

Would Microsoft Azure Stream Analytics Be a Suitable Foundation for an Event Processing Network Model?

Arne Koschel¹ ^a, Anna Pakosch¹ ^b, Christin Schulze¹,
Irina Astrova², Christian Gerner¹ and Matthias Tyca¹

¹*Hochschule Hannover, DataH, University of Applied Sciences and Arts, Hannover, Germany*

²*Department of Software Science, School of IT, Tallinn University of Technology, Tallinn, Estonia*

Keywords: Event Processing Network (EPN), Event Processing Network Model, Azure Stream Analytics.

Abstract: This article looks at a proposed list of generalized requirements for a unified modelling of event processing networks (EPNs) and its application to Microsoft Azure Stream Analytics. It enhances our previous work in this area, in which we recently analyzed Apache Storm, Amazon Kinesis Data Analytics and earlier also the EPiA model, the BEMN model, and the RuleCore model. Our proposed EPN requirements look at both: The logical model of EPNs and the concrete technical implementation of them. Therefore, our article provides requirements for EPN models based on attributes derived from event processing in general as well as existing models. Moreover, as its core contribution our article applies those requirements by an in depth analysis of Microsoft Azure Stream Analytics as a concrete implementation foundation of an EPN model.

1 INTRODUCTION

Intelligent data management and processing has changed: Collecting large amounts of data from various sources happens in every company today, often called 'Big Data'. It is not longer sufficient to store data in relational DBs, log files or events separately. Information from data combined from different sources is important for the competitiveness of enterprises.

Batch Processing is an established approach for processing 'Big Data'. At its core data is collected and processed 'in batches' (Shaikh, 2019). Therefore, data is collected for a certain period of time before being processed. The drawback is that no real-time processing is possible. First, data is collected for some time, before processing takes place.

Recently, (Event) Stream Processing joined the field (Shaikh, 2019). An approach to process data directly after generation. Through this near real-time processing, an action can happen immediately after processing. Enterprises can react faster to changes.


For the implementation of Stream Processing a modeling technique called (complex) Event processing network (EPN) found its way into practice. This


approach gives a guideline, included components and also requirements, how such a Stream Processing should occur.

Along with the rise of Stream Processing tools were developed to model and implement EPNs. Therefore, we contribute here an evaluation of different recent tools, which support EPN realization as automatically as possible. The evaluation also addresses the following questions: What are EPNs and which requirements are important to provide them? A detailed look is taken at Apache Storm, Amazon Kinesis Data Analytics and – in this paper – Microsoft Azure Stream Analytics.

The following article enhances our work from (Schulze et al., 2023) and (Koschel et al., 2023) and provides the following contributions: First, we briefly look at our – compared to our work from (Koschel et al., 2017) – more formally structured list of generalized EPN model requirements (as shown in (Schulze et al., 2023) in detail). Second, we provide – as the key contribution of the present article – an in-depth evaluation of Microsoft Azure Stream Analytics with respect to our EPN model requirements. Moreover, we also briefly compare Microsoft Azure Stream Analytics to Apache Storm and Amazon Kinesis Data (AKD) Analytics.

The remainder of this paper is structured as follows: After discussing related work in Section 2 we

^a  <https://orcid.org/0000-0001-5695-2893>

^b  <https://orcid.org/0009-0001-6867-4488>

place a brief introduction to the topic and provide our EPN model requirements in Section 3. Next, we take an in-depth look at Microsoft Azure Stream Analytics in Section 4 followed by a brief comparison to Apache Storm and Amazon Kinesis Data (AKD) Analytics in Section 5. Finally, Section 6 summarizes our results and leads to a conclusion.

2 RELATED WORK

The basis of our project builds on authors in the scope of EPN and Complex Event Processing (CEP) for example, Dunkel and Bruns (Dunkel & Bruns, 2015) and (Bruns & Dunkel, 2010). We also used foundations from our earlier work on EPNs, namely (Koschel et al., 2017), (Koschel et al., 2018) and (Astrova et al., 2019). There we more informally establish the requirements for EPNs and apply them to different EPN modeling approaches and tools (EPiA, BEMN, RuleCore). With the present paper we extend our work with slightly refined and more formally structured requirements as well as a deep look at more recent tools, here in particular at Microsoft Azure Stream Analytics.

Compared to our earlier work, here we cast the requirements into a template from (Rupp & Pohl, 2021) that means we somewhat formalize them. We use a template to define the requirements in a standardized form and to show their importance. To ensure the quality of the requirements, we validated them against the quality criteria from (IEE, 1998).

Furthermore, we use the *IEEE830-1998* standard (IEE, 1998) for quality criteria for requirements. There exists a newer standard *IEEE/ISO/IEC29148-2011*, which describes the quality criteria from *IEEE830-1998* in more summarized form. We still meet all the quality criteria from both versions, except for the singularity. That one is new in *IEEE/ISO/IEC29148-2011*.

For the description and evaluation of the different tools we used the documentation of the publishers. Apache offers large documentation in (Str, 2021a) for Apache Storm, Amazon Web Services provides information in (Str, 2021b) for Amazon Kinesis Data (AWD) Analytics and Microsoft introduces Microsoft Azure Stream Analytics in (Microsoft, 2021b).

In contrast to other publications, we distinguish ourselves by standardizing and validating the requirements of EPNs and by evaluating various tools with different open or closed source characteristics, effort, and costs. With this variety, we aim to give an overview of different tools and support the decision for an individually suitable tool.

3 EPNs AND REQUIREMENTS

This section curtly presents the basics of EPNs and the requirements for this kind of systems.

According to (Luckham, 2002) an event is a significant change of state. Event Processing Networks (EPNs) can be seen as generalized software systems that allow the processing of events. However, EPN models lack standardization, which is where our work aims to contribute.

3.1 Basics of Event Processing Networks

EPNs are build on the basis of the Event-Driven Architectures (EDA) and Complex Event Processing (CEP). These both approaches

'[...] represent a new style of enterprise applications that places events at the center of the software architecture - event orientation as an architectural style.' — Dunkel and Bruns (Bruns & Dunkel, 2010, p. 4)

In this context, EDA is more about the design of event-driven architectures as a design style. CEP describes a technology for dynamic processing of large datasets (Bruns & Dunkel, 2010). Thus, CEP is a part of an EDA, which can be used for processing data within it. In detail CEP describes the dynamic processing of large data streams (also called *event streams*) in real-time. An event is any happening in the system. Here the change of state of a fact or an object is represented (Bruns & Dunkel, 2010).

The processing of events within a CEP is realized by using rules. These rules contain knowledge about handling events or event sequences (Dunkel & Bruns, 2015). For the realization of these rules and the processing of the data CEP contains Event Processing Agents (EPA).

An EPN is a set of EPAs, which are interconnected and exchange information during and about processing of the data (Dunkel & Bruns, 2015). An EPN can be interpreted as a graphical tool for modeling the flow of events for event processing systems (Koschel et al., 2017). Thus, the main components of EPN are EPAs in order to be able to perform CEP. EPAs contain various components like Event Model, (Event) Rules, and Event Procession Engine (Dunkel & Bruns, 2015).

Other components, for example producers, are also further elements of EPNs and can be taken from (Dunkel & Bruns, 2015) and (Koschel et al., 2017). The next part explains the requirements for EPNs.

3.2 Requirements

We evaluate the selected tools following an identical set of requirements, which we have put into a standardized form as shown next.

3.2.1 Handling the Requirements

The set of requirements originates from our work in (Koschel et al., 2017). We have standardized the form of these requirements in (Schulze et al., 2023) by applying (Rupp & Pohl, 2021) and (IEE, 1998). The reason was that the original requirements were just described as bullet points, had no formal structure and were partially a little ambiguous.

To address these issues, we evaluated various requirement templates how they address issues such as writeability, readability and learnability and common use (Robertson & Robertson, 2012). We have chosen (Rupp & Pohl, 2021) because it provides a straightforward structure for requirements.

In addition, we apply the quality criteria of IEEE 830-1998 (IEE, 1998) to achieve high quality requirements in structure and content. Specification of the quality criteria according to (IEE, 1998) are requirements, that are correct, unambiguous, complete, consistent, verifiable, modifiable, traceable and ranked for importance and/or stability.

The requirements are formulated according to a template and fulfill all quality criteria. Those templates achieve writeability, readability and learnability and are therefore efficient. This also satisfies the modifiable criteria from IEEE 830-1993.

Correctness, unambiguousness and completeness are achieved by splitting, expanding and substituting specialist terms. Requirements are checked to be consistent, verifiable, traceable and they are ranked by importance (see more details in (Schulze et al., 2023)).

3.2.2 The Requirements

Our standardized requirements are as follows:

- **EPNR1.** The tool shall offer the developer to model events with their inherent attributes as the central component of the engine.
- **EPNR2.** The tool shall map real world descriptions to events as scenarios.
- **EPNR3.** The tool shall offer event structures as simple, complex or aggregated. Simple events can be created and used independently. In addition, complex events have dependencies and references to other events. Also, aggregated events can be grouped logically.

- **EPNR4.** The tool shall offer possibilities to express the relativity of events and their temporal and causal relationships, e.g. sequence, preconditions and postconditions.
- **EPNR5.** The tool shall process and show the flow of events through the system.
- **EPNR6.** The tool shall offer the modeling of EPN by components, their properties and used patterns.
- **EPNR7.** The tool shall offer the modeling of components outside the system boundary and the behavior between inside and outside components.
- **EPNR8.** The tool should be expressive in usage, about readability, writability, learnability and efficiency.
- **EPNR9.** The tool should offer the developer further possibilities to create the model, e.g. IDE, graphical event programming.

In (Schulze et al., 2023) we evaluated Apache Storm (Str, 2021a) and in (Koschel et al., 2023) we evaluated Amazon Kinesis Data Analytics. As the major contribution of the present paper we will evaluate Microsofts Azure Stream Analytics against our requirements.

4 MICROSOFT AZURE STREAM ANALYTICS

In this section, Microsoft Azure Stream Analytics is examined and evaluated. It was chosen by the authors for its closed source characteristics and because of Microsofts significant position in the market.

4.1 Overview of Microsoft Azure Stream Analytics

Azure Stream Analytics is a real-time CEP engine that is designed to analyze and process high volumes of fast streaming data. Data can be obtained from multiple sources simultaneously. It can be categorized as a Platform-as-a-Service (PaaS) and thus it is fully managed within Azure (Microsoft, 2021b).

Azure Stream Analytics guarantees exactly-once event processing semantics and at-least-once event delivery, to avoid losing events. It also provides built-in checkpoints to maintain the status of your job and delivers repeatable results (Microsoft, 2021a).

For realizing the stream processing, Azure Stream Analytics provides the following components (Microsoft, 2021b):

4.1.1 Jobs

The core of Azure Stream Analytics are jobs. Jobs have to be created by developers themselves and consist of *Input, Query Processing Engine and Output* (see Figure 1). A job is a self-contained unit. However, jobs can be attached to each other so that the output of a previous job would be the input of the following job. This allows complex events to be processed in multiple stages. Furthermore, a job can have multiple inputs and outputs.

4.1.2 Input

As *Input* Azure Stream Analytics provides IoT Hubs, Event Hubs, Azure Blob Storage or Azure Data Lake Storage. Hubs and Storages also provide an interface for external producers to transfer data into a job. Hubs are classic message queues that collect data, store it temporarily, and transmit it to the Query Processing Engine on demand.

4.1.3 Output

The *Output* can provide Azure resources for storage and archiving, data warehousing, dynamic visualization or sending alerts and executing actions. Azure resources are possible as output, however these can transfer the output to external targets.

4.1.4 Query Processing Engine

Azure Stream Analytics and its Query Processing Engine is built on Trill – a streaming query processor developed by Microsoft Research (Chandramouli et al., 2014). In order to examine the Query Processing Engine it is necessary to take a closer look at Trill, since the engine is subject to the concept.

Trill (Trillion events per day) is an in-memory streaming analytics engine, which reads input data exactly once. It is designed to handle a wide range of data, both real-time and offline, and is based on a temporal data and query model. Trill can be used as a streaming engine, a lightweight relational in-memory engine, and a progressive query processor.

In addition, Trill was developed to address the following specific requirements (Chandramouli et al., 2014):

- **Query Model.** A Stream Analytics Engine must be able to process various data. On one hand, offline data should be processed, such as logs; on the other hand, real-time data should be processed in order to immediately publish warnings in case of possible problems. Real-time data should be able to be processed with historical data to detect

correlations. Finally, progressive data should be processed to provide fast and continuous information.

- **Fabric & Language Integration.** A Stream Analytics Engine must be easily usable by a high-level language (HLL) in order for the engine to be used by an application. HLLs such as Java and C# offer a rich volume of data types, libraries and custom logic that must be supported by the engine.
- **Performance.** Processing large offline data sets requires high throughput, while real-time monitoring requires low latency.

Trill meets all the above requirements with the help of the hybrid system architecture (Chandramouli et al., 2014). There are two modes of Trill defined as *Library mode* and *Multi-core mode*. In Library mode, which is the default, Trill is only executed in one thread and can thus be integrated into other applications. Compared to the Library mode, higher performance is achieved in Multi-core mode. Here several threads are executed in parallel. It supports a new two-level streaming temporal *MapReduce* operation, executed using a lightweight optional scheduler (Chandramouli et al., 2014).

Trill offers a query language called Trill-LINQ. This language enables CEP and also forms the basis of Query Processing Engine. The following core concepts are provided by Trill-LINQ (Chandramouli et al., 2015):

- **Filtering and Projection.** Similar to SQL, the projection of an event can be complete or transform e.g. selected attributes. Furthermore, events can be filtered based on conditions.
- **Windowing.** Time intervals within the processing are defined so that each event is assigned to a time window and processed inside of it. Events always contain a timestamp since they are processed by Trill strictly non-decreasing. Based on the length of the time intervals the developer controls the size of the batches and thus also the performance with latency and throughput. There are several windows, which are described in the context of Azure Stream Analytics.
- **Aggregation.** Close to SQL aggregation can be used to combine events and transform them into new values, e.g., number of events.
- **Grouped Computation.** Similar to SQL a grouping is performed, where the result of a sub-operation is provided to a specific key.
- **Correlation and Set Difference.** Trill allows to perform temporary join operations on different

streams or datasets and merge them. Any combination of real-time data, historical data or offline data is possible.

- **Data-Dependent Windowing.** The windows are selected based on the data. The window can be a time interval, as in classic windowing, or an event that can occur and thus shorten the time interval.

Trill has only been described rudimentarily, tailored to the article to explain Azure Stream Analytics. A detailed description would exceed the scope of this paper, but can be found in (Chandramouli et al., 2014) and (Chandramouli et al., 2015).

By incorporating Trill as a library, Azure Stream Analytics was up and running within 10 months. Queries are created using Transact SQL and then translated into Trill expressions by a SQL compiler (Terwilliger, 2018).

Since Trill is designed for high compatibility with high-level languages and their numerous data types, the Query Processing Engine can process data in JSON, AVRO or csv format. Due to the possible nesting of JSON and AVRO, events of arbitrary structure and complexity can be processed. This flexibility means, that Transact SQL can be extended with JavaScript and C# user-defined functions (Microsoft, 2021c). As known from SQL, this language also has operations to easily filter, sort, aggregate, and join streaming data.

As in Trill time is important since it is essential in CEP. Thus, it must be ensured that it is easy to work with this time component. Windowing was introduced for this case. In Azure Stream Analytics events are also processed based on their timestamp. In total there are five different time windows in Azure Stream Analytics (Microsoft, 2022):

- **Tumbling Window.** Divide into disjoint time intervals, events, whose time stamps are greater than the interval start and less than or equal to the interval end, are assigned to this interval. E.g. 12:00 PM – 12:05 PM window will include events that happened exactly at 12:05 PM, but will not include events that happened at 12:00 PM.
- **Hopping Window.** Intervals of fixed length overlapping and offset by one HOP parameter. E.g., interval length is 10 minutes and HOP parameter is 5 minutes, which means the first interval is from 12:00 PM - 12:10 PM and the second interval is from 12:05 PM - 12:15 PM. Events whose timestamp is 12:06 PM - 12:10 PM are in both intervals.
- **Sliding Window.** A fixed length interval is shifted using the time axis. In order not to generate an infinite number of possible shifts, a new window

is only created when events have left or joined this window.

- **Session Window.** The intervals depend on the arrival of the events and the maximum interval length. It is defined by a timeout and the interval length. If no further event occurs within the timeout after the arrival of an event, the interval is closed. As soon as a new event arrives, a new interval is opened. However, even with permanently arriving events, the interval length is never longer than the maximum length.
- **Snapshot Window.** All events with the same timestamp are assigned to one snapshot.

In addition to windowing, queries can be nested. So, it is possible to define sub-queries and process them within the query of a job. Within the sub-queries, different sources can be used and correlated.

The Query Processing Engine offers multiple functions through Trill to process streams. Beside the windowing and the accompanying division of the streams, events can be processed with a SQL based query language. Queries can be nested to structure the queries as needed. Functions can be created to make the processing more precise. Finally, jobs can be attached to each other. In addition to the stream, other data such as historical, reference, or scoring can be used to implement processing or correlate relationships (see Figure 1).

This part introduced the functionality of Azure Stream Analytics and Trill. By using a SQL-based language, streams can be examined and analyzed. The next part examines the requirements.

4.2 Evaluation of Microsoft Azure Stream Analytics for Modeling EPN

This part will argue, which requirements are fulfilled or not fulfilled by Microsoft Azure Stream Analytics.

- **EPNR1 - fulfilled:**
Events can be processed in JSON, AVRO or csv format. Both, JSON and AVRO data, can be structured and contain some complex types such as nested objects (records) and arrays.
- **EPNR2 - fulfilled:**
Due to the freely selectable structure of JSON and AVRO data, things of the real world can be described and mapped in scenarios.
- **EPNR3 - fulfilled:**
Events can be grouped and aggregated by queries. In addition, real-time data can be correlated with historical data to identify patterns. Finally, there is the windowing concept, in which groupings can be created based on time intervals.

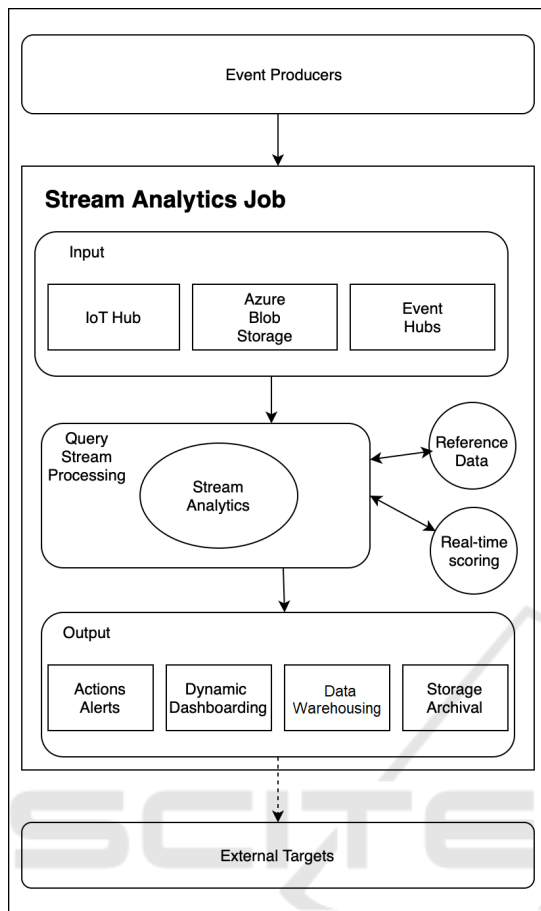


Figure 1: Stream processing concept in Azure (based on (Microsoft, 2021b)).

- **EPNR4** - fulfilled:
Events can be processed causally on multiple levels. Several jobs can be executed in sequence, furthermore within one job there can be several queries and processing steps.
- **EPNR5** - fulfilled:
Events are processed in a granular manner using at-least-once delivery and exactly-once processing. Job diagrams within the Azure interface visualize the flow of events from exactly one job.
- **EPNR6** - fulfilled:
The given components in Azure can be used to model an EPN. As described, there are Input, Query Stream Processing and Output. In essence, the Query Stream Processing offers the possibility of processing using the Stream Analytics Query Language. Extensions are supported by user-defined functions.
- **EPNR7** - fulfilled:
Event producers do not need to be part of the system. Any kind of data can be imported from out-

side, e.g. from IoT devices, weather data, protocols.

- **EPNR8** - fulfilled:
Various tutorials, from step-by-step beginner to advanced concepts and examples for end to end scenarios.
- **EPNR9** - fulfilled:
Implementation is possible in Azure Portal or Visual Studio (Code), Azure CLI, Terminals.

Azure Stream Analytics also meets all the requirements for modeling an EPN. Due to the PaaS character, developers contain a platform that enables them to specifically implement the use cases of stream processing. However, the given functions and features of Azure Stream Analytics, even though it is SQL-based, are very complex and can be a barrier to entry.

5 COMPARISON

In this article, we analyzed Microsoft Azure Stream Analytics in Section 4 in-depth with respect to our standardized set of requirements from Section 3.2.2. As we stated there, Microsoft Azure Stream Analytics nicely fulfills our requirements and thus it is well suited for the realization of EPNs.

Mainly the collected information for the presentation of Microsoft Azure Stream Analytics (as well as other tools) was taken from the documentation of the publishers or developers of the tools. For this reason, some information may be presented subjectively, as companies would like to widely distribute their tool in any case. Moreover, – in contrast to Apache Storm – Amazon and Microsoft Azure are costly tools, so an advertising factor within the documentation cannot be ruled out. Furthermore, information may be incomplete, because companies want to keep their implementations private.

To provide some more distinctive criteria to other tools we took in particular a more developer-oriented perspective. The result is summarized in table 1, which briefly compares all three tools under evaluation, namely Apache Storm, Amazon Kinesis Data (AKD) Analytics and Microsoft Azure Stream Analytics. Developers may use this table to identify the most important criteria that argue for or against a tool. In particular open source nature, price, convenience, and potential vendor lock in some distinctive factors.

While Apache Storm is open source and free of charge, Amazon Kinesis Data (AKD) Analytics and Microsoft Azure Stream Analytics are not. However, while Apache Storm has to be maintained by one selves, Amazon Kinesis Data (AKD) Analytics and

Table 1: Tools Criteria and Characteristics.

	Apache Storm	Amazon Kinesis Data Analytics	Microsoft Azure Stream Analytics
Basis	MapReduce	MapReduce	Trill
Support	no support	Plattform-as-a-Service (PaaS)	Plattform-as-a-Service (PaaS)
Costs	no costs	costs based on usage	costs based on usage
Effort	high	low	average
Eventformat	Tuple	Tuple	JSON, AVRO, csv
Environment	JVM	AWS	Azure
Language	topology in Java, others arbitrary	Java, Scala, Python, SQL	SQL based, JavaScript or C# user-defined functions
Distribution	Thread-based	Thread-based in Apache Flink	no details
Maximum Processing Rate	over one million tuples per second and per node	real-time	real-time
Reliability	guaranteed by spouts	guaranteed by AWS	99.9% promised
Data protection	own realization	Server location can be set	Server location can be set
Security	own realization	provided by AWS	provided by Azure

Microsoft Azure Stream Analytics are nicely hosted and maintained by Amazon and Microsoft and possibly easier to be used.

Thus, there is no clear winner between the three tools, but more a question of individual developer taste and skills as well as company preferences. For example, if a company prefers open source tools and has good development and hosting skills in house, then Apache Storm seems to be more suitable. If a company is an AWS shop anyway, wants likely less maintenance effort and is able to pay the costs for Kinesis, then Kinesis could be more favorable. Likewise Microsoft Azure Stream Analytics could be the favourite option, if a company is a strong Microsoft Azure user anyway.

6 CONCLUSION

In this article we had a deep look at Event Processing Network Models as a foundation of Event Stream Processing tools (cf. (Schulze et al., 2023)) and presented our enhanced (compared to our earlier work) standardized set of requirements for EPN models in Section 3.2.2.

As the key contribution of this article we apply those requirements, in order to have an in-depth look at Microsoft Azure Stream Analytics in Section 4. It turns out that Microsoft Azure Stream Analytics is a well suited tool for modeling and implementation of EPNs. Additionally we summarized our comparison

of Microsoft Azure Stream Analytics, Amazon Kinesis Data Analytics (Koschel et al., 2023), and Apache Storm (Schulze et al., 2023) in Section 5.

Since all those tools mostly fulfilled our requirements comparisons may need other criteria as well. The suitability of a tool depends on more individual circumstances, such as which kind of 'shop' you are – for example, Amazon vs. Microsoft –, but also how high the own development and administration effort should be.

Therefore, the decision for a suitable tool is based on the effort, the control and the costs involved. For these reasons, no absolute recommendation can be made. Rather the authors recommend examining each individual use case or at least a set of typical ones, in order to select the ideal tool. Since the field of event processing is more important than ever, we will continue to evaluate upcoming tools in this space within future work of ours.

REFERENCES

- (1998). IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*, (pp. 1–40).
- (2021a). Apache Storm. *Apache Software Foundation*. Online: <https://storm.apache.org/> [retrieved: 04, 2022].
- (2021b). Streaming Data Solutions on AWS. *Amazon Web Services Inc.* Online: <https://docs.aws.amazon.com/whitepapers/latest/streaming-data-solutions-amazon-kinesis/welcome.html> [retrieved: 04, 2022].

- Astrova, I., Koschel, A., Kobert, S., Naumann, J., Ruhe, T., & Starodubtsev, O. (2019). Evaluating RuleCore as Event Processing Network Model. In *Proc. 15th International Conference on Web Information Systems and Technologies (WEBIST 2019)* (pp. 297–300). Vienna, Austria: SCITEPRESS — Science and Technology Publications.
- Bruns, R. & Dunkel, J. (2010). *Event-Driven Architecture - Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse (Software architecture for event-driven business processes)*. Springer.
- Chandramouli, B., Goldstein, J., Barnett, M., DeLine, R., Fisher, D., Platt, J. C., Terwilliger, J. F., & Wernsing, J. (2014). Trill: A high-performance incremental query processor for diverse analytics. *Proceedings of the VLDB Endowment*, 8(4), 401–412.
- Chandramouli, B., Goldstein, J., Barnett, M., & Terwilliger, J. F. (2015). Trill: Engineering a library for diverse analytics. *IEEE Data Eng. Bull.*, 38(4), 51–60.
- Dunkel, J. & Bruns, R. (2015). *Complex Event Processing - Komplexe Analyse von massiven Datenströmen mit CEP (Complex analysis of massive data streams with CEP)*. Springer Vieweg.
- Koschel, A., Astrova, I., Kobert, S., Naumann, J., Ruhe, T., & Starodubtsev, O. (2017). Towards Requirements for Event Processing Network Models. In *Proc. 8th International Conference on Information, Intelligence, Systems, Applications (IISA 2017)* (pp. 27–30). Larnaca, Cyprus: IEEE.
- Koschel, A., Astrova, I., Kobert, S., Naumann, J., Ruhe, T., & Starodubtsev, O. (2018). On Requirements for Event Processing Network Models Using Business Event Modeling Notation. In *Proc. 2018 Conf. Intelligent Computing. Advances in Intelligent Systems and Computing (SAI 2018)* (pp. 756–762). London, UK: Springer.
- Koschel, A., Astrova, I., Pakosch, A., Gerner, C., Schulze, C., & Tyca, M. (2023). Is Amazon Kinesis Data Analytics Suitable as Core for an Event Processing Network Model? In *Proc. 16th International Conference on Agents and Artificial Intelligence (ICAART 2024)* (pp. 1036–1043). Rome, Italy: INSTICC, SCITEPRESS.
- Luckham, D. (2002). *The Power of Events*. Addison Wesley, USA.
- Microsoft (2021a). Event Delivery Guarantees (Azure Stream Analytics). *Microsoft Documentation*. Online: <https://docs.microsoft.com/en-us/stream-analytics-query/event-delivery-guarantees-azure-stream-analytics> [retrieved: 09, 2024].
- Microsoft (2021b). Introduction to Azure Stream Analytics. *Microsoft Documentation*. Online: <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction> [retrieved: 04, 2022].
- Microsoft (2021c). Parse JSON and Avro data in Azure Stream Analytics. *Microsoft Documentation*. Online: <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-parsing-json> [retrieved: 04, 2022].
- Microsoft (2022). Windowfunctions (Azure Stream Analytics). *Microsoft Documentation*. Online: <https://docs.microsoft.com/de-de/stream-analytics-query/windowing-azure-stream-analytics> [retrieved: 09, 2024].
- Robertson, S. & Robertson, J. (2012). *Mastering the Requirements Process: Getting Requirements Right*. Addison-Wesley Professional.
- Rupp, C. & Pohl, R. (2021). *Basiswissen Requirements Engineering (Basic knowledge Requirements Engineering)*. dpunkt.verlag.
- Schulze, C., Gerner, C., Tyca, M., Koschel, A., Pakosch, A., & Astrova, I. (2023). Analyzing Apache Storm as Core for an Event Processing Network Model. In *Proc. International Conference Intelligent Systems (IntelliSys 2023), LNNS 824* (pp. 397–410). Amsterdam, Netherlands: Springer, Cham.
- Shaikh, T. (2019). Batch Processing — Hadoop Ecosystem. *K2 Data Science and Engineering*. Online: <https://blog.k2datascience.com/batch-processing-hadoop-ecosystem-f6da88f11cae> [retrieved: 04, 2022].
- Terwilliger, J. (2018). Microsoft open sources Trill to deliver insights on a trillion events a day. *Microsoft Blog Developer*. Online: <https://azure.microsoft.com/de-de/blog/microsoft-open-sources-trill-to-deliver-insights-on-a-trillion-events-a-day/> [retrieved: 09, 2024].