Temporal Stream Algebra

Simon Brodt
Institute for Informatics
University of Munich
http://www.pms.ifi.lmu.de/
simon.brodt@ifi.lmu.de

François Bry
Institute for Informatics
University of Munich
http://www.pms.ifi.lmu.de/
bry@lmu.de

ABSTRACT

Data stream management systems (DSMS) so far focus on event queries and hardly consider combined queries to both data from event streams and from a database. However, applications like emergency management require combined data stream and database queries. Further requirements are the simultaneous use of multiple timestamps after different time lines and semantics, expressive temporal relations between multiple time-stamps and flexible negation, grouping and aggregation which can be controlled, i. e. started and stopped, by events and are not limited to fixed-size time windows. Current DSMS hardly address these requirements. This article proposes Temporal Stream Algebra (TSA) so as to meet the afore mentioned requirements. Temporal streams are a common abstraction of data streams *and* database relations; the operators of TSA are generalizations of the usual operators of Relational Algebra. A in-depth 'analysis of temporal relations guarantees that valid TSA expressions are non-blocking, i. e. can be evaluated incrementally. In this respect TSA differs significantly from previous algebraic approaches which use specialized operators to prevent blocking expressions on a "syntactical" level.

Categories and Subject Descriptors

D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*semantics*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Algebraic approaches to semantics, Operational semantics, Program analysis*; H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms

Theory, Algorithms, Languages, Verification

Keywords

CEP, Event Processing, DSMS, Data Stream, Query Algebra, Incremental Evaluation, Temporal Analysis

.

1. INTRODUCTION

The continuous and timely analysis of event streams, Complex Event Processing (CEP), has become a widely recognized discipline. Data stream management systems (DSMS) [17, 23, 43, 6, 19, 54, 32, 1], derive higher-order knowledge in the form composite events from a stream of low-level basic events. Although active databases have addressed several issues close to CEP, combined data stream and database queries have hardly been considered so far. Recent DSMS typically focus on event queries. If at all, queries to persistent data, e. g. from a database, are usually restricted to event-condition-action rules where the condition is executed after the event query succeeded and hence serves as additional filter before the action is actually executed.

Emerging applications require combined data stream and database queries, though [25]. One of these applications is the detection and management of emergencies in large infrastructures like metro system, airports or power grids. Section 2 contains a simple example adopted from the metro use-case [51, 52, 53], illustrating the need for combined data stream and database queries.

The contributions of this article are as follows:

1. A common data model for data streams and database relations
2. An algebra of operators for querying data streams and database relations

Salient properties of this algebra are:

- Timestamps after different time lines/time-semantics
- Expressive temporal relations
- Rich negation, grouping and aggregation

The paper is structured as follows: Section 2 motivates our approach using an example from emergency management. Section 3 introduces temporal streams and the common data model for data streams and database relations. Section 4 defines the operators of Temporal Stream Algebra (TSA). Section 5 explains how propagated constraints on temporal relations can be used to decide whether an operator application or TSA expression respectively is blocking or not. Section 6 describes the incremental evaluation of TSA. Sections 7 informs about related work and Section 8 points to the presented achievements and to future work.

2. MOTIVATION

Combined data stream and database queries. Use cases from emergency management need combined queries to data stream and database data[1] [51, 52, 53]. Consider a train in which a fire has broken out and that has to stop inside a metro station. Unfortunately the fire is close to one staircase which therefore is likely to quickly fill with smoke and should not be used for evacuation. For correctly assessing the situation and choosing an appropriate reaction, the system needs to combine the alarm events from smoke and temperature sensors with static data about the location of sensors and staircases. This way the system can conclude that one of the staircases being too close to the fire should not be used for evacuation as it will be impassable due to smoke shortly. Since persistent data are required for interpreting the volatile data arriving on the stream, combined data stream and database queries are needed.[2]

Declarativity. We argue that a high-level user-language for combined data stream and database queries should be declarative. A declarative language has a number of well-known advantages, like ease of programming and clear and (relatively) comprehensible semantics. This is particularly important for emergency management where rules have to be written (or at least have to be verified) by security experts which typically have limited programming skills. Furthermore emergency management requires predicable results, thus a clear semantic is mandatory. The work presented in this paper is part of the implementation of the event, state and action language Dura [10, 11, 12, 29].

Relational Algebra. For queries to database relations declarative query languages, particularly SQL, are widely used. Their system-level counterpart is Relational Algebra [2, 26]. Temporal Stream Algebra (TSA) generalizes the data model and operators of Relational Algebra to apply to both data streams and database relations. Such a generalization is by no means trivial and, so far, has not been proposed. Preserving the data model and the operators of Relational Algebra has at least two advantages: First the properties of the operators are well understood. For example the laws on operator permutations are very important for the optimization of Relational Algebra expressions. Second, there exist reliable techniques for the evaluation and optimization of Relational Algebra expressions, like specialized join algorithms, heuristics for operator reordering or cost-models, that base on these laws and on other properties of the operators of Relational Algebra. As TSA preserves the operators and their characteristics these techniques and algorithms can be reused or easily adopted for TSA. Third, preserving the operators of Relational Algebra for combined data stream and database queries means that the representation of database queries does not change at all and allows to query data streams in the same way that is already familiar from database relations.

[1]Within this article the terms "database data" or "database relation " refer to static data as opposed to stream data.

[2] Persistent data is likely to change less frequently than the streaming data itself, but does not have to be completely static. E.g. sensor locations may change, as for instance sensors in a moving train. Using TSA it is possible to handle dynamically changing persistent data in a semantically precise manner (i. e. without race conditions). This also holds for the incremental evaluation of TSA (Section 6). However details on this point it are out of the scope of this article.

Blocking Operators. Some operators of Relational Algebra, the so called "blocking operators", like set difference or grouping and aggregation, in general need to process the complete input before the may produce the correct results. For a data stream the complete input is only known at the end of the stream, which in the case of an CEP application is only reached when the application is terminated. However CEP queries are required to deliver results as the data arrives on the stream and not when the application is terminated. Therefore the use of "blocking operators" within CEP queries must be restricted to situations where each result only depends on a limited section of the stream and thus can be produced before the complete stream is known.

Previous approaches [22, 37], including those not related to Relational Algebra [19], tackle this problem on a "syntactical" level. They basically restrict operators in such a way that they may not lead to potentially blocking query expressions. In other words, the "syntax" does not allow to write blocking expressions. However solving this "semantic" problem on a "syntactic" level puts unnecessary limits to expressivity. By contrast TSA does not prevent blocking query expressions on the "syntactical" level. Instead TSA uses a "semantic" analysis of the temporal relations specified in the query expression, to determine whether the expression is blocking or not (See Sections 4 and 5 for details).

Common Data Model. Keeping the operators of Relational Algebra calls for a common data model for data streams and database relations as the same operators must work on both data sources. Using common operators for both data sources has two advantages: The number of operators is kept low and the common operators allow arbitrary combinations of both stream and database data. Note that a common data model for stream and database data should incorporate constraints on temporal relations providing the information that the temporal analysis needs to determine the validity of an query expression with respect to "blocking" or "non-blocking". Thus the data model is a key elements of TSA (see Section 3).

Multiple Time Lines. Modeling the emergency management use cases showed that a single, predefined time model (regardless of being based on time points or intervals) is insufficient. In emergency management even atomic events may refer to at least three times. The first time is the one at which an event, e. g. sensor message, is emitted. This time is known as application time. The second time is set when the event message is received by the CEP system. This time is known as system time. The third time is the time simulated data refer to. Simulated events are used for predictions on the future development of an emergency. For simulated events the time where the underlying data is available (application/system time) differs from the future time for which the event makes a prediction. Whether these times are modeled as time point or intervals depends on their usage and both, the common data model and the algebra proposed in this article, leave this decision open.

Supporting multiple time lines requires more than having multiple timestamp attributes. For example different time lines impose different orders on the events. An in-order processing is therefore only possible for at most one of the time lines. For all other time lines the evaluation is inevitably out-of-order. However previous algebraic approaches [22, 37] assume in-order processing and current DSMS offer only limited support for out-of-order processing, and thus favor a sin-

gle time line. By contrast TSA treats all time lines equally and imposes no limitation on their number. The presented incremental evaluation of TSA does a bulk-wise out-of-order processing for all time lines (Section 6).

Negation, Grouping and Aggregation. Finally emergency management needs flexible negation and grouping and aggregation that can be controlled, i. e. started and stopped, by events. The "event-controlled" grouping and aggregation is required for queries like "Count the number of people that have left the building from the detection of the fire to the arrival of the fire-brigade". Similar examples exist for negation. Negation or grouping and aggregation fixed-sized time-windows are too limited.

3. DATA MODEL: TEMPORAL STREAMS

The common data model for data streams and database relations is an essential part of this work. Temporal streams generalize the concept of (finite) relations from Relation Algebra so as to include data streams as well. The basic idea behind the definition is that streams may have potentially infinite size, however they are expected to have only a finite history at each point of time, i. e. up to some point of time only a finite amount of data may arrive on the stream.

DEFINITION 1. **(Temporal Stream)**
A *temporal stream* is a (possibly infinite) relation R with attribute schema $A(R) \subseteq ATTR$ and timestamp attributes $A_{temp}(R) = \{p_1, \ldots, p_k\}$ that has a finite history at each point in time. I. e. for all $s_1, \ldots, s_k \in \mathbb{Q}$ the number of tuples $r \in R$ occurring before that point in time is finite:

$$|\{r \in R \mid r(p_1) \leq s_1 \wedge \cdots \wedge r(p_k) \leq s_k\}| \; < \; \infty$$

where $ATTR$ is a set of attribute names and $A_{temp}(R)$ denotes the set of timestamp attributes of R. For data streams $A_{temp}(R)$ contains at least one timestamp attribute. For ordinary (finite) database relation $A_{temp}(R)$ may be empty. As database relations are always finite they trivially fulfill the condition formulated above.

The definition of temporal streams bases on the observation that each data item in a data stream is carrying temporal information, i.e. timestamps, that is correlated with the sequence order of the data items in the stream.[3] The timestamps may refer to different time-models, none of the timestamps might define a total order on the incoming events and events may arrive out-of-order with respect to each of the timestamps. However as long as the timestamps refer to clocks that all increase over time, i.e. time does not stop for any of the clocks, also the minimum value for each timestamp of any future event increases over time.[4] In other words, as more and more data of the stream arrives we observe temporal progress with respect to each of the timestamps. TSA is very flexible with respect to the used time models. TSA only requires a total order on the domain of a time model,[5] and supports the simultaneous use of multiple different time models. However, for sake of simplicity and without loss of generality this paper only uses a single time model, namely $\mathbb{Q}$.

In Relational Algebra relations are always finite. In some way data streams, and thus the corresponding relations, are finite, too. The reason is that no application will run forever and so only a finite amount of data may arrive on the stream. However this view is misleading as it wrongly suggests that Relational Algebra could be applied to data streams just as it is (See "Blocking Operators" in Section 2).

To stress the fact that we must not wait for the end of a data stream,[6] TSA models data streams as potentially infinite relations adopting the idea in [22, 20]. The payload of each data item including the associated timestamps are stored in attributes of the corresponding type. However those timestamps that are correlated to the sequence order,[4] i.e. refer to the temporal progress of the stream, play special role in the validation and the evaluation of TSA expressions. A so-called "stream bound formula" within the schema remembers the special role of these timestamps.

The schema of a relation in Relational Algebra is just its set of attributes. For temporal streams the attribute schema is accompanied by a "temporal relation formula" and a "stream bound formula". These formulas are formally defined in Section 5. The "temporal relation formula" (Definition 18) describes the temporal relations between the timestamp attributes. The "stream bound formula" (Definition 19) tells about the ability of timestamp attributes to obtain a finite prefix of a temporal stream matching the schema. Basically a timestamp attribute is able to yield a finite prefix of a temporal stream if it is correlated to the temporal progress of the stream as described above. The two formulas carry the necessary information for the validation and evaluation of a TSA expressions (See Section 5).

DEFINITION 2. **(Temporal Stream Schema)**

1. A *temporal stream schema* is a triple $S = (A, G, H)$ such that A is an attribute schema, G is a temporal relation formula and H is a stream bound formula and all attributes occurring in G and H are in the attribute schema (i.e. $attr(G) \subseteq A_{temp}$ and $attr(H) \subseteq A_{temp}$).
2. A temporal stream schema $S = (A, G, H)$ is *valid* if the set of all timestamp attributes A_{temp} is a stream bound with respect to G and H (See Definition 22).
3. A relation R matches a temporal stream schema $S = (A, G, H)$ if R has attribute schema $A(R) = A$ and both G and H hold in R (See Definitions 20,22). If S is valid, the latter implies that R is a temporal stream.

4. TEMPORAL STREAM ALGEBRA – TSA

The common operators for queries to data streams and database relations are the second essential element of TSA. TSA generalizes the operators of Relational Algebra without changing their definition of the result sets. As mentioned in Section 2 this introduces the problem of "blocking" operators and query expressions. It is a fundamental observation, that the answer to the question whether a certain query expression is non-blocking, depends on the temporal relations between timestamp attributes imposed by the input streams and the query expression itself. In Relational Algebra this

[3] Even the sequence numbers themselves can be seen as a kind of timestamp, admittedly with respect to a quite unusual time model. TSA is able to manage even this special case.

[4] The data items may also contain timestamps, that are not correlated to the sequence order, i.e. are not related to the progress of time or the availability of data. Particularly all timestamps of database relations fall into this category. Such timestamps are treated as ordinary data.

[5] This does not impose a total order on the data items.

[6] This implies the need for an incremental evaluation

information is not propagated, though. Thus subsequent operators may not rely on the information for determining whether they can be applied correctly.

The solution of this problem (and one of the central ideas of TSA) is the propagation of constraints on temporal relations inside the schema of TSA expressions. The propagation of the "temporal relation formulas" (Definition 18) and of the "stream bound formulas" (SBFs, Definition 19) contained in the schema of TSA expressions is therefore the most important part of the TSA operator definitions.

Within a TSA program, temporal relations are established in several ways. First by constraints that are part of the schema of an temporal stream definitions. Temporal stream definitions are the starting point for all TSA expressions.

DEFINITION 3. **(Temporal Stream Definition)** A *temporal stream definition* is a pair $D = (n, S)$ where $n \in STREAM$ is a name for an temporal stream, $S = (A, G.H)$ is a temporal stream schema, $STREAM$ is a set of names and the following restrictions hold:

1. G and H do not contain variables
2. H is a disjunction of atomic SBFs, i. e. has structure $bounded(p_1, b_1) \vee \ldots \vee bounded(p_k, b_k)$
3. The atomic SBFs $bounded(p_i, b_i)$ in H define a bijection $b^D : A_{temp} \rightarrow BOUND$ with inverse p^D.

The first restriction allows to validate whether the schema of a query expressions matches the schema of the output stream. The third restriction is merely technical, it is important when generating the necessary formulas and functions for the incremental evaluation. The second restriction in the above definition expresses that each stream bounds (see Definition 22) of a temporal stream definitions must contain a stream bound that consists only of a single timestamp attribute.[7] For example, if you would like to express that both system- (p_{sys}) and application-time (p_{app}) are stream bounds of a stream $D = (n, S)$ then the stream bound formula H of schema S would look like this:

$H = bounded(sys, n{:}sys) \vee bounded(app, n{:}app)$

All TSA expressions E have an associated schema $S(E)$. The schema of a temporal stream definition $D = (n, S)$ is $S(D) = S$. The schema $S(E)$ of composite TSA expressions is defined inductively in the definitions of the TSA operators.

Definition 4 shows that temporal relations may also be derived from conditions on timestamp attributes imposed by the selection operator. The definition of the resulting relation/temporal stream is the same as for Relational Algebra.

DEFINITION 4. **(Selection σ)** Let E be a TSA expression with schema $S(E) = (A, G, H)$ and let R be a temporal stream which matches the schema of E and C a set of conditions with $domain(C) \subseteq A$.

$$\sigma[C](R) := \{r \in R \mid r \text{ satisfies } C\}$$
$$S(\sigma[C](E)) := (A, (G \wedge C_{temp}), H')$$

where C_{temp} is the TRF that results from C when replacing every non-temporal atom by $\top$ if it occurs with positive polarity or by $\bot$ if it occurs with negative polarity.

[7] Stream bounds with multiple elements that do not fulfill this condition are allowed for the schema of TSA expressions. They result from the application of binary operators like cross-product, set-difference or union.

The definition of C_{temp} extracts the maximum temporal information from the condition C. Basically C_{temp} results from C by replacing all non-temporal atoms in such a way that the condition C is fulfilled as much as possible.

Finally temporal information is introduced by the definition of new timestamp attributes relative to existing ones. At that point the presentation of the TSA operators slightly differs from the usual presentation of the operators of Relational Algebra. The usual projection operator of Relation Algebra allows both discarding attributes and defining new attributes based on existing ones. In TSA we split these two tasks into two operators: The projection operator of TSA which only discards attributes and the imbed operator which allows to define new attributes. These changes in the presentation help to keep the definitions simple, as each definitions only cares for a single aspect of the propagation of temporal relation formulas and stream bound formulas. The following definition uses the notion of "temporal terms" that are formally introduced in definition 17. Temporal terms are a powerful mean to define relative timestamps.

DEFINITION 5. **(Imbed ι)** Let E be a TSA expression with schema $S(E) = (A, G, H)$ and let R be a temporal stream which matches the schema of E. Let $a' \notin A$ be a new attribute, t the term defining a' and $a_1, \ldots, a_k \in A$ the attributes occurring in t.

$$\iota[a' = t](R) := \{r' \mid \exists r \in R \text{ such that } r'(a) = r(a) \text{ for } a \in A \text{ and } r'(a') = f_t(r(a_1), \ldots, r(a_k))\}$$
$$S(\iota[a' = t](E)) := (A \cup \{a\}, G', H)$$
$$G' := \begin{cases} G \wedge (a' = t) & \text{if } t \text{ is a temporal term} \\ G & \text{else} \end{cases}$$

where $f_t : dom(a_1) \times \ldots \times dom(a_k) \rightarrow dom(a')$ is the function of the values of $a_1, \ldots, a_k \in A$ defined by t.
If t is a temporal term, then t defines a relative timestamp.

The main point of the above definition is, that the definitions of relative timestamp attributes are preserved within the temporal relation formula G' of the schema.

For projection and grouping the definition of the resulting temporal stream is almost the same as for Relational Algebra. The only difference is that the projection may not introduce new attributes. For both operators the temporal relation formula and the stream bound formula are propagated in the same way: The discarded timestamp attributes are replaced by temporal variables. The simple definition of grouping might be surprising, as grouping is one of the "blocking" operator which are usually considered as "problematic". But as TSA solves the problem of blocking query expressions on the basis of the propagated temporal constraints and not on the level of the operators, the definition of the grouping operator can actually be straight-forward.

DEFINITION 6. **(Projection π)** Let E be a TSA expression with schema $S(E) = (A, G, H)$ and let R be a temporal stream which matches the schema of E and $A_1 \subseteq A$ the set of retained attributes.

$$\pi[A_1](R) := \{r' \mid \exists r \in R \text{ such that } r'(a) = r(a) \text{ for } a \in A_1\}$$
$$S(\pi[A_1](E)) := (A_1, \xi(G), \xi(H))$$

where ξ is an injective mapping of the discarded timestamp attributes in $A_{temp} \setminus A_{1temp}$ to variables that do not occur in G or H.

DEFINITION 7. **(Grouping γ)** Let E be a TSA expression with schema $S(E) = (A, G, H)$ and let R be a temporal stream which matches the schema of E and $A_1 \subseteq A$ the set of grouping attributes. Let $a_1, \ldots a_k \in A \setminus A_1$ and $a'_1, \ldots, a'_k \notin A_1$ and $F_1, \ldots, F_k$ aggregation functions like min, max, sum or avg.

$$\gamma[A_1][a'_1 = F_1(a_1), \ldots, a'_k = F_1(a_k)](R) := \{r' \mid \exists r \in R \text{ such that } r'(a) = r(a) \text{ for } a \in A_1 \text{ and } r'(a'_i) = F_i((r_\gamma(a_i))_{r_\gamma \in R_r}) \}$$
$$S(\gamma[A_1][a'_1 = F_1(a_1), \ldots, a'_k = F_1(a_k)](E)) := (A_1, \xi(G), \xi(H))$$

where $R_r = \{r_\gamma \in R \mid r_\gamma(a) = r(a) \text{ for } a \in A_1\}$ and ξ is an injective mapping of the discarded timestamp attributes $A_{temp} \setminus A_{1temp}$ to variables that do not occur in G or H.

The next definition is the second deviation from the usual presentation of operators in Relation Algebra. The basic TSA operators do not include a join but a cross-product operator.Thus using the basic operators of TSA, a join must be expressed as a combination of cross-product and selection. Again this change helps to keep the definitions simple.

However omitting the join as basic operator does not restrict the expressivity or efficiency of TSA in any way. Further operators, like the usual projection, renaming, join, semi-join and anti-join, can easily be added to TSA. Such easy extensions are not described in this article for pace reasons. All these operators can be expressed as combinations of the basic operators presented in this paper. Sections 5.1 and 6 show that the incremental evaluation of TSA solely depends on the correct propagation of the temporal constraints, but is independent from the concrete operators contained in an TSA expression. Thus, defining a composite operator with respect to semantics, constraint propagation and incremental evaluation is just done by providing its decomposed representation by means of basic operators.

DEFINITION 8. **(Cross Product $\otimes$)** Let E_1 and E_2 be TSA expressions with schema $S(E_1) = (A_1, G_1, H_1)$ and $S(E_2) = (A_2, G_2, H_2)$ that have disjunct attributes $(A_1 \cap A_2 = \emptyset)$ and let R_1 and R_2 be temporal streams which match the schema of E_1 and E_2 respectively.

$$R_1 \otimes R_2 := \{r' \mid \exists r_1 \in R_1, r_2 \in R_2 \text{ with } r'(a) = r_1(a) \text{ for } a \in A_1 \; r'(a) = r_2(a) \text{ for } a \in A_2\}$$
$$S(E_1 \otimes E_2) := (A_1 \cup A_2, \xi_1(G_1) \wedge \xi_2(G_2), \xi_1(H_1) \wedge \xi_2(H_2))$$

where ξ_1 and ξ_2 are injective substitutions, such that the $\xi_1(G_1)$, $\xi_1(H_1)$ and $\xi_2(G_2)$, $\xi_2(H_2)$ use disjunct sets of temporal variables.

The definition of the cross-product operator is straightforward. The renaming of the temporal variable avoids unintended interferences between the temporal relation formulas and stream bound formulas of the two subexpressions.

DEFINITION 9. **(Union $\cup$)** Let E_1 and E_2 be TSA expressions with schema $S(E_1) = (A, G_1, H_1)$ and $S(E_2) = (A, G_2, H_2)$ that have the same attributes and let R_1 and R_2 be temporal streams which match the schema of E_1 and E_2 respectively.

$$R_1 \cup R_2 := \{r' \mid \exists r \text{ with } r \in R_1 \text{ or } r \in R_2 \text{ and } r'(a) = r(a) \text{ for } a \in A\}$$
$$S(E_1 \cup E_2) := (A, G', (\xi_1(H_1) \wedge \xi_2(H_2)))$$
$$G' := (G'_1 \wedge G''_1 \wedge \xi_1(G_1)) \vee (G'_2 \wedge G''_2 \wedge \xi_2(G_2))$$
$$G'_1 := (\xi_1(p_1) = p_1) \wedge \ldots \wedge (\xi_1(p_k) = p_k)$$
$$G''_1 := \bigwedge_{v \in attr(H_2) \cup vars(H_2)} (\xi_2(v) \leq p_1 + -\infty) \wedge \ldots \wedge (\xi_2(v) \leq p_k + -\infty)$$
$$G'_2 := (\xi_2(p_1) = p_1) \wedge \ldots \wedge (\xi_2(p_k) = p_k)$$
$$G''_2 := \bigwedge_{v \in attr(H_1) \cup vars(H_1)} (\xi_1(v) \leq p_1 + -\infty) \wedge \ldots \wedge (\xi_1(v) \leq p_k + -\infty)$$

where ξ_1 and ξ_2 are injective substitutions, that map all timestamp attributes $\{p_1, \ldots, p_k\} = A_{temp}$ to temporal variables and replace temporal variables by other variables such that $\xi_1(v) \neq \xi_2(w)$ for any two timestamp attributes or temporal variables v and w occurring in A, G_1, G_2, H_1 or H_2.

It is far from obvious why the definition of the union is not as simple as the definition of the cross-product. With regards to the temporal relation formulas (TRFs) only, it would in fact be possible to define $G' = \xi_1(G_1) \vee \xi_2(G_2)$ analogously to the definition in the cross-product operator.[8] The crucial point is that the attributes from the two subexpressions, despite their identical names, are actually different, as they originate from different inputs. This does not matter for the TRFs at the first place. However it does matter for the SBFs. Identifying the attributes from the different inputs could result in a wrong analysis of the stream bounds (Definition 22) and other properties of the TSA expression. Therefore the substitutions ξ_1 and ξ_2 and the formulas G'_1 and G'_2 are used to decouple the formulas from the two inputs. The intuitive meaning of G''_1 and G''_2 is that the part of G' corresponding to the first subexpression should not care about the requirements $\xi_2(H_2)$ on stream bounds from the the second subexpression and vice versa.

DEFINITION 10. **(Set Difference $\setminus$)** Let E_1 and E_2 be TSA expressions with schema $S(E_1) = (A, G_1, H_1)$ and $S(E_2) = (A, G_2, H_2)$ that have the same attributes and let R_1 and R_2 be temporal streams which match the schema of E_1 and E_2 respectively.

$$R_1 \setminus R_2 := \{r_1 \in R_1 \mid \neg\exists r_2 \in R_2 \text{ with } r_1 = r_2\}$$
$$S(E_1 \setminus E_2) := (A, G_1 \wedge (G'_1 \vee \xi_2(G_2)), (H_1 \wedge \xi_2(H_2)))$$
$$G'_1 := \bigwedge_{v \in attr(H_2) \cup vars(H_2)} (\xi_2(v) \leq p_1 + -\infty) \wedge \ldots \wedge (\xi_2(v) \leq p_k + -\infty)$$

where ξ_2 is an injective substitution for temporal variables, such that the substituted variants $\xi_2(G_2)$, $\xi_2(H_2)$ of G_2, H_2 and the original formulas G_1, H_1 use disjunct sets of temporal variables.

The constraint propagation for set-difference is a little bit tricky, too. On the one hand, the TRF G_2 does not impose any constraints on the output tuples of the expression. Thus we could just choose G_1 as TRF for the schema of the expression. On the other hand the information in G_2 needs to be propagated to correctly determine the stream bounds with respect to the negative expression. The solution is as follows: First G_1 is assumed to hold for the negative subexpression as well, as those tuples from the negative stream that do not fulfill G_1 are irrelevant anyhow. Second $\xi_2(G_2)$ is combined disjunctively with G'_1. As in the definition for the union G'_1 basically tells that in case of the positive subexpression one does not need to care about the requirements $\xi_2(H_2)$ on stream bounds from the negative subexpression. A symmetric counterpart of G'_1 for the negative subexpression is not necessary, as G_1 also holds for

[8] The definition for the TRF in the result schema is equivalent to the simple definition w.r.t. Definition 20.

the negative subexpression. However G'_1 does not impose any temporal relations between the timestamp attributes of E. Thus the same holds for $G'_1 \vee \xi_2(G_2)$. TSA makes only weak, "syntactic" checks when defining temporal streams or TSA expressions. Further "semantic" checks are done at a later stage. The next definitions reference Definition 22 from Section 5 about temporal analysis.

DEFINITION 11. **(Validity of Temporal Stream Definitions)** A temporal stream definition D with schema $S(D) = (A, G, H)$ is *valid* iff the set of all timestamp attributes A_{temp} is a stream bound with respect to G and G. The definition implies that any relation R matching the schema $S(D)$ is a temporal stream.

DEFINITION 12. **(TSA Query)** A TSA query is a pair $q = (D, E)$ such that E is a TSA expression with schema $S(E) = (A, G_E, H_E)$ and D is a temporal stream definition for the output stream with schema $S(D) = (A, G_D, H_D)$ which both have the same attributes.

DEFINITION 13. **(Validity of TSA Queries)** Let $q = (D, E)$ be a TSA query as defined in Definition 12 and let $D_1, \ldots, D_k$ be the temporal stream definitions occurring in E. The TSA query q is *valid* iff

1. $D_1, \ldots, D_k$ are valid.
2. The schema $S(E)$ of the TSA expression E matches the schema $S(D)$ of the definition for the output stream, i. e. G_E implies[9] G_D and all stream bounds with respect to G_D and H_D are stream bounds with respect to G_E and H_E.

The most important property of valid TSA queries is, that they are non-blocking, i. e. can be evaluated incrementally.

PROPOSITION 14. **(Non-Blocking Queries)** Let $q = (D, E)$ be a TSA query and let $D_1, \ldots, D_k$ be the temporal stream definitions occurring in E.

If q is *valid* the q is *non-blocking*. This means, for any temporal streams $R_1, \ldots, R_k$ matching the schema of $D_1, \ldots, D_k$ respectively and any stream bound $\{p\}$[10] with respect to $S(D)$, the finite prefix for limit $s \in \mathbb{Q}$

$$\begin{aligned}&\{r \in E(R_1, \ldots, R_k) \mid r(p) \leq s\} = \\ &\{r \in E(\{r_1 \in R_1 \mid r_1(p_{1,1}) \leq s_{1,1} \vee \cdots \vee r_1(p_{1,l_1}) \leq s_{1,l_1}\}, \\ &\qquad\qquad \vdots \\ &\qquad \{r_k \in R_k \mid r_1(p_{k,1}) \leq s_{k,1} \vee \cdots \vee r_1(p_{k,l_k}) \leq s_{1,l_k}\} \\ &) \mid r(p) \leq s\}\end{aligned}$$

where $\{p_{i,1}\}, \ldots, \{p_{i,l_i}\}$ are stream bounds for $S(D_i)$ and $s_{i,1}, \ldots, s_{i,l_i} \in \mathbb{Q}$ for $1 \leq i \leq k$, i. e. the prefix depends only on finite prefixes of $R_1, \ldots, R_k$.

PROOF (SKETCH). Let $S(D) = (A, G_D, H_D)$ and $S(E) = (A, G_E, H_E)$. Without loss of generality one may assume that the temporal stream definitions $D_1, \ldots, D_k$ use different stream bound identifiers, as Proposition 14 and none of its indirectly referred Definitions depend on the actual names of stream bounds.

As q is a valid query, any stream bound $\{p\} \subseteq A_{temp}$ with respect to $S(D)$ is a stream bound with respect to $S(E)$.

Let $\overline{G}_E$ be the normalized form of G_E (see Section 5.1) and the DNF of $\overline{G}_E$ of G_E be $dnf(\overline{G}_E) = C_1 \vee \cdots \vee C_l$, where $C_1, \ldots, C_l$ are conjunctions of normalized atomic temporal relation formulas (Definition 18).

For each D_i, $1 \leq i \leq k$ and each C_j there must exist at least one[11] atomic stream bound formula $bounded(v_{i,j}, b_{i,j})$ in H_E[12] (Definition 19) where $v_{i,j} \leq p + dist_{C_j}(v_{i,j}, p)$ (Definition 21) and $b_{i,j}$ belongs to D_i.

However $v_{i,j}$ is actually a renamed version of the attribute $p_{i,j} := p^{D_i}(b_{i,j})$ associated to $b_{i,j}$ in D_i. Thus if $p \leq s$ in the result relation then $p_{i,j} \leq s_{i,j} := s - dist_{C_j}(v_{i,j}, p)$ in the source relation R_i, at least for "case" C_j of G_H.

Proposition 15 allows to apply the formal results from the formulas to the actual relations.

Finally the disjunction $r_i(p_{i,1}) \leq s_{i,1} \vee \cdots \vee r_1(p_{i,l}) \leq s_{i,l}$ provides the condition required for the proof for each temporal stream definition D_i. This disjunction could be condensed but this is needed for the proof. □

The proof of Proposition 14. bases on the following proposition on the well-definedness of the TSA operators.

PROPOSITION 15. **(Well-Definedness)** Let E be a valid TSA expression where $D_1, \ldots, D_k$ are temporal stream definitions occurring in E. If $R_1, \ldots, R_k$ are temporal streams matching the schema of $D_1, \ldots, D_k$ respectively then the $E(R_1, \ldots, R_k)$ and matches the schema $S(E)$.

PROOF (SKETCH). The proof is straight-forward by induction over the structure of E. □

PROPOSITION 16. **(Relational Completeness)** TSA is *relational complete* [18] for finite (non-stream) relations.[13]

PROOF. On finite non-stream relations, TSA is equivalent to Relational Algebra. □

5. TEMPORAL RELATIONS

Temporal relations are an essential part of expressive data stream queries as discussed in Section 4.

Temporal terms are used to specify new timestamps relative to existing timestamps. This is for example needed when defining the timestamps p of a composite event (see Definition 5). Assume that the composite event is composed from two events with timestamps p_1 and p_2. A usual definition for the timestamp p of the composite event would be $p = \max\{p_1, p_2\}$. Temporal terms are also used when establishing a temporal relation, that is not just "equal", between two existing timestamps using the selection operator (see Definition 4). For example if a timestamp p_1 should occur at most 2 seconds after timestamp p_2 then this would translate to $p_1 <= p_2 + 2sec$ where $p_2 + 2sec$ is a temporal term defining a new timestamp relative to p_2. To the best of our knowledge temporal terms allow to realize any of the usual temporal semantics for timestamps (also intervals) of composite events [31, 6, 22, 8].

[9] Can be checked if right side does not contain variables. By definition G_D and H_D do not contain variables.

[10] The 2. restriction in Definition 12 and Proposition 14 imply the analogous proposition for arbitrary stream bounds.

[11] If D_i describes a static relation this does not hold. But in that case there is nothing to show. One can choose $l_i = 0$.

[12] Note that H_E is in CNF with exactly one clause per $D_1, \ldots, D_k$. This follows immediately from the definition of temporal stream definitions and of TSA operators.

[13] The term "relational complete" compares the expressive power of some formalism for querying *finite relations* to the expressive power of Relational Algebra. A generalization to temporal streams is, if at all possible, non-trivial.

DEFINITION 17. **(Temporal Terms)**
The set of *temporal terms* is defined inductively:

1. $v \in ATTR \cup VAR$ is an atomic temporal term
2. $t + c$ is a temporal term iff t is a temporal term and $c \in \mathbb{Q}$
3. $min\{t_1, \ldots, t_k\}$, $max\{t_1, \ldots, t_k\}$ are temporal terms iff $t_1, \ldots, t_k$ are temporal terms

where VAR is a set of temporal variables, $VAR \cap ATTR = \emptyset$.

Temporal relation formulas (TRFs) describe temporal relations between temporal terms, particularly between timestamp attributes and/or variables. Variables are basically required for handling relations on attributes that are discarded by the projection or the grouping operator of TSA (See Definitions 6,7). Temporal relations in TSA are based on timestamps (in contrast to intervals). However time intervals can be defined using two time stamps and a TRF stating that the first timestamp is smaller then the second. Furthermore as TRFs enable conjunctions and disjunctions[14] all 2^{13} interval relations of Allen's interval algebra [5] can be expressed.

DEFINITION 18. **(Temporal Relation Formulas)**
The set of *temporal relation formulas* (TRFs) is defined inductively:

1. $\top$ and $\bot$ are atomic TRFs
2. $t_1 \,\mathsf{op}\, t_2$ is an atomic TRF for $\mathsf{op} \in \{<, \leq, =, \geq, >, \neq\}$ iff t_1 and t_2 are temporal terms
3. $G \wedge G'$, $G \vee G'$ and $\neg G$ are TRFs iff G, G' are TRFs

TRFs store the information about temporal relations between timestamp attributes. When using multiple timestamp attributes there shows another effect which is not covered by TRFs, though. The definition of temporal streams (Def. 1) requires that upper limits to the values of all timestamp attributes of a temporal relation result in a finite prefix of the stream. However, if input streams carry more than one timestamp then it often suffices to limit a subset of the timestamp attributes of the temporal stream to obtain a finite prefix. Consider for example, an input stream with application- and system-time timestamps. An upper limit for the values of one of the two timestamps suffice to obtain a finite prefix of the input stream. If two input streams of this kind are combined by the cross-product operator then any subset of timestamp attributes containing at least one timestamp attribute from each stream, can be used to get an finite prefix of the result stream. The purpose of stream bound formulas (SBFs) is to propagate the information which combinations of timestamps are suited for obtaining a finite prefix of a temporal stream. The SBF for a (static) database relation may just be $\top$ as we do not need to obtain a finite prefix in that case.

DEFINITION 19. **(Stream Bound Formulas)** The set of *stream bound formulas* (SBFs) is defined inductively:

1. $\top$ is an atomic SBF
2. $bounded(v, b)$ is an atomic SBF for $v \in ATTR \cup VAR$ where $b \in BOUND$ is a stream bound identifier.
3. $H_1 \wedge H_2$ and $H_1 \vee H_2$ are SBFs iff H_1 and H_2 are SBFs.

[14] By contrast Point Algebra [40] allows only conjunctions.

Definition 19 assigns so-called "stream bound identifiers" to the atoms of a SBF. As these identifiers do not change during the propagation process (in contrast to the names of attributes and variables) the identifiers allow to identify the input relation that a specific atom in a SBF originates from. This way the temporal analysis of a TSA query expression that is needed for validation and incremental evaluation may base solely on the propagated formulas and does not need to analyze the TSA query expression recursively.

5.1 Temporal Analysis

As stated throughout the whole article, TSA analyses the temporal constraints for a number of reasons, e. g. for the decision on the validity of a query expression and for its incremental evaluation. This section provides the relevant definitions and briefly describes the employed algorithms.

Normalization of Temporal Relation Formulas. Temporal relation formulas (TRFs) can be normalized using the following equivalences in ($\mathcal{NORM}$).[15] After normalization the TRF is equal to $\top$ or $\bot$ or all atomic subformulas have the form $v \leq w + c$ and the normalized TRF does not contain negation. This normalization is essential for following definitions and the algorithmic analysis of TRFs.

$$
\begin{aligned}
Neg1 &: \neg(G \wedge G') \Leftrightarrow (\neg G \vee \neg G') \\
Neg2 &: \neg(G \vee G') \Leftrightarrow (\neg G \wedge \neg G') \\
Neg3 &: \neg\neg G \Leftrightarrow G \\
Neg4 &: \neg(t_1 \,\mathsf{op}\, t_2 + c) \;\Leftrightarrow\; t_1 \,\mathsf{op}^{-1}\, t_2 + c \\
Neg5 &: \neg\top \Leftrightarrow \bot \text{ and } \neg\bot \Leftrightarrow \top \\
Top &: G \wedge \top \Leftrightarrow G \text{ and } G \vee \top \Leftrightarrow \top \\
Bot &: G \wedge \bot \Leftrightarrow \bot \text{ and } G \vee \bot \Leftrightarrow G \\
Zero &: v \,\mathsf{op}\, w \;\Leftrightarrow\; v \,\mathsf{op}\, w + 0 \\
Eq &: t_1 = t_2 + c \;\Leftrightarrow\; (t_1 \leq t_2 + c) \wedge (t_1 \geq t_2 + c) \\
Neq &: t_1 \neq t_2 + c \;\Leftrightarrow\; (t_1 < t_2 + c) \vee (t_1 > t_2 + c) \\
Geq &: t_1 \geq t_2 + c \;\Leftrightarrow\; t_2 + c \leq t_1 \\
Gr &: t_1 > t_2 + c \;\Leftrightarrow\; t_2 + c < t_1 \\
Less^{16} &: t_1 < t_2 + c \;\Leftrightarrow\; t_1 \leq t_2 + (c - \varepsilon) \\
Arith1 &: t_1 + c \,\mathsf{op}\, t_2 \;\Leftrightarrow\; t_1 \,\mathsf{op}\, t_2 + (-c) \\
Arith2 &: t_1 \,\mathsf{op}\, (t_2 + c) + d \;\Leftrightarrow\; t_1 \,\mathsf{op}\, t_2 + (c + d) \\
Min1 &: t \leq \min\{t_1, \ldots, t_k\} + c \;\Leftrightarrow \\
&\quad\; t \leq t_1 + c \wedge \ldots \wedge t \leq t_k + c \\
Min2 &: \min\{t_1, \ldots, t_k\} \leq t + c \;\Leftrightarrow \\
&\quad\; t_1 \leq t + c \vee \ldots \vee t_k \leq t + c \\
Max1 &: t \leq \max\{t_1, \ldots, t_k\} + c \;\Leftrightarrow \\
&\quad\; t \leq t_1 + c \vee \ldots \vee t \leq t_k + c \\
Max2 &: \max\{t_1, \ldots, t_k\} \leq t + c \;\Leftrightarrow \\
&\quad\; t_1 \leq t + c \wedge \ldots \wedge t_k \leq t + c
\end{aligned}
$$

for temporal terms $t_1, \ldots, t_k$ and $c, d \in \mathbb{Q}$ and TRFs G, G' and $v, w \in ATTR \cup VAR$ and $\mathsf{op} \in \{<, \leq, =, >=, >, \neq\}$ and $<^{-1} \mapsto >$, $\leq^{-1} \mapsto \geq$, $=^{-1} \mapsto \neq$, $\geq^{-1} =\leq$, $>^{-1} \mapsto <$, $\neq^{-1} \mapsto =$

DEFINITION 20. **(Temporal Relations)** A temporal relation formula G holds in R, denoted $R \models G$, iff for all tuples $r \in R$ the instantiation $\sigma_r(G)$ of G is satisfiable in $\mathbb{Q}$:

$\models_{\mathbb{Q}} \exists v_1, \ldots, v_l : \sigma_r(G)$

where $A_{temp} = \{p_1, \ldots, p_k\}$ and $\{v_1, \ldots, v_l\} = vars(G)$ and $\sigma_r := \{p_1 \mapsto r(p_1), \ldots, p_k \mapsto r(p_k)\}$

[15] In case of the algorithmic analysis, the equivalences are to be read from left to right.

[16] Imagine ε as infinitely small value with $(c - \varepsilon) + (d - \varepsilon) = ((c + d) - \varepsilon)$ and $c - \varepsilon < c$ but $d < c - \varepsilon$ if $d < c$.

The temporal distance[17] of two temporal variables or timestamp attributes is the maximum that the value of the second temporal variable is smaller than the value of the first [13]. This information is essential for the incremental evaluation (see Section 6) and for automatic garbage collection[18].

DEFINITION 21. **(Temporal Distance)** Let G be a temporal relation formula and R be a temporal stream with $A(R) = A$. The *temporal distance* of two attributes or variables $v, w \in ATTR \cup VAR$ with respect to G is

$$dist_G(w,v) := \max_{C \in dnf(\overline{G})} \{dist_C(w,v)\}$$
$$dist_C(w,v) := \min\{c \mid \mathcal{TD}, C \models_{\overline{\mathbb{Q}}} v \leq w + c\}$$

where $u, v, w \in ATTR \cup VAR$ and $c, d, +\infty, -\infty \in \overline{\mathbb{Q}}$ and $\overline{G}$ is the normalized form of G and C is a conjunction of atomic TRFs of the form $v \leq w + c$ and $\mathcal{TD}$ contains

$$Ref: \quad v \leq v$$
$$Trans: \; u \leq v + c \wedge v \leq w + d \;\Rightarrow\; u \leq w \pm (c + d)$$
$$Inf: \quad v \leq w + +\infty$$

The algorithmic analysis of the temporal distances is closely related to the simple temporal problem (STP) [48] and the disjunctive temporal problem DTP [35]. Basically the (naive) analysis algorithm is as follows: The TRF is normalized and converted into disjunctive normal form. Each conjunction is an STP instance. The distance of all pairs of attributes and variables for this instance can determined using any algorithm for the all-pair shortest path problem, e. g. the Floyd Warshall algorithm [24], or specialized algorithms for STP. The distance of two attributes or variables for the whole TRF is then the maximum distance of the two attributes or variables in any of the conjunctions.

The following definition is about so-called "stream bounds" that can be derived from a SBF and its corresponding TRF. Stream bounds are those sets of timestamp attributes of a temporal stream that are suited to limit the stream to a finite prefix. Intuitively speaking, stream bounds do not let any part of the stream pass infinitely. The decision whether some TSA query is blocking or not depends on the analysis of stream bounds.

DEFINITION 22. **(Stream Bounds)** Let G be a temporal relation formula and H be a temporal relation constraint and let R be a temporal stream with $A(R) = A$.

1. A set $\{p_1, \ldots, p_k\} \subseteq ATTR$ is a *stream bound* with respect to G and H iff

   $bounded(p_1, b_1), \ldots, bounded(p_k, b_k), G, \mathcal{TD}, \mathcal{SB} \models H$

   for any $b_1, \ldots, b_k \in BOUND$[19] and

   $\mathcal{SB}: v \leq w + c\,,\; c < +\infty\,,\; bounded(w, b_w) \Rightarrow bounded(v, b_v)$

   for $v, w \in ATTR \cup VAR$, $b_v, b_w \in BOUND$ and $c \in \overline{\mathbb{Q}}$
2. A set $\{p_1, \ldots, p_k\} \subseteq ATTR$ is a *stream bound* with respect to schema $S = (A, G, H)$ if $\{p_1, \ldots, p_k\}$ is a stream bound with respect to G and H.
3. The set $\{p_1, \ldots, p_k\} \subseteq A_{temp}(R)$ of timestamp attributes is a *stream bound* for R iff every upper limit $s_1, \ldots, s_k \in \mathbb{Q}$ for the values $p_1, \ldots, p_k$ yields a finite prefix of R:

   $$|\{r \in R \mid r(p_1) \leq s_1, \ldots r(p_l) \leq s_l\}| < \infty$$
4. H holds in R with respect to G, denoted $R \models_G H$, iff all stream bounds $\{p_1, \ldots, p_l\} \subseteq A_{temp}$ with respect to G and H are stream bounds of R.

[17] Note that the temporal distance is usually asymmetrical.
[18] Automatic garbage collection is not described in this paper, see [20, 13] for the idea
[19] Actually the names of atomic SBFs do not play a role here.

The algorithmic analysis of the stream bounds is similar to the one for temporal distances. The TRF G is normalized and converted into disjunctive normal form. For each conjunction C the following is done: First the distance between all attributes and variables in the conjunction is computed. Second for each atom in the SBF H, the atom is set to true if the distance from one of the attributes of the potential stream bound $\{p_1, \ldots, p_k\}$ to the attribute or variable in the atom is finite. Otherwise the atom is set to false. If the H holds under this interpretation, then $\{p_1, \ldots, p_k\}$ is a stream bound with respect to C. If $\{p_1, \ldots, p_k\}$ is a stream bound with respect to all conjunctions, then $\{p_1, \ldots, p_k\}$ is a stream bound with respect to G and H.

Increment formulas are the basis for the increment conditions that are used to transform a usual TSA expression into an incremental expression. Basically the increment conditions are derived from the increment formulas, by replacing names $b \in BOUND$ by their current value at run-time. In other word, Increment formulas are the parametrized versions of the increment conditions.

DEFINITION 23. **(Increment Formulas)** Let H be a SBF. The *increment formula* $\triangle H$ to H is defined recursively:

$$\triangle H_{\top} = \top$$
$$\triangle H_{bounded(v,b')} = v \leq b$$
$$\triangle H_{H_1 \wedge \ldots \wedge H_k} = F_{H_1} \wedge \ldots \wedge F_{H_k}$$
$$\triangle H_{H_1 \vee \ldots \vee H_k} = F_{H_1} \vee \ldots \vee F_{H_k}$$

where $v \in ATTR \cup VAR$ and $b \in BOUND$. $\triangle H$ results from H by replacing each atom $bounded(v, b)$ in H by $v \leq b$.

Stream bound functions are used to compute the progress of an output stream of a TSA query with respect to the progress of the input streams of the query (see Section 6). Stream bound function formulas are an intermediate step in the generation of these functions. Due to their special structure, stream bound function formulas are probably a good basis for optimizing stream bound functions using techniques for the optimization of Boolean functions [34].

DEFINITION 24. **(Stream Bound Functions)** Let G be a TRF, H be a SBF, $b \in BOUND$ the name for a stream bound and $p \in ATTR$ a timestamp attribute that is assigned to b (e. g. by $bounded(p, b)$ in a temporal stream definition).

1. The *stream bound function formula* for p, b, G and H is defined recursively:

$$F_{b,p,G,H} = \bigwedge_{C \in dnf(\overline{G})} F_{b,p,C,H}$$
$$F_{b,p,C,\top} = \top$$
$$F_{b,\{p\},C,v \leq b'} = \begin{cases} \top & \text{if } dist_C(p,v) = -\infty \\ \bot & \text{if } dist_C(p,v) = +\infty \\ b \leq b' - dist_C(p,v) & \text{else} \end{cases}$$
$$F_{b,p,C,(H_1 \wedge \ldots \wedge H_k)} = F_{b,p,C,H_1} \wedge \ldots \wedge F_{p,b,C,H_k}$$
$$F_{b,p,C,(H_1 \vee \ldots \vee H_k)} = F_{b,p,C,H_1} \vee \ldots \vee F_{b,p,C,H_k}$$

   where $v \in ATTR$ and $b' \in BOUND$ and $\overline{G}$ is the normalized form of G and C is a conjunction of atoms of the form $v \leq b'$.
   The basic idea is, to enforce $v \leq b'$ using p and b and the inequation $v \leq p + dist(p, v) \leq b + dist(p, v) \leq b'$.

The $\bigwedge$ in the first case results from the fact that H must hold in each case of G (compare Definition 19).

2. Let F be a stream bound function formula for b and any p, G, H. The *stream bound function* f_F of F is defined recursively:

$$\begin{array}{ll} f_\top & = +\infty \qquad f_\bot = -\infty \\ f_{b \le b'-d} & = b' - d \\ f_{F_1 \wedge \ldots \wedge F_k} & = \min\{f_{F_1}, \ldots, f_{F_k}\} \\ f_{F_1 \vee \ldots \vee F_k} & = \max\{f_{F_1}, \ldots, f_{F_k}\} \end{array}$$

where $b' \in BOUND$ and $d \in \overline{\mathbb{Q}}$.

3. The *stream bound function* for p, b, G and H is

$$f_{p,b,G,H} = f_{F_{p,b,G,H}}$$

4. The *stream bound function* for a TSA query $q = (D, E)$ with $S(D) = (A, G_D, H_D)$ and $S(E) = (A, G_E, H_E)$ and b occurring in H_D

$$f_{b,q} = f_{p^D(b),b,G_E,H_E}$$

where p^D is the function defined in Definition 3.

The stream bound function formula F can be normalized to be $\top$ or $\bot$ or not to contain $\top$ or $\bot$ at all. In that case f_F is either $-\infty$ or $+\infty$ or does not contain $-\infty$ or $+\infty$ at all. If $F = \bot$, thus $f_F = -\infty$, then p is not a stream bound with respect to G and H. If $f_F = +\infty$ then G and H belong either to a static relation or a stream that is actually empty because its imposed temporal relations are unsatisfiable.

6. INCREMENTAL EVALUATION

An incremental evaluation is obviously crucial for TSA. The incremental evaluation allows to derive results continuously as the data arrives on the stream. Without an incremental evaluation results could only be derived at the end of the stream which is too late for CEP application and particularly for emergency management. Proposition 14 in Section 4 shows an important property of valid TSA queries: Any prefix of the result stream of a query q depends only on finite prefixes of the input streams of q.

Without loss of generality the definitions for the incremental evaluation of TSA assume that all temporal stream definitions of a TSA program use different stream bound identifiers $b \in BOUND$. This can easily be realized if $BOUND = STREAM \times ATTR$ and for temporal stream definition $D = (n, S)$ and stream bound $\{p\}$ of S the stream bound identifier $b = b^D(p)$ corresponding to p in D has the form $b = n : p$.

The incremental evaluation of TSA allows for an asynchronous evaluation of the queries of a TSA program.[20] However the queries are not fully independent from each other, but need to exchange information on the progress of their respective output streams. This is necessary as the maximum achievable progress of of the output stream of a query depends on the progress of the referred input streams.

The incremental evaluation uses values assigned to the stream bound identifiers occurring in the temporal stream definitions and queries, to propagate this progress information. These values are used to instantiate the increment formula (Definitions 23) for the current increment computation of a query. The values for stream bound identifiers serve for a similar purpose as the "punctuations" of [57]. However they are not only used for input streams but are propagated through the evaluation process.

[20] Actually every fair sequence for the increment computations will compute the correct result. The actual sequence may significantly affect the efficiency and the response-time of the evaluation, though.

DEFINITION 25. **(Stream Bound Values)** Let P be a set of TSA Queries. For a TSA Query $q = (D, E)$ let $s_{q,i} \in \mathbb{Q}$ and $e_{q,i} \in \mathbb{Q}$ denote the start and end time of the ith ($i \ge 1$) increment computation for q. The following is defined simultaneously:

1. Value of b for query q at the ith increment computation for q and $i \ge 0$

$$v_{q,0}(b) = -\infty$$
$$v_{q,i}(b) = f_{b,q}(v_{s_{q,i}}(b_1), \ldots, v_{s_{q,i}}(b_l))$$

where $f_{b,q}$ is the function term from Definition 24 and $b_1, \ldots, b_l$ are the stream bound names occurring in $f_{b,q}$.

2. Value of b for query q at time $t \in \mathbb{Q}$

$$v_{q,t}(b) = \begin{cases} v_{q,0}(b) \text{ for } t < e_{q,1} \\ v_{q,i}(b) \text{ for } e_{q,i} \le t < e_{q,i+1} \end{cases}$$

3. Value of b for temporal stream definition D at time t

$$v_{D,t}(b) = \min_{\substack{q' \in P \\ \text{out}(q') = D}} \{v_{q',t}(b)\}$$

The value $v_{D,t}(b)$ for input streams must be provided. The values for each D must not decrease and should exceed every limit after some finite amount of time. If so, then the same holds for the derived values, assuming a fair execution sequence for the queries. [21]

4. Value of b at time t

$$v_t(b) = v_{D_b,t}(b)$$

where D_b is the temporal stream definition corresponding to b.

The definition is well defined as $s_{q,i} < e_{q,i}$ for $i \ge 1$.

The incremental evaluation is able to work with conservative approximations of the stream bound values, too. This is particularly useful for a parallel or even distributed evaluation of a TSA program, as the values for the stream bound identifiers do not need to be perfectly synchronized. Furthermore one could conservative approximations for the functions $f_{b,q}$. The functions $f_{b,q}$ are optimal with respect to the achievable progress, however approximate,i. e. simpler versions of the functions may help to reduce the computational overhead that is potentially introduced by the computation of the stream bound values.

As the each step of the incremental evaluation of a query only depends on the current values for the stream bound identifiers, it is easily possible to stop and resume the incremental evaluation by storing the values persistently. This is particularly useful for crash recovery, but also allows to switch the scheduling strategy or even to re-optimize the executed program.

[21] In case of system time this is basically the current value of the local clock minus 1."Minus 1" ensures that definitely all new data item will receive a greater timestamp than the returned value. This assumes that the value of a clock is never decreasing. For application time, a mechanism similar to "punctuation" [57] had to be used.

DEFINITION 26. **(Increment Expression)**
Let $q = (D, E)$ be a *valid* TSA query with output schema $S(D) = (A, G_D, H_D)$. The *incremental expression* for the $i+1$th ($i >= 0$) increment computation for q is:

$\triangle E_{i+1} = \sigma[\triangle H_{D\,i+1} \wedge \neg \triangle H_{D\,i}](E)$

where $\triangle H_D$ is the increment formula for H_D from Definition 23, $\triangle H_{D\,i+1}$ results from $\triangle H_D$ by replacing each stream bound name b by its new value $v_{q,i+1}(b)$ and $\triangle H_{D\,i}$ results from $\triangle H_D$ by replacing each stream bound name b by its old value $v_{q,i}(b)$ from the ith execution of $\triangle E$.

The increment expression for a query can be seen as parametrized expression. This is very useful as it avoids to compile each of the increment expressions at the run-time of the TSA program. Instead the parametrized version of the increment expression can be compiled only once.

Furthermore, the increment expression is in fact an ordinary Relational Algebra expression, i.e. the incremental evaluation reduces the evaluation of TSA expressions to the evaluation of Relational Algebra expressions. In other words, TSA generalizes Relational Algebra to data streams and the incremental evaluation of TSA reduces the evaluation of TSA to the evaluation of Relational Algebra. Thus an efficient implementation of Relational Algebra could easily be enhanced to an implementation of TSA that allows for processing database relations *and* streams.

PROPOSITION 27. **(Correctness)** The consecutive execution of the increment statements $\triangle E_1, \triangle E_2, \triangle E_3, \ldots$ for a TSA query $q = (D, E)$ yields the same result as if E had been applied to the whole stream at once.

7. RELATED WORK

Most current DSMS focus on event streams and hardly account for database data. Generally speaking current DSMSs and their corresponding query languages have a built-in semantics for timestamps and do not support the use of timestamps from multiple time lines. Furthermore the time window and sequencing constructs offered by the query languages make it hard or even impossible to express complex temporal relations. The insufficient treatment of temporal relations of also results in a very limited support for negation and grouping and aggregation. Typically DSMS either offer no out-of-order processing or treat it as "necessary evil". A bulk-wise processing of events is hardly accounted for.

Temporal Algebras [55, 41, 45] are used in so-called temporal databases which are one of the (many) ancestral fields of CEP. Temporal algebras do neither account for the stream aspect of CEP nor for the incremental evaluation needed to the potentially infinite relations representing event streams. Queries are just evaluated against the currently available data and do not care whether data arriving in the future might invalidate the result of the query.

Composition Operator Approaches are characterized by the use of composition operators, particularly the sequence operator, for building complex events and expressing temporal relations. The classification comes from [21] which classifies these approaches as "composition operator languages". The following systems and approaches belong to this category: Amit [4], ruleCore [54, 44], CAYUGA [19, 31] and [28, 27, 16, 3, 39, 59, 7, 58, 14, 49, 42, 30, 15, 9, 50, 46]. The underlying semantics (if such) typically uses the term Event Algebra, however these algebras have little in common with Relational Algebra and thus TSA. One main difference to TSA is that temporal relations are expressed by composition operators like the sequencing operator and not within a selection as in TSA.[22] Combined queries to data streams and databases are hardly supported. Furthermore only single timestamps or time intervals are used. The operational semantics of Event Algebras is usually defined in terms of automatons where the automatons perform state transitions as reaction to new events. By contrast TSA uses a set based incremental evaluation.

Data Stream Approaches typically use an SQL-like query language and are classified as "data stream query languages" in [21]. Due to their SQL-like query they are relatively close to Relational Algebra. The following systems fall into this category: PIPES / Logical Stream Algebra [36, 37], TelegraphCQ [17, 47], Aurora / Borealis / StreamBase / StreamSQL [56, 32, 1], CQL / STREAM [6], Esper [23], DataCell [33, 38], Coral8 [43] and [25]. Some of these systems[38, 25] support combined queries to events and database relations. Only few of the systems have a formal semantics. One of those is PIPES with the Logical Stream Algebra that is an adoption of Relational Algebra [37]. The differences of TSA and the Logical Stream Algebra are explained in the next paragraph. Some systems combined queries to data streams and databases. A general difference of the preceding systems and TSA is that TSA enables multiple timestamps and direct and expressive temporal relations within the selection operator, whereas the systems mentioned above use only a single timestamp or time interval and temporal relations can only be expressed in a limited and indirect way by means of time-windows. DataCell [33, 38] is a special as it has a very procedural semantics based on petri-nets. However DataCell is one of the few systems that also support queries to database data and it even account for bulk-processing.

Logic Stream Algebra is the semantic foundation of the PIPES system [37] which belongs to the data stream approaches mentioned before and shares their general differences to TSA. The Logic Stream Algebra adopts Relational Algebra but it does so in quite a different way as TSA. We want to list some differences here: The Logic Stream Algebra requires a discrete time domain whereas TSA also copes with dense time domains like $\mathbb{Q}$. The Logic Stream Algebra uses only a single timestamp in contrast to multiple timestamps in TSA. In the Logic Stream Algebra temporal relations have to be expressed indirectly using the time-window operator. Thus temporal relations are very limited. In TSA expressive temporal relations can directly be expressed as part of selection conditions. The window operators of the Logic Stream Algebra (conceptually) makes a copy of each tuple for each time point within the window. The other operators are somehow defined with respect to the snapshots of tuples that are valid at a certain point of time. Thus the operation is actually defined independently for each time point (see snapshot-reducibility in [37]). Together with the property of the time-window operator mentioned before this enables a comparably simple garbage collection however at the price of a low expressivity for temporal relations. The temporal streams in TSA contain only a single tuple for each incoming tuple. The TSA operators combine tuples from different times. Garbage collection bases on temporal relevance con-

[22] CAYUGA gives access to the duration of an event within a selection condition.

ditions, e. g. time windows, that are *implicitly* derived from the specified temporal relations using an evolved temporal analysis of the TSA expressions.

CERA is the Complex Event Relational Algebra of Michael Eckert [20, 22] proposed as operational semantics for the logic event query language XChange^{EQ}. TSA picks up many of the ideas of CERA but somehow makes a restart as to achieve minimal and orthogonal operators which are better suited for optimizations. TSA provides more expressive temporal relations, e. g. all interval relations of Allen's interval algebra [5] are supported, a significantly more flexible grouping a better temporal analysis. Furthermore TSA enables multiple timestamps in input and output streams whereas CERA only has some support for multiple timestamps in intermediate computations.

8. CONCLUSION

We present Temporal Stream Algebra (TSA) that generalizes the data model and operators of Relational Algebra to apply to both data streams and database relations. TSA providesa common data model and an algebra of operators for querying data streams and database relations.

TSA proposes a new approach for coping with "blocking" operators based on an analyis of the temporal relations specified in query expressions so as to determine whether the expression is blocking or not. TSA supports multiple timestamps after different time lines and time semantics. TSA allows to specify expressive temporal relations including all Allan's interval relations. Finally, TSA provides a flexible, event-controlled negation and grouping and aggregation that can be controlled, i. e. started and stopped, by events.

Though TSA generalizes Relational Algebra, the incremental evaluation of TSA reduces the evaluation of TSA expressions to that of Relational Algebra expressions. Thus an existing implementation of Relational Algebra can be enhanced to a system, processing data streams as well.

The incremental evaluation of TSA allows for an asynchronous query processing. Even the progress information that is exchanged between queries does not have to be strictly synchronized. Both properties are useful for distributed processing. The incremental evaluation of TSA performs a bulk-wise out-of-order processing of events. We expect that this contributes to a high throughput, though it probably does not achieve minimum response times for single events, and helps to cope with peaks in the event load. These expectations still have to be verified by experimental evaluations. A prototype implementation based on TSA of a high-level event, state and action language, Dura, has been completed.

The combination of asynchronous and bulk-wise processing opens a number of interesting questions with regards to scheduling. Though any fair execution sequence for the increment expressions of the queries will yield the correct results, scheduling affects the efficiency and the response-time of the evaluation. Determining a good or even optimal sequence is one of our current research issues.

Furthermore the asynchronous and bulk-wise processing enables a situation dependent query prioritization. The increment expressions of less important queries could just be executed less frequently. This increases the response time of the less important queries, but as the efficiency of the whole system increases due to the greater bulks, the response time for the prioritized queries decreases at the same time.

Finally garbage collection is very important for the efficiency of the proposed incremental evaluation as it may significantly reduce the size of intermediate results. Garbage collection can be seen as symmetric counterpart of incremental evaluation. The increment-condition removes tuples from the intermediate result that are not *yet* relevant. The condition for garbage collection removes tuples that are not relevant *anymore*. In other words, the increment-selection poses an upper bound to relevant events, where as the condition for garbage collection poses a upper bound to irrelevant, i. e. a lower bound to relevant events. This suggests that applying the temporal analysis for the incremental evaluation to lower bounds instead of upper bound could yield good garbage collection conditions [13].

9. ACKNOWLEDGMENTS

The work presented in this article is part of the research project EMILI (`http://www.emili-project.eu/`) funded by the European Commission under grant agreement number 242438. Special thanks go to Fabian Groffen and Martin Kersten from the MonetDB team at CWI for their support in the prototype implementation of TSA and to Steffen Hausmann for the excellent cooperation in the design of Dura and TSA and for numerous fruitful discussions.

10. REFERENCES

[1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Aurora: A new model and architecture for data stream management. *The VLDB Journal*, 12(2), 2003.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] R. Adaikkalavan and S. Chakravarthy. SnoopIB: Interval-based event specification and detection for active databases. *Data and Knowledge Engineering*, 1(59):139–165, 2006.

[4] A. Adi and O. Etzion. Amit — the situation manager. *The VLDB Journal*, 13(2):177–203, 2004.

[5] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[6] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[7] R. S. Barga and H. Caituiro-Monge. Event correlation and pattern detection in CEDR. In *Proc. Int. Workshop Reactivity on the Web*, volume 4254 of *LNCS*. Springer, 2006.

[8] R. S. Barga, D. Gannon, and D. A. Reed. The client and the cloud: Democratizing research computing. *IEEE Internet Computing*, 15(1), 2011.

[9] M. Bernauer, G. Kappel, and G. Kramler. Composite events for XML. In *Proc. Int. Conf. on World Wide Web*, pages 175–183. ACM, 2004.

[10] S. Brodt, S. Hausmann, and F. Bry. Deliverable D4.3: Dura – Concepts and Examples. `www.emili-project.eu`, 2011.

[11] S. Brodt, S. Hausmann, and F. Bry. Deliverable D4.5: Implementation. `www.emili-project.eu`, 2011.

[12] S. Brodt, S. Hausmann, and F. Bry. Deliverable D4.6: Modularization Mechanisms. `www.emili-project.eu`, 2012.

[13] 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*, pages 289–300. ACM, 2008.

[14] F. Bry, M. Eckert, and P.-L. Pătrânjan. Reactivity on the Web: Paradigms and applications of the language XChange. *J. of Web Engineering*, 5(1):3–24, 2006.

[15] J. Carlson and B. Lisper. An event detection algebra for reactive systems. In *Proc. ACM Int. Conf. On Embedded Software*, pages 147–154. ACM, 2004.

[16] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proc. Int. Conf. on Very Large Data Bases*, pages 606–617. Morgan Kaufmann, 1994.

[17] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[18] E. F. Codd. Relational completeness of data base sublanguages. *In: Database Systems, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.

[19] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White. Cayuga: A general purpose event monitoring system. In *CIDR*, pages 412–422, 2007.

[20] M. Eckert. *Complex Event Processing with XChange$^{\mathrm{EQ}}$: Language Design, Formal Semantics and Incremental Evaluation for Querying Events*. PhD thesis, Institute for Informatics, University of Munich, 2008.

[21] M. Eckert, F. Bry, S. Brodt, O. Poppe, and S. Hausmann. A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed. In S. Helmer, A. Poulovassilis, and F. Xhafa, editors, *Reasoning in Event-based Distributed Systems*, volume 347 of *Studies in Computational Intelligence*, chapter 3. Springer, 1 edition, 2011.

[22] M. Eckert, F. Bry, S. Brodt, O. Poppe, and S. Hausmann. Two Semantics for CEP, no Double Talk: Complex Event Relational Algebra (CERA) and its Application to XChangeÊQ. In S. Helmer, A. Poulovassilis, and F. Xhafa, editors, *Reasoning in Event-based Distributed Systems*, volume 347 of *Studies in Computational Intelligence*, chapter 4. Springer, 1 edition, 2011.

[23] EsperTech Inc. Event stream intelligence: Esper & NEsper. `http://esper.codehaus.org`.

[24] R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345–, June 1962.

[25] M. J. Franklin, S. Krishnamurthy, N. Conway, A. Li, A. Russakovsky, and N. Thombre. Continuous analytics: Rethinking query processing in a network-effect world. In *CIDR*, 2009.

[26] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2001.

[27] S. Gatziu and K. R. Dittrich. Detecting composite events in active database systems using petri nets. In *Proc. Int. Workshop on Research Issues in Data Engineering: Active Database Systems*, pages 2–9. IEEE, 1994.

[28] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Compose: A system for composite specification and detection. In *Advanced Database Systems*, LNCS, pages 3–15. Springer, 1993.

[29] S. Hausmann and F. Bry. Complex Actions for Event Processing. Submitted for publication, 2012.

[30] A. Hinze and A. Voisard. A parameterized algebra for event notification services. In *Proc. Int. Symp. on Temporal Representation and Reasoning*, pages 61–65. IEEE, 2002.

[31] M. Hong. *Expressive and Scaleable Event Stream Processing*. PhD thesis, Cornell University, 2009.

[32] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. B. Zdonik. Towards a streaming sql standard. *PVLDB*, 1(2):1379–1390, 2008.

[33] M. Kersten, E. Liarou, and R. Goncalves. A query language for a data refinery cell. In *Proc. Int. Workshop on Event-Driven Architecture, Processing and Systems*, 2007.

[34] B. R. King, S.-V. A. L., M. C. T., and H. G. D. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984.

[35] M. K. Kostas Stergiou. Backtracking algorithms for disjunctions of temporal constraints. *Artif. Intell.*, 120, June 2000.

[36] J. Krämer and B. Seeger. Pipes: a public infrastructure for processing and exploring streams. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, New York, NY, USA, 2004. ACM.

[37] J. Krämer and B. Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM Trans. Database Syst.*, 34, April 2009.

[38] E. Liarou, R. Goncalves, and S. Idreos. Exploiting the power of relational databases for efficient stream processing. In *Int. Conf. on Extending Database Technology (EDBT)*, volume 360, pages 323–334. ACM, 2009.

[39] M. Mansouri-Samani and M. Sloman. GEM: A generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997.

[40] P. v. B. Marc Vilain, Henry Kautz. *Constraint propagation algorithms for temporal reasoning: a revised report*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[41] L. E. McKenzie, Jr. and R. T. Snodgrass. Evaluation of relational algebras incorporating the time dimension in databases. *ACM Comput. Surv.*, 23, 1991.

[42] D. Moreto and M. Endler. Evaluating composite events using shared trees. *IEE Proceedings — Software*, 148(1), 2001.

[43] J. Morrell and S. D. Vidich. Complex Event Processing with Coral8. `www.coral8.com/system/files/assets/pdf/Complex_Event_Processing_with_Coral8.pdf`, 2007.

[44] MS Analog Software. ruleCore(R) Complex Event Processing (CEP) Server. `http://www.rulecore.com`.

[45] G. Ozsoyoglu and R. T. Snodgrass. Temporal and real-time databases: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 7:513–532, August 1995.

[46] N. W. Paton, editor. *Active Rules in Database Systems*. Springer, 1998.

[47] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein. Enabling real-time querying of live and historical stream data. In *SSDBM*, page 28, 2007.

[48] J. P. Rina Dechter, Itay Meiri. Temporal constraint networks. *Artif. Intell.*, 49:61–95, May 1991.

[49] C. Roncancio. Toward duration-based, constrained and dynamic event types. In *Proc. Int. Workshop on Active, Real-Time, and Temporal Database Systems*, volume 1553 of *LNCS*, pages 176–193. Springer, 1997.

[50] C. Sánchez, M. Slanina, H. B. Sipma, and Z. Manna. Expressive completeness of an event-pattern reactive programming language. In *Int. Conf. on Formal Techniques for Networked and Distributed Systems*, LNCS. Springer, 2005.

[51] N. Seifert and M. Bettelini. Deliverable D3.1: Use Cases Requirements Analysis and Specification (Main Report). `www.emili-project.eu`, 2010.

[52] N. Seifert, M. Bettelini, and S. Rigert. Deliverable D3.2: Concrete Use Case Models. `www.emili-project.eu`, 2011.

[53] N. Seifert, M. Bettelini, S. Rigert, S. Vraneš, V. Mijović, N. Tomaševiś, G. Konečni, V. Janev, L. Kraus, P. L. Kroner, D. Siller, A. Braun, Y. Leontyeva, J. L. M. E. nol, and J. R. H. Gonzales. Deliverable D3.3: Use case modelling for implementation in SITE. `www.emili-project.eu`, 2012.

[54] M. Seiriö and M. Berndtsson. Design and implementation of an ECA rule markup language. In *Proc. Int. Conf. on Rules and Rule Markup Languages for the Semantic Web*, volume 3791 of *LNCS*, pages 98–112. Springer, 2005.

[55] G. Slivinskas, C. S. Jensen, S. Member, R. T. Snodgrass, and S. Member. A foundation for conventional and temporal query optimization addressing duplicates and ordering. *IEEE TKDE*, 13:2001, 2001.

[56] StreamBase Systems. *StreamSQL Guide*, 2011. `http://streambase.com/developers/docs/latest/streamsql/index.html`.

[57] P. Tucker and D. Maier. Exploiting punctuation semantics in data streams. In *Proc. ICDE*, page 279, 2002.

[58] E. Wu, Y. Diao, and S. Rizvi. High-performance Complex Event Processing over streams. In *Proc. Int. ACM Conf. on Management of Data (SIGMOD)*. ACM, 2006.

[59] D. Zhu and A. S. Sethi. SEL, a new event pattern specification language for event correlation. In *Proc. Int. Conf. on Computer Communications and Networks*. IEEE, 2001.