Normalization of Unstructured Log Data into Streams of Structured Event Objects

Daniel Tovarňák
Institute of Computer Science
Masaryk University
Brno, Czech Republic
tovarnak@ics.muni.cz

Tomáš Pitner
Faculty of Informatics
Masaryk University
Brno, Czech Republic
tomp@fi.muni.cz

***Abstract*—Monitoring plays a crucial role in the operation of any sizeable distributed IT infrastructure. Whether it is a university network or cloud datacenter, monitoring information is continuously used in a wide spectrum of ways ranging from mission-critical jobs, e.g. accounting or incident handling, to equally important development-related tasks, e.g. debugging or fault-detection. Whilst pursuing a novel vision of new-generation event-driven monitoring systems, we have identified that a particularly rich source of monitoring information, computer logs, is also one of the most problematic in terms of automated processing. Log data are predominantly generated in an ad-hoc manner using a variety of incompatible formats with the most important pieces of information, i.e. log messages, in the form of unstructured strings. This clashes with our long-term goal of designing a system enabling its users to transparently define real-time continuous queries over homogeneous streams of properly defined monitoring event objects with explicitly described structure. Our goal is to bridge this gap by normalizing the poorly structured log data into streams of structured event objects. The combined challenge of this goal is structuring the log data, whilst considering the high velocity with which they are generated in modern IT infrastructures. This paper summarizes the contributions of a dissertation thesis „Normalization of Unstructured Log Data into Streams of Structured Event Objects“ dealing with the matter at hand in detail.**

***Index Terms*—log management, logging, data integration, normalization, stream processing, monitoring**

I. Introduction

Computer logs are one of the few mechanisms available for gaining visibility into the behavior of an IT infrastructure and its elements. They are also considered to be one of the richest and most valuable sources of such behavior-related monitoring information. However, log data are repeatedly reported to be of poor quality, mainly because a considerable portion of logs is unstructured by nature. This renders them to be unsuitable for straightforward automated processing and analysis. In many cases, even semi-structured log data can be considered sub-optimal for direct processing, i.e. when being processed by systems that expect some kind of *schema* to be imposed on the processed data. During the operation of any modern IT infrastructure, vast floods of heterogeneous log data are generated by many distributed producers spread across the infrastructure's layers and tiers.

These facts directly clash with a vision of a new-generation event-driven monitoring system enabling its users to transparently define real-time *continuous queries* over homogenous streams of properly defined monitoring events. The continuous queries would be used to detect *complex events*, for example, *one thousand of unsuccessful logins of user root in 5 minutes*, representing patterns of simpler events present in the monitoring information, e.g. *user login* in this case. Bluntly put, a holistic application of the Complex Event Processing approach to the monitoring and log analysis domain.

II. Context and Problem Statement

In the context of our work, an ideal state of affairs would be if all the log data, generated by the given IT infrastructure, were accessible in an interoperable and scalable manner as streams of structured event objects. *Structured event object* is a serialized piece of data, representing an occurrence, which is described via an explicit and strict data schema. *Event stream* is an infinite sequence of such objects adhering to the same schema. This fact would allow for all of the log data to be accessed in a transparent and unified way within the notion of a loosely coupled event-driven architecture. As a result, the log data consumers would not only be able to directly utilize the Complex Event Processing approach for advanced correlation and monitoring queries, but it would be also possible to research novel monitoring approaches, e.g. based on machine learning, pattern mining, or predictive modelling. All of this over high-quality source data.

In its current state, log data are continuously generated at high rates by many distributed producers using several transport protocols and many heterogeneous representations. Moreover, a predominant portion of *log entries* takes the form of unstructured or semi-structured data with the main piece of information, i.e. *log message*, represented as a free-form string mixing natural language with run-time context variables.

We propose to close this gap by the means of *data normalization*, i.e. transformation and unification of data transport, data representation, data types, and data structures resulting in a common format. Normalization is a recognized data integration pattern in the context of message-driven and, in turn, event-driven architectures. The presented dissertation thesis [1] deals with multiple knowledge gaps in areas inherent to the normalization of heterogeneous low-grade log data into streams of properly defined event objects.

978-3-903176-15-7

A. Logging Mechanisms

Logging is a programming practice. It is used by software developers to communicate information outside the scope of a program in order to trace its execution. Whilst the unstructured nature of log entries, i.e the respective elements written into *log*, can be remedied with a reasonable effort, the unstructured nature of log messages is stemming from the way they are created. The default logging mechanism of a vast majority of programming languages is traditionally based on a simple *string parameterization* allowing the developers to mix natural language with *logging variables* encapsulating execution context. Whilst very flexible, it immensely hinders the automated processing of log messages, which need to be explicitly parsed in order to impose some structure on them.

Since the existing literature dealing with the improvement of log data quality at their imminent source is very scarce, we have decided to research the possibility of designing a logging mechanism allowing the developers to communicate log messages in a manner resulting in the generation of fully structured log data. Due to the intended audience of this paper, this research area will not be extensively discussed here.

B. Log Abstraction

Log abstraction is one of the most crucial tasks in the process of log data normalization. It addresses the unstructured nature of log messages in a reactive manner, i.e. after the log data are generated. Simply put, log abstraction is the separation of the static and dynamic part of the log message so that both parts can be accessed independently. The static part is referred to as *message type*, which corresponds to the parameterized log message template in logging code, and the dynamic part corresponds to the actual *logging variables* and their values. Regular expressions corresponding to individual message types are typically used in practice to facilitate the actual abstraction.

Log abstraction can be seen as a two-tier procedure. First, a set of message types and corresponding matching patterns must be defined/discovered, and only after that can be this pattern-set used to pattern-match each incoming log message in order to extract dynamic information and impose some structure on it. We have identified challenges in both these tiers.

C. Log Data Normalization Description and Execution

In the course of the normalization process, pattern-matching and log abstraction of log messages is only one type of many different tasks that are usually needed to transform the log data into the desired state. In our case, the desired state is represented by structured event objects. The other tasks include, but are not limited to: string manipulation, e.g. whitespace trimming or pattern replacement; structure alteration, e.g. movement or renaming of data fields; type manipulation, e.g. type conversion or date parsing. These tasks must usually follow some kind of an execution logic and a conditional execution based on the content of the log data is also often needed. Moreover, there must be a way to define explicit data schemes of the transformed data in order to render it fully structured.

Currently, we are not aware of any general-purpose data transformation language that would be able to describe such log data transformation logic, let alone provide a way to execute it. On the other hand, we deem the use of general-purpose programming languages unsuitable due to the limited flexibility and inconvenience for domain experts. We believe that in this case, the use of some *domain-specific language (DSL)* is a proper path. This is indeed the path taken by many of so-called log data management tools, which have emerged due to the need for log data transformation and normalization.

Unfortunately, none of these tools is capable enough for the transformation of unstructured and semi-structured log data into fully structured event objects with explicitly defined schemes, which primarily stems from their internal orientation on semi-structured data and from untyped nature of their DSLs. Therefore, we orient on researching the possibility of describing and executing the above-mentioned transformations in a manner allowing for the normalization of heterogeneous log data into streams of event objects.

III. Research Goal and Contributions

In terms of methodology, we follow the one of *design science*, which deals with the design and investigation of *artifacts* in context, so that they can better contribute to the achievement of some goal that benefits its stakeholders. In the light of the above-mentioned facts, the **primary research goal** of the thesis is to *improve the way log data can be represented and accessed in order to allow the log analysis practitioners to analyze them in a unified and interoperable manner*. The main contributions of the thesis are represented by the design and evaluation of the following computer science artifacts.

Original Contributions

Design and evaluation of two prototypes of *structured logging mechanisms for Java programming language*. Both mechanisms allow the developers to communicate structured log messages, including their explicit data schemes, yet they differ in the provided flexibility and imposed overhead.

Design and evaluation of a *message type discovery algorithm based on word frequency clustering*, which is addressing several deficiencies of the existing algorithms for mining historical log data. The algorithm exhibits superior accuracy and improved usability in practice.

Design and evaluation of a *multi-pattern matching approach based on a special trie-based data structure*. The approach is highly-scalable with respect to the number of matching patterns, and its prototypical implementation exhibits a very respectable performance for real-world pattern sets.

Design and evaluation of a *log data normalization approach based on prototype-based programming* consisting of a *DSL* with the ability to describe log data transformations in an object-oriented manner, and of a *normalization engine* with the ability to execute these transformations, consequently resulting in the log data taking the form of streams of event objects.

Table I
EXAMPLE OF MESSAGE TYPE DISCOVERY IN THE TASK OF LOG ABSTRACTION

Log Messages		Message Types		Regular Expressions
User Jack logged in User John logged out Service sshd started User Bob logged in Service httpd started	⇒	User * logged * : [$1, $2] Service * started : [$1]	⇒	User (\w+) logged (\w+) Service (\w+) started

IV. LOG ABSTRACTION – MESSAGE TYPE DISCOVERY

The discovery of message types for the purposes of log abstraction is a tiresome process when done entirely manually. Log data generated by a single application or software project can contain hundreds of unique message types since their logging code can contain hundreds of unique logging statements. For this reason, the research in this respect is focused on automated approaches for message type discovery. In the literature, two orthogonal groups of approaches can be identified – the message type discovery can be based either on source code analysis, or on data mining techniques.

Since the source code of the targeted log data producer may not be always available for analysis, which is true especially for proprietary hardware and software appliances, we have turned our attention towards approaches that discover message types from historical log data via data mining techniques.

A number of works emerged in this area over the years, utilizing different approaches primarily based either on cluster analysis or custom heuristics, eventually the combination of both. We have studied the existing algorithms (and their implementations) and identified several deficiencies, mainly in terms of their practical usability for our goals.

1) The algorithms often produce overlapping message types, i.e. it is possible for an individual log message to correspond to more than one discovered message type, which is not suitable for the purposes of log abstraction.
2) It is common for the discovery algorithms to be fine-tuned to yield the best results. However, there are algorithms that do not support any fine-tuning at all, or in contrast, provide up to 5 mostly unbounded parameters.
3) The common step of each approach is the tokenization of the log messages. The studied approaches predominantly use space as a fixed delimiter, unable to work with multiple delimiters, which decreases their accuracy.
4) The algorithms do not support multi-word variable positions leading to sub-optimal results.

A. Our Approach

In order to address the above-mentioned deficiencies, we have decided to design a new message type discovery algorithm combining different techniques used in the studied approaches. The approaches used in this area take advantage of the observation that although the log messages are free-form, they are generated by a limited set of fixed logging statements and thus the generated log messages are likely to form clusters with respect to variable positions.

We refer to the algorithm as to the *Extended Nagappan-Vouk* (*ENG*) since it is based on the original idea of frequency table and intra-message word frequency proposed in [2]. Other than that, the algorithm is significantly improved in order to support multiple delimiters for tokenization, support multi-word variables, report distinct message types with no overlapping, and finally, the algorithm can be parameterized via a single parameter controlling its sensitivity and the granularity of the reported message types. The discovery algorithm is able to generate pattern sets in a special format directly suitable for log abstraction via pattern matching as can be seen in Listing 1.

```
regexes: # regex tokens
  INT:        [integer, "[0-9]+"]
  BOOL:       [boolean, "\btrue\b|\bfalse\b"]
  WORD:       [string,  "[0-9a-zA-Z]+"]

patterns: # patterns describing the message types
  grp0:
    mt1: 'User %{WORD:var1} logged %{WORD:var2}'
    mt2: 'Service %{WORD:var1} started'
```

Listing 1: Example output of the designed algorithm

B. Evaluation Summary

The accuracy evaluation of message type discovery is a typical task that can be achieved via classic information retrieval techniques for clustering evaluation [3]. Assuming the discovery does not produce overlapping message types, each discovered message type induces a *strict cluster* of log messages in the original data set and it is possible to calculate a number of external criteria that evaluate how well the discovered clustering matches the *gold standard* classes (message types). Similarly to many others, we use *F-measure (F_1 score)* as the external criterion to be used for message type discovery accuracy evaluation. F-measure is a harmonic mean of *precision* and *recall*, other common external criteria.

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Thanks to the work of He et al. [4] we have been able to evaluate the accuracy of our algorithm on externally provided heterogeneous log message data sets and accompanying ground truths, which adds to the evaluation validity. Moreover, we were able to compare the algorithm's accuracy with accuracies reported for some other algorithms for message type discovery. In their evaluation study, the authors used five real-life log message data sets ranging from supercomputers (BGL and HPC), through distributed systems (HDFS and Zookeeper), to standalone desktop software (Proxifier), in order to evaluate

accuracy of four different message type discovery algorithms (*SLCT*, *IPLoM*, *LKE*, and *logSig*). The data sets were randomly sampled for 2000 log messages from each dataset in order to shorten the running times of some of the more computationally intensive algorithms. The ground truth (gold standard) was created manually by the authors. The reported results (F-measures) of the evaluated algorithms as well as results our algorithm are summarized in Table II.

It can be seen that the proposed algorithm exhibits a superior accuracy in an evaluation based on five real-world data sets with externally provided ground truth. When using its default settings (*ENG*), the algorithm achieved very high accuracy with an average *F-measure* of 0.953. When considering the best algorithm settings for each data-set (*ENG**), it exhibited an average *F-measure* of 0.996.

Table II
F-MEASURES FOR EVALUATED ALGORITHMS

	BGL	**HPC**	**HDFS**	**Zookeeer**	**Proxifier**	***AVG***
SLCT	0.61	0.81	0.86	0.92	0.89	*0.818*
IPLoM	0.99	0.64	0.99	0.94	0.90	*0.892*
LKE	0.67	0.17	0.57	0.78	0.81	*0.600*
LogSig	0.26	0.77	0.91	0.96	0.84	*0.748*
ENG	0.9251	0.986	1.00	0.9999	0.8547	***0.953***
ENG*	0.9985	0.986	1.00	0.9999	1.00	***0.996***

V. LOG ABSTRACTION – MULTI-PATTERN MATCHING

Given a set of matching patterns representing individual message types and an input log message, the combined goal of pattern-matching for log abstraction is to determine if the input fully adheres to any of the message types, and, if so, uniquely identify it and extract the values present on the respective variable positions of the message type. A commonly used naïve approach is based on the iteration of the given pattern set until a match is found. However, this is infeasible for velocities in which the log data are currently generated since there can be hundreds or even thousands of message types in a single pattern set. Therefore, scalable approaches for log message abstraction based on multi-pattern matching are needed. We have recognized two approaches, which can be utilized for multi-pattern matching in terms of log abstraction, both based on limiting the searched pattern-space – multi-regex matching and tree-based organization.

Multi-regex matching is based on combining the respective finite automata corresponding to the individual regular expressions into an equivalent finite automaton consequently used for pattern matching. However, we have learned that in terms of practical multi-regex matching implementations suitable for log abstraction, the situation is unsatisfactory, and there is an inherent complexity when implementing such an approach.

A. Our Approach

In our work, we have focused on addressing the problem from a different direction – what if we wanted to avoid the complexity of multi-regex matching altogether by leveraging the specific goals of log abstraction and characteristics of the matching patterns created for this purpose? *Tree-based approaches* address the problem of multi-pattern matching in a more straightforward way by organizing the matching patterns in various tree-like structures with the goal of segmenting and limiting the searched pattern-space. This tree-like organization can be either inter-pattern, i.e. organizing the individual patterns as a whole with respect to some observed knowledge, or intra-pattern, i.e. organizing the individual pattern components, words for example, into a tree-like matching structure.

We have designed an elegant multi-pattern matching algorithm based on a clever intra-pattern organization that is able to practically eliminate the need for multi-regex matching, whilst imposing only minimal limitations on the way the matching patterns can be created. The basic idea of our approach is based on organizing the pattern set into a special data structure we refer to as regex trie (REtrie) as seen on Figure 1.

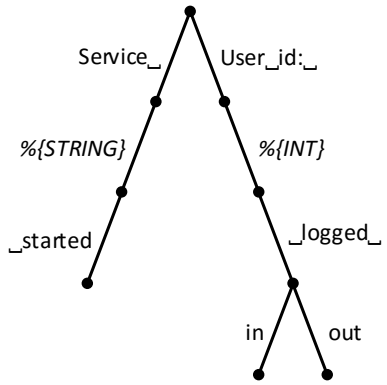

Figure 1. Regex trie (REtrie) containing three matching patterns

Trie is a tree-like data structure used for storing strings. Alongside with hash table, trie is one of the fastest data structures for string retrieval when implemented properly. In our case the search process follows a depth-first traversal, i.e. it backtracks when it is unable to continue on the current trie path. Thanks to the explicit priorities, the most specific patterns are tried first. In the case a path can be found from the trie root to a node with a non-empty leaf value, a successful match is returned, together with values captured by the regex tokens. This way, each log message can be matched against the whole pattern set represented by the regex trie at once.

B. Evaluation Summary

We have performed a series of experiments based on two real-world pattern sets and also partially generated data sets in order to evaluate the practical implementation of the presented data structure and the related multi-pattern matching approach. The results showed that the performance of regex trie scaled well with respect to the number of patterns (thousands) as well as the number of CPU cores. The tested Erlang implementation consisting of mere 300 lines of code exhibited a decent speed-up, stopping at an overall throughput of more than 1.9 million abstracted log lines per second on 8 cores[1].

[1]Intel® Xeon® CPU E5-2650 v2 @ 2.60GHz with 64GB RAM.

VI. End-to-End Log Data Normalization

From the data integration perspective, normalization can be performed on four different translation layers. The *transport* layer determines the way the data are transferred over the network. The second layer determines the *data representation*, i.e. how the data are serialized into individual elements, consequently determining if they are unstructured, semi-structured, or structured. The *data types* layer is extremely important since it defines the data types on which the domain model is based. The fourth, *data structures* layer, describes a top-level domain model, i.e. what logical entities will be dealt with and what relationships will they have, if any. In terms of data integration, the most loosely coupled outcome of normalization takes the form of a *Canonical Data Model*, i.e. a common data format unifying the three top layers – the bottom layer is assumed to be based on messaging. In our case, the *Canonical Data Model* is represented by structured event objects and their individual types, whose data schemes are explicitly defined.

In the course of the normalization process, the parsing of different formats of log entries and abstraction of log messages is only one type of many different tasks that are actually needed to transform the log data into the desired state. Other common tasks that are somewhat inherent to the log data normalization process include: input and output adaptation, data serialization and deserialization, parsing, transformation, and enrichment.

Whilst some of the already existing log management tools are quite capable and they support many of these tasks, in one form or another, they have very limited capabilities in terms of structuring the log data into event objects, which mainly stems from their orientation on basic semi-structured data manipulations and predominantly untyped nature of their corresponding domain-specific languages. Although we have been able to implement end-to-end log data normalization logic by using these tools, it was always at the cost of manual type enforcement, ex-post schema definition, and combination with additional external functionality, which was rather error-prone.

A. Our Approach

To address the problems pointed out above we have first designed an abstract *log data normalization approach* that allows for the data transformations to be carried out in a statically typed object-oriented paradigm, instead of being oriented on dynamically typed or untyped transformations of associative arrays, as it is common in practice. Then, we have created a *domain-specific language* and related execution logic implementing this approach that is covering the most common transformation operations with specific orientation on data lacking explicit structural information, i.e. unstructured and semi-structured data. Last, but not least, we have created a *normalization engine prototype* that is able to perform this execution logic whilst handling the tasks that are not necessarily the responsibility of the DSL, e.g. data serialization, or timekeeping. A simple result of log data normalization, as discussed throughout this paper, is illustrated by Listing 2.

1) Prototype-Based Normalization: The designed normalization approach can be described as a series of object-to-object transformations, which is partially based on the notion of *prototype-based object inheritance*, sometimes also referred to as prototype-based programming. Prototype-based programming is a variant of object-oriented programming in which new objects are created by reusing attributes of existing objects, which serve as prototypes [5]. There is no notion of instantiation as in the case of class-based programming.

In our approach, every piece of data intended for normalization starts as an object with a properly defined object type it belongs to. As soon as an object is created/constructed, it is immutable, i.e. the object and its attributes cannot be further modified. The only way to achieve such a modification is to clone the existing object, referred to as the *prototype*, and perform a finite sequence of attribute manipulations, i.e. additions, deletions, and transformations, which will subsequently result in the construction of a new immutable object that is based on the prototype. The typed data objects that are the result of this object-to-object transformation represent the normalized event records that can be serialized into structured event objects and exposed as data streams.

2) Domain-Specific Language: The simple domain-specific language that implements the normalization approach presented above is based on YAML data format and the actual compilation/execution logic is backed by Erlang programming language. The normalization logic is described via a *transformation descriptor* written in the DSL, which is then compiled into a sequence of instructions that can be executed in Erlang. During the compilation, a basic type-checking is performed and an explicit type information and external schemes are generated, which are describing all the defined object types. This means that it is possible to enumerate all the event/object types that can be yielded by the normalization process *before* the execution.

3) Normalization Engine: We have aimed at a minimalistic design of the normalization engine with the goal of keeping the necessary requirements to a bare minimum. The engine, written in Erlang, instantiates the input adapters as per their definition in the transformation descriptor, executes the transformation logic, and serializes the resulting event records via data serialization format of choice. The engine is also responsible for schema generation. The normalized event objects are then written into a messaging system via an output adapter. Currently, Apache Kafka serves as the primary delivery system.

```
<137>Apr 5 19:31:10 serena audd[631]: User xtovarn logged in
```

```
UserSession() {
    syslog=SyslogInfo() {
        timestamp=1459877470000, severity=1, facility=17,
        hostname="serena", app_name="audd", procid=631
    },
    user="xtovarn",
    action="LOGIN"
}
```

Listing 2: Example of an unstructured *Syslog* log entry with log message in natural language and a corresponding normalized event object representing a successful user login

B. Evaluation Summary

In a real-world setting and the context of online data processing we consider *throughput* to be one of the most important performance metrics. We have evaluated the presented approach in terms of end-to-end throughput on real-world data sets for a workload consisting mainly of log message abstraction. The performed experiments showed that the approach is able to normalize approximately two hundred thousand unstructured log entries per second, with the normalization engine running on a single commodity server, and the delivery system running on three dedicated machines. The hardware setup of the benchmarking cluster is shown in Table III.

Table III
HARDWARE SETUP FOR THE CONDUCTED EXPERIMENTS

Node type (#)	Hardware
Benchmarking (1×)	• Intel® Xeon® E5410 @ 2.33GHz • 4 cores, 16GB RAM, SATA7.2k
Normalization (1×)	• Intel® Xeon® E5-2650 v2 @ 2.60GHz • 8/16 HT cores, 64GB RAM, SATA7.2k
Messaging (3×)	• 2 × AMD Opteron™ 4284 @ 3.0GHz • 2 × 8 cores, 64GB RAM, SATA7.2k

VII. CONCLUSION AND FUTURE WORK

The thesis [1] summarized in this paper represents a comprehensive material dealing with one of the richest sources of behavior-related monitoring information, i.e. log data. Although the value of log data is widely recognized, so is their poor quality, which is rendering them unsuitable for automated processing. In this work, we have dealt with the primarily unstructured nature of log data, and especially log messages, which typically represent the most important information present in the generated log entries.

We have addressed the matter at hand by improving the quality of log data, their structure, representation, and the way they can be accessed, by their normalization into fully structured event objects with defined data schemes, which can be exposed as data streams. The results related to this thesis were published on multiple occasions [6], [7], [8], [9], [10], [11], [12], [13], [14], [15].

The achieved results offer virtually endless possibilities with respect to new approaches for log data analysis, correlation, storage, mining, pattern detection, prediction, root cause analysis, or machine learning in many application areas. In addition, thanks to the proposed concepts, it is possible to implement an architecture that allows for ingestion and normalization of large amounts of heterogeneous monitoring data into a central location, rendering them readily available for real-time analysis, detection, alerting, and long-term retention.

We plan to reap the benefits of such a unified access to high-quality event data in our future endeavours. One of our biggest ambitions in this area is the utilization of the presented results for a holistic realization of the distributed event-driven monitoring architecture for real-time security monitoring based on information from corresponding log data producers and other important security information sources, e.g. IP flows.

ACKNOWLEDGEMENTS

The publication of this paper and the follow-up research was supported by the ERDF „CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence“ (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

REFERENCES

[1] D. Tovarnak, “Normalization of Unstructured Log Data into Streams of Structured Event Objects [online],” *Dissertation thesis*, Masaryk University, Faculty of Informatics, Brno, 2017, available from <https://is.muni.cz/th/rjfzq/thesis-twoside-final-bw.pdf> [cit. 2018-10-28].

[2] M. Nagappan and M. A. Vouk, “Abstracting log lines to log event types for mining software system logs,” in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 114–117.

[3] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[4] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, “An evaluation study on log parsing and its use in log mining,” in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016, pp. 654–661.

[5] M. Abadi and L. Cardelli, *A Theory of Objects*, 1st ed. Springer-Verlag New York, Inc., 1996.

[6] D. Tovarnak and T. Pitner, “Towards Multi-tenant and Interoperable Monitoring of Virtual Machines in Cloud,” in *Proceedings of the 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, ser. SYNASC ’12. IEEE Computer Society, 2012, pp. 436–442. [Online]. Available: http://dx.doi.org/10.1109/SYNASC.2012.55

[7] D. Tovarnak, A. Vasekova, S. Novak, and T. Pitner, “Structured and Interoperable Logging for the Cloud Computing Era: The Pitfalls and Benefits,” in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, ser. UCC ’13, 2013, pp. 91–98.

[8] D. Tovarnak, “Towards Distributed Event-driven Monitoring Architecture [online],” *Ph.D. thesis proposal*, Masaryk University, Faculty of Informatics, Brno, 2013, available from <http://theses.cz/id/0jawn5/?lang=en> [cit. 2017-02-02].

[9] D. Tovarnak, F. Nguyen, and T. Pitner, “Distributed Event-Driven Model for Intelligent Monitoring of Cloud Datacenters,” in *Proceedings of the 7th International Symposium on Intelligent Distributed Computing*, ser. IDC ’13. Springer International Publishing, 2014, pp. 87–92. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-01571-2_11

[10] F. Nguyen, D. Tovarnak, and T. Pitner, “Semantically Partitioned Peer to Peer Complex Event Processing,” in *Proceedings of the 7th International Symposium on Intelligent Distributed Computing*, ser. IDC ’13. Springer International Publishing, 2014, pp. 55–65. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-01571-2_8

[11] D. Tovarnak and T. Pitner, “Continuous Queries Over Distributed Streams of Heterogeneous Monitoring Data in Cloud Datacenters,” in *Proceedings of the 9th International Joint Conference on Software Technologies - Volume 1: ICSOFT-EA*, ser. ICSOFT ’14, INSTICC. SciTePress, 2014, pp. 470–481.

[12] D. Tovarnak, “Practical Multi-Pattern Matching Approach for Fast and Scalable Log Abstraction,” in *Proceedings of the 11th International Joint Conference on Software Technologies - Volume 1: ICSOFT-EA*, ser. ICSOFT ’16, INSTICC. SciTePress, 2016, pp. 319–329.

[13] M. Cermak, D. Tovarnak, M. Lastovicka, and P. Celeda, “A Performance Benchmark for NetFlow Data Analysis on Distributed Stream Processing Systems,” in *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium*, ser. NOMS ’16, 2016, pp. 919–924.

[14] T. Jirsik, M. Cermak, D. Tovarnak, and P. Celeda, “Toward Stream-Based IP Flow Analysis,” *IEEE Communications Magazine*, vol. 55, no. 7, pp. 70–76, 2017.

[15] J. Vykopal, R. Oslejsek, P. Celeda, M. Vizvary, and D. Tovarnak, “KYPO Cyber Range: Design and Use Cases,” in *Proceedings of the 12th International Conference on Software Technologies - Volume 1: ICSOFT*, ser. ICSOFT ’17, INSTICC. SciTePress, 2017, pp. 310–321.