Anomaly Management using Complex Event Processing

Extending Data Base Technology Paper

Bastian Hoßbach, Bernhard Seeger
Department of Mathematics and Computer Science
University of Marburg, Germany
{bhossbach,seeger}@mathematik.uni-marburg.de

ABSTRACT

During the last decade, complex event processing (CEP) has emerged as a technological foundation for many time-critical monitoring applications. CEP is powerful, effective, easy to use and low in costs at the same time. Common CEP applications are for example stock-market analysis, detection of fraudulent credit card use, traffic monitoring and consumption forecasting in power grids. Many application domains are still hard to target by CEP, because state of the art CEP technology is characterized by a static behavior and by a signature-based detection paradigm. In this paper, we motivate substantial improvements of CEP technology by making the behavior of the infrastructure dynamic and by switching the detection paradigm from signatures to anomalies. This leads to multiple changes in the infrastructure that raise interesting and challenging research questions. The resulting dynamic CEP infrastructure not only makes existing applications more powerful and easier to maintain but also enables novel application domains.

1. INTRODUCTION

Huge or complex systems and processes require monitoring to run smoothly. The most suitable monitoring is continuous and in real-time. For this reason, *complex event processing* (CEP) has been developed and implemented in many application domains during the last decade. Today, CEP is used as back-end in monitoring solutions that e.g. analyze stock markets, search for fraudulent use of credit cards or mobile phones and perform forecasting in traffic networks or power grids [14, 20, 22, 24, 27].

One of the most important tasks of CEP is the detection of risks and chances, summarized under the term *situations of interest* (SOI). This kind of use can be found in nearly every CEP application in practice. Because the detection is done in near real-time, it is possible to react immediately in order to minimize the imminent damage of risks or to maximize the arising benefit of chances. Typically, SOI are specified in the form of signatures (*blacklist approach*). On the one hand, this strategy is expensive (each single SOI has to be specified as a signature), potentially incomplete (there might exist unknown or unexpected SOI without an active signature) and hard to maintain. On the other hand, this method detects every SOI for that a signature exists and produces no false positives. A well-known example of signature-based systems is antivirus software.

A different way of detecting SOI is searching for anomalies. Anomalies are significant differences from a normal state and behind each anomaly a risk or a chance could be concealed (*whitelist approach*) [7]. This strategy can produce false positives, but is inexpensive, complete (every anomaly is detected) and easy to maintain. Additionally and in comparison to the signature-based strategy, the detection of anomalies discovers unknown SOI automatically.

The state of the art CEP technology follows the signature-based approach. Users define one or more continuous queries (in terms of signatures) for each SOI and execute them in a CEP infrastructure that searches for all specified SOI. In some important application domains, e.g. information security monitoring, the anomaly-based approach will become more important than the signature-based approach [12]. Therefore, CEP infrastructures should also support the anomaly-based approach besides the signature-based one.

In this overview paper, we study limitations of current CEP technology in supporting anomaly management. We discuss possible solutions for each limitation and present some initial ideas. Overall, we propose the extension and improvement of the CEP infrastructure. The proposed dynamic CEP infrastructure is able to capture the normal behavior of monitored objects and to create all necessary continuous queries for the detection of anomalies automatically. Additionally, the infrastructure is adaptive to changes in the application context at runtime. This dramatically reduces the effort of building and maintaining CEP applications and ensures a high quality at any time. The user has still the opportunity to define queries and can so extend the monitoring logic. This results in the best of both worlds: an automatically created anomaly-based monitoring setup that can be extended by the user with little effort.

In Section 2, we present state of the art CEP technology. We introduce important application domains that require anomaly-based, adaptive and reactive monitoring tools in Section 3. In Section 4, we discuss limitations of the current CEP infrastructure in supporting these monitoring tools and propose specific improvements. We conclude the main results of the paper and outline ongoing work in Section 5.

EDBT 2013, March 18–22, 2013, Genoa, Italy.

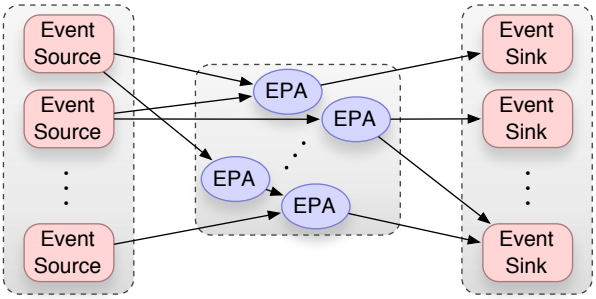

Figure 1: General CEP Infrastructure

2. CEP TECHNOLOGY

The current state of CEP technology has been developed and established mainly in the area of stream processing during the last decade. *Stream processing engines* (SPE) are used in addition to traditional database systems to support the analysis of streaming data in near real-time. The main aim of CEP is the detection of SOI in streaming data rather than manipulating data streams [26]. Besides tailor-made CEP systems (e.g. Cayuga [6], ZStream [18]) that only support the needed functionality, general-purpose SPE (e.g. Esper [11], Odysseus [2]) can also be used to implement CEP applications, because they are extensible or powerful enough to support all necessary CEP functions.

All available CEP systems and SPE provide the same infrastructure to build CEP applications. So it is possible to define state of the art CEP technology based on a general and abstract infrastructure. This CEP infrastructure, shown in Figure 1, is completely independent of specific systems.

The general CEP infrastructure consists of three main components: a set of event sources, a set of *event processing agents* (EPA) and a set of event sinks. EPA are continuous queries that perform typical CEP tasks like filtering, aggregating and correlating events or searching for event patterns. They are connected to event sources or other EPA and event sinks are connected to EPA. The implementation of a CEP application comprises three steps: First, each event source is registered at the system. Then, EPA are defined on the registered event sources. Finally, event sinks are registered at the system and connected to the created EPA to consume their results. Once a CEP application has been deployed and started, no changes in the sets of event sources and sinks are expected and the behavior of each event source is also expected to be stable. So there is no reason why the monitoring logic (that is the set of EPA) should be dynamic. In fact, adding and removing of event sources, EPA and event sinks are supported by most CEP systems and SPE, but all changes have to be done manually by the user. For the best of our knowledge, there is no system that supports arbitrary updates of the definitions of EPA at runtime. This is why we call this kind of CEP technology static.

In practice, almost all CEP applications, which run for months or years, are context-sensitive. This point gets more important in an anomaly-based perspective. For example, the normal behavior of a temperature sensor changes from one season to another and the total network traffic of internet service providers at weekends differs significantly from the one during the week. In nearly every application domain it is impossible to define a static normal behavior. Instead, the normal behavior of an object changes over time [23] and depends on other objects. The realization of anomaly-based monitoring tools with the current CEP infrastructure is a difficult task because of its static structure and behavior.

3. MOTIVATING EXAMPLES

In the following, we review some traditional application domains of CEP and show how better capabilities for the management of anomalies can improve existing applications. After this, we describe novel application domains from the field of information security that can only be addressed adequately by anomaly-based CEP technology.

Fraud Detection: Systems for the detection of fraud are needed for example in the areas of credit cards and mobile phones. The aim is to protect customers in case of stolen credit cards or mobile phones against financial damage. Because a fraudulent use differs from a non fraudulent one, real-time anomaly detection is an ideal monitoring concept. A challenging problem in realizing such systems is that individual customers have different normal behaviors that can vary over time. Another domain, where fraud detection is necessary, are social online games that are free to play and financed by advertising. There is a need in the detection of bots (players controlled by software), because they lead to costs (e.g. servers and network traffic) but produce no income by consuming the advertising. Bots can be detected based on their behavior that differs from the behavior of human players.

Healthcare: Critically ill patients are often separated in intensive care units, where multiple sensors on the body of the patient measure important vital signs like heart rate, temperature and blood pressure. The continuous measurements are visualized for healthcare professionals on screens. Recently, the utilization of the data streams has been increased by adding a CEP system for continuous and automated analysis, especially in the absence of medical staff [15]. It is very important that every change in the physiological state is detected and reported to healthcare professionals in a timely fashion. Anomaly detection is the best suitable approach for the monitoring.

JIT-Logistics: In *just-in-time* (JIT) logistics goods are delivered at the very moment when they are needed. For example, the car manufacture Volkswagen in Germany gets roofs for cabriolets delivered by Webasto from Portugal just-in-time [5]. Typically, JIT-logistics are implemented in supply chains like in the previous example. In JIT-logistics there is no need for stores. This significantly reduces costs, but every difference in the logistic processes can cause problems very fast in a depending manufacture (e.g. no cabriolet can be completed without a roof). In the worst case, the whole production comes to a standstill. This results in huge financial damage. So the real-time monitoring of all involved transporters of a logistic process (e.g. ships, trucks, airplanes) is very important. Anomaly management is the right monitoring paradigm, because every difference in processes of JIT-logistics is usually sub-optimal and can cause economic loss.

SLA Monitoring: To guarantee customers, for example of a cloud web service, some quality of the service, *service level agreements* (SLA) are used. SLA have to be monitored in real-time, because violations normally results in contractual penalties for the provider. The correct execution of a service can be easily defined as the compliance with all associated SLA. Therefore, anomaly-based monitoring tools are more suitable than signature-based ones again.

Stock Market Analysis: Participants and players in the stock markets are highly interested in information about SOI as fast as possible, because they commonly arise unpredictably and suddenly. Continuous and automated anomaly management is a solid foundation for the exploration of SOI at stock markets.

A novel and challenging application domain of CEP is the management of security anomalies in computer systems. We discuss this domain in more detail, because it is an ongoing project in our research group. This is the reason why we have identified limitations in the application of current CEP technology in this domain. The goal of our *ACCEPT* project [1] is the development of novel security monitoring tools for computer systems. Because such tools have to analyze enormous amounts of streaming data in a timely manner, the use of CEP for the analysis is advisable. In information security the detection of anomalies instead of predefined signatures has become important today and will be critical in the future. For example, in the last quarter of 2011 more than 33 % of all web malware was zero-day malware [8]. Zero-day means that the malware infected numerous computer systems before signatures and patches could have been developed and distributed. The ratio of zero-day malware is growing. So today's signature-based monitoring systems will become less effective and important [12].

But malware is not the only reason why tools like *ACCEPT* are required in information security. We want to master the detection of every security anomaly. This could also be an intruder or a SQL injection. Intruders are hard to detect, especially when they use valid (stolen) credentials to get access to a computer system or network. In such cases, the only possibility to detect intruders is based on their behavior. For example, regular users would never delete logfiles, install root-kits or scan the whole network for open ports. SQL injections are another good example and one of the most dangerous security risks nowadays (number one security risk in the CWE/SANS Top 25 [10] in 2011). The normal behavior of an (web) application that interacts with a database can be defined by a set of SQL patterns that describe all normal SQL queries the application usually sends to the database or by simple metadata (e.g. tables and attributes that are queried). SQL injections will not match the patterns and metadata (e.g. the database catalog is queried) and this is how to detect them. Future security monitoring tools should not need to be configured by thousands of signatures. Instead, they should learn the normal behavior of monitored objects (e.g. applications, users, processes) by themselves and detect suspicious behavior automatically.

All examples have in common that there exist many objects of the same type (e.g customers, patients, processes, trucks, SLA, stocks) with individual normal behaviors that can vary over time.

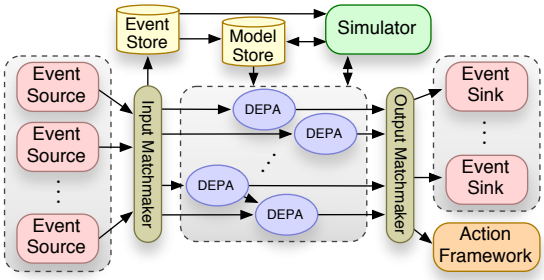

Figure 2: Proposed Dynamic CEP Infrastructure

4. ISSUES AND CHALLENGES

In this section, we discuss all limitations of the general CEP infrastructure we already have identified during the design phase of *ACCEPT*. The most limiting issues in supporting anomaly management are its inflexibility and static behavior. This makes it hard to create applications (everything has to be defined manually like connections and EPA) as well as to maintain their lifecycles. In fact, applications are not intended to have a lifecycle because of the static infrastructure. To target this issue, we propose the extension of the infrastructure by a more flexible event flow management instead of hard-wired connections and to allow EPA to change their definitions at runtime. Additional changes concern improvements in the adaptive and reactive capabilities. Lastly, we bring earlier proposals for a standardization of stream processing back into use. The final dynamic CEP infrastructure that we propose is shown in Figure 2 and discussed in the following.

4.1 Matchmakers

In order to achieve a more flexible event flow, all static connections are replaced by components that decide for each incoming event individually to which consumers it should be forwarded to. We call these components *matchmakers*. For each side (input side and output side) there is exactly one matchmaker needed.

The *input matchmaker* establishes connections between event sources and EPA. Each event source and each EPA has exactly one connection to the input matchmaker. EPA are now defined on the basis of content rather than on the basis of sources (this concerns e.g. the from-clause in SQL-based event processing languages). The task of the input matchmaker is to forward each incoming event to all EPA that are interested in some information contained in the event. The input matchmaker also identifies differences in the schemas (data types, names, metrics, ...) between an incoming event and an interested EPA and solves the conflict through a transformation of the event before forwarding it (analogous to data fusion [19] in data warehouses or schema evolutions [9]). This allows event sources to send different types of information without having a fixed schema and to send incomplete information.

The *output matchmaker* is similar to the input matchmaker. Each EPA and each event sink has exactly one connection to the output matchmaker and the definition of connections by hand becomes superfluous again. The task of the output matchmaker is to take each result from the EPA and to forward it to all interested event sinks. Like EPA, event

sinks are now defined on the basis of content instead of specific EPA. Transformation techniques are also an integrated part of the output matchmaker.

Because efficient filtering and transforming are the main tasks of the matchmakers, it might be possible to realize them completely or in parts by EPA again. This approach would benefit from internal optimization techniques. It also raises some interesting research questions (declarative languages for the configuration of matchmakers and procedures for automatic creation of the EPA are required).

4.2 Dynamic EPA

Once an EPA has been defined and started in current systems, the only actions that can be taken afterwards are stopping and restarting it. In dynamic environments that are the rule rather than the exception every change in the context (this concerns not only the time dimension but also all other dimensions of the application as for instance the location or the states of objects) affects the behavior of the EPA immediately [23]. As a consequence, a CEP applications should be able to adapt to changes in its context. A natural way to target this issue is to allow the definitions of EPA to be modified at runtime. Therefore, we propose updatable EPA and call them *dynamic event processing agents* (DEPA). Algorithms have to be developed that on the one hand are able to make the new definition of an EPA available as fast as possible and that on the other hand preserve clear and deterministic semantics of the monitoring logic.

4.3 Event Store

The *event store* is a highly optimized and append-only database for recording all incoming events. It has the best possible write performance. At least on the time dimension of stored event streams an index is required in order to support fast temporal point and range queries. In the best case, events are stored direct in multi-version indexes that not only support the time dimension but also all other dimensions of event streams. Because event streams are potentially unbounded, old events are aggregated or deleted after some lifetime. The main purpose of the event store is to hold the history of all incoming event streams. This data is needed by the model store, the simulator and the action framework of the dynamic CEP infrastructure.

4.4 Model Store

Anomaly detection is the search for abnormalities that can be only defined if there exists a specification of the usual. For this reason, the event store is analyzed in order to derive models that describe the normal behavior of monitored objects. Suitable models can be simple statistics (e.g. averages or histograms) as well as more complex models (e.g. Bayesian networks, support vector machines or hidden Markov models). The *model store* is responsible for storing and managing such models. The selection and production of models should be done automatically rather than by the user. In order to achieve this goal, challenging and comprehensive tasks have to be mastered. First of all, the whole event store has to be analyzed to figure out what models can be learned from specific stored events. For example, individual objects and spatial data have to be detected. This step can use methods from the areas of knowledge discovery and data mining. Second, instances of all possible model types have to be produced for each event stream. In most cases, multiple instances of the same model type have to be learned, because model types (especially complex ones) are parameterized and each parameter has to be guessed. Finally, the produced models have to be evaluated (test data is taken from the event store again). Only accurate models are added to the model store. Because the normal behavior of monitored objects can change, the creation of models is no one-time task. Instead, models are produced continuously. Inaccurate models in the model store are updated or removed and new models are added.

The CEP system of the infrastructure consumes the stored models. For each model there are running DEPA that monitor the normal behavior and report all anomalies. The creation of these DEPA is done automatically. For each model type there have to be multiple DEPA with free variables predefined. Then, each instance of a model type creates its predefined DEPA and replaces all free variables by constant values. DEPA are necessary instead of EPA, because the normal behavior is usually context-sensitive. Every time a stored model is updated or it predicts a change in the normal behavior, all associated DEPA are adapted (that is, former free variables are replaced by new constant values). For example, a simple predefined DEPA that can monitor time series data is given by:

```
SELECT * FROM [value:Number IN Celsius]
WHERE  value < normal_value - tolerance
OR     value > normal_value + tolerance;
```

The DEPA above is defined on events that contain a numeric attribute named "value" that is measured in the unit of celsius temperature. It also contains two free variables that specify the expected value of incoming measurements (*normal_value*) and a distance (*tolerance*) by that they are allowed to differ from the expected value before they are classified as anomalous. The incoming measurements could be produced by temperature sensors for example. Based on the historical data in the event store, a model that specifies the expected temperature for different periods of time (e.g. 25 during the day and 12 at night) could have been learned. It is also possible to compute the variance of the temperature (to set the tolerance) in all time periods. This model is now used to maintain the free variables.

The power of DEPA is not limited to thresholds. Other parts of DEPA are also updatable at runtime (e.g. event patterns, aggregates, predicates or the size of time windows). Besides the challenges in building the model store and managing all active models, we do not believe that existing model types can be easily reused. Because we want the model types to be both generally applicable (usually results in many model parameters) and automatically produceable (only possible in the presence of very few model parameters), they have to be tailor-made to fulfill all requirements. This raises new challenges in machine learning.

4.5 Simulator

The *simulator* is used for executing DEPA inside a secure and isolated environment. Real data is taken from the event store to perform the execution. This component is needed for several reasons. First of all, the simulator can be used during the design phase of DEPA to test and debug them. Especially defining parameters like event patterns, thresholds and sizes of time windows is complicated in most cases and users have to try several configurations until they find a well-working setting. The second task of the simulator is

supporting what-if analyses. Every time a CEP application has not detected a SOI, the SOI in question can be replayed from the event store in order to find the missing DEPA that is able to detect it. Additionally, the simulator is needed after changes on one or more DEPA to ensure that they are still able to find all SOI (that is, the changes have not decreased the quality of the monitoring logic).

The third function of the simulator is continuous and automated meta-monitoring of the CEP application itself. Each DEPA is defined within the scope of a specific context that can change. It is important to detect every change in these contexts, because the definition of associated DEPA might become obsolete. Therefore, every time a new or updated DEPA gets active, it is executed inside the simulator on the freshest data in the event store. Afterwards, metadata that describe the behavior of the DEPA with respect to the actual context (e.g. produced results per time period) are computed and stored in the model store. During the execution of the CEP application, the simulator periodically takes a copy of every active DEPA, executes it on the freshest data from the event store and recomputes the metadata. After this, the recomputed metadata are compared to the stored ones. If the deviation between the different versions of metadata is significant, the context might have changed. In cases of a context switch, the system can inform the user or try to solve the conflict automatically (e.g. changing the size of time windows if the output size has become much bigger/smaller). The most interesting questions are: What metadata are suitable to describe the behavior of an arbitrary DEPA with respect to its context? Is it possible to define strategies to solve conflicts automatically?

4.6 Action Framework

Real-time detection of SOI without real-time reaction is a toothless tiger. In some cases, only immediate reactions to the occurrence of a SOI can minimize damage or maximize benefit and the opportunity for optimal reactions lapses quickly. In other cases, there are reactions needed very frequently. The current CEP infrastructure simply forwards the results from EPA to event sinks. At this point its responsibility ends. Nearly all CEP applications we studied in practice use messaging (e.g. E-Mail, SMS or pager) or logging as event sinks. This contradicts the real-time requirements of many CEP applications.

We propose for future CEP infrastructures reactive capabilities. One solution could be an *action framework* besides traditional event sinks on the output side. Inside the action framework, actions are defined for specific SOI. Every time an action gets informed about a SOI it is responsible for, it triggers predefined reactions (e.g. cancel a credit card, ban a player, buy/sell a stock or stop/restart a process). The action framework poses interesting questions: how to prevent detect-react-cycles? This means that a SOI triggers one or more reactions that change the context, the changed context produces new SOI that trigger reactions again and so on (in the worst case, the CEP application collapses). Additionally, it should not be possible to start actions that are in conflict with other actions or that are harmful. Besides techniques from the active databases community (e.g. event-condition-action triggers), which have currently been successfully adopted by CEP [25], methods of program verification and model checking are also needed. Whenever an application is allowed to make decisions by itself, it is important to be able to reproduce afterwards why the application has made a certain decision. For this reason, the event store is required to replay periods in history. Additionally, the semantics of all (D)EPA and actions have to be deterministic. Provenance is an upcoming and interesting topic in the stream processing community and indispensable for reactive CEP applications.

4.7 Quality

The following topic concerns not only CEP but also stream processing in general. In traditional database applications a lot of work has been done in managing data quality. There are textbooks (e.g. [17]), best practices, tools and techniques (e.g. entity-relationship models or normal forms) to measure, preserve and increase data quality. In stream processing applications the roles of queries and data are reversed in comparison to database applications [13]. Continuous queries represent the most important asset of stream processing applications. Therefore, we propose the management of their quality with the same effort as quality of data is managed in database applications. Most quality dimensions for data [21] can be reused unchanged for continuous queries: the set of all running queries should be e.g. complete, free of error, up-to-date and value-adding. We would like to see more work that helps users to create stream processing applications with such attributes. This issue opens an interesting field of research. In a broader sense, it also includes automatic creation of queries based on the desired output [3]. In the best case, the complete and minimal set of required queries is created automatically.

4.8 Standardization

Another significant difference between databases and SPE is the degree of standardization. Database systems are highly standardized. The syntax and semantics of the query language SQL are defined in an international standard and all SQL database systems support this standard. Additionally, the API of database systems can be highly abstracted. This results in interfaces (e.g. ODBC/JDBC) that provide database functionality completely independent of specific database products. Each professional database system can be used and easily exchanged behind these interfaces.

In the area of stream processing the reverse is true. Nearly every SPE has its own query language. The different languages vary in syntax, semantics and expressiveness [4, 16]. Once a specific SPE has been integrated in an application, it is very hard and expensive to replace it by another. The API and internal data models differ so strongly that the application code has to be completely rewritten. Besides the technical difficulties, all continuous queries have to be entirely redefined because of differences in syntax and especially in semantics of the query languages. So the exchange of SPE products not only is expensive but also introduces errors with high probability. We propose the introduction of standards for both the query language and the API.

Besides the benefits of standards for the success of commercial products, also the research community could profit from. Only in the presence of standards benchmarking and testing is absolutely objective and comparable. The research on new systems, concepts, models and algorithms would become easier to evaluate and proof. Lastly, standards can be the missing piece of great success stories (just recall the important meaning of SQL for database systems).

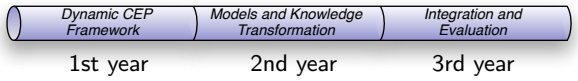

Figure 3: Schedule of the ACCEPT Project

5. CONCLUSION AND FUTURE WORK

Motivated by challenging and future monitoring applications, we have suggested an extended and improved *complex event processing* (CEP) infrastructure that is able to fulfill all their requirements. The proposed draft is structured into different sections. In each section interesting research questions are revealed and for some of them we have presented initial ideas.

During the next years, we will work on a subset of the presented research problems. Figure 3 shows the planned schedule of our *ACCEPT* project. In the first year, we want to develop an abstract event processing framework that run on top of different *stream processing engines* (SPE). The framework serves a dual purpose. First, it abstracts from the specific query language and API of SPE so that we can use different SPE beneath and easily exchange them. Second, the framework extends the API by a method to update the definitions of arbitrary continuous queries at runtime in a fast and safe way. We prefer to implement an abstract framework, because implementing yet another SPE would not provide any additional value. After the first year, we plan to have a running prototype that allows dynamically changing its monitoring logic. In the second year, we will integrate techniques from knowledge discovery, data mining and machine learning. The system will derive and update models that describe the normal behavior with respect to the context (e.g. time) automatically. Furthermore, the creation and modification of continuous queries will be triggered by models from a model store. Therefore, we have to define mappings between specific model types and continuous queries. In the last year, we will build tools for information security monitoring using our dynamic CEP infrastructure. Integrated in these tools, we will be able to evaluate them and the new kind of CEP infrastructure simultaneously.

6. ACKNOWLEDGMENTS

This work has been supported by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF) under grant no. 01BY1206A.

7. REFERENCES

[1] ACCEPT. `http://www.accept-projekt.de/`

[2] H. Appelrath et al. Odysseus: a highly customizable framework for creating efficient event stream management systems. In *DEBS*, pages 367–368, 2012.

[3] C. Binnig, D. Kossmann and E. Lo. Towards automatic test database generation. In *IEEE Data Engineering Bulletin*, 31(1), pages 28–35, 2008.

[4] I. Botan et al. SECRET: a model for analysis of the execution semantics of stream processing systems. In *PVLDB*, 3(1), pages 232–243, 2010.

[5] G. Brar and G. Saini. Milk run logistics: literature review and directions. In *WCE*, pages 797–801, 2011.

[6] L. Brenna et al. Cayuga: a high-performance event processing engine. In *SIGMOD*, pages 1100–1102, 2007.

[7] V. Chandola, A. Banerjee and V. Kumar. Anomaly detection: a survey. In *ACM Computing Surveys*, 41(3), pages 15:1–15:58, 2009.

[8] Cisco Systems Inc. Cisco 4Q11 global threat report. Technical report, 2012.

[9] C. Curino, H. Moon, A. Deutsch and C. Zaniolo. Update rewriting and integrity constraint maintenance in a schema evolution support system: PRISM++. In *PVLDB*, 4(2), pages 117–128, 2010.

[10] CWE/SANS Top 25. `http://cwe.mitre.org/top25/`

[11] Esper. `http://esper.codehaus.org/`

[12] Gartner Inc. Effective security monitoring requires context. Technical report G00227893, 2012.

[13] L. Golab and M. Özsu. Data stream management. Morgan & Claypool Publishers, 2010.

[14] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez and P. Valduriez. StreamCloud: a large scale data streaming system. In *ICDCS*, pages 126–137, 2010.

[15] H. Han, H. Ryoo and H. Patrick. An infrastructure of stream data mining, fusion and management for monitored patients. In *CBMS*, pages 461–468, 2006.

[16] N. Jain et al. Towards a streaming SQL standard. In *PVLDB*, 1(2), pages 1379–1390, 2008.

[17] Y. Lee, L. Pipino, J. Funk and R. Wang. Journey to data quality. The MIT Press, 2006.

[18] Y. Mei and S. Madden. ZStream: a cost-based query processor for adaptively detecting composite events. In *SIGMOD*, pages 193–206, 2009.

[19] F. Naumann, A. Bilke, J. Bleiholder, M. Weis. Data fusion in three steps: resolving inconsistencies at schema-, tuple-, and value-level. In *IEEE Data Engineering Bulletin*, 29(2), pages 21–31, 2006.

[20] K. Patroumpas and T. Sellis. Event processing and real-time monitoring over streaming traffic data. In *W2GIS*, pages 116–133, 2012.

[21] L. Pipino, Y. Lee and R. Wang. Data quality assessment. In *Communications of the ACM*, 45(4), pages 211–218, 2002.

[22] N. Schultz-Møller, M. Migliavacca and P. Pietzuch. Distributed complex event processing with query rewriting. In *DEBS*, pages 4:1–4:12, 2009.

[23] Y. Tan, X. Gu and H. Wang. Adaptive system anomaly prediction for large-scale hosting infrastructures. In *PODC*, pages 173–182, 2010.

[24] K. Teymourian, M. Rohde and A. Paschke. Knowledge-based processing of complex stock market events. In *EDBT*, pages 594–597, 2012.

[25] D. Wang, E. Rundensteiner, R. Ellison. Active complex event processing: applications in real-time health care. In *PVLDB*, 3(2), pages 1545–1548, 2010.

[26] L. Woods, J. Teubner and G. Alonso. Complex event detection at wire speed with FPGAs. In *PVLDB*, 3(1), pages 660–669, 2010.

[27] Q. Zhou, Y. Simmhan and V. Prasanna. Towards an inexact semantic complex event processing framework. In *DEBS*, pages 401–402, 2011.