Event Data Warehousing for Complex Event Processing

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

Josef Schiefer, Hannes Obweger, Szabolcs Rozsnyai
Senactive GmbH
Innstraße 1-7
3400 Klosterneuburg, Austria
{jschiefer, hobweger, srozsnyai}@senactive.at

Abstract — In the last few years, Complex Event Processing (CEP) has emerged as a new paradigm for event-driven applications. The research focus in this area has so far been primarily on operational issues and not on the ex-post analysis of event data. On the other side, approaches like data warehousing have proven useful in the past to extract further valuable information from the data available within an organization. In this paper, we elaborate the concept of a fully-implemented event data warehouse as an add-on for CEP that allows to efficiently archive and query valuable event data for later analysis. We outline the overall architecture and describe the relevant meta-models for an integrated data management approach. The data management itself is implemented using a RDBMS, and its schema is automatically synchronized with the CEP models. Finally, we present a real-world use case to illustrate the application of the event data warehouse in practice.

Complex Event Processing, Data Warehousing, Event Streams

I. INTRODUCTION

Complex Event Processing (CEP) provides a new paradigm to collect and process data flowing through an organization in (near) real time in order to automate and accelerate decision cycles while facilitating an agile approach to keep the underlying business logic adaptable to changing business requirements.

Until today, the main research focus in the field of CEP has been devoted towards operational issues, e.g. performance, query and rule implementations, event type models or event correlation [11]. The persistence of event data, even for purely operational tasks like recoverability or scalability, has not yet been studied in greater detail in any scientific materials that we are aware of. Nevertheless, the usage of business data for ex-post analytical processing has been studied in great detail in the context of data mining and data warehousing (DWH) [9].

In this paper, we propose a DWH approach to be used for data processed by CEP applications, i.e., event data. The advantages of such an approach are not limited to those of traditional data warehousing: Knowledge gained from event-data analysis can also help to adapt and improve the underlying CEP applications. The seamless integration between CEP and an event data warehouse system allows reusing event-type- and event-correlation information available in CEP solutions, which facilitates the time-consuming and expensive task to setup and change the data integration processes.

In this paper, we describe an approach to store event data processed by CEP applications efficiently in an event data repository for ex-post analysis. We refer to the repository as an Event Data Warehouse (EDWH) in the remainder of this paper. We extend the DWH definition of Bill Inmon [9] and define an EDWH as follows:

An Event Data Warehouse (EDWH) is a subject-oriented, integrated, time-variant and non-volatile collection of event data in support of operational or management's decision-making process. The EDWH stores data items which originate from a continuous stream of events and stores the following types of data items: 1) data items representing the original events in event streams, 2) data items capturing relationship information for event sequences resulting from event correlations, and 3) derived or calculated data items from events or event streams. An EDWH supports a query language for accessing all three types of data items.

In order to analytically process event data ex-post, it has to be permanently archived somewhere. The common solution for this problem is to make use of a data warehouse (DWH) approach where data is added periodically in a batch using an extract, transform and loading (ETL) process. We identified a list of requirements which existing CEP systems do not sufficiently address:

Analytical Processing. Existing CEP systems support flexible pattern-matching mechanisms; most of them SQL-based or rule-based. Although pattern matching can be quite complex, there is no sophisticated analytical data processing such as any kind of forecasting, classification, clustering, or association rule learning.

Data Aggregations. CEP systems allow to aggregate data of event streams within a certain time window. Nevertheless, this data aggregation is, in most cases, limited to pattern detection. There is no simple way of accessing generated aggregates or the current value of aggregates. Furthermore, it is difficult to have aggregate levels for data items, which is a prerequisite for many types of data analysis.

Processing Historical Data. CEP systems only have a temporary memory and capture historical event data only for a certain period of time (time windows). If a CEP system goes...
offline, all event data is gone. Many business applications require at least some persistent memory in order to maintain state for event stream processing.

**Traceability.** Existing CEP systems are not able to adequately maintain and track event processing steps, event correlations, calculated aggregates, etc. Many CEP system vendors offer debugging tools for finding out what is going on during the event processing; however, there is no way to ask the system for all generated downstream or upstream artifacts for a single event.

### II. RELATED WORK

**Related work** in this field can be split up into CEP and DWH approaches. Several different academic and commercial CEP implementations exist today, varying to a great extent in expressiveness of rules and queries, ease of use, performance, and the way they are integrated into the overall IT landscape of an organization. Many solutions use an SQL-based approach to query a stream of events, e.g., the open-source CEP engine Esper [7] or Aurora [3] as well as its successors Borealis [2] and Medusa [29]. The queried data itself is, nevertheless, transient and kept in memory only. Another important concept in CEP is the definition of rules, which is a focus of RuleCore [21], an event-driven rule processing engine based on Event-Condition-Action (ECA) rules. Correlation of events, i.e. to create (an ordered) set of related events from an incoming event stream is a second essential concept in CEP studied by Wu et al. [28] and Chen et al. [6].

AMIT [4] is another event stream engine that provides a mature user interface to model business situations with four types of entities: events, situations, lifespans and keys. The definition of an event is the base entity which can be related to other events and is comparable to the event type definition used in this paper. Lifespans are used to define time intervals between related events to describe certain situations. A situation is the main instance for specifying queries.

The importance of a solid event model and event typing is also crucial for event mining applications. Moen [12] [13] describes algorithms which use typed events to determine similarities between event objects in order to discover similar event patterns. An EDWH complements this approach by providing the data needed to feed such event-mining applications.

The second field related to the proposed solution is the domain of DBMS and traditional data warehouses. CEP can be seen as advancement from Active Databases [27]. While CEP employs an “inbound” approach to process the incoming stream of events, Active Databases rely on an “outbound” processing model. The difference, as pointed out by Stonebraker [22], is that while “outbound” processing requires all data to be stored beforehand to be potentially processed afterwards, “inbound” processing allows the data to be processed first and only optionally stored afterwards, resulting in a significant rise in performance. Another shortcoming of Active Databases is the lack of full-fledged programming support to execute the actions derived from evaluating a set of rules, usually in the form of database triggers.

Stonebraker [22] further argues that there is no “One Size Fits All” solution for data-centric applications and predicts that the commercial world will fracture into a collection of independent database engines. We tend to agree with Stonebraker that event stream processing within a single database system is no longer applicable. However, we also believe that separating database engines opens a completely new set of technical problems which have to be solved.

DWH and especially Active DWH [20] pursue a similar approach as we do in this paper. Active Data Warehouses combine active mechanisms based on ECA (Event-Condition-Action) rules with analysis capabilities of data warehouse solutions to extend (passive) systems with reactive capabilities. They are event-driven since data is continuously updated, providing a near real-time perspective on an organization.

Aside from sharing the same shortcomings as Active Databases, especially their “outbound” processing model, the biggest difference to an EDWH is their loosely coupled integration into an IT landscape, involving time-consuming activities to implement them from scratch or adapt them according to changing business needs.

### III. ARCHITECTURAL OVERVIEW

The EventBase architecture provides, in contrast to other traditional DWH approaches, the ability to store and maintain events in real time in a central repository, together with historical and analytical data. This type of persistent organization of events allows for seamless access on real-time and historical event data. Furthermore, the typical ETL steps of a DWH and analytical solutions are managed by the event processing model.

Figure 1 illustrates the SARI architecture. The bottom of the figure shows different source systems (i.e. the event producing components) continuously generate event notifications. A sense layer represents the adapters of SARI that can be docked to event-producing systems or the communication infrastructure. The adapters gather events in either a push or pull process and propagate them into the event processing realms.
The internal communication infrastructure uses an event bus for publishing the received events to the event processing models. SARI uses sockets as a generic interface for sending and receiving events to and from event processing models. The processing of event streams is performed in event processing maps, where the flow of events is modeled with various components according to the business requirements. For the event-data storage, SARI uses sockets for “forwarding” event data to the database. In other words, users can define for any type of socket, whether received events shall be stored in the database.

The EventBase extends SARI’s event processing model with an efficient up-to-date operational storage together with retrieval mechanisms for business events for analytical as well as operational purposes. The query language for retrieving near real-time events and creating conjunctions with historical events, metrics and scores is SARI-SQL, which, in contrast to Event Cloud’s indexing approach [12][16][26], is a structured query approach. SARI-SQL is tailored to satisfy the special requirements of analytical business users and meet the characteristics of events and their relationships.

The core access component of the EventBase is a query engine supporting SARI-SQL. Set on top of the EventBase data repository, it exposes its services through programming interfaces and a graphical user interface. SARI-SQL can be used by a wide range of frontend tools for accessing the data within the EventBase; this includes reporting, incident tracking, as well as data mining tools, which are used by analysts to examine the business environment based upon events processed by the event-based system. A detailed discussion of SARI-SQL is outside the scope of this paper. The interested reader may refer to Rozsnyai [14].

IV. EVENT STREAM META-MODEL

Before we discuss the data model and further data management issues, we present the relevant meta-models. In the following, we outline meta-models of the event-based system SARI. We will use the meta-models as a foundation for mapping and preparing event data to a data repository. We distinguish two types of meta-models: 1) meta-models for deriving the data model, and 2) meta-models for storing and preparing event-data in runtime.

A. Meta-Models for Deriving the Data Model

For generating the data model from a stream of events, the following information is required:

- Structural information of various types of events (including data types of attributes, nested event types, constraints for attributes)
- Information on how structural information is reused (inheritance, exheritance)
- Relationship information on how events are correlated with each other
Figure 2 shows the event and correlation meta-model [18] of the SARI system. The event meta-model includes information on various event types which capture the meta-data carrying the structural information of events. Please note that the event meta-model is simplified and includes only relevant information for the data mapping. For a detailed discussion of the event meta-model, please refer to Rozsnyai et al. [15]. An event type in SARI can have a set of attributes complying with an attribute type (e.g., String, Integer, etc.). SARI supports special event types such as virtual, dynamic, and duck types. Virtual event types define views on existing event types (similar as views for tables in the relational database world). Dynamic event types automatically derive the structural information from a set of existing event types. Duck types can be used for temporary deactivating the typing system (e.g., for event data staging purposes) and, at a later point in time, the appropriate event type for the unclassified event data can be inferred. For further details on the various event types, please refer to [15].

On the right side of Figure 2, you can see the correlation meta-model of the SARI system. SARI uses so-called correlation sets for modeling these event relationships between events. In other words, correlations between events are declaratively defined in a model which is used by the correlation engine during runtime. Correlation sets are able to define relationships based on matching key-value pairs of attributes among event types. Correlation sets can have a set of correlation bands which define a sequence of events that use the same matching approach for the event correlation. For instance, if we want to correlate order events with transportation events (of the same order), we might have one band defining the correlation between order events (e.g., OrderCreated, OrderShipped, OrderFulfilled) and a second band defining the correlation between transport events (e.g., TransportStart, TransportEnd). The correlation set is able to link multiple bands, thereby correlating the events of all its bands. SARI supports various types of correlation bands such as simple correlation (elementary matching), semantic matching correlation, self-referencing correlation, and bridges for correlation sets (for chaining multiple correlation sets).

Figure 3 gives an overview of the rule model of SARI. Rules have one or more rule triggers which can have other rule triggers as a precondition. Based on a set of preconditions, event actions can be triggered. In SARI, an event action can be the generation of a new event object, a calculation of a score or metric, or the data collection for an entity. All these actions generate data items (event objects, scores, metrics, entity data) which are stored in the database and linked to the triggering event(s).

B. Meta-Models for Event Data Staging

When storing event stream data to a database, we need an infrastructure for 1) mapping the event data to database tables, and 2) generating downstream artifacts from event data such as consolidated event data or performance indicators. SARI solely uses the meta-models for event types for the mapping of event data to database tables and correlation. Nevertheless, when integrating the event data during runtime, SARI uses an event processing model and a rule model for generating downstream artifacts based on discovered event patterns.

Figure 3
The rule model of Figure 3 needs to be interpreted by a rule engine. SARI uses so-called Event Processing Maps (EPMs) for defining event processing flows. Similar to a construction kit, an EPM offers various adapters and services for the event processing. Depending on the business requirements, these components can be flexibly conjoined or disconnected. For instance, EPMs can be used to clean or enrich event data before it is processed by a rule service. The processing of the rules is performed by a rule service, which is one of many components within an EPM.

Figure 4 shows the elements of the EPM which are relevant for storing event data in a database. An EPM consists of customizable components which can be combined in an event processing flow. A processing flow always starts with a sense socket and ends with a response socket. The sense and respond socket correspond to adapters for receiving or sending the event data. Sockets can also be used for connecting multiple EPMs or as a central distribution hub for events.

For the event data storage, SARI uses sockets for “forwarding” event data to the database. In other words, users can define for any type of socket, whether the received events of the socket should be stored in the database.

In summary, the EPMs form the data staging area of the CEP system. Rules are used to match event patterns and, subsequently, to generate downstream artifacts from the event pattern, such as scores, metrics, and entity information. Figure 4 only shows the relevant elements of EPMs for the data staging. SARI supports many other elements such as filters, modifiers, and synchronization blocks [19], whose discussion is out of the scope of this paper.

Some parts of the database schema, namely the tables for typed events, metrics and scores, are changed dynamically based on the definitions of the underlying CEP application. Figure 5 shows an exemplary ER diagram for a simple CEP application with three different event types A, A’ and B where event type A’ inherits its attributes from A. Furthermore, one metric, a score and an entity have been defined in the CEP application.

In the following, we describe the purpose of each table and discuss how they are used to fulfill the requirements of an EventBase.

**Events**, which are at the core of this schema, are stored in the main Events table. It contains the generic information about all the events stored in the EventBase and is used to link events part of the same correlation to metrics, scores and entities on which they had an influence. Apart from the event type identifier and the timestamp of the event’s occurrence, a serialized version of the event is stored, which can be used to reconstruct the event object.

For each event type, a separate table exists which splits up the event attributes of this event type into database columns as shown for event types A, A’ and B in Figure 5. These columns allow for efficient querying of event attributes by analytical applications. Each event type attribute, which has a runtime type supported by the underlying database, is directly mapped to a database column of a corresponding database type. Event attributes having unsupported runtime types are available only after deserializing the whole event from the serialized version in the Events table.

**V. RELATIONAL MAPPING OF EVENT DATA**

In this section, we describe the database schema used to map events, correlations between those events as well as associated metrics, scores and entities to a relational database which serves as an EDWH repository.

![Event Processing Map](image)

**Figure 4. SARI – Event Processing Map.**

![Entity Relation (ER) Model](image)

**Figure 5. Entity Relation (ER) Model.**

Events of an inheriting event type are only stored to the inheriting event type’s table, thus all event attributes are included in this table; see, for example, table Events A’ shown in Figure 5. Inheritance is then implemented through the use of database views, as is shown in Figure 6. A view is created for
each event type but, unlike the event type tables before, a view may contain events of different event types, i.e. event types inheriting from the same base event type. For example, Events A View contains events from the base event type A as well as from the event type inheriting from A, i.e., A'. Therefore, a query against the Events A View returns all the events sufficing event type A, hence also those events of the inheriting event type A'.

Indexes are individually created for those event attributes which might often be queried for analysis of the CEP application. A full-text search, as known from Web search service providers, is provided on-demand if supported by the underlying database. The serialized XML version of the event contains all the event attributes in textual format to serve as the full-text searched column. For a trade-off in performance, the same feature is available to search in a set of correlated events by appending the serialized XML versions of these events in the database [16].

During the lifetime of a CEP, applications may change; this includes the addition, removal or change of event type definitions. While adding and removing event type definitions as a whole can be handled by simply dropping the associated tables and views, changing an event type definition with related events already stored in the EventBase may be difficult. Dropping an event attribute can be treated the same way as dropping an event type, i.e., by either dropping the column or keeping the column and inserting NULL values for new events from now on. Which approach is correct solely depends on the CEP application at hand.

Correlations are persisted in the Correlation table with their ids, temporal information about the correlation and a foreign key to their associated correlation set. Correlations have an n-to-m relationship to events, which is established through the Events2Correlation table. Consolidated serialized XML versions of all events part of the same correlation can furthermore be included to provide the possibility for full-text searching over correlations.

Metrics and Scores values calculated from various (correlated) events are stored in tables created specifically for each metric definition similar to events. The difference between metrics and scores is that metrics are always (re-)calculated as a whole, whereas for scores, the current value is either increased or decreased. Nevertheless, on a database level they can be treated analogously and, therefore, only metrics will be described in greater detail in this section.

As described in previous sections, metric values are updated upon the arrival (or absence) of events after certain preconditions have been met within a rule. Metrics consist of an identifier, a value and a set of properties usually populated with certain attributes of an associated event and are stored in two different tables. The metric table, e.g. Metric A depicted in Figure 5, contains the current metric value for a certain set of properties. The metric history table (e.g. MetricHistory A) contains the history of all metric values and is associated with the event from the events table which triggered the calculation of this metric, resulting in a specific metric value. Each time a metric calculation is triggered by an event, the new record is added to the metric history table, and an update is executed against a metric table containing the current metric values for the given metric properties.

The event-driven calculation of metrics is a common task executed by CEP applications to detect certain situations of interest, and almost all decisions made at the rule-processing end of these applications rely on this information. Most of the time, it will therefore be of great interest for ex-post analytical processing, and the preservation of this information proves to be a useful add-on for an EDWH and spares us to recalculate this information again later on. The archiving of this information may have an impact on performance; nevertheless, an ex-post calculation may require storing many other, generally irrelevant events just to come up with correct numbers afterwards.

Entities. As practice showed, metric properties usually refer to the real-world entities, like customers, regions or products. Entity tables are used to manage and enrich the information about an entity. This information can later on be used to find related metrics via common properties, or to find related entities using social network analysis methods. Furthermore, entities are linked to those events which helped to enrich the entity in any way.

VI. SCALABLE EVENT DATA STAGING

The data staging process for an EDWH includes multiple phases of preparing the event data for analytical purposes. One major challenge for continuously integrating event data is that the integration process cannot take for granted that it has free reign to drop tables, re-load tables and conduct other major database operations without disturbing simultaneous end-user queries. In order to work around this challenge, data that changes throughout the day can be loaded into a parallel set of tables, either through a batch process that runs every few
minutes or through a continuous trickle feed. Once the load interval (e.g., five minutes) is up, the freshly loaded tables are simply swapped into production and the tables with the now stale data are released from production [10]. This can be accomplished through the dynamic updating of views, by simple renaming tables or by swapping partitions.

The downside of this type of n-minute cycle-loading process is that the data in the EventBase is not truly real time. For applications where true real-time data is required, the best approach is to continuously trickle-feed the changing data from the source system directly into the EventBase. Obviously, near real-time data integration cannot be as fast as bulk load integration. Therefore, it is necessary to carefully identify critical data, which needs to be integrated continuously with minimized latency.

In the following, an approach is introduced for integrating trickle-feed event data with event trace capture and loading (ETCL). A key difference of ETCL to traditional bulk loading of data warehouse systems can be summarized as follows:

**Trace Files instead of Flat Files.** Traditional ETL tools use flat files where records of the same type are collected and periodically loaded into the database. ETCL uses event traces, which collect in memory the records resulting from the processing of a single event. Thereby, various types of records (e.g., records for event data, metrics, scores, etc.) are created. Finally, the event trace is applied to the database.

**Size of Data Chunks.** The data loads for ETCL handle smaller data sets and is performed after the completing the processing of an event. The size of the data to be loaded into the database depends on the event processing results.

**Loading Data from Memory.** During the event processing, the SARI system collects event trace data in memory. When loading the data into the EventBase, the event trace data is directly transferred to the database without using intermediate storage (e.g., a flat-file).

**Consistent Event Data.** A trace file keeps a record of the result from event processing tasks and thereby maintains the temporal order the created records. When bulk-loading data, the records are split into flat files which are individually loaded into the database. During the data loading, simultaneous user queries must be turned off in order to avoid locked database records and inconsistent query results. ETCL solves the problem by continuously applying consistent fragments of the event data trace during the loading process thereby keeping the stored event data consistent.

When capturing data for the event trace, the event data is processed in multiple stages. Each stage produces its own set of records, which are added to the trace. On the higher level, we distinguish the following types of records: 1) the attribute data for events, which contain the elementary information of event objects, 2) information on relationships between events, and 3) metrics and scores which are calculated during the data staging. The records are continuously created during the event processing and finally loaded in chunks into the EventBase database. Figure 7 illustrates this process.

The SARI system allows the user to configure which artifacts, i.e., which parts of the event trace, shall be stored to the EventBase. This configuration is called event trace outline. Note that there are several dependencies between the various trace levels: For instance, tracing correlations of events (by activating the trace for correlation sets) requires that the correlated events are also captured in the trace. Otherwise, the event trace contains incomplete records when writing the trace to the database.

**A. Methods for Storing Event Traces**

In the following, we discuss multiple methods for loading the records of event traces into a database system.

**Transactional Inserts.** The simplest way of loading the data into the database is to generate and execute INSERT SQL statements. By using normal INSERT SQL statements, the processing is fully transactional, similar to OLTP applications. Since data is only added to the EventBase, a low isolation level can be used for the transactions. A major shortcoming of this approach is a high number of transactions and database round trips for storing the event trace records in the EventBase.

**Transactional Batch Processing.** The processing of the transactional INSERT statements can be optimized by processing them in a batch mode, i.e., by sending multiple INSERT commands to the database in a batch. Another approach is to use stored procedures, which can consume the event trace information as input and perform inserts on database level. Using batch processing for the data inserts can significantly improve the performance. However, one shortcoming of this approach is that the database system still has the overhead for processing each record with a separate SQL statement.

**Transactional In Memory Bulk Loading.** This approach combines the transactional batch processing with bulk loading capabilities. Instead of loading the data from flat files, records from the event trace are directly loaded from memory into the database.
database system. However, many of today’s database systems do not support large in-memory data-loads directly. In order to circumvent this problem, mechanisms can be used to load data from in-memory documents: For instance, Microsoft’s SQL Server or Oracle database systems allow preparing XML documents in memory, which can then be used to efficiently query data items for inserting them in one step into the database. The data inserts for a single XML document is performed in a single transaction.

**Using Transactional Logs for Event Traces.** In this approach, a database-specific event trace format is used for storing record information. When applying the event trace to the database system, the trace can be applied directly by the database system, thereby minimizing the processing and time for the data integration. The key advantage for this approach is that data is inserted in a highly efficiently manner. A major shortcoming of this approach, however, is that transactional logs for database system are proprietary and vary from system to system.

**B. Performance Experiments with ETCL**

Using an event processing application from the fraud domain [23], we conducted an experiment for storing the event processing results with event tracing. We ran SARI, using the transactional batch mode, on two nodes with four-CPU Intel servers. We used a separate machine with the same technical specification as the database server, running Microsoft SQL Server 2008. The following table summarizes the performance results.

<table>
<thead>
<tr>
<th>TABLE I. EVENT DATA STAGING WITH ETCL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td>Number of source events: 168,232 events</td>
</tr>
<tr>
<td>Average number of attributes per event: 18 event attributes</td>
</tr>
<tr>
<td>Number of event types: 14 event types</td>
</tr>
<tr>
<td>Event tracing mode: Transactional batch</td>
</tr>
<tr>
<td>Time to process data: 4 minutes, 40 seconds</td>
</tr>
<tr>
<td>Time to store data: 4 minutes, 40 seconds</td>
</tr>
<tr>
<td>Time to store data: 4 minutes, 40 seconds</td>
</tr>
</tbody>
</table>

We were able to store the events with the full event trace at a rate of about 600 events per second. Please note that the processing of an event potentially generates new events (e.g. alert events) which are also stored as part of the event trace.

We conducted another experiment using in memory bulk loading. By using this storage mode with SQL Server 2008, we created XML documents which containing the full event trace information and sent these XML documents to a stored procedure for inserting the data. With this storage mode, we were able to store 424 events per second. The lower throughput is caused by the significant amount of XML processing which increases the CPU utilization on the SARI nodes as well as on the database node.

Finally, we did an experiment with the transactional insert storage mode, which allowed us to store 127 events per second. The significantly lower throughput was caused by the higher number of database activity due to more database roundtrips.

**VII. EVENT-DRIVEN BUSINESS INTELLIGENCE**

As yet, we presented the concept of EDWH and showed how events, correlations and other artifacts are persisted in the EventBase. In this section, we focus on the analysis part of SARI: Event-driven BI means the creation of knowledge about underlying business environments from historic event data. As such knowledge can be used to adapt and further improve real-time event-processing logic, applied event-driven BI is essential for up-to-date and continuously refining event-based systems.

In the following, we demonstrate event-driven BI through a real-world use-case from the fraud detection domain. Fraud detection and prevention is a major issue across technology-driven business domains relying on online payment solutions and customer interactions.

Suntinger et al. [23] showed that fraud analysis fits particularly well with the event-driven approach: Fraud analysis requires root-cause and cause-chain analysis on the level of single user-actions both in near real time (for the in-time prevention of ongoing fraud) and posteriori (for the continuous improvement of the knowledge base). Event-based systems use a rule-based approach to continuously evaluate the stream of customer interactions. For the expert-driven analysis of past customer data, relevant events are stored in the EventBase.

As an exemplary analysis framework, we utilize the EventAnalyzer [24]. The EventAnalyzer is the first grown-to-maturity BI tool built upon the EventBase. It offers a range of visualization opportunities tailored to the characteristics of event data. The key visualization techniques are derived from the Event Tunnel metaphor of seeing past events flowing through a 3D cylinder and providing different viewing on it. Figure 8 above shows a screenshot of the analysis framework. A detailed description of the depicted panels will be given throughout the below example.

In the following, we assume that a well-established online-betting provider uses a SARI-based fraud-detection system to continuously monitor ongoing customer interactions. We furthermore assume that the system automatically generates alerts for users that match known fraud patterns.

After receiving two temporally close fraud alerts, the business analyst decides to further investigate the affected user accounts. In the analysis framework, the analyst formulates a query selecting the corresponding account-history correlation-sessions from the Event Base.
In the EventAnalyzer, the definition of queries is facilitated by the query builder (Figure 8a): Here, analysts can select those event types that are considered relevant from the overall list of event types defined in the given event-model; the same is provided for correlation sets. If required for their investigations, analysts can define more sophisticated clauses, as, for instance, on event attribute values.

The event-tunnel top-view in Figure 8c shows the results of the above query: Single events are plotted as glyphs, with user-defined mappings (Figure 8b) from event attributes to color, shape and size. Correlations between events are plotted as colored bands, connecting the correlated events by their times of occurrence. As depicting similar-looking glyphs at similar points in time, the plot in Figure 8c unveils that the suspicious users’ account histories correlate strongly over time. Also, from the glyphs’ coloring, the analyst detects that both users failed to place a “free throw” bet, as they were recognized as officials by the system.

From visualizations of the EventBase, business analysts gain deep insight into the various business processes persisted therein. Query-driven and tailored for a quick and step-wise navigation through the EventBase, analytical applications such as the Event Analyzer push it one step further: From an initial clue such as the above fraud alarms, they allow the analyst detecting previously unconsidered facts and relationships.

Consider the business analyst formulating a new query on the event-data warehouse now selecting all events from the BetPlaced-event table that are a), “free throw” bets, and b), occurred recently. The corresponding event-tunnel side-view in Figure 8d scatter-plots the result set based on the events’ times of occurrence (x-axis) and the events’ bet amount (y-axis). It is easy to see that soon after the detected fraud attempts (mapped to blue based on their account IDs), an outlier showing a significantly higher bet amount occurs (mapped to red as exceeding a threshold). For an experienced analyst, this data point represents a valuable link to related and possibly conspicuous data. One possible interpretation could be that the outlier was placed by a so-called putter-on, a fraudster that places bets for people who are prohibited to bet. The suspicion is confirmed by the fraudster’s “account balance” metric as available in the EventBase and depicted over time in Figure 8e: The step chart shows that the fraudster’s account is widely inactive, with sequences of cash-ins, placing a bet, winning a bet and cash-outs occurring strictly every few days.

The example shows that with tailored analysis tools such as the EventAnalyzer, the EventBase serves as a valuable and easy to access source for knowledge discovery. Thus far, the visual analysis of event data was proven useful; adoptions of well-known data-mining techniques such as similarity searching are, however, in current development. For more information on the EventAnalyzer, the Event Tunnel and its application in fraud detection, readers may refer to Suntinger et al. [24].

VIII. CONCLUSION AND OUTLOOK

In this paper we have addressed several open issues regarding event persistence and analysis of existing CEP solutions. In particular, we have presented a solution for efficient event data repository management as an extension for CEP solutions in order to enable post analysis of events and break up current event processing limitations.

The presented solution is based on a formal and efficient data schema maintained by a common RDBMS. It solves common problems to efficiently persist events, their relationships, the preservation of calculated event metrics and the representation of non-native database types including
advanced concepts such as event inheritance. The implementation and the evaluation have been conducted upon the existing event processing solution SARI.

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. Furthermore, we plan to evaluate the integration of the Event Data Warehouse (EDW), respectively the linking of EventBase entities to external DWH dimensions to bundle dynamic real-time event information with large historical information repositories for better situation detection and verification of decision making processes.

ACKNOWLEDGEMENT

We want to thank the Senactive development team for their valuable discussions and for implementing this research work.

REFERENCES


