687–698, 1999.

[15] A. David and M. D. Mölle. From HUPPAAL to UPPAAL - a translation from Hierarchical Timed Automata to Flat Timed Automata, 2001.

[16] Y. Diao and M. Franklin. Query processing for high-volume XML message brokering. In *Proc. Int. Conf. on Very Large Data Bases*, pages 261–272. VLDB Endowment, 2003.

[17] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan. Runtime semantic query optimization for event stream processing. In *Proc. Int. Conf. on Data Engineering*, pages 676–685. IEEE Computer Society, 2008.

[18] L. Ding, N. Mehta, E. A. Rundensteiner, and G. T. Heineman. Joining punctuated streams. In *Proc. Int. Conf. on Extending Database Technology*, 2004.

[19] L. Ding, E. A. Rundensteiner, and G. T. Heineman. MJoin: A metadata-aware stream join operator. In *Proc. Int. Workshop on Distributed Event-Based Systems*, pages 1–8. ACM, 2003.

[20] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, J. A. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. 2001.

[21] P. M. Fischer, K. S. Esmaili, and R. J. Miller. Stream schema: Providing and exploiting static metadata for data stream processing. In *Proc. Int. Conf. on Extending Database Technology*, 2010.

[22] L. Golab, T. Johnson, N. Koudas, D. Srivastava, and D. Toman. Optimizing away joins on data streams. In *Proc. Int. Workshop on Scalable Stream Processing System*, pages 48–57. ACM, 2008.

[23] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata. In *Proc. Int. Conf. on Database Theory*, 2002.

[24] A. K. Gupta and D. Suciu. Stream processing of XPath queries with predicates. In *Proc. Int. Conf. on Management of Data*, 2003.

[25] D. Harel. Statecharts: A visual formalism for complex systems. volume 8. Elsevier Science, 1987.

[26] T. Johnson, M. S. Muthukrishnan, V. Shkapenyuk, and O. Spatscheck. Query-aware partitioning for monitoring massive network data streams. In *Proc. Int. Conf. on Management of Data*, 2008.

[27] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. FluXQuery: An optimizing XQuery processor for streaming XML data. In *Proc. Int. Conf. on Very Large Data Bases*, 2004.

[28] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. Schema-based scheduling of event processors and buffer minimization for queries on structured data streams. In *Proc. Int. Conf. on Very Large Data Bases*, 2004.

[29] S. Lasota and I. Walukiewicz. Alternating Timed Automata. volume 9. ACM, 2008.

[30] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Record*, 34(1):39–44, 2005.

[31] J. Liu, Q. Wu, and W. Liu. Temporal restriction query optimization for event stream processing. In *Advances in Web and Network Technologies, and Information Management*, 2010.

[32] B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou. A transducer-based XML query processor. In *Proc. Int. Conf. on Very Large Data Bases*, pages 227–238. VLDB Endowment, 2002.

[33] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 49–60. ACM, 2002.

[34] G. N. Paulley and G. K. Attaluri. Semantic query optimization in object-oriented databases.

[35] O. Poppe and F. Bry. The grammar and the declarative semantics of ESCL. Research report, Institute for Informatics, University of Munich, 2011.

[36] H. Su, E. A. Rundensteiner, and M. Mani. Semantic query optimization in an automata-algebra combined XQuery engine over XML streams. In *Proc. Int. Conf. on Very Large Data Bases*, 2004.

[37] H. Su, E. A. Rundensteiner, and M. Mani. Semantic query optimization for XQuery over XML streams. In *Proc. Int. Conf. on Very Large Data Bases*, 2005.

[38] P. A. Tucker, D. Maier, T. Sheard, and P. Stephens. Using production schemas to characterize strategies for querying over data streams. *IEEE Transactions on Knowledge and Data Engineering*, 19(9), 2007.

[39] S. D. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *Proc. Int. Conf. on Management of Data*, 2002.