SARI-SQL: Event Query Language for Event Analysis

Szabolcs Rozsnyai
Senactive
Inkustrasse 1-7
3400 Klosterneuburg, Austria
srozsnyai@senactive.com

Josef Schiefer
Institute for Software Technology and Interactive Systems
Favoritenstrasse 9-11/188
1040 Vienna, Austria
js@ifs.tuwien.ac.at

Heinz Roth
Secure Business Austria
Favoritenstrasse 16
1040 Vienna, Austria
roth@securityresearch.at

Abstract

Complex Event Processing (CEP) systems are capable of processing large amounts of events, utilizing them to monitor, steer and optimize business in real time. The lack of tracking events and maintaining the causal relationships and traceability between those events, as well as aggregating them to higher-level events, is a problem that is currently investigated by many research groups. In this paper, we present SARI-SQL, which is a domain-specific event-query language, (EQL) that is designed for business analysts to easily gain insight into business events. SARI-SQL enables the retrieval of near real-time events and can process historical events, metrics and scores for analytical purposes. We introduce the SARI-SQL syntax and show infrastructural components for the query engine. We further show examples to illustrate the query language, and propose a reference implementation for the query engine.

1. Introduction

Business processes evolved to networked workflows that are complex and executed in parallel with little human involvement to meet the needs of today's agile and adaptive business environments [1]. Such contemporary business requirements call for agility, flexibility and service orientation. A simplified summary of this widely discussed and necessary trend can be reduced to the demand that today's businesses have to adapt their processes and organizations faster than their competitors. Organizations that are able to handle critical situations faster than their competitors will end up us winners in today's globalized and fast-paced business climate.

The pillars of such agile models are loosely coupled, distributed and service- or event driven-oriented systems that generate huge amounts of events at various granularity levels. The lack of tracking those events and maintaining the causal relationships and traceability between those events, as well as aggregating them to high-level events or correlating them, is a problem that is currently investigated by many research groups [2][3][4].

Event-based systems are increasingly gaining widespread attention for classes of problems that require integration with loosely coupled and distributed systems for time-critical business solutions. The field of event-based or event-processing systems is a quite young area of research and is mainly influenced by the publish-subscribe paradigm, relational databases and later on by active- and zero-latency data warehousing.

A promising solution for these problems is Complex Event Processing (CEP). CEP defines a set of technologies to process large amounts of events, utilizing them to monitor, steer and optimize the business in real time [2]. Such an event-based system continuously processes and integrates the data included in events without any batch processes for extracting and loading data from different sources and storing it to a data warehouse for further analysis. CEP solutions capture events from different sources, with different time order and take events with various relationships between each other into account.

The purpose of this paper is to introduce the CEP domain specific language SARI-SQL, its sub-language EAExpression and the necessary infrastructural components for managing events and their relationships for retrieval and analytical purposes.

The query language SARI-SQL for retrieving near real-time events and creating conjunctions with historical events, metrics and scores is in contrast to the Event Clouds indexing approach [5][6][7] which is a formally structured solution that extends ANSI-SQL. SARI-SQL can be allocated to the group of domain-specific languages and, therefore, it is capable of satisfying the special requirements and meeting the characteristics of events and their relationships. SARI-SQL creates an abstraction of the event type model by encapsulating a lot of overhead and by creating an abstraction layer over events and their internal data structures. The user of this language can concentrate on only expressing the required results instead of putting effort into making the "things run". As a consequence, it allows domain experts to easily gain insights due to the level of abstraction of the specific problem domain.

2. Related Work

The roots of event-based systems basically come from two different fields. The first one has grown out of append-only databases and can be seen in the world of triggers and zero/active data warehousing. The second root can be allocated to the development of middleware systems, the publish/subscribe paradigm and the various filter and subscription techniques. Such subscription efforts formed the first rudimentary event processing engines including languages comparable to a certain extent to SARI-SQL.

Each of those fields formed event-based systems with a partly different terminology and understanding. However, these solutions mostly provide the same capabilities. An ongoing debate that reflects this issue is currently about event stream processing and complex event processing where both provide comparable functionalities, but their roots and their event processing approaches are different. Another example is the view of event processing systems and their scope. Solutions coming from the publish/subscribe field tend to consider the communication infrastructure as an essential part of an event-based system as they usually aim for wide-scale event processing with the trade-off for expressive event processing languages.

SIENA [8] is a multi-broker event notification service based on the publish/subscribe paradigm, focusing on maximizing the subscription expressiveness while maintaining the scalability in wide-area networks. SIENA's subscription language allows to define filters and patterns for retrieving event notifications. Furthermore, the subscription distribution is pushed as far as possible towards notification producers in such networks in order to save bandwidth as events, in which no consumers are interested in, so it can be filtered out immediately.

JEDI (Java Event-based Distributed Infrastructure) [27] is a distributed communication middleware based on the publish/subscribe communication paradigm. The main characteristics of JEDI is that it is based on multicast, nor the destination or the producing source is defined, the notification delivery is guaranteed and mobility is ensured through the concept of reactive objects [28]. The event notification routing is carried out through so called event dispatchers, which is physically either organized as a central component or by multiple distributed components that are then interconnected through a hierarchical topology. Subscriptions are forwarded from the issuing nodes to the top root node. The data model of JEDI is built upon key-value pairs where each of event notification consists of a name and attributes. Subscriptions are called event profiles in JEDI and contain filters also based on key value pairs.

Gryphon [9] is a content-based publish/subscribe notification system which has a special focus on creating a redundant broker network for routing notifications with the ability of providing guaranteed delivery while preserving scalability and availability. The system is based on acyclic directed information graphs which are used to define the exchange of event notifications between the producers and consumers. Each of the nodes in the graph can contain specific types of events, whereas the edges are capable of selecting/filtering or transforming them.

Hermes [10][11] is a distributed middleware based on the publish/subscribe paradigm making use of type- and attribute-based subscription mechanisms while taking traditional middleware features like interoperability, reliability and usability into account. The system is based on event broker networks which form a logical routing network for event notifications. The subscription language itself is a message that contains attribute filters on specified event types. A filter is an XPath expression that is evaluated against incoming event notifications.

The Aurora system [12] developed an architecture to process the data streams with some QoS requirements by decomposing queries into a few predefined operators. These works mainly focus on the system architecture, continuous query execution (i.e., scheduling and various non-blocking join algorithms), and QoS delivery mechanisms.

Medusa is stream processing system basically using Aurora's query capabilities [13][14][15], with the goal to enable the distribution of evaluation of Aurora queries. Medusa is focusing, in contrast to the Aurora project, on developing a distributed infrastructure in order to create a loosely coupled network of stream processing components.

Borealis [16] is a distributed stream processing engine built upon Aurora [15][17] and can be seen as its successor. The project explicitly addresses [16] the problem of dynamic revision of query results, in which it is sometimes necessary to correct errors in previously received data. Further, Borealis is aiming at providing support for query modifications during runtime, with a low overhead, fast and automatic modifications and providing high scalability with growing demands in heterogeneous environments that include optimization strategies and fault tolerance.

Esper [18][19] is an Open Source event stream processing solution for analyzing event streams. Esper supports conditional triggers on event patterns, event correlations and SQL queries for event streams. Esper's query language is comparable to ANSI-SQL, but is extended with concepts to conform the requirements of event processing. Such features are, for instance, moving time windows and the support of aggregations.

Amit [3] is an event stream engine whose goal is to provide high-performance situation detection mechanisms. The situation detection can be defined through the usage of a number of operators. Amit supports quantifier attributes, join operators, counter functions, temporal operators and also event absence operators that can be applied within a define lifespan.

SASE [48] considers itself as a complex event processing system and offers an expressive and user-friendly continuous query language. The language allows defining sequence patterns which take the temporal order of events into account and support the definition of sliding time windows that can be spanned over long durations (hours or days). SASE is capable of persisting event processing results and performing combinations of continuous query results with database queries.

3. Event-Driven BI Architecture

Traditional data warehousing approaches are used to understand the business situation based on a collection of historical data and do a good job as a decision-making tool at a strategic level and for understanding the business situation based on a collection of historical data [26].

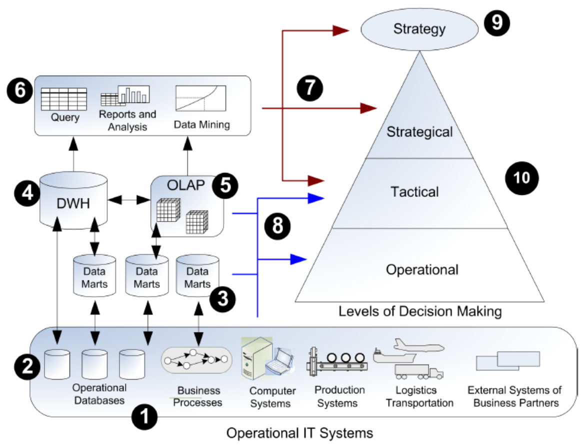

Figure 1: Decision-Making Pyramid adapted from [26]

Figure 1 illustrates the loop of information that is currently in place in most of the organizations to support decision making. At the bottom of the figure, there is a list of exemplary IT systems from various organizations working together to accomplish a task. Most of these systems contain some sort of data storage managed by operational databases (2). All of these systems can be attached to ETL (extract, transform and load) processes to extract data for either local data marts (3) of specific organizations or for data warehouses (4) containing global information. The data integration process (ETL) is a time and resource-consuming task which is often performed regularly overnight in order to not disturb the daily business operations. This task requires high efforts of adaption in case the underlying systems change their data structure. To extract valuable information from the collected data, it is required to perform analysis over the data collections. There is a wide range of business intelligence tools on the market to perform the tasks of data analysis (6).

Users of such systems can place custom queries, create reports or dig around in the data by performing data mining tasks. By exercising these tasks, the users of such systems are able to extract valuable information, perform analysis on them and, at the end, interpret the information and apply the gained insights to business.

On the right side, there is a pyramid (10) representing the levels of decision making. On the very top, there is an enterprise vision and strategy (9) formulated by clear goals, including objectives, concrete numbers and formulated means how to reach these goals - in short, quantitative and qualitative goals underlined with actions to be taken to a certain level of abstraction.

Depending on the level of decision making, information from different sources is required on different aggregation levels. The level of decision making also defines the requirements of the freshness of the data used for decision making. Decisions made on a strategic level require less fresh data than decisions made on an operational level. The time of making a decision is an important key factor on lower levels of the pyramid. Long decision cycles on operational levels can cause risks or losses of business opportunities. Today, business requires fast-paced decision making on operational and tactical decision-making levels. Hackathorn [20] points out that the business value of an event decreases by the amount of time that passes until an appropriate action is taken.

Although analyses on historical data using OLAP, for instance, are a way to gain deeper insights, data warehouses are not meant to provide process knowledge in the first place, and regularly they often don't provide data in a real-time fashion [76]. This leads to the lack of making a decision on a tactical or operational level in a timely manner in order to preserve the value of the information until an action has been taken.

Analysis and mining tools [5][22] are integrated with the Event-Base for discovering causal relationships between events, mining events, finding patterns within events as well as providing a facility for visualizing events, their relationships and patterns.

Event Cloud [5][6][7][23] was one of the first approaches to allow users to search for business events and patterns of business events within a repository of historical events. Event Cloud processes events, thereby creating an index for events and correlations between events in order to enable an effective event search. It provides a historic view of events with drill-down capabilities to explore and discover different aspects of business processes based on event correlations. Event Cloud allows users to investigate events, such as picking up single events and displaying their content and discovering related events or event patterns.

The Event Analyzer is a research effort coupled to SARI and the Event-Base that provides a toolset for visualizing streams of events making use of the tunnel metaphor [22]. The Event Analyzer is based on retrieving the desired information from the Event-Base through placing SARI-SQL queries against the Event-Base. By applying queries, the Event Analyzer users are capable of retrieving events, persevered with their correlations in order to create a tunnel-like visualization.

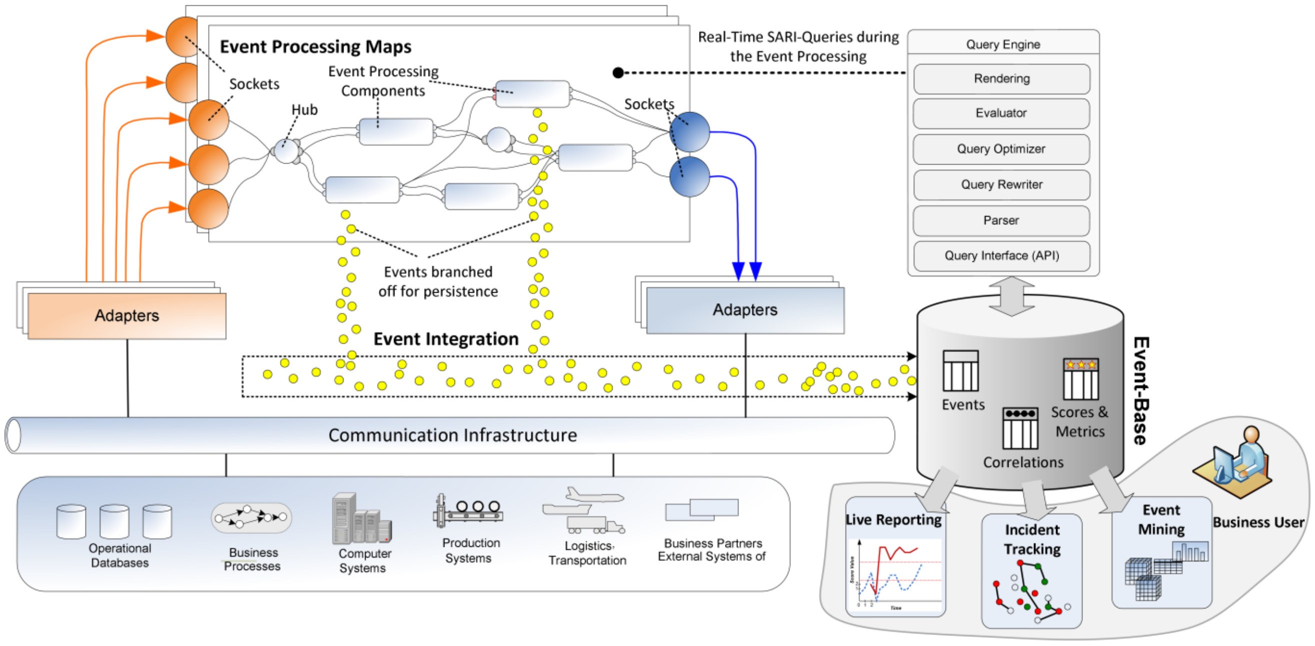

Figure 2: SARI Architecture and Event-Base

3.1 Architecture and Terminology

One of the most promising concepts that approaches the problems of gaining real-time business knowledge enables closed loop decision-making on operative levels and delivering real-time information on processes is Complex Event Processing (CEP). The main application field for CEP is generally in areas where someone needs low latency times in decision cycles [20] combined with a high event throughput for observing relevant business events of predefined or exceptional situations, indicating opportunities or problems.

A CEP system continuously processes and integrates the data included in events without any batch processes for extracting and loading data from different sources and storing it to a data warehouse for further processing or analysis. CEP solutions capture events from different sources, with different time order and take events with various relationships between each other into account.

In terms of Business Intelligence, active or zero latency data warehouses can significantly reduce the refresh cycles, but the missing process knowledge that is implicitly provided by events is still absent. The main gap that CEP solutions are addressing is the delay in analysis and, thus, in taking actions, which can result in a loss of business value [20]. CEP systems can handle the flood and the noise of events coming from operational systems and are capable of significantly reducing the latency if deployed properly. They are capable of enriching the information from historical sources (for instance gained insights from statistical analyses) and performing decisions and actions automatically if desired. The key characteristics of a CEP system are its capability of handling complex event situations, detecting patterns, creating correlations, aggregating events and making use of time windows.

SARI (Sense And Respond Infrastructure) [21] is solution that is capable of fulfilling the requirements for CEP systems. In SARI, business situations and exceptions are with sense and respond rules which have been designed to be created and modified by business users. SARI offers a user-friendly modeling interface for event-triggered rules, which allows to model rules by breaking them down into simple, understandable elements.

The Event-Base is a data repository that can be considered as a next-generation database for the purpose of managing events, their correlations, continuously calculated metrics and correlations between events. It provides an efficient up-to-date operational storage together with retrieval mechanisms for business events for analytical as well as operational purposes without the costly data staging processes known from established data warehousing solutions. By providing access to such processed and prepared real-time events, it is possible to derive and generate new knowledge in order to provide facilities for decision making with low latency and, thus, it allows catching business opportunities or threats in a timely manner.

Current solutions for the analysis of business events are usually a combination of several tools, whereas the tools are not capable to provide a unified view on real-time events of an organization out of the box. This drawback of existing solutions means that the data integration of events requires large efforts and possibly extensive use of consulting services and, thus, it leads to high costs.

The Figure 2 illustrates the overall components and the collaboration of the event processing instances within the Event-Base. On the bottom of the figure shown, source systems (i.e. the event producing components) continuously generate event notifications. The Sense Layer represents the adapters of SARI that can be docked to the event producing systems or the communication infrastructure. The adapters can gather events in either a push or pull process and propagate them into the event processing realms. The event processing flow is modeled with various components according to the business requirements [21].

In the event processing engine that evaluates the event processing maps, there is an Event-Base Publishing Component that is responsible for propagating the events into the Event-Base. This special event component can be arbitrarily connected, in a declarative way, to event services, hubs or adapters. An Event-Base publishing component is consuming events by applying filters, creating correlations between events, calculating metrics and then mapping those events to the Event-Bases data repository. This process is highlighted in Figure 2 by the term Event Integration which is represented as a flow of events from the event processing maps to the Event-Base repository for further application. During the flow of the events, the correlations are created and the metrics are calculated.

Similar to database management systems, SARI can create different numbers of Event-Bases for various event processing applications. Each Event-Base contains application data and the infrastructure for processing events. The events stored to the Event-Base are preprocessed by a full-text indexing engine for providing fast access to them and to also preserve the relationships between the events. The processes of indexing events, their relationships and performance discussions can be found in [5][7].

The core access component is a query engine supporting SARI-SQL. SARI-SQL is extending the common SQL capabilities with special semantics that take the special nature of events into account. The query engine consists of six logical components, an API, a SARI-SQL language parser, a query rewriter, a query optimizer, a result evaluator and a rendering engine for returning the results of a placed query.

3.2 Event Analysis

Analysis and mining tools [22][5] are integrated with the Event-Base for discovering causal relationships between events, mining events, finding patterns within events and they provide a facility for visualizing events, their relationships and patterns.

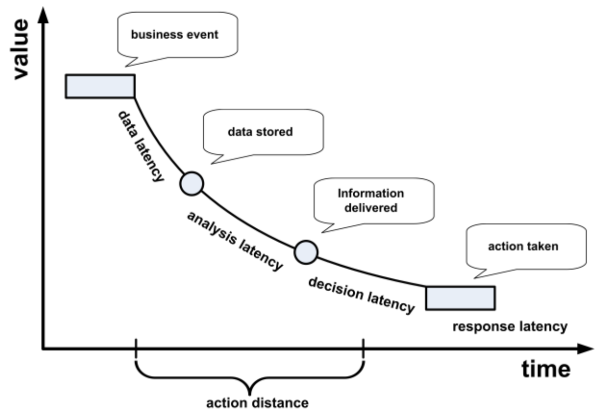

Figure 3: The decrease of the business value of events over time according to [20]

Current solutions that exist today, especially in the field of business intelligence and data mining, relating to the analysis of historical data and/or the appliance of OLAP, are capable of providing deeper insights into business operations. However these traditional analysis tools are not capable of providing process knowledge and they regularly lack of providing data in a real time fashion. This leads also to a lack of decision making at operational and tactical levels in order to preserve the value of information which is decreasing by time [20] (compare with Figure 3: The decrease of the business value of events over time according to [20]). Furthermore the traditional approaches don't consider the special characteristics of events that represent the flow of information in businesses, instead they align collections of historical data along dimensions for further evaluation. The relationships (e.g. correlations) and specific aspects of events are not available anymore in classical representations.

David Luckham presents in [2] a compilation of requirements for CEP analysis tools, whereas the requirements for those tools are addressed by the Event Analyzer. The list of requirements from Luckham were reduced by Vecera in [23] to the most relevant points.

Display parameters and attributes of an event. This requires that a user needs to have facility to examine events, their attributes and their relationships at different granularity level without being overloaded. Further the item requires the (graphical) representation of events according their temporal order.

Trace the causal history of an event in an event execution. Requirement that allows the backtracking of causal event chains that made a specific event to fire.

Graphically present events, event timelines and event correlation. An event analysis tool should be capable of graphically represent events, their relationships to each other (e.g. correlations) and some way the temporal propagation during time.

Search Patterns. This is one is possibly the mightiest functionalities that is required by an event analysis tool. An analysis tool should be capable of searching through large repositories of events in order to detect patterns that represent some sort of exceptional situations.

Drill-Down. The tool should support event granularity abstraction mechanisms. Such mechanisms should allow the users to create mind-size large abstractions of given situations and then allow to drill down deeper to more detailed levels.

The Event Analyzer is a research effort, coupled to SARI and the Event-Base, that provides a toolset, for visualizing streams of events making use of tunnel metaphor [22]. The Event Analyzer is based on retrieving the desired information from the Event-Base through placing SARI-SQL queries against the Event-Base. By applying queries the Event Analyzer users are capable of retrieving events, persevered with their correlations, in order to create a tunnel like visualization.

3.3 Data Management

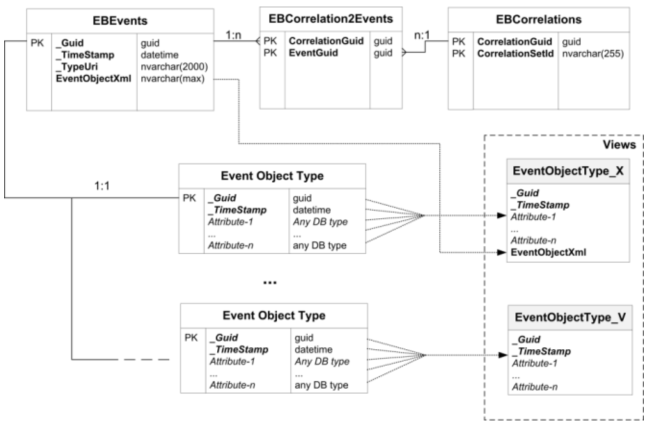

Figure 4: EventBase Data Model

This section pays attention to how events and their correlations are processed on a conceptual level to carry them on to the underlying data structures and, further, how the events are maintained for later access. This topic is highly relevant for SARI-SQL, as it is the data repository that is used by the SARI-SQL engine to retrieve the data that is requested by a given query.

Figure 4 shows the Entity-Relationship (ER) diagram of the EventBase's underlying data structure for managing the events and their correlations.

The event typing model of SARI [24] consists of event object type definitions which have one or more attributes. An attribute type can be either a single-value type, collection/dictionary type or another nested event object type. Single value types correspond to an ordinary runtime type such as Integer, String and so forth. During execution runtime of the EventBase, relations for all event object types maintained in the event object type library are created. Single value type attributes are represented by relational-attributes with a database type corresponding to the event object attribute type. In contemporary RDBMS, for every runtime type, there is a corresponding database attribute data type available. For each event object type, there is a unique identifier and a timestamp defined during execution runtime.

For nested event object types and collection/dictionary types, there are no additional relational-attributes created. The representation of such nested types is solved by the *EventObjectXml* attribute in the relation *EBEvents* which also contains a unique type resource identifier with a reference to the corresponding *EventBase* repository that it belongs to. The *EBEvents* contains a 1:1 mapping to the corresponding event object type over the unique identifier _*Guid* of the Event Object. The relational-attribute *EventObjectXml* contains the serialized XML representation of the event objects, including the nested types like dictionaries, event object type attributes or unknown attributes.

The relation *EBCorrelations* represents correlation sets and consists of a unique identifier *CorrelationGuid* and a *CorrelationSetId* containing the correlation identifier (e.g. name). The two relations *EBCorrelations* and *CorrelationGuid* form a n:m relationship which is resolved by *EBCorrelation2Events*.

For example, let's consider *TransportStart* and *TransportEnd* events that correlated over an *OrderId* attribute with each other. During event processing runtime, the event objects of both events are inserted into the *EBEvents* relations and, in parallel, into their corresponding event object type relation for the *TransportStart* and *TransportEnd* events. Correlated *TransportStart* and *TransportEnd* events are brought into relationship through an entry in the *EBCorrelation2Events* relation, whereas the events in a specific correlation can be retrieved through the *EBCorrelations* by querying for the *CorrelationSetId*.

Now that the event type specific relations don't contain the whole event object data like nested types or any other specific header attributes, the extended *EventObjectType_X* View is introduced. Such a view exists for every event object type maintained in the event object type library. The view contains every "flat" attribute (e.g. single value type attribute) of an event and, in addition, it offers the *EventObjectTypeXml*, including the nested types for access. Basically, the *EventObjectType_X* can be accessed to retrieve the full information of an event.

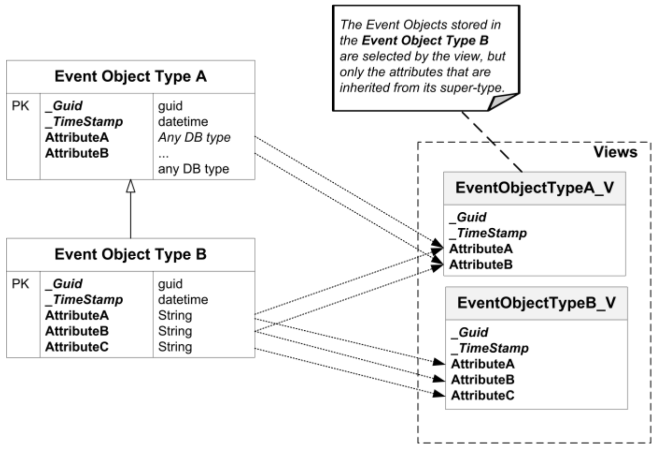
The Event Objects stored in the **Event Object Type B** are selected by the view, but only the attributes that are inherited from its super-type.

Figure 5: Event Object Type Inheritance Data Management

The last issue of this topic is the management of event object type inheritances. Advanced event typing concepts are discussed in [24]. Event object types can be specialized by inheriting attributes from parent types similar to object-oriented programming languages.

Figure 5 shows an illustration of two event object types where the *Event Object Type B* is inherited from *Event Object Type A*. Originally, *Event Object Type A* contains the two attributes (*AttributeA* and *AttributeB*). The *Event Object Type B* additionally introduces the *AttributeC*. In case of inherited types, the attributes of the super-types are duplicated in the relations of the specialized types. If the view where the super-types are accessed, every event object is included from all derived types, but with the difference that only the attributes of the super-types are available.

4. SARI-SQL

EAExpression and SARI-SQL are two languages independent from each other. However, SARI-SQL includes EAExpressions in order to be able to express the access of events in its queries.

EAExpression is an expressive and easy to understand language for accessing events, both during the design phase of event processing applications and also during runtime to perform evaluations on events. The difference is that during the design phase event object types, known to the event processing realm, are under evaluation. The event object types define the data structure of events, including their attributes and their types. This information is stored in event object type libraries of the corresponding Event-Bases. The EAExpression processor is capable of using library maintained event type information to validate given expressions according to the accessed event types.

The EAExpression language pays special attention to the nature of events and their specific characteristics. The access language is tightly coupled to the SARI's underlying event object type model [24] and, during runtime, to the evaluation of expressions on event objects. However, the syntax and semantics of the language can be decoupled completely from the underlying models with certain efforts. EAExpressions play an important role in the event processing models of SARI. Possibly the most important application is in the event-driven rules described in [25].

SARI-SQL, on the other hand, is a query language that can be used to retrieve events, correlations, metrics and scoring information from the Event-Base. The language is, from a syntactical point of view, comparable to ANSI-SQL and is also a declarative language. Special attention was paid to extend the concepts of ANSI-SQL by special event processing related concepts, such as providing easy access to correlated events without breaking too much with the syntax-style of ANSI-SQL. This is because SARI-SQL is aimed at power users applying queries through APIs in their programs and, on the other hand, at domain experts (e.g. business users) that try to retrieve knowledge from the Event-Base. As SQL is an industry wide standard and is part of many business related applications, a lot of non-technical people are familiar with the language. The special capabilities and extensions that are offered can be found very intuitively.

The SARI-SQL syntax structure is basically relying on four main key clause-constructs:

SELECT *EAExpression*
FROM *[EventObjectType|Scores|Metrics|Joins]*
WHERE *EAExpression*
OVERCORR *CorrelationIdentifiers*

- **SELECT.** The SELECT clause represents the projection part of the query, as is known from ANSI-SQL. It allows to select attributes but, in contrast to normal SQL, these are attributes of event object types instead of tables (i.e. relations). Furthermore, each of the projections in this clause allow to define EAExpressions with certain restrictions that will be explained later on.
- **FROM.** The FROM clause defines the data sources that can be either an event object type, a metric or a score information which are under examination in terms of applicability in the projection (SELECT) clause and the condition (WHERE) clause. In the relational database SQL world, the FROM clause allows to select relations that consist of attributes (e.g. columns). In SARI-SQL, these are the attributes of event object types or metrics and scores of a specific type.
- **WHERE.** The WHERE clause is the conditional part of an SARI-SQL query and allows to define filters upon attributes and, further, it allows to create inner joins over event object type attributes, metrics and scores similar to the relational model.
- **OVERCORR.** The OVERCORR construct allows to define a kind of a preselector over events by restricting the space of available event object types. This is a construct that allows to retrieve events only from a certain defined correlation set.

Due to space limitations, we will focus on illustrative examples to introduce the syntactical characteristics of SARI-SQL and the evaluation procedures. Figure 6 shows three event object types (*ShipmentCreated*, *TransportStart*, *TransportEnd*) forming two correlations *ShipmentToTransport* and *TransportInfo*. This example will be used to explain the syntactical concepts of SARI-SQL throughout in this paper.

Example 1 - Projecting Event Object Type Attributes.

```
a) SELECT start.StartLocation, end.EndLocation
   FROM TransportStart start, TransportEnd end
b) SELECT ShipmentID, FreightValue - Costs,
   EAAvg(Product.Price) FROM ShipmentCreated
```

The first example selects two different attributes (*StartLocation* and *EndLocation*) from two different event object types (*TransportStart*, *TransportEnd*). As there is more than one event object type in the FROM clause, it is required to define an alias for them. As aliasing is activated, it is also mandatory to apply it in every other clause in order to be able to access the attributes of event object types and avoid ambiguity. This is an important point, as in EAExpressions, the dot notation separates event object type identifiers from the attributes. In SARI-SQL, the outer most left identifier in a dot concatenated expression is an alias if there is more than one event object type in the FROM clause. The last example illustrates the possibility of applying EAExpressions in SELECT clause items, such as the EAExpression-Average function on the collection of Prices.

The nature of event object types describes that there are so-called multi-value attribute types where dictionaries, collections and nested event object types as attributes belong to. As these types of attributes represent multidimensional data, they are returned as data objects preserving the data dimensions. This is an important difference to the relational model where the structures (i.e. relations) are "flat".

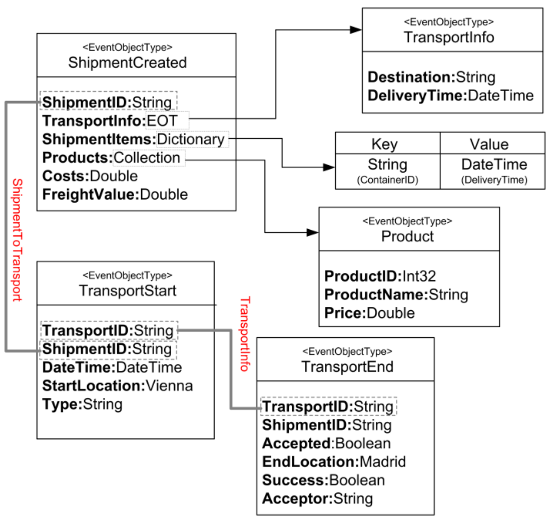

Figure 6: Event Object Type Example for the SARI-SQL

Example 2 - Retrieving Event Object Types of Correlations.

```
a) SELECT start.StartLocation, end.EndLocation
   FROM TransportStart start, TransportEnd end
   OVERCORR TransportInfo
   WHERE start.StartLocation = 'Vienna'
b) SELECT start.StartLocation, end.EndLocation
   FROM Corr1.TransportStart start, Corr2.TransportEnd end
   OVERCORR TransportInfo Corr1, TransportInfo Corr2
```

The example a) selects the *StartLocation* and *EndLocation* from the types *TransportStart* and *TransportEnd*. Both event object types must be contained in sessions of a *TransportInfo* correlation, with the restriction that the *StartLocation* of a transport is settled in Vienna. The example b) takes two correlation sets which results in a Cartesian product because of the two event object types in the FROM clause. This example is meant for illustrating the usage of aliasing with more than one correlation set used in a query.

Example 3 – Accessing Event Metrics.

```
a) SELECT end.TransportId
   FROM Metric('AvgTransportDuration') avg, TransportEnd end
   WHERE end.EndLocation = "Madrid" AND avg.value > 3
```

This example selects all classifiers from the metric *AvgTransportDuration*. Each classifier of the metric is treated as a column in the SELECT clause and, thus, it is also possible to define joins or constraints in the WHERE clause on them like shown above.

Example 4 – Defining Implicit Time Windows.

```
a) SELECT * FROM TransportStart
   WHERE 01.02.2009 < @timeCreated < 29.02.2009
```

Event processing solutions such as Esper, [18] for instance, make use of continuous query languages for the purpose of creating queries over streams of events. For performance reasons, these types of solutions provide means to define a time frame in which the events are valid to be run against a query. This could be, for instance, a query that is calculating the average amount of attributes in a time frame of 15 minutes. Every event object in SARI contains event header attributes which are inherited from the BaseEvent type and capture metadata such as the creation time or priority of an event. The header attribute, *timeCreated*, contains the timestamp of the event creation.

This example shows how the *timeCreated* header attribute can be used to set a time frame over events independent from explicitly modeled timing related attributes.

4.1 SARI-SQL Evaluation

The evaluation process of SARI-SQL is split up into four stages where, in contrast to EAExpressions, the validation and evaluation process are not explicitly divided from each other. The four stages of the SARI-SQL query engine processes are introduced subsequently in this section. The concrete implementation is not within the scope of this paper; therefore, the query engine is described from a conceptual point of view to create a separation from implementation details.

Figure 7 represents the overall query evaluation process starting from the query input to the query rendering that passes over the results.

The SARI-SQL validation processes consist of syntax parsers that check the syntactical correctness of a given query and the abstract syntax tree parser that handles the semantic correctness of the query.

The semantic correctness of a query checks, for instance, if the accessed event object types and their attributes exist in the Event-Base.

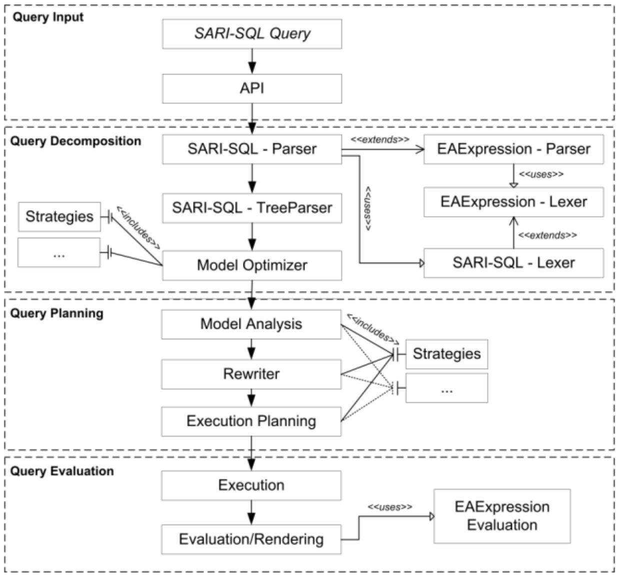

Figure 7: SARI-SQL Query Processing Evaluation

The query validation processes is more comparable to a prepared statement from the Java-World. The SARI-SQL query engine prepares the query for execution by transforming the given query from an abstract syntax model into a more efficient internal data structure that is then used to perform optimization strategies and apply more optimal query rewriting methods. This prepared query model is then available and can be reused during runtime for several executions. If a specific query has to be reused more than once, it can save significant time during execution.

The preceding paragraphs will introduce the four SARI-SQL evaluation blocks and the query processing from a conceptual point of view.

Query Input. The query input consists of an API to access and propagate the query statement to the underlying SARI-SQL query engine. Further, it manages and returns syntax, semantic and evaluation (e.g. query execution) error states, query engine messages (e.g. "explain") for tracking the internal query processing steps and the query result retrieval.

Query Decomposition. The main job of the decomposition block is to prepare a given SARI-SQL query statement for execution (e.g. the query is pre-processed, optimized and made ready for execution against an Event-Base) and to catch errors within the query.

Query Planning. The query planner components analyzes the query that which was previously assembled and optimized in the query decomposition stage, and it performs a rewriting of the query into the target Event-Base repository schema. The rewriting itself adapts the query to the target RDBMS system of the Event-Base and is, therefore, database-vendor independent. The analyzer components further determine the execution strategy whose information is required for post-processing the result set.

Query Evaluation. The query evaluation block is in charge of executing the prepared and pre-processed SARI-SQL query. Due to the analysis and rewriting strategies in the previous Query Planning processing steps, it is necessary to reevaluate the returned results from the Event-Base in order to deliver the required results. The EAExpressions evaluation processes are partly used in this stage. Therefore, this SARI-SQL processing block can also be seen as a post-processing step before the result is turned over to the API.

Example 5 – Rewriting of Correlation Queries

The *OVERCORR* clause is applied after the FROM clause and holds a list of correlation name items. In case more than one correlation is used, aliasing is applied in order to be able to define which event object type in the FROM clause belongs to which correlation to avoid ambiguity. In case that a FROM item is not aliased, but aliasing is used for correlations, then all event objects of the corresponding types are selected.

The planning process of the correlation clause evaluation is quite simply structured. For every correlation item, the corresponding referenced items are walked through and a rewriting strategy is applied. The most complex part of this process is to perform the rewriting for correlations and their corresponding FROM items.

```
a) SELECT start.StartLocation, end.EndLocation
   FROM TransportStart start, TransportEnd end
   OVERCORR TransportInfo
b) SELECT start.TransportStart, end.AttrInt2 FROM (SELECT
   ETgeneratedId0.*, corr.* FROM EBCorrelations corr,
   EBCorrelations2Events corr2event, dbo.[TransportStart_X]
   ETgeneratedId0 WHERE corr.CorrelationSetId = 'TransportInfo'
   AND corr2event.CorrelationGuid = corr.CorrelationGuid
   AND corr2event.eventguid = ETgeneratedId0._guid) start FULL
   OUTER JOIN (SELECT ETgeneratedId1.*, corr.* FROM
   EBCorrelations corr, EBCorrelations2Events corr2event,
   dbo.[TransportEnd_X] ETgeneratedId1 WHERE
   corr.CorrelationSetId = 'TransportInfo' AND
   corr2event.CorrelationGuid = corr.CorrelationGuid
   AND corr2event.eventguid = ETgeneratedId1._guid) end
   ON start.CorrelationGuid = end.CorrelationGuid
```

To give an impression of the resulting rewriting (b), the reader should consider the simple SARI-SQL query example (a) containing correlation access where the target RDBMS of the Event-Base is the MS SQL Server 2005.

Example 6 – Post-evaluation of Nested ypes

As the analysis and rewriting strategies created a result overhead for handling unrepresentable items, such as nested types, it is necessary to post-process in-memory result tables. SARI-SQL spans up internal trees of sub-expressions in order to create decomposition points. Parts of those decomposed-expressions can be directly pushed onto the Event-Base, and parts of the conditions and joins must be post-evaluated in memory with the help of EAExpressions.

```
SELECT ShipmentID, Costs  FROM ShipmentCreated
WHERE TransportInfo.Destination = "Vienna"
```

In order to ease the understanding of this process, the steps are illustrated with an example in Figure 8.

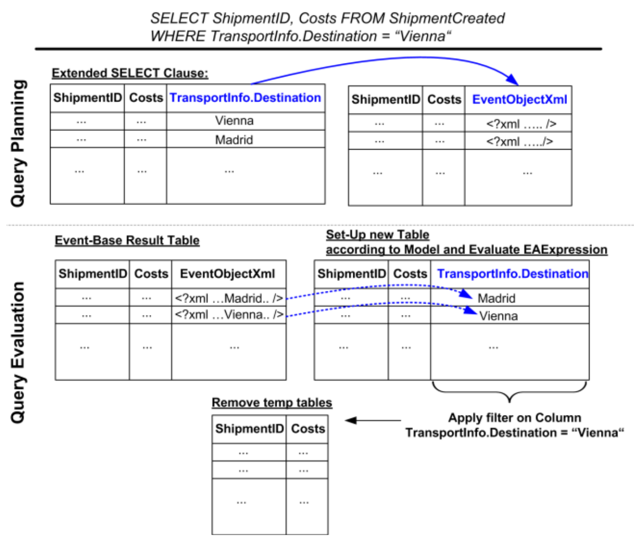
SELECT ShipmentID, Costs FROM ShipmentCreated
WHERE TransportInfo.Destination = "Vienna"
Query Planning
Extended SELECT Clause:
ShipmentID
Costs
TransportInfo.Destination
Vienna
Madrid
EventObjectXml
<?xml ..... />
Query Evaluation
Event-Base Result Table
Set-Up new Table according to Model and Evaluate EAExpression
<?xml ...Madrid.. />
<?xml ...Vienna.. />
Remove temp tables
Apply filter on Column TransportInfo.Destination = "Vienna"

Figure 8: Post-Evaluation Process Example

The first action was taken in the Query Planning block where the *TransportInfo.Destination* item of the SELECT clause was created, as it is a nested type and, thus, a non-DB type. Further, that item was rewritten to access the corresponding *EventObjectXML* which is, in this example, the *ShipmentCreated* event object type. The result table in the Query Evaluation block contains the *ShipmentID*, Costs and the *EventObjectXml* holding the event object instances wrapped into a XML structure. Now the new result table is created holding the original *TransportInfo.Destination* instead of the *EventObjectXml*. During the process, every row of the original result table is walked through and, as the first two columns were DB types, they can be directly taken into the final result table. However, the third column (i.e. the *TransportInfo.Destination*) is treated as an EAExpression that is evaluated against the event object, wrapped inside XML of the corresponding row. The result of the EAExpression evaluation process is then stored in the new result table. At the end of the process, the WHERE clause is rewritten to match the label of the *TransportInfo.Destination* column. Then, it is applied as a conditional filter against the new result table and every tuple is excluded that does not contain "Vienna" in the third column. Finally, the overhead helper columns are stripped away so that only the explicitly defined projection items of the SARI-SQL SELECT clause are left over in the result table.

5. Conclusion and Future Work

In this paper, we have introduced and discussed the CEP domain specific Event-Query Language (EQL) SARI-SQL and the necessary infrastructural components SARI and the Event-Base. This DSL creates an abstraction of the event type model by encapsulating a lot of overhead and putting a layer over events and their internal data structures. Due to space limitations, we extracted several key features and discussed interesting capabilities on conceptual levels. For instances, SARI-SQL allows a simple access to correlations of events and the application of defined constraints. The results can then be used by domain experts to analyze and gain insight of the actual problems. The work presented in this paper is part of a larger, long-term research effort aiming at developing a comprehensive set of technologies and tools for event analysis. We are looking forward to introduce more powerful evaluation strategies as well as language enhancements that allow to create conjunctions between live event streams (i.e. Continuous query languages) and historical data.

6. References

[1] D. Luckham, A. Manens, S. Bhansali, W. Park, and S. Daswani. Modeling and causal event simulations of electronic business processes. 2005.

[2] D. Luckham. The Power Of Events. Addison Wesley, 2005.

[3] S. Babu and J. Widom. Continuous queries over data streams. SIG-MOD Record, 30(3):109_120, 2001.

[4] J. Schiefer and C. McGregor. Correlating events for monitoring business processes. In ICEIS (1), pages 320_327, 2004.

[5] S. Rozsnyai, R. Vecera, J. Schiefer, and A. Schatten. Event cloud - searching for correlated business events. In CEC/EEE, pages 409_420. IEEE Computer Society, 2007.

[6] S. Rozsnyai. Efficient indexing and searching in correlated business events. Diploma thesis, Technische Universität Wien, 2006.

[7] R. Vecera, S. Rozsnyai, and H. Roth. Indexing and search of correlated business events. Ares, pages 1124_1134, 2007.

[8] A. Carzaniga, D.S. Rosenblum, and A.L.Wolf. Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems, 19(3):332_383, 2001.

[9] IBM. Publish/subscribe over public networks. Technical Report, 2001.

[10] P.R. Pietzuch. Hermes: A Scalable Event-Based Middleware. PhD thesis, University of Cambridge, 2004.

[11] J.M. Bacon and P.R. Pietzuch. Hermes: a distributed event-based middleware architecture. Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on, 2002.

[12] ANTLR. Antlr webpage. http://www.antlr.org/, 02 2008.

[13] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S.B. Zdonik. Scalable distributed stream processing. In CIDR, 2003.

[14] M. Balazinska, H. Balakrishnan, J. Salz, and M. Stonebreaker. The medusa distributed stream-processing system. http://nms.lcs.mit.edu/projects/medusa, 01 2008.

[15] S. Zdonik, M. Stonebraker, M. Cherniack, U.C. Etintemel, M. Balazinska, and H. Balakrishnan. The aurora and medusa projects, May 06 2003.

[16] D.J. Abadi, Y.Ahmad, M.Balazinska, U. Cetintemel, M. Cherniack, J.H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S.B. Zdonik. The design of the borealis stream processing engine. In CIDR, pages 277_289, 2005.

[17] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S.B. Zdonik. Aurora: a new model and architecture for data stream management. VLDB J, 12(2):120_139, 2003.

[18] Esper Codehaus. Esper. http://esper.codehaus.org/, 01 2008.

[19] T. Bernhardt and A. Vasseur. Esper: Event stream processing and correlation. http://www.onjava.com/lpt/a/6955, 01 2008.

[20] R. Hackathorn. Current practices in active data warehousing. DMReview, 2002.

[21] J. Schiefer and A. Seufert. Management and controlling of time-sensitive business processes with sense & respond. In CIMCA/IAWTIC, pages 77_82. IEEE Computer Society, 2005.

[22] M. Suntinger, H. Obweger, J. Schiefer, and Groeller. The event tunnel: Interactive visualization of complex event streams for business process pattern analysis. Technical report, Institute of Computer Graphics and Algorithms – Vienna University of Technology, 2007.

[23] R. Vecera. Efficient indexing, Searching and Analysis of Event Streams. PhD thesis, Technische Universität Wien, 2007

[24] S. Rozsnyai, J. Schiefer, and A. Schatten. Concepts and models for typing events for event-based systems. In DEBS, pages 62_70. ACM, 2007.

[25] J. Schiefer, S. Rozsnyai, C. Rauscher, and G. Saurer. Eventdriven rules for sensing and responding to business situations. In DEBS, pages 198_205. ACM, 2007.

[26] M. Golfarelli, S. Rizzi, and I. Cella. Beyond data warehousing: what's next in business intelligence? In Il-Yeol Song and Karen C. Davis, editors, DOLAP, pages 1_6. ACM, 2004.

[27] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. Software Engineering, IEEE Transactions on, 27, 2001.

[28] G. Cugola, E. Di Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In ICSE, 1998.