Processing Big Events with Showers and Streams

Christoph Doblander[1,2], Tilmann Rabl[1,3], Hans-Arno Jacobsen[1]

[1] Middleware Systems Research Group
arno.jacobsen@msrg.org
[2] TU München, Germany
doblande@in.tum.de
[3] University of Toronto, Canada
tilmann.rabl@utoronto.ca

Abstract. Emerging use cases derived from the area of cloud computing, smart power grids, and business process management require a set of capabilities not met by traditional event processing systems. These use cases were chosen to illustrate the capabilities required from systems that are able to process what we refer to as *Big Events*, that is Big Data in motion. To further illustrate *Big Events*, we identify three use cases and analyze the characteristics of the events involved. Based on this analysis, we specify requirements regarding the event schema, event query language, historic event processing needs, event timing, and result accuracy. Collectively, we refer to the constellation of state changes in a given system that exhibits these characteristics as *event showers*, referring to the collective of these events, similar to the notion of an event stream in the context of event stream processing. We call systems that offer capabilities for meeting these requirements *event shower processing systems* in contrast to traditional *event (stream) processing systems*. The use cases we picked, demonstrate that additional value can be captured by having shower processing systems in place. The benefits lie in the new possibilities to gain additional insights, increase observability, and to further exert control and opportunities for optimizations in the given applications.

1 Introduction

As storage prices continue to drop, more and more data is stored for subsequent analysis. This trend has recently been coined as the era of *Big Data* [27]. As more and more data can be stored, the value of data analysis increases, since ever more patterns can be mined and data that previously was not of interest can be monetized. However, still many data sources are unused since their value perishes quickly. This kind of fast-paced data that needs to be processed in real-time or near-real time is often called *Big Events* [17] and refers to the processing of *Big Data* in motion.

The possibility to process *Big Events* by either exposing events from large systems or by sensing events from many sources needs a powerful processing system. We call systems that can process *Big Events*, *(event) shower processing* *systems*. Events represent state transitions in the environment, conveyed as *event messages* to the system. The terms event messages and events in this context are typically used interchangeably.

More formally, an *event shower* is a partially ordered set of events, either bounded or unbounded, where the partial orderings are imposed by the causal, timing, and other relationships between the events. Others have referred to similar notions as *event clouds* [25, 26], which due to the affinity in terminology to cloud computing, we would like to avoid to reuse here. Informally speaking, an *event shower* represents the constellation of events over time resulting from considering the collective of events originating from disparate event sources in a distributed system.

Events can be sensed from the environment or can be exposed by existing systems and applications. As we show in our use case analysis, valuable information can be derived from correlating and analyzing event showers. Since our society becomes more dependent on technology and systems become more complex, observability is a critical requirement. Creating interfaces between systems requires a lot of specification and testing. In case of a malfunction, reproducing errors is a hard problem since the interactions internal to a system can not be easily reproduced or since the unique conditions leading up to the failure only occur once in a while.

Without doubt, debugging functionalities in software development tools increase observability for programmers, while event exposure increases observability in complex systems [41]. When event showers originating from multiple interacting systems can be analyzed, these systems become more observable and transparent. In case of a malfunction in an observable system, it may be possible to find ways to recover, if a well-behaved state is reached [12].

The difference between *event stream processing systems* and *event shower processing systems* are defined by the characteristics of the events involved. In stream processing systems, events tend to originate from one to a few data sources, while in shower processing systems, events originate from many data sources. Consequently, for event showers, it is impossible to accurately synchronize time across the publishing data sources. Therefore, logical clocks or other mechanism are required to establish some form of event ordering. In stream processing, events are often implicitly timestamped, relative to the single source they are emitted from or relative to the stream processor that received them in a given order, often arrival order.

In stream processing systems, the stream schema is known a priori, in *shower processing systems*, event schemas are subject to change and may not be known a priori, because the systems exposing events are not within the organizational control of the system which correlates and aggregates the events. Accordingly, *shower processing systems* have to be able to deal with schema-less information. More of the differences are discussed in Section 3.

To this end, systems are needed, which can analyze massive amounts of events from multiple systems and can deal with the characteristics of these events. With shower processing systems, it is possible to discover emergent behavior, mine for patterns, observe the interactions between disparate systems and, thus, increase the overall observability, while this is less of a concern for stream processing.

In this paper, we analyze the characteristics of events derived from considering three emerging use cases. Based on our analysis, we formulated the requirements for systems, which can deal with these kind of events, which we refer to as event shower processing system. We show how the different elements (i.e., query language, publish/subscribe semantics, and consistency requirements) fit together and outline future research required in this area in order to establish event shower processing systems. Throughout the text, we point to literature that describes systems exhibiting some of the requirements we postulate.

The rest of the paper is organized as follows. In Section 2, we describe three different use cases: cloud computing, smart power grids, and business process management. In Section 3, we define the required feature set for an event shower processing system, discuss how it differs from existing approaches, and present the individual elements of such a system. Finally, in Section 4, we discuss how event shower processing systems can be beneficial in the presented use cases.

2 Use Cases

We identified three emerging use cases where additional insights can be gained by analyzing and correlating events from multiple systems. The domains were chosen because of recent interest from the research community and where affordable sensors could bring increased observability.

Figure 1 shows an exemplary overview of a system which processes event showers. Multiple systems expose events or sense events from the environment. Operators formulate queries which compile to pattern matching rules, content-based routing topologies, aggregations and correlations. The topology is optimized for low latency or maximum throughput.

2.1 Cloud Computing

With cloud computing, storage and computing resources get commoditized. Cloud providers offer on-demand configurable computing resources [28].

Monitoring plays an important role when making systems running on cloud computing platforms resilient (e.g., [39]). This helps to understand how systems operate. In case of failure, monitoring helps with the root-cause analysis or to discover potential weaknesses. As abstraction layers are added to the software stack, one looses observability because typically performance problems are understood at the very lowest layer of the stack [10].

Shower processing systems make it possible to correlate the data streams of the cloud environment with other event streams. While system-level events are well defined in terms of schema, application specific events can change frequently. A potential use case is to correlate exposed events from business processes and performance indicators of cloud environments [29]. With this information it is,

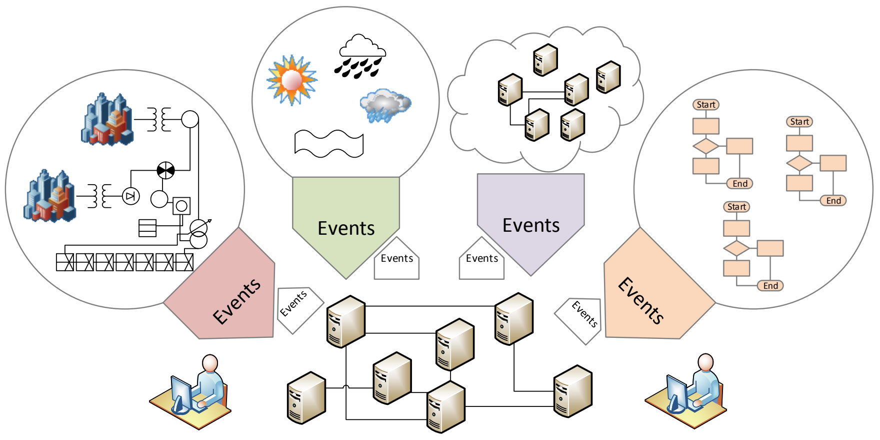

Fig. 1. Event shower processing system

for example, possible to optimize the latency of a specific business process or to forecast the impact on the infrastructure when a specific business process gets executed more often.

2.2 Smart Power Grids

The smart grid is the next evolution of the electrical power grid by enabling bidirectional communication and control of energy generators and consumers [3]. Energy demand and generation has to be exactly balanced. Until recently, the demand was given and then matched with the generation. As the portion of fluctuating energy sources, like solar and wind increases, the demand must increasingly be matched to the generation.

Observability is a key ingredient to control and balance production and demand in the smart grid [9]. This can be seen in the following example. The total production in Germany at 12:00 o'clock on June 20th, 2013, a sunny weekday, was 70 GW. 18 GW or 26% of the total production were generated by solar cells [1]. The expected solar production, published the previous day was only 13 GW, resulting in a prediction error of 5 GW. Based on the previous day's prediction, power plants are scheduled. This results in monetary losses due to severe over-provisioning. Furthermore, the gap between production and demand had to be compensated by lowering the output of conventional generators. Even more expensive is the inverse situation, where the expected solar production is not met. In this case, the higher demand has to be compensated through the spot market or more costly alternatives, with short-term regulation energy. While shifting demand is already done by huge power consumers, such as cold storages, there is a huge potential in controlling large numbers of electric vehicles and smaller devices like heat pumps or fridges [16,31,33]. Recent approaches also show how consumption forecasting could be done more accurate [44].

To control the huge number of individual devices, the current state must be observable to estimate the potential effect of control. This requires massive sensing infrastructures and near real-time processing of event showers. Since the events may have different kinds of latencies caused by changing networking conditions, the system has to deal with missing information and respect those in the overall state estimation. An area where low latency is required is, for example, phasor measurements. Phasor measurement units include GPS clocks, which provide an external time stamp for potentially correlating events, subject to the achievable GPS clock accuracy [20]. As the cost of GPS clocks and phasor measurement units decreases, it also becomes affordable to install them in low voltage grids, to pro-actively take actions in developing situations [30].

Event shower processing can add significant value to this scenario. Events from weather stations, buildings, generators, shiftable demand and energy storages can be correlated and aggregated and control can further be optimized.

2.3 Business Process Management

Exposing events from business processes can provide valuable insights if combined and correlated with external data. Instead of mining existing log files one can think of automatically exposing events via the business process execution engine [41] or to leverage a process execution engine, already designed and implemented through a publish/subscribe approach [24], thus, naturally exposing events.

Business process mining has been shown to be applicable to real-world scenarios [32]. An extended scenario could be the correlation of events from business processes with click-stream events from Web shops and weather data. An exemplary scenario is as follows: Historic click-stream events show that people tend to do more online shopping on rainy days [4]. Analysis of the events exposed from a business process engine could show that these sales have higher return rates. Thus, returns increase and customers are not satisfied. Hence, additional personnel resources are needed to deal with the returns. This shows that correlating weather data and historic events could help provision personnel accordingly.

Exposing events can also benefit white box testing in SOA environments [41]. This shows that increased observability can be used also for testing. More generally speaking, event exposure can be though of as an approach to expose unstructured information over system boundaries to enable the above described scenarios.

3 Definitions

The main characteristics of the events in the presented use cases above are the following: The events are exposed implicitly, which makes it difficult to define an event schema. They cross organizational boundaries or systems, which makes it difficult to standardize and prescribe a given event schema. Also, events may be exposed from proprietary and legacy systems, so changing the events is not easily possible. Furthermore, events from inexpensive sensors may lack exact timing information.

It is difficult to support the above use cases with existing event (stream) processing systems. While existing event processing systems exhibit capabilities to handle some of these event characteristics, event shower processing systems are representatively covering all requirements (see Table 1).

	Showers	Streams
Schema	Optional	Defined
Boundaries	Distributed across multiple systems	Part of a system
Routing	Implicit publish/subscribe semantics	-
Historic	Historic and current events	Only current events
Query language	Declarative	Can be declarative
Timing	External or logical clocks	Ordered by system arrival
Consistency	Eventual consistent	Consistent

Table 1. Event showers vs. event streams

Event stream processing and complex event processing: Event stream processing systems typically do not cross organizational or system boundaries. If those boundaries are crossed, typically the event schema is specified, e.g., in financial markets. Complex event processing can combine multiple data sources but the correlation and aggregation of the events is done within a single system. An exemplary software which can be used in such an environment would be IBM Infosphere [2].

A system capable of processing event showers can distribute the aggregation and correlation of events across multiple systems and can consider infrastructure concerns to optimize the topology. This could be done by leveraging existing publish/subscribe-style event processing and overlay networks. [23, 43]

Rule-based systems: In rule-based systems it is possible to derive deductions [11]. Event shower processing systems take this approach one step further by enabling deductions on multiple event streams by supporting a declarative logic-based query language.

Publish/Subscribe systems: Publish/Subscribe systems consist of publishers which produce events, subscribers which register for events, and brokers which route the events through an overlay network [14]. New event sources are advertised by a broadcast message. The advertisement contains schema and additional information regarding event shape and timing.

Event shower processing systems compile queries to aggregations and correlations, which are essentially join operations [22, 23]. The operations are compiled

```
:-type(production(timestamp:number(integer), household:string(varchar)),
                  watt:number(integer))).
:-type(household(hhid:string(varchar), connectedTo:string(varchar))).
:-type(solarcell(hhid:string(varchar), kwpeak:number(integer))).
:-type(windturbine(hhid:string(varchar), kwpeak:number(integer),
                   diameter:number(integer))).
:-type(transformer(trid:string(varchar), a:string(varchar),
                   b:string(varchar))).
```

Listing 1. Type definition Datalog

to subscriptions that attract publications as intermediate results and pass matching publications on throughout the system to higher-level subscriptions. Event shower processing systems create these implicit subscriptions to events expressed in the query language and spread the correlation and aggregation throughout the topology [22, 23].

3.1 Event Schema

An *event schema* is a formal definition of the structure of data. Events can be observed or are automatically exposed by systems or databases [40, 41]. When events are exposed implicitly, the schema of these events can change, if new features are introduced in the underlying system. Hence, an event shower processing system must be able to map unstructured, semi-structured and structured data to a schema. An adaptor can map unstructured data to structured data, see Figure 2 for an illustration. Existing approaches, which have been designed to deal with semi-structured data are NoSQL databases [34]. It is also possible to infer types based on discovered schemas. That is *type providers* [37] offer type safe access inside a statically typed programming language. Listing 1 shows some types from the smart grid domain. With type providers the corresponding types and adapters could be generated automatically. This could be done based on an advertisement, which contains type information or by discovering the schema of events.

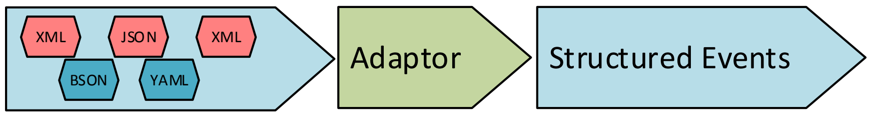

Fig. 2. Schema adaptor

3.2 Historic Event Data And Databases

In a system, which can process event showers, there is no difference between current events and historic event data, see Figure 3. This is an important feature serving the discovery and correlation between event streams or to train machine learning models. Accessing historic event data can be implemented as a feature of publish/subscribe systems [18, 21] or as part of a hybrid event processing architecture, such as MADES [38], which is a distributed event store that can query historic event data in the same way as process current and future events.

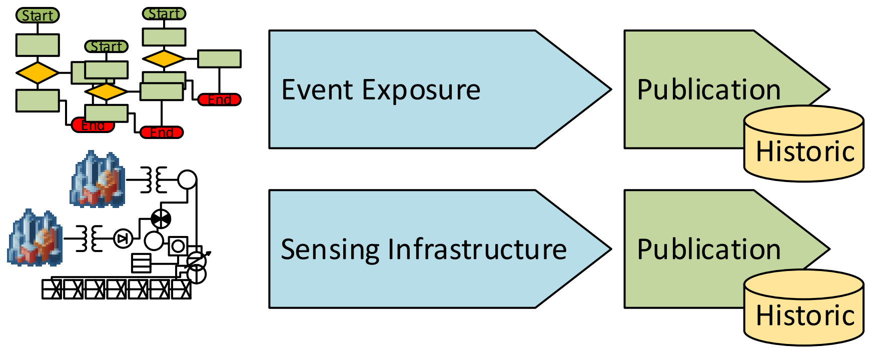

Fig. 3. Event sources

The NoSQL database CouchDB [7] can expose notifications when the underlying data changes. Relational databases have the possibility to expose events by triggers. The notifications could be further exposed as events. This point of view challenges the implementation of the query processor, which should be capable to process historic data in a batch-like fashion and also incorporate current events.

Recent work [42] shows how the map-reduce model for batch processing can be combined with event stream processing. Consequently, event shower processing systems can be seen as the next evolutionary step of big data systems.

3.3 Query Language

To discover knowledge in event streams a powerful language is needed. Datalog [11] can be viewed as a subset of general logic programs. It also supports recursion which has advantages when querying graph structures, e.g., social networks or electrical grid topologies. Datalog queries are guaranteed to terminate and can be run safely [35]. It has also been demonstrated [8] that complex events can be derived from simpler events by means of deductive rules. To use Datalog

```
\% transformers, transactional data
mediumhighvoltagegrid, distributiongridnorth).
transformer(munichsouth, mediumhighvoltagegrid,
distributiongridsouth).  transformer(munich, highvoltagegrid,
mediumhighvoltagegrid).

\% households, transactional data
household(hh2, distributiongridsouth).
solarcell(hh2, 12).
windturbine(hh2, 23, 7).
...

\% current production, eventstreams with unix timestamp
production(1375688745, hh1, 15).
production(1375688745, hh2, 18).
production(1375688746, hh3, 4).
...
```

Listing 2. Sample data and events

for event processing some extensions are needed, for example, to reason about temporal relations [5]. Also deductions and inductions have to respect temporal semantics [6]. It has already been shown that Datalog can be executed in parallel [15] and, therefore, Datalog processing can also utilize massive parallel hardware like Graphic Processing Units (GPU) or Field Programmable Gate Arrays (FPGAs). Historic data and exemplary events are shown in Listing 2. Listing 3 shows how this data is queried and aggregated. The output of the query refreshes continuously as new data becomes available or static data is updated.

3.4 Timing

Preserving ordering in a distributed system is a challenging task since clocks can not be accurately synchronized. Ordering can be preserved by logical clocks which need coordination. In case of large distributed event showers this is not practicable. Corbett et al. [13] show how to use GPS clocks to preserve ordering in a truly global distributed database. Depending on the consistency requirements approximately synchronizing to within a Δ_{max} may suffice [19]. Other approaches use heartbeat signals in streams [36]. Systems which can process event showers consequently rely on logical clocks, GPS clocks or must be aware of the synchronization error.

3.5 Accuracy

Sensors may be deployed in the field with unreliable network connections or inaccurate readings. Also, shower processing systems can span multiple organizations domains, operating world-wide, and, thus, must be self-aware of the latency they introduce. The immediate output of the event shower system may still not be

```
all_renewables_kw(HH, KW) :-
  windturbine(HH, KW, _).

all_renewables_kw(HH, KW) :-
  solarcell(HH, KW).

nearest_transformer(HH, TR) :-
  household(HH, GRID),
  transformer(TR, _, GRID).

sum_renewables_kw(SumSolar) :-
  sum(all_renewables_kw(HH, KW), KW, SumSolar).

sum_renewables(HH, TR, SumSolar) :-
  nearest_transformer(HH, TR),
  sum(all_renewables_kw(HH, KW), KW, SumSolar).

sum_renewables(TR, SumRenewable) :-
  sum(sum_renewables(_, TR, ToSum), ToSum, SumRenewable).

current_production(TR, Prod) :-
  nearest_transformer(HH, TR),
  production(TS, HH, Prod).

sum_production(TR, SumProd) :-
  sum(current_production(TR, Prod), Prod, SumProd).
```

Listing 3. Queries in Datalog

accurate and as additional events arrive at the system, the result becomes more accurate. For example, if the maximum latency of incoming events from one event stream was more than a second, the result stream must be delayed at least a second to produce an accurate result.

4 Conclusions

We showed that there are emerging use cases in cloud environments, smart grids and business process management where current state of the art event processing systems are unable to cope. The events may be implicitly exposed from legacy system, business process engines or are sensed from the environment with cheap sensors and high latency network connections. Analyzing and correlating these kind of events needs additional capabilities in the query language and the underlying system, dealing, among others, with schemaless events, timing and accuracy.

We outlined possibilities to overcome those hurdles when dealing with that kind of events. First, by using a higher level logic-based query language which abstracts from publish/subscribe. Second, by adding adapters to be able to deal with unstructured events in a type-safe way. Third, by including historic data and databases which can be used to train machine learning classifiers and to discover correlations. Finally, by adding the possibility to adapt accuracy by delaying the result or by supporting logical clocks or GPS clocks.

References

1. EEX Transparency Platform (Jun 2013), `http://transparency.eex.com`
2. IBM InfoSphere (Jun 2013), `http://www.ibm.com/software/data/infosphere/`
3. NIST Smart Grid Definition (Jun 2013), `http://www.nist.gov/smartgrid/beginnersguide.cfm`
4. Online spending soars as shoppers stay warm in wintry conditions (Jun 2013), `http://www.thisismoney.co.uk/money/markets/article-2305003`
5. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM 26 (Nov 1983)
6. Alvaro, P., Marczak, W.R., Conway, N., Hellerstein, J.M., Maier, D., Sears, R.: Dedalus: datalog in time and space. In: Proceedings of the First International Conference on Datalog Reloaded. Datalog'10, Springer-Verlag (2011)
7. Anderson, J.C., Lehnardt, J., Slater, N.: CouchDB: The Definitive Guide. O'Reilly Media, Inc., 1st edn. (2010)
8. Anicic, D., Fodor, P., Stojanovic, N., Stühmer, R.: Computing complex events in an event-driven and logic-based approach. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems. DEBS, ACM, New York, NY, USA (2009)
9. Bayegan, M.: A Vision of the Future Grid. Power Engineering Review, IEEE 21(12), 10–12 (2001)
10. Cantrill, B.: Hidden in Plain Sight. ACM Queue 4(1) (Feb 2006)
11. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Transactions on Knowledge and Data Engineering 1(1) (1989)
12. Chen, C., Ye, C., Jacobsen, H.A.: Hybrid context inconsistency resolution for context-aware services. In: IEEE International Conference on Pervasive Computing and Communications (PerCom) (2011)
13. Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., Woodford, D.: Spanner: Google's globally-distributed database. In: Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (2012)
14. Fidler, E., Jacobsen, H.A., Li, G., Mankovski, S.: The PADRES distributed publish/subscribe system. In: 8th International Conference on Feature Interactions in Telecommunications and Software Systems (2005)
15. Ganguly, S., Silberschatz, A., Tsur, S.: A framework for the parallel processing of Datalog queries. In: Proceedings of the ACM SIGMOD international conference on Management of data. ACM, New York, NY, USA (1990)
16. Goebel, C., Callaway, D.: Using ICT-Controlled Plug-in Electric Vehicles to Supply Grid Regulation in California at Different Renewable Integration Levels. IEEE Transactions on Smart Grid (2013)
17. Jacobsen, H.A.: Big events. In: Third International Workshop on Big Data Benchmarking (2013)
18. Jacobsen, H.A., Muthusamy, V., Li, G.: The PADRES Event Processing Network: Uniform Querying of Past and Future Events. it - Information Technology 51(5), 250–260 (2009)
19. Jerzak, Z., Fach, R., Fetzer, C.: Adaptive Internal Clock Synchronization. In: SRDS '08. IEEE Symposium on Reliable Distributed Systems (2008)

20. Li, F., Qiao, W., Sun, H., Wan, H., Wang, J., Xia, Y., Xu, Z., Zhang, P.: Smart transmission grid: Vision and framework. IEEE Transactions on Smart Grid 1(2), 168–177 (2010)
21. Li, G., Cheung, A., Hou, S., Hu, S., Muthusamy, V., Sherafat, R., Wun, A., Jacobsen, H.A., Manovski, S.: Historic data access in publish/subscribe. In: Proceedings of the DEBS 2007. ACM, New York, NY, USA (2007)
22. Li, G., Jacobsen, H.A.: Composite subscriptions in content-based publish/subscribe systems. In: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware. Middleware, Springer-Verlag New York, Inc. (2005)
23. Li, G., Muthusamy, V., Jacobsen, H.A.: Adaptive content-based routing in general overlay topologies. In: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware. Middleware, Springer-Verlag New York, Inc. (2008)
24. Li, G., Muthusamy, V., Jacobsen, H.A.: A distributed service-oriented architecture for business process execution. ACM Trans. Web 4(1) (Jan 2010)
25. Luckham, D., Schulte, R.: Event Processing Glossary - Version 2.0 (Jun 2013), `http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/`
26. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
27. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.H.: Big Data: The next frontier for innovation, competition, and productivity (2011), `http://www.mckinsey.com/mgi/publications/big_data/pdfs/MGI_big_data_full_report.pdf`
28. Mell, P., Grance, T.: NIST Cloud Computing Definition. Tech. rep. (Jul 2009), `http://www.csrc.nist.gov/groups/SNS/cloud-computing/`
29. Muthusamy, V., Jacobsen, H.A.: BPM in Cloud Architectures: Business Process Management with SLAs and Events. In: Lecture Notes in Computer Science Business Process Management, vol. 6336. Springer Berlin Heidelberg (2010)
30. Novosel, D., Madani, V., Bhargava, B., Vu, K., Cole, J.: Dawn of the grid synchronization. IEEE Power and Energy Magazine 6(1) (2008)
31. V. del Razo, C. Goebel, H.A.J.: Benchmarking a car-originated-signal approach for real-time electric vehicle charging control. Tech. rep. (2013)
32. Reijers, H.A., Weijters, A.J.M.M., Dongen, B.F.V., Medeiros, A.K.A.D., Song, M., Verbeek, H.M.W.: Business process mining: An industrial application. Information Systems 32(5) (2007)
33. Rivera, J., Wolfrum, P., Hirche, S., Goebel, C., Jacobsen, H.A.: Alternating direction method of multipliers for decentralized electric vehicle charging control. In: Proceedings of the IEEE CDC (2013, In press)
34. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional (2012)
35. Sagiv, Y., Vardi, M.Y.: Safety of datalog queries over infinite databases. In: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. ACM (1989)
36. Srivastava, U., Widom, J.: Flexible time management in data stream systems. In: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM (2004)
37. Syme, D., Battocchi, K., Takeda, K., Malayeri, D., Petricek, T.: Themes in information-rich functional programming for internet-scale data sources. In: Proceedings of the 2013 workshop on Data driven functional programming. ACM (2013)

38. Tilmann Rabl, Mohammad Sadoghi, K.Z., Jacobsen, H.A.: DEBS '13: Poster: MADES - A Multi-Layered, Adaptive, Distributed Event Store. ACM (2013)
39. Tseitlin, A.: The Antifragile Organization. ACM Queue (2013)
40. Ye, C., Jacobsen, H.A.: Whitening SOA Testing Via Event Exposure. Software Engineering, IEEE Transactions on (2013)
41. Ye, C., Jacobsen, H.A.: Event Exposure for Web Services: A Grey-Box Approach to Compose and Evolve Web Services. In: The Smart Internet. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2010)
42. Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I.: Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In: Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing. pp. 10–10. HotCloud'12 (2012)
43. Zhang, K., Sadoghi, M., Muthusamy, V., Jacobsen, H.A.: Multicast group membership management in high speed wide area networks. In: 33rd International Conference on Distributed Computing Systems (2013)
44. Ziekow, H., Doblander, C., Goebel, C., Jacobsen, H.A.: Forecasting Household Electricity Demand with Complex Event Processing: Insights from a Prototypical Solution. In: Proceedings of the 14th International Middleware Conference. Middleware (2013)