AID4TRAIN: Artificial Intelligence-based Diagnostics for TRAins and INdustry 4.0

Marcello Cinque[1,2], Raffaele Della Corte[1,3], Giorgio Farina[2], and Stefano Rosiello[3]

[1] Università degli Studi di Napoli Federico II, via Claudio 21, 80125 Naples, Italy
[2] Consorzio Interuniversitario Nazionale per l'Informatica, M.S. Angelo, Via Cinthia, 80126 Naples, Italy
[3] Critiware s.r.l., via Carlo Poerio 89/A, 80121, Naples, Italy
{macinuqe, raffaele.dellacorte2, giorgio.farina}@unina.it, stefano.rosiello@critiware.com

Abstract. Diagnostic data logs generated by systems components represent the main source of information about the system run-time behavior. However, as faults typically lead to multiple reported errors that propagate to other components, the analysts' work is hardened by digging in cascading diagnostic messages. Root cause analysis can help to pinpoint faults from the failures occurred during system operation but it is unpractical for complex systems, especially in the context of Industry 4.0 and Railway domains, where smart control devices continuously generate high amount of logs.
The AID4TRAIN project aims to improve root cause analysis in both Industry 4.0 and Railway domains leveraging AI techniques to automatically infer a fault model of the target system from historical diagnostic data, which can be integrated with the system experts knowledge. The resulting model is then leveraged to create log filtering rules to be applied on previously unseen diagnostic data to identify the root cause of the occurred problem. This paper introduces the AID4TRAIN framework and its implementation at the current project stage. Further, a preliminary case study in the railway domain is presented.

Keywords: Artificial intelligence · Fault model · Railway · Industry 4.0.

1 Introduction

Diagnostic data logs are the primary source of data for understanding the behavior of a system. Diagnostic data logs are sequences of text lines –typically stored in log files– reporting on the runtime behavior of a system [3], including entries highlighting the occurrence of system failures. Their analysis has been extensively used for troubleshooting [7, 19, 2].

Root cause analysis is a well-established practice aiming at identifying the fault originating failures occurred during system operation. Understanding the fault underlying the occurrence of a failures is of paramount importance

since it provides insights about the potential corrective actions to prevent failures from appearing later on and to avoid severe consequences [8], such as data and economical loss, damage to the environment. However, root cause analysis is often carried out in a manual fashion. Human experts generally rely on their experience to identify suspicious log entries, often by querying for predefined keywords. Understanding and traversing the diagnostic data logs from different components demands for substantial cognitive work by human experts. Highly specialized analysts are expected to face a number of challenges, which encompass the high volume and heterogeneity of data, the presence of different error and failure mode, the presence of corrupted, duplicated or redundant data, or even worst, the absence of data to infer the root cause of the occurred problem, the absence of consolidated and automatic analysis procedures.

This is especially true in the context of **Railway** and **Industry 4.0** domains, where smarts and control objects continuously generate high amount of diagnostic data logs. In these domains the root cause analysis is important to improve the reliability and safety of the operation. Complex systems like Train Control and Monitoring System (TCMS) [5] and smart manufacturing leveraging Industrial Internet of Things (IIoT) technologies [1] encompass a wide set of heterogeneous subsystems and components, each one generating diagnostic data logs. In addition, the presence of a fault typically leads to errors propagating through the system components, generating cascading diagnostic messages that increase the complexity of the root cause analysis and the cost of maintenance.

In this paper, we introduce the **AID4TRAIN** (*Artificial Intelligence-based Diagnostics for TRAins and INdustry 4.0*) project, which aims to support and improve the root cause analysis in Industry 4.0 and Railway domains leveraging Artificial Intelligence (AI) and data analytics approaches. The project purses the idea to bring together the system view provided by diagnostic data logs and the system expert knowledge, modeled as fault trees. AI and data analytics approaches are used to infer the fault model of the target system from historical diagnostic data in automatic way, which is then checked and integrated with the knowledge of system experts. The inferred fault tree model is subsequently leveraged to create log filtering and correlation rules to be applied on previously unseen diagnostic data to identify the root cause of occurred problems. The project is being developed by Critiware S.r.l. and an industrial railway partner acting as problem owner and data provider (Hitachi Rail Italy), and with the support of a national research center (CINI). The paper describes the concepts underlying the AID4TRAIN framework, its main components, and reports preliminary results obtained in a railway case study.

2 Problem Statement

Diagnostic systems available on modern train are able to automatically report information related to potential faults or functional anomalies. However, as anticipated, the consequences of a fault are hardly confined on the component that is first affected by the problem, and can have cascade effects (and related diagnos-

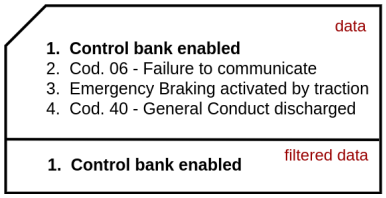

Fig. 1. Example of diagnostic log and filtering.

tic messages) on many components. This leads to logs with an excessive number of lines which hardens the root cause analysis task, impacting to the overall life-cycle cost of a working train, due to the need of highly specialized maintenance staff. The analysis is impaired by several factors: data volume and heterogeneity, variety of fault types, presence of corrupted, redundant or repeated data, and, in rare cases, lack of useful information to reconstruct the event of interest. On the other hand, modern train supervision and control systems are equipped with more powerful computing resource, if compared to past systems, opening to novel opportunities to automate the on-board selective diagnostic task.

While there is still the need for human experts, to ultimately judge the hypothesized cause of a fault and program a maintenance action, the problem we aim to solve is to reduce the gap between the expert and the raw data, by providing a simplified view of knowledge automatically extracted from data in the form of fault trees. The challenge we address in this paper is to infer these models by mining potential correlations across huge volumes of data, spanning significant hours of functioning of the on-board equipment, in production environments. Learned grouping rules can be then used, in a later stage of the project, to automatically filter raw data on-board, hence reducing the amount of information to be delivered to the specialist, in case of anomalies.

As an example, a very common scenario pointed out by analyzing diagnostic data logs provided by the railway partner of the project is related to the *activation of a train control block*. When a train driver enables the control bank, typically a huge number of anomalies are reported into the diagnostic log. As shown in the *data* block of Fig. 1, this event (i.e., `Control bank enabled`) is recorded in diagnostic logs along with the cascading anomalies[4] (e.g., `Cod.06-Failure to communicate`, `Emergency braking activated by traction`, `Cod.40-General Conduct discharged`). The maintenance experts inferred that these cascading events are related to the activation of the control block (and then to the diagnostic systems not ready yet or still in startup phase), after time consuming manual analysis. Our framework aims to easy out this manual process by pinpointing only primary events (i.e., `Control bank enabled` in this example) and identifying secondary events (e.g., `Emergency braking activated by traction`), generating a data log where secondary events can be easily filtered out (as shown the *filtered data* block in Fig. 1) to easy out both operation and maintenance of the train system. Therefore, the aim here is to generate rules allowing to classify events in primary and secondary, through the analysis of historical diagnostic data logs collected during the system runtime.

[4] Only a fragment of the log is reported due to space limitations.

3 Proposed Framework

The core of AID4TRAIN is a framework aiming to support smart maintenance processes by enhancing failure diagnostics. The core of the framework is a central *Fault-Tree Database* (FTD), which includes causal relationships between events occurring during the system behavior. The FTD is composed by a set of fault-tree models. These models are either provided by the diagnostics experts, by means of the *Fault-Tree Editor* (FTE), or proposed automatically by looking at past diagnostic events occurred in production, by means of the *Artificial Intelligence-based Log Analyzer* (AILA), as depicted in Fig. 2.

The FTE makes it easier for the domain expert to model well-known relationships between detected anomalies and possible root causes (e.g., from technical datasheet of the target system and its components). The AILA tool both identifies new failure modes and enhances existing fault trees by applying artificial intelligence algorithms to the historical field failure data logs. This component will automatically propose new fault trees to the domain experts. The expert can discover subtle fault chains leading to some system error that are not previously known. The tool can also propose new root events, in case a sequence of reported diagnostic events occurs multiple time in the same conditions but a root cause is not know. These discovered root events should be further analyzed by the experts and refined by means of the FTE tool.

The FTD stores all the relationship to effectively help both the system operators and the maintenance teams to quickly identify anomalies in all the diagnostics event produced by the system. To this purpose, another key component of the framework is the *Fault-Tree Model Compiler* (FTMC), which translates fault-tree models in log filtering rules. Those rules classify all the diagnostics events occurring during the system operation in either "primary" (e.g., the root cause) and "secondary" (e.g., the related propagating events). Diagnostic logs and filtering rules are imported and analyzed by means of the *Log File Analyzer* (LFA). This component analyses the set of anomalies reported in diagnostic logs, and it applies the rules given by the FTMC to point out automatically the possible primary causes of the events.

At the current project stage we are focusing on the design and implementation of the AILA component, which is presented in Section 4.

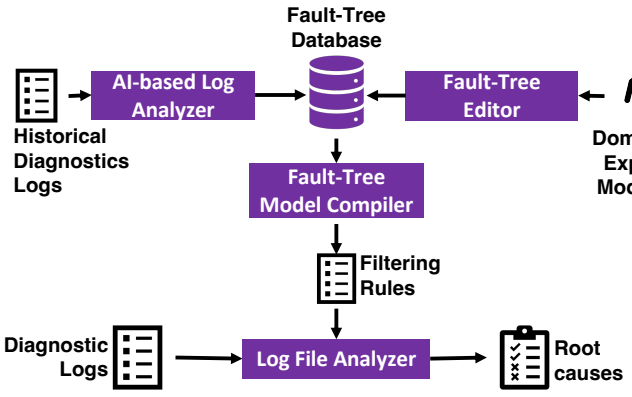

Fig. 2. AID4TRAIN framework components.

4 The AI-based Log Analyzer

The Artificial Intelligence-based Log Analyzer (AILA) of the AID4TRAIN framework is a Python-based component, leveraging both temporal coalescence and unsupervised AI techniques to infer potential filtering rules. AILA implements the flow shown in Fig. 3, which encompasses the following three steps.

Tupling. The event-tupling is a well-known technique [4] for temporal coalescence. A tuple is a collection of events that are close in time. The principle is due to the empirical observation that often multiple events that are reported together are due to the same underlying fault. Moreover, the same fault may persist, repeat often over time, and propagate to other components, which in turn may report an additional event into the log file. Two subsequent events are included in the same tuple if the time elapsed between the two occurrences is less than a fixed window size. Otherwise, they will be grouped in two different tuples. In this process the choice of the window size is a crucial factor, and represents a configuration of AILA. The obtained tuples are then represented by a binary sequence. Each element represent the presence of a particular event in the tuple.

Clustering. The AILA component assumes that the historical diagnostic data are not labeled (as in our railway case study), which prevents the use of supervised AI techniques. Therefore, in this step the tuples including a similar set of events are grouped together in a cluster leveraging an unsupervised AI technique, in order to learn a potential filtering rule. To this purpose we apply a hierarchical clustering technique based on single linkage [20]. To measure the distance between two tuples, we use the hamming distance, which is directly proportional to the number of different events occurred in the tuples. For each cluster we compute the number of events within the clusters, the probability of occurrence of the event within the cluster, and the average duration of the tuples within the cluster. Each cluster is considered to be a potential learned filtering rule if it contains at least two events and at least two tuples. Indeed, clusters with a single tuple are likely due to chance. Moreover, clusters with less

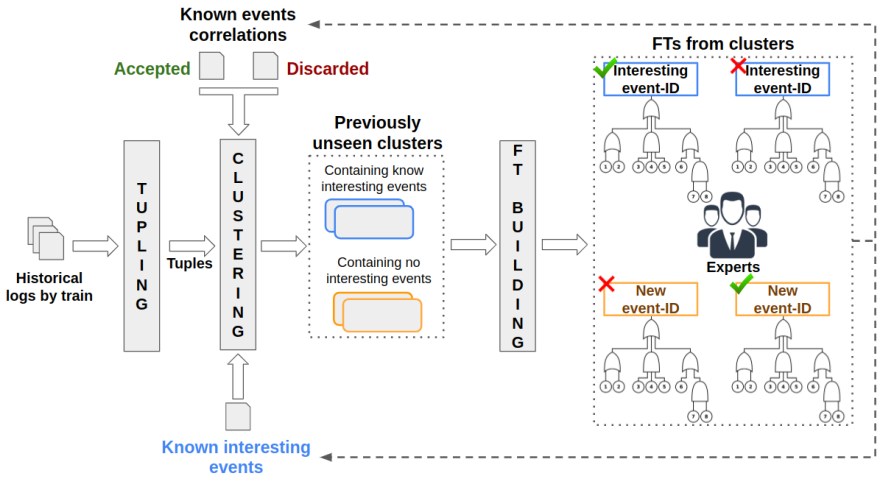

Fig. 3. The Artificial Intelligence-based Log Analyzer component.

than two events are not useful to learn potential correlations between events. The clustering step leverages a list of *known interesting events*, i.e., events that domain experts already known to be relevant in the target system, e.g., events representing root cause of failures. The clustering step leverages this list to divide the generated clusters in two groups: (i) clusters containing known interesting events and (ii) clusters containing no interesting events.

Fault Tree building. Each cluster is transformed in a two-level fault tree. The root of the tree is the *primary event*, which is selected depending on the group the cluster belongs to. If the cluster contains an interesting event, this event is used as primary event of the tree, i.e., *Interesting event-ID* in Fig. 3; otherwise, the primary event is represented by a potential unknown event, i.e., *New event-ID* in Fig. 3, which needs to be investigated from domain experts. The remaining events of the cluster are used as leaves of the tree, representing *secondary events*. The inferred trees represent filtering rules, which can be summarized as follows: when all the events composing the fault tree occur within the related time window (i.e., the average duration of the tuples within the cluster), the root event of the tree is marked as primary, while all other events as secondary. This allows analysts to easily filter-out all the secondary events from diagnostic logs, focusing only on the primary ones when analyzing diagnostic data for root cause analysis.

Before considering the rules consolidated, the domain experts are expected to review the obtained fault trees. To this purpose each rule is translated in a XML standard representation of a fault-tree, which can further elaborated, discarded and accepted by a domain expert by means of a graphical fault-tree editor[5]. It is important to note that the reviews made by experts are provided as inputs to the clustering step, which accepts the list of both *accepted* and *discarded* rules (in order to prevent that rules already analyzed by experts will be reviewed again). Further, if the experts identify new primary events during the analysis, they will be included within the *known interesting events* list. The accepted trees will be then provided to the Fault-Tree Model Compiler to translate the trees in filtering rules, which can be interpreted by the Log File Analyzer. As already mentioned, at this stage of the project we start addressing the design and implementation of the AILA component, and leverage an existing editor as FTE; FTMC, FTD and LFA components will be addressed later during the project lifetime.

5 Preliminary case study

As a preliminary case-study, we collected two weeks of diagnostic events logs related to a single train during operation by means of the train control room facility at our industrial partner. The collected dataset is not labeled; therefore, no information is provided about the relationship of the events.

Before providing the dataset to the AILA component, we execute a **data preparation** step, in order to transform the data in the format expected by

[5] We use FTEdit as fault-tree editor (https://github.com/ChuOkupai/FTEdit) and the Open-PSA Model Exchange Format as XML representation.

AILA as well as to address suggestions provided by our industrial partner. First, the raw output of the diagnostic systems is converted in a table, containing for each event: a timestamp, the event-type and the event-code (which are described in the train model of the specific train), a description in natural language of the event and the name of the component affected. The obtained table is then analyzed to discard duplicated events (i.e., events with the same event-code and timestamp), according to the train control room team indications.

The dataset obtained after the data preparation step is composed by 57,653 events. In order to properly configure the AILA component, we first performed a sensitivity analysis in order to select the tupling window the AILA should use for the tupling step. Fig. 4 shows the number of tuples by the selected tupling window. According to a previous study [4], a good tradeoff is represented by the knee point of the curve. Therefore, we obtained a tupling window of 5 seconds, corresponding to 5,744 tuples. The AILA component is then configured with the obtained tupling window, and it is executed on the prepared dataset. The clustering step performed by AILA generates 970 clusters. Only 348 clusters contains more than a single tuple, and only 249 contains at least two diagnostic events. Therefore, AILA considers the 249 remaining clusters as potential filtering rules learned in an automatic way, with a reduction of about 95% with respect to the number of tuples. Fig. 5 shows the distribution of the number of events for each learned rule. The 70% of the total rules learned is composed by less than 5 events. Rules with more than 20 events represent only less than 7%.

In our preliminary study we also analyzed the effect of the tupling window on the number of potential rules obtained from the clustering step. Fig. 6 shows that by varying the tupling window from 1 to 18 seconds, the number of potential rules ranges between 228 and 278, with an average of 253 rules, which is quite near to the 249 rules obtained with the selected tupling window of 5 seconds.

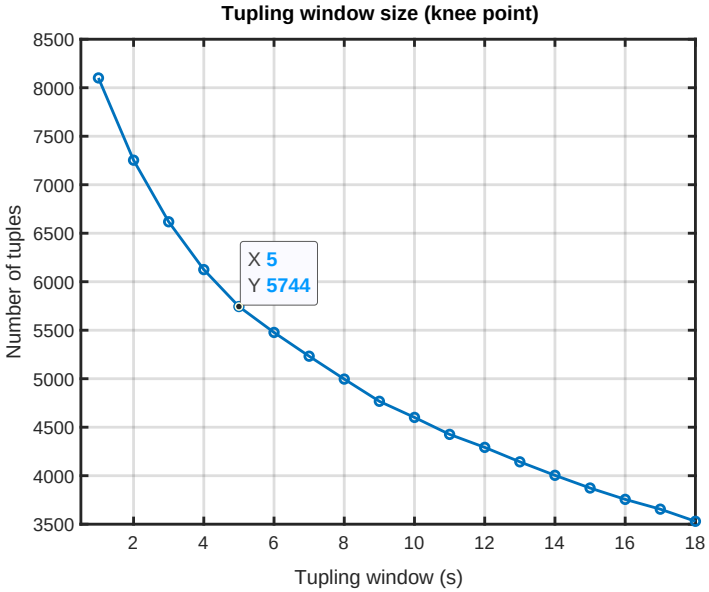

Fig. 4. Tupling windows size (knee point analysis).

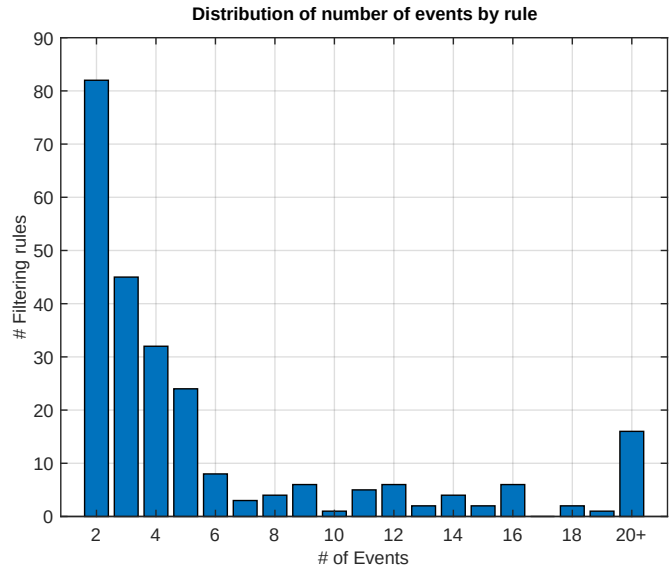

Fig. 5. Distribution of the number of events in each learned rule.

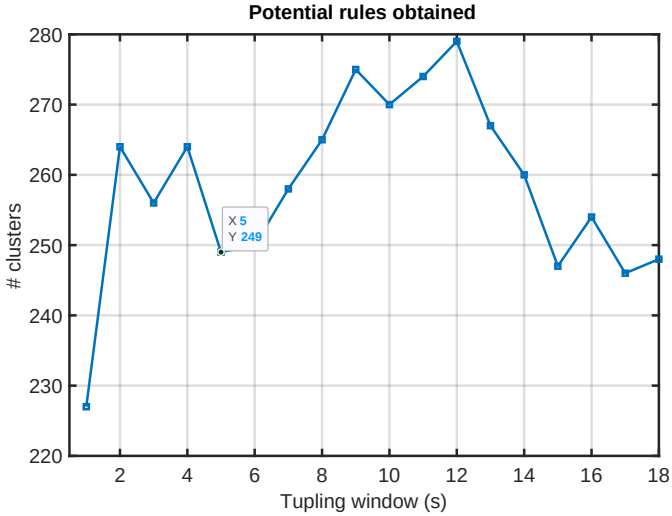

Fig. 6. Number of potential rules learned by tupling window selection.

As indicted in Section 4, the resulting clusters are then converted in the Open-PSA fault-tree XML representation. As an example, the code fragment in Listing 1.1 and the corresponding graphical output in Fig. 7 show the tree representation of a rule inferred by a cluster, and which should be analyzed by a domain expert. To facilitate the analysis of the expert, each node of the tree (see Fig. 7) is enriched with the following information:

- an *eventID*, which is a combination of event name and event code extracted from the train model of the system, e.g., `017-_CloseIR_`;
- the *time window* (for the root event only), i.e., @window parameter, indicating the maximum time in which the rule applies;

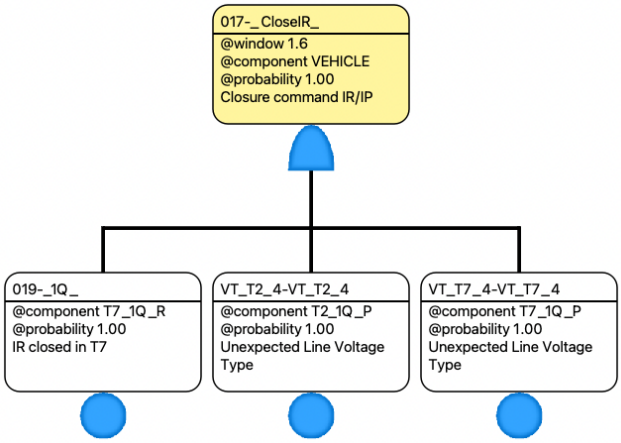

Fig. 7. Example of graphical representation of a learned rule.

- the *component* to which this event belongs to, i.e., @component parameter;
- the *probability* of occurrence, i.e., @probability parameter, which is defined as the ratio of number of times the event occurred in the cluster and the total number of tuples of the cluster;
- a natural language *description* of the event.

In the provided example, the cluster includes the events `017`, `019`, `VT_T2_4` and `VT_T7_4`, which have been grouped together by AILA. These events happened at least two times within a time window of 1.6 seconds. The event type `CloseIR (017)` has been indicated by the domain expert as a potential interesting event (through the dedicated list provided to AILA). Therefore, it has been placed at the root of the fault-tree. The obtained tree represented an example of rule obtained in an automatic way by using the AILA component, and that can be analyzed and modified by experts through the FTE component. In this respect, AID4TRAIN framework allows to obtain potential filtering rules from historical data in an automatic way as well as to combine the view provided by logs with the expertise of the analysts.

Listing 1.1. Fault-tree XML Open-PSA Representation.

```
<opsa-mef author="AID4TRAIN">
    <define-basic-event name="019-_1Q_">
      <label>
        @component T7_1Q_R
        @probability 1.00
        IR closed in T7
      </label>
      <attributes><attribute value="false" name="keep"/>
      </attributes>
    </define-basic-event>
    <define-basic-event name="VT_T2_4-VT_T2_4">
      <label>
        @component T2_1Q_P
        @probability 1.00
        Unexpected Line Voltage Type
      </label>
      <attributes><attribute value="false" name="keep"/>
      </attributes>
    </define-basic-event>
```

```
20      <define-basic-event name="VT_T7_4-VT_T7_4">
21        <label>
22          @component T7_1Q_P
23          @probability 1.00
24          Unexpected Line Voltage Type
25        </label>
26        <attributes><attribute value="false" name="keep"/>
27        </attributes>
28      </define-basic-event>
29      <define-fault-tree name="471">
30        <attributes><attribute value="017-_CloseIR_" name="top-event"/
            ></attributes>
31        <define-gate name="017-_CloseIR_">
32          <label>
33            @window 1.6
34            @component VEHICLE
35            @probability 1.00
36            Closure command IR/IP
37          </label>
38          <and>
39              <basic-event name="019-_1Q_"/>
40              <basic-event name="VT_T2_4-VT_T2_4"/>
41              <basic-event name="VT_T7_4-VT_T7_4"/>
42          </and>
43        </define-gate>
44      </define-fault-tree>
45  </opsa-mef>
```

6 Related Work

Event logs include empirical evidence about the errors occurred in a software system, hence, log analysis is efficient in classifying the propagation of errors and failure modes. Through heuristic tupling, the log errors are coalesced in tuples in order to associate them with a failure mode. For instance, error events occurring close in time are coalesced to represent a single failure mode. The validity of heuristic models for time coalescence in event logs is discussed by Hansen et al. [6]; their sensitivity analysis is also adopted in this paper. Spatial coalescence heuristics are adopted in the analysis of larger systems such as data-centers and supercomputers [15] [9]. Collisions are the main issue of tupling heuristics: errors of different failure modes are associated to the same failure mode, for instance, due to the tuning of the time window or because the chance that independent failures occur on different nodes is not negligible. In this paper, the tupling heuristics are adopted to identify independent log events root causes. Once we get the tuples, we adopt the clustering to support the tuples.

Clustering techniques group the objects that are similar between them and dissimilar between the objects of other clusters. Clustering methods includes Hierarchical and Partitional clustering [21]. Partitional clustering defines at-priori the number of clusters, and searches the partition that maximizes a given function cost. For instance, k-means [12] tries to minimize the total intra-cluster variance at each iteration. Hierarchical clustering can be divisive or agglomerative. Initially, agglomerative algorithms assume that each cluster (leaf) contains a single object; subsequently, at each step, the "closest" clusters are joined to get a larger cluster. Measures of similarity between clusters are necessary to link the clusters. Linkage methods, such as Single-link, Average-link

and Complete-link, calculate the inter-cluster distance considering all combinations of points between the two clusters [17] [18], while geometric methods adopt geometric centers to represent the clusters, and to calculate the inter-cluster distance. For instance, the Ward's method is a geometric method which minimizes the intra-cluster variance. Divisive algorithms instead initially consider a partition formed by a single large cluster containing all the elements, and then divide it iteratively. Divisive clustering, as opposed to agglomerative ones, needs to measure the density or sparsity of points within a cluster to decide whether or not to proceed with the division.

Fault Tree (FT) is a well-known method to model the propagation of component failures in the system. FT is a tree, or more generally a directed acyclic graph, composed of one Top-Level Event (TLE, the root of the tree) and several intermediate events and basic events (the leaves). The events of an FT are linked though Boolean gates generating new intermediate events [16]. By describing the FT as Disjunctive Normal Form (DNF), each conjunction represents a cut-set, and each cut-set is a root cause of the TLE. Usually, the FT is adopted in FTA (Fault Tree Analysis) to qualitatively decompose the system failure in a hierarchical structure in order to identify the possible root causes as cut sets, or to quantify system dependability attributes. Building a fault tree requires a lot of manual effort, however, several studies discussed the possibility to build a fault tree from observational data making assumptions on the data set and conducting statistical tests [14] [13] [10] [11]. Nauta et al. [13] is the first completely automated tool to test the causality in the FT construction statistically.

We adopt the Fault Tree to model the filtering rules to detect the root causes in diagnostic logs. In that case, we are not interested in the accuracy of the fault tree in terms of a precise reproduction of the real hardware structure.

7 Conclusion

The paper described the key aspects underlying the AID4TRAIN project, which will provide Artificial Intelligence and data analytics approaches for supporting and improving the root cause analysis in Industry 4.0 and Railway domains. The paper introduced the software framework envisioned by the project, the design and implementation of its AI-based component, and a preliminary railway case. Future work will focus on the extensive implementation and validation of the AID4TRAIN framework in the context of real-world case study.

Acknowledgment

This work has been supported by the Meditech Competence Center under the AID4TRAIN Project (I65F21001010005).

References

1. Abuhasel, K.A., Khan, M.A.: A secure industrial internet of things (IIoT) framework for resource management in smart manufacturing. IEEE Access **8** (2020)

2. Chuah, E., Jhumka, A., Browne, J.C., Barth, B., Narasimhamurthy, S.: Insights into the diagnosis of system failures from cluster message logs. In: 11th European Dependable Computing Conference (EDCC). pp. 225–232 (Sept 2015)
3. Cinque, M., et al.: An empirical analysis of error propagation in critical software systems. Empirical Software Engineering **25**(4), 2450–2484 (2020)
4. Di Martino, C., Cinque, M., Cotroneo, D.: Assessing time coalescence techniques for the analysis of supercomputer logs. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). pp. 1–12. IEEE (2012)
5. Goikoetxea, J.: Shift2rail connecta: The next generation of the train control and monitoring system. Zenodo (Apr 2018). https://doi.org/10.5281/zenodo.1421620
6. Hansen, J.P., Siewiorek, D.P.: Models for time coalescence in event logs. [1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing pp. 221–227 (1992)
7. Kalyanakrishnam, M., Kalbarczyk, Z., Iyer, R.K.: Failure Data Analysis of a LAN of Windows NT based Computers. In: Proceedings of the International Symposium on Reliable Distributed Systems (SRDS 99). pp. 178–187. IEEE Computer Society
8. Lal, H., Pahwa, G.: Root cause analysis of software bugs using machine learning techniques. In: 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence. pp. 105–111. IEEE (2017)
9. Liang, Y., Zhang, Y., Sivasubramaniam, A., Jette, M.A., Sahoo, R.K.: Bluegene/l failure analysis and prediction models. International Conference on Dependable Systems and Networks (DSN'06) pp. 425–434 (2006)
10. Linard, A., Bucur, D., Stoelinga, M.: Fault trees from data: Efficient learning with an evolutionary algorithm. ArXiv **abs/1909.06258** (2019)
11. Linard, A., Bueno, M.L.P., Bucur, D., Stoelinga, M.: Induction of fault trees through bayesian networks. Proceedings of the 29th European Safety and Reliability Conference (ESREL) (2019)
12. MacQueen, J.: Some methods for classification and analysis of multivariate observations (1967)
13. Nauta, M., Bucur, D., Stoelinga, M.: Lift: Learning fault trees from observational data. In: QEST (2018)
14. Nolan, P.J., Madden, M.G., Muldoon, P.R.: Diagnosis using fault trees induced from simulated incipient fault case data (1994)
15. Oliner, A.J., Stearley, J.: What supercomputers say: A study of five system logs. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07) pp. 575–584 (2007)
16. Ruijters, E., Stoelinga, M.: Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15**, 29–62 (2015)
17. Sneath, P.H.A.: The application of computers to taxonomy. Journal of general microbiology **17 1**, 201–26 (1957)
18. Sørensen, T., Sørensen, T., Biering-Sørensen, T., Sørensen, T., Sorensen, J.T.: A method of establishing group of equal amplitude in plant sociobiology based on similarity of species content and its application to analyses of the vegetation on danish commons (1948)
19. Tian, J., Rudraraju, S., Li, Z.: Evaluating web software reliability based on workload and failure data extracted from server logs. Soft Eng, IEEE Transactions on **30**(11), 754–769 (Nov 2004). https://doi.org/10.1109/TSE.2004.87
20. Xu, D., Tian, Y.: A comprehensive survey of clustering algorithms. Annals of Data Science **2**(2), 165–193 (2015)
21. Xu, R., Wunsch, D.C.: Survey of clustering algorithms. IEEE Transactions on Neural Networks **16**, 645–678 (2005)