Information and Software Technology 197 (2026) 108166

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

What you model is what you get: A model-driven dashboard generation approach

Maria Teresa Rossi [a,*], Alessandro Tundo [b], Leonardo Mariani [a]

[a] Department of Informatics, Systems and Communication, University of Milano-Bicocca, viale Sarca 336, Milan, 20126, Italy
[b] Institute of Information Systems Engineering, Technische Universität Wien, Favoritenstrasse 9-11, Vienna, 1040, Austria

ARTICLE INFO

Keywords:
Dashboard generation
Model-driven engineering
KPI
Grafana

ABSTRACT

Context: Dashboards play a pivotal role in cloud systems monitoring, as they facilitate the visualization of the Key Performance Indicators (KPIs) that are continuously gathered from the system under observation. To timely and easily identify malfunctions and unexpected behaviors, operators have to configure, design, and maintain dashboards, so that the right set of indicators is properly visualized. Unfortunately, cost-effectively manipulating dashboards is a challenge, also for experts.
Objectives: This paper proposes a model-driven approach that supports both the cost-effective definition (*generation*) and modification (*adaptation*) of dashboards.
Method: The key idea is that a model-driven representation of a dashboard can be more easily manipulated than interacting with the GUI of dashboard management systems. Once a dashboard's model is defined, the actual dashboard can be generated automatically with model-transformation techniques.
Results: Our empirical results with popular Grafana Labs and Dynatrace dashboards show that the interpretability of the dashboards generated automatically is similar to the one of the manually configured dashboards. Moreover, the model-driven customization of the dashboard allows non-expert operators to act more efficiently, sometime as efficient as expert users.
Conclusions: Overall results show that the model-driven approach can be used to cost-effectively generate useful dashboards, with an effectiveness close to that of experts.

1. Introduction

Motivation and challenges. It is well-known that the constituent services of cloud systems may exhibit failures and unexpected behaviors, which can have difficult-to-predict consequences for the entire system [1,2]. To timely detect these issues, operators can exploit dashboards, which visualize Key Performance Indicators (KPIs) that are continuously collected from the running services and components. As such, dashboards are a pivotal visual-based tool for analysis and assessment of cloud systems.

There are several tools that can be used to implement and configure dashboards. Among the most popular, there are Grafana,[1] Dynatrace,[2] and Kibana,[3] which can flexibly visualize the data stored in time series databases through a collection of visualizations and panels. Unfortunately, configuring these dashboards could be particularly cumbersome and time consuming [3–5]. Creating or updating a dashboard may require the execution of long sequences of interactions with the GUI pages designed to manipulate visualizations. Moreover, dashboards often need to be updated, to reflect new requirements about visualizations or to visualize newly collected data [6].

A fundamental challenge, and the main motivation for our work, lies in streamlining both the *generation* (i.e., automated, model-driven creation of dashboards from abstract representations, typically at design-time) and the *adaptation* (i.e., customizations applied during system operation or in response to evolving requirements). Current solutions often concentrate on only one aspect: some approaches address automatic dashboard generation but provide limited support for subsequent customization by operators, while others expect operators or administrators to design or adapt visualizations manually, which is resource-intensive and error prone [3,5,7–10].

Research question. Our work addresses the discussed research challenges driven by the following main research question: "*Can operators*

* Corresponding author.
E-mail addresses: maria.rossi@unimib.it (M.T. Rossi), alessandro.tundo@tuwien.ac.at (A. Tundo), leonardo.mariani@unimib.it (L. Mariani).

[1] https://grafana.com/.
[2] https://www.dynatrace.com.
[3] https://www.elastic.co/kibana.

https://doi.org/10.1016/j.infsof.2026.108166
Received 8 October 2025; Received in revised form 22 April 2026; Accepted 23 April 2026
Available online 12 May 2026

with basic MDE skills and little prior dashboard experience, efficiently generate and flexibly adapt dashboards achieving results comparable to dashboard experts?"

Contribution. We propose a model-driven approach and supporting tool that (1) enables the *automatic generation* of dashboards from a minimal specification, namely, the set of KPIs to be visualized and the queries needed to retrieve the KPIs from a target time series database; (2) allows operators to *cost-effectively adapt* the generated dashboards, modifying their structure and content as needs evolve; and (3) generates a *technology-agnostic representation* of a dashboard that can be ported to different dashboard target platforms. In our vision, operators can obtain an initial, well-structured dashboard and subsequently refine it to accommodate specific needs, using intuitive model-editing operations rather than repetitive, low-level GUI interactions. Further, they can visualize the generated dashboard on the preferred dashboard target platform.

We evaluated the proposed dashboard generation approach with respect to two different aspects, namely, the *comprehensibility* of the automatically generated dashboards compared to dashboards designed by experts, and the *flexibility* in dashboard adaptation granted by our approach to operators with knowledge in MDE, compared to both non-experts and dashboard experts.

Concerning comprehensibility, results of two user-studies with 20 professionals and 10 researchers show little differences in the accuracy of the answers to questions that require the inspection of the manually created and automatically generated dashboards to be answered. This result confirms the intuition that the heuristics used in the dashboard creation process are likely to satisfy the expectations of the operators.

Concerning flexibility, we observed that tasks performed with our model-based tools can be completed faster than using the GUI of dashboards, and in some specific cases even faster than experts who know how to reconfigure a dashboard via code. This result confirms that our approach can be used to make operators highly productive when facing new dashboard technologies.

This paper extends our previous work [11] on this topic, and provides the following key contributions:

- A rigorous description of the proposed model-driven approach that originally combines automatic dashboard generation, operator-centric adaptation, and a technology-agnostic solution;
- An empirical evaluation with actual Grafana Labs and Dynatrace Playground dashboards involving professionals and researchers, focusing on comprehensibility and time savings;
- A tool implementation and the associated reproduction package [12].

Structure of the paper. The paper is structured as follows. Section 2 discusses related work. Section 3 describes the challenges of configuring monitoring dashboards. Section 5 presents the proposed MDE approach for the generation of dashboard. Section 6 describes the empirical evaluation about the comprehensibility and time savings of the generated dashboard. Section 7 provides final remarks.

2. Related work

To position our contribution, we analyze existing approaches along three key dimensions: the level of automation in dashboard generation, the support for systematic dashboard customization, and the technology-independence of the generated dashboard.

Automatic dashboard generation. Da Col et al. [13] propose DashBot, a system that leverages Multi-Armed Bandits to guide operators in the creation of data-driven customized dashboards. It achieves this by iteratively recommending the most relevant groups (panels), which can be refined based on feedback and explanations of the operators. DashBot is primarily intended for interactive dashboard construction and relies on multiple iterations to complete the exploration and exploitation cycles of the Multi-Armed Bandits algorithm, thus limiting full automation and requiring continuous user involvement.

Belo et al. [9] and Dabbebi et al. [14] present two distinct methods for enabling the construction of personalized dashboards tailored to users' needs. The first method employs agents to observe users' behavior while they interact with dashboards, identifying usage patterns and automatically rearranging the dashboards. The second method introduces a conceptual model for developing learning analytics dashboards that can be customized based on the user's context and requirements. These methods focus on refining and personalizing an already existing dashboard rather than supporting its systematic generation from a minimal and abstract specification.

In the smart cities scenario, Santos et al. [10] propose an approach for automatically suggesting KPIs to be displayed using knowledge graphs and ontologies. However, the operator remains responsible for selecting the most appropriate visualizations and customizing the dashboard. Deng et al. [15] propose deep reinforcement learning to generate analytical dashboards by leveraging visualization knowledge. Their approach utilizes a training environment where learning agents can explore and imitate human behavior in the dashboard generation process. However, in both cases the operator remains heavily involved, as significant manual input is still required for visualization selection and dashboard configuration.

To partially overcome this limitation, Tundo et al. [3] propose a declarative approach to automatically create dashboards, envisioning the use of meta-model layouts to determine the visualization style when organizing a set of input KPIs. However, the paper only anticipates developing a technology-agnostic solution in future work, and the proposed approach simply focuses on visual features, with little actual usage of models.

Overall, these approaches emphasize data-driven or learning-based recommendation, but they lack an explicit and adaptable model of the dashboard. As a consequence, automation is often achieved at the cost of limited transparency, limited controllability, and reduced support for systematic customization and adaptation.

MDE for dashboard generation. Erazo-Garzon et al. [8] present an approach to implement IoT dashboards as web applications, leveraging MDE techniques. Specifically, they define a Domain-Specific Language (DSL) for modeling dashboards and their visualizations. In this context, the operator is responsible for choosing the type of visualization to display. Additionally, the authors envisioned a specific ecosystem tied to a limited set of data sources and formats. The same is implemented by Kintz et al. [4]. The paper develops an approach in the context of business process monitoring. In both cases, the operator is responsible for explicitly specifying key visualization choices, and the proposed solutions are tightly coupled to specific application domains and data ecosystems.

Vázquez-Ingelmo et al. [5] present a dashboard generator that relies on the software product line paradigm to retrieve dashboard features. The paper proposes a configuration process that considers operators' goals and characteristics to infer high-level visualization features, such as the number of channels or visualizations within a dashboard, and then uses visual mapping to convert this abstract information into concrete visualizations. The paper highlights the benefits of using meta-modeling techniques to describe abstract dashboards, as this supports visual mapping and ensures that basic guidelines and constraints are followed. Moreover, they confirm the advantages of developing a method based on a template-driven generator engine. However, in these approaches the responsibility for specifying the target dashboard remains largely with the operator, limiting the level of automation achievable in practice.

Jiang et al. [16] propose a framework for model-driven dashboard generation to quickly model, prototype, and validate dashboard designs. Further, they propose a GUI-driven customization approach that,

Table 1
Comparison of existing approaches in terms of automation, systematic customization, and technology-independence.

Paper	Automation	Adaptation	Tech. Ind.
Da Col et al. [13]	**Automatic** using Multi-Armed Bandits **from dataset schema**	GUI only	–
Belo et al. [9]	**Automatic** using usage patterns mining from full specification	GUI only	–
Dabbebi et al. [14]	Semi-automatic using contextual models from full specification	GUI only	–
Santos et al. [10]	**Automatic** using knowledge graph from full specification	–	–
Deng et al. [15]	**Automatic** using deep reinforcement learning from full specification	GUI only	–
Tundo et al. [3]	**Automatic** using meta-model layouts from full specification	GUI only	**Yes**
Erazo-Garzon et al. [8]	**Automatic** using DSL models from full specification	**Full dashboard**	–
Kintz et al. [4]	**Automatic** using process and KPI models from full specification	GUI only	–
Vázquez-Ingelmo et al. [5]	Semi-automatic using software Product Line from full specification	GUI only	–
Jiang et al. [16]	**Automatic** using multi-level interpreters from full specification	GUI only	–
Our Approach	**Automatic** using M2T transformations **from KPIs list**	**Full dashboard**	**Yes**

by means of visual widgets, helps users continuously improve the dashboard design with a low-code perspective. This work shows that model-driven technologies are promising for the automatic generation of dashboards, as they effectively capture real-world design decisions and facilitate dashboard creation.

While these works confirm the suitability of MDE for dashboard generation, they mainly treat models as intermediate artifacts for code generation or prototyping. In contrast, dashboard models can be exploited as first-class runtime artifacts, explicitly designed to support both generation and cost-effective customization and adaptation.

Table 1 summarizes the discussed approaches according to the three identified aspects, highlighting how our approach combines automation, systematic customization, and technology-independence. We highlight approaches that are (i) automatic, (ii) do not require a full specification to produce a dashboard, (iii) generate the full dashboard implementation and not only a GUI detached from data sources, and (iv) the technology independent solutions. our approach provide a unique combination of features.

Beyond dashboard-specific solutions, low-code and no-code platforms represent a broader class of approaches aimed at simplifying application development through visual modeling. Low-code platforms such as *BESSER* [17], Mendix,[4] OutSystems[5] and Microsoft PowerApps[6] promise higher productivity through visual drag-and-drop modeling. Although marketed for citizen developers, most of these platforms inherit core principles from MDE's abstract syntax definition, model validation, and code generation, confirming that low-code can be viewed as a MDE approach with a modern user experience [18]. Whereas purely GUI-based solutions excel in usability, an explicitly model-driven foundation offers stronger abstraction, formalization, and automation. We therefore view MDE and Low-/No-code as complementary: the latter paradigm lowers the entry barrier, while the former paradigm guarantees rigor and extensibility. Our work adopts this dual stance by coupling a rigorous MDE backbone with interactive editors, thereby balancing automation and user control.

[4] https://www.mendix.com.
[5] https://www.outsystems.com.
[6] https://www.microsoft.com/en-us/power-platform/products/power-apps.

3. The challenges of configuring monitoring dashboards

A cloud monitoring system is usually composed of four primary components: (i) a set of probes to collect data of Key Performance Indicators (KPIs) for the resources under observation, (ii) a time series database to store the collected data, (iii) a data transfer channel for processing and transferring data from the probes to the database, and (iv) a set of dashboards to access and visualize the KPIs [19].

Let us consider the case of monitoring a computing cluster running Kubernetes,[7] the market leader system for automating deployment, scaling, and management of containerized applications. The set of KPIs collected for the Kubernetes internal components sum up to 53 KPIs (i.e., stable and beta level),[8] while the set of KPIs representing the state of Kubernetes objects[9] (i.e., the health of the various objects inside the cluster, such as deployments, nodes and containers) are more than 200. To visualize such KPIs, dashboards can be created by means of dashboard platforms such as Grafana,[10] Dynatrace,[11] or Kibana,[12] which in some cases offer pre-defined dashboards to visualize the (sub)set of the collected KPIs for common tools or systems (e.g., Kubernetes).

However, this is not always the case, and pre-defined dashboards may not always be compatible with the diverse monitoring needs of different organizations or businesses. This necessity gives rise to the requirement for customized dashboards that address a substantial number of KPIs and visualizations. To illustrate, we consider the process of visualizing KPIs using Grafana.

In particular, Fig. 1 reports the main tasks involved in the creation of a dashboard extracted by the Grafana documentation.[13] This process envisages a main flow and various sub-flows. A dashboard is composed mainly by panels that contain charts and can be organized in rows.

[7] https://kubernetes.io/.
[8] http://bit.ly/3I5MkCt.
[9] http://bit.ly/4l7kILE.
[10] https://grafana.com.
[11] https://www.dynatrace.com.
[12] https://www.elastic.co/kibana.
[13] https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/create-dashboard/.

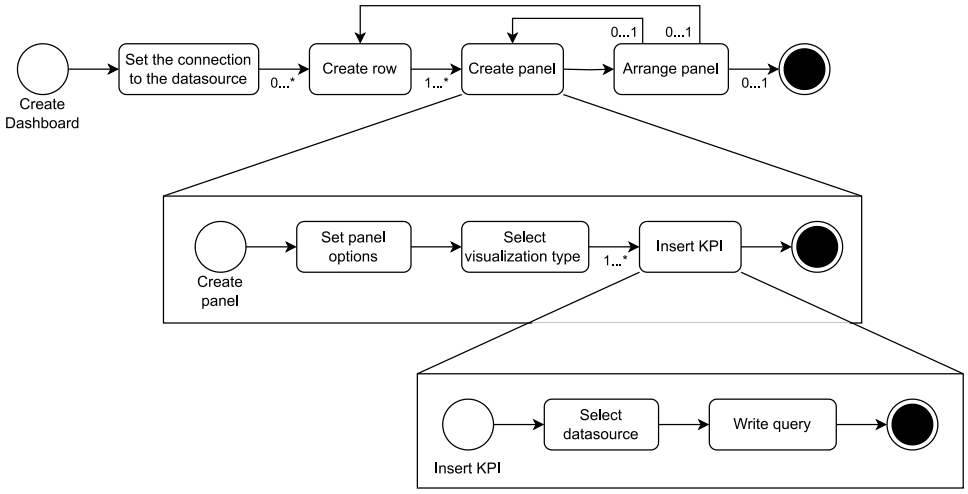

Fig. 1. Overview of the dashboard creation process on the Grafana platform.

For this reason, the main flow starts by setting the data source where data is taken from, and then continues with the definition of the rows that must include the dashboard's panels. Every panel, once created, has to be properly positioned in the dashboard, otherwise it would be positioned by default on the top. The positioning of the panels has to be performed manually by the operator[14] by dragging and dropping elements. The creation of every panel implies the customization of the panel options (e.g., title, description, series colors) and the choice of the visualization type. Further, during the creation of the panel, the operator has to specify the shown KPIs. In particular, when creating a panel, to insert a KPI, the operator must select the data source from which to retrieve the data and write the query needed to extract it. These two tasks have to be repeated for every KPI the operator wants to visualize in any panel. A same workflow of activities must be completed to obtain the same result with other dashboard platforms, such as Dynatrace.

When a large number of KPIs are under consideration, as in the case of monitoring a Kubernetes cluster previously outlined, the execution of such tasks can become repetitive and cumbersome, impeding the capacity for prompt adaptation to the evolving demands of monitoring. Moreover, operators involved in monitoring a cloud system may have different expertise and varying roles, such as business managers interested in high-level monitoring of the cloud services offered by their company, or engineers focused on performance monitoring [20]. For this reason, using dashboard platforms like Grafana and Dynatrace can be challenging for management-level operators who may lack the expertise to leverage its advanced functionalities effectively. This limitation can hinder the ability of all the operators to select and query the KPIs of interest efficiently.

We distill four main challenges in the manual configuration of monitoring dashboards by considering the flow and the type of the required tasks involved in the process, and the different skill set and knowledge requirements for the operators:

- `C1` - **Task Repetitiveness**: For dashboards of large size with a high number of rows and panels, the operator has to perform many times the same manual tasks. This dramatically slows down the process of setting up and managing dashboards.
- `C2` - **Evolution & Adaptation**: Modifying dashboards to respond to changing requirements often involves manual effort, such as re-aligning panels or rewriting queries to accommodate new KPIs. This process can be cumbersome and tedious.
- `C3` - **Levels of Complexity**: Creating manual dashboards requires accomplishing tasks with varying levels of complexity. For instance, tasks such as establishing the connection to the data source and writing queries require technical knowledge about how to connect and query data sources. This aspect might completely prevent non-experts to craft their own dashboards, and reduces the time expert operators can dedicate to styling and fine-tuning.
- `C4` - **Logical Organization**: The need to manually position panels can introduce inconsistencies or make dashboards harder to read. Moreover, poor organization caused by inexperience in visual design may reduce the overall clarity and usability of the dashboard.

The aforementioned challenges demonstrate the core difficulties in configuring monitoring dashboards for cloud systems, thereby prompting the question of whether an automated approach can provide significant advantages over manual methods. Indeed, an automated dashboard generation approach has the potential to reduce the time required to create and modify dashboards, particularly for large and complex systems. This approach can eliminate slow, repetitive, and cumbersome tasks while enabling adaptation to evolving operator needs, and enhancing flexibility and responsiveness. Furthermore, it has the potential to reduce the learning curve for non-experts and to leverage templates that facilitate better organization, thereby enhancing the interpretability of dashboards.

To this end, we propose a model-driven approach to address three main *goals* (i) automatically generate dashboards from a minimum set of information, that is, the set of KPIs to be visualized and the specification of the query to extract the KPIs to be visualized from a target time series database; (ii) allow the cost-effective customization of the automatic result, including the structure and content of visualizations, to satisfy any specific need of the operators that may escape the automatic generation process; and (iii) support portability to different web dashboarding platforms ensuring technology-independence.

To assess the comprehensibility and flexibility of the proposed approach and to compare it with manually created dashboards, we investigate two *research questions* (RQs):

RQ1 (Comprehensibility) - To what extent the comprehensibility of dashboards automatically generated by a model-driven approach are comparable to manually defined dashboards? This research question investigates the comprehensibility of dashboards automatically generated through a model-driven approach in comparison to manually created dashboards with two user studies.

RQ2 (Flexibility) - How fast can automatically generated dashboards be created and adapted to changing operator's needs?

[14] We define the operator as any user performing tasks on a dashboard to monitor a cloud system.

This research question investigates the flexibility and practicality of a model-driven approach in comparison to creating and modifying dashboards using the native platform.

4. Design rationale of the proposed approach

The approach was designed through a structured and iterative process grounded in (i) the analysis of practical dashboard configuration challenges, (ii) established principles from model-driven engineering (MDE), and (iii) guidelines from the information visualization literature.

The identification of the challenges (C1–C4) presented in Section 3 was informed by three complementary sources. First, we systematically analyzed the documented workflow of widely adopted dashboard platforms (e.g., Grafana), reconstructing the sequence of configuration tasks required to define panels, queries, visualizations, and layout structures. Second, we considered representative cloud monitoring scenarios (e.g., about Kubernetes clusters) characterized by a high number of KPIs and heterogeneous operator roles, highlighting issues of scalability, repetitiveness, and usability. Third, we drew on existing literature on dashboard usability and monitoring complexity [21], which emphasizes cognitive load, interpretability, and the need for consistent organization of indicators.

In parallel, the definition of visualization selection heuristics was informed by established visualization principles [22], including semantic coherence, perceptual effectiveness, and consistency of quantitative comparison. In particular, guidelines recommending the grouping of related measures, the preservation of unit homogeneity within the same visualization, and the alignment between data characteristics (temporal, categorical, scalar) and visualization types were explicitly considered during the design of the automatic visualization rules.

Overall, the design of the approach was grounded in core MDE principles, such as abstraction, separation of concerns, explicit metamodeling, and model-to-text transformation. These principles guided the decision to separate (i) the abstract specification of KPIs and dashboard structure from (ii) the platform-specific generation logic.

The steps of the proposed approach were defined by systematically mapping the identified challenges (C1–C4) to concrete design decisions.

- **Addressing Task Repetitiveness (C1)** led to the introduction of an abstract KPI modeling phase, where indicators are specified once at the model level and later reused during generation. This reduces repeated manual configuration at the platform level.
- **Addressing Evolution and Adaptation (C2)** motivated the separation between automatic generation and subsequent model-level customization. By treating the dashboard model as a first-class artifact, structural changes can be performed declaratively and propagated automatically to the generated dashboard.
- **Addressing Levels of Complexity (C3)** led to the abstraction of technically demanding tasks (e.g., panel configuration) into higher-level modeling constructs, thus lowering the operational burden while preserving expert-level control.
- **Addressing Logical Organization (C4)** informed the definition of grouping and layout rules, ensuring consistent structural organization of KPIs and visualizations.

The approach reflects a progressive abstraction-to-concretization workflow, consistent with MDE practices. First, KPIs and structural information are specified at an abstract level through the metamodel. This corresponds to a domain-level specification independent of any target platform. Second, automatic rules derive visualization groupings and layout structures based on semantic and data-type information. This intermediate phase enriches the abstract model with derived elements while maintaining platform independence. Finally, model-to-text transformations generate the concrete dashboard configuration for a specific platform (e.g., Grafana). This step operationalizes the abstract specification into executable artifacts. This sequencing ensures traceability between high-level modeling decisions and the generated dashboard artifacts, while preserving the possibility of manual refinement at the model level before code generation.

We did not defined the approach in a single step but we refined it iteratively. We evaluated intermediate versions of the metamodel and generation rules against realistic monitoring scenarios to verify their ability to (i) reduce repetitive configuration effort, (ii) preserve interpretability of the resulting dashboards, and (iii) maintain sufficient flexibility for customization. This iterative refinement process led to simplifying certain modeling constructs and privileging transparent, explainable generation heuristics over more complex or opaque automation strategies. The resulting artifact therefore represents a deliberate balance between automation, expressiveness, and usability.

Overall, the proposed approach can be interpreted as an instantiation of model-driven principles applied to dashboard engineering, where practical challenges, theoretical foundations, and iterative refinement jointly shaped the final artifact.

5. Model-driven generation of dashboards

We designed an approach for model-driven dashboard generation based on the activities shown in Fig. 2. Specifically, the activities in yellow are performed by the final user, and the blue ones represent the automated generative steps of our approach. The rectangles represent input/output data.

The process starts with the *KPIs Definition* step, which consists of the manual definition of the list of KPIs that shall be visualized in the resulting dashboard. The KPIs are defined using a graphical modeling language in a *dashboard model*, which must conform to a *dashboard metamodel*. Unless the operator needs to customize the resulting dashboard, this is the only manual intervention needed.

The *Automatic Definition of KPIs Visualization* step automatically generates a complete definition of a dashboard, enriching the input dashboard model with the specification of the visualizations, their grouping, and placement. The only input is the list of KPIs defined in the previous step. The output is a *dashboard model with visualizations*, which must be complaint with the dashboard metamodel. Such a model is a technology-agnostic representation of the dashboard, that is, the visualizations are defined according to the dashboard metamodel, without referring to any specific dashboard platform.

The *automated dashboard generation* step uses a *platform-specific rule-based template* to obtain the *dashboard source code* from the input dashboard model with visualizations, that is, it generates the implementation of the dashboard for the target platform. This last step generates the actual ready-to-use dashboard code, for instance, the Grafana dashboard JSON model.

Note that the operators, after the *dashboard uploading*, are not forced to use the automatically generated dashboard, although it is quite practical to stay with the automatic result, but they can intervene manually modifying the dashboard model with visualizations obtained in the second step (*changes to visualizations* activity in Fig. 2).

The operators do not have to execute the main steps sequentially but can flexibly move back and forth across the activities adjusting the results as needed, as suggested by the decision nodes in the figures. This means that the modeling artifacts involved in the process can be manually changed without compromising the functioning of the whole approach. For instance, if the operator is not satisfied with the automatic grouping proposed by the *Automatic Definition of KPIs Visualization* step, she can manually modify it, and run the last activity of the approach, generating the actual dashboard. The same can happen to the primary dashboard model containing the list of KPIs to monitor. The operator can update such a list and re-run the whole approach.

In summary, following the process in Fig. 2 the user may (i) *automatically generate* the abstract representation of a dashboard (*dashboard model with visualizations*), (ii) *customize the structure and content* of

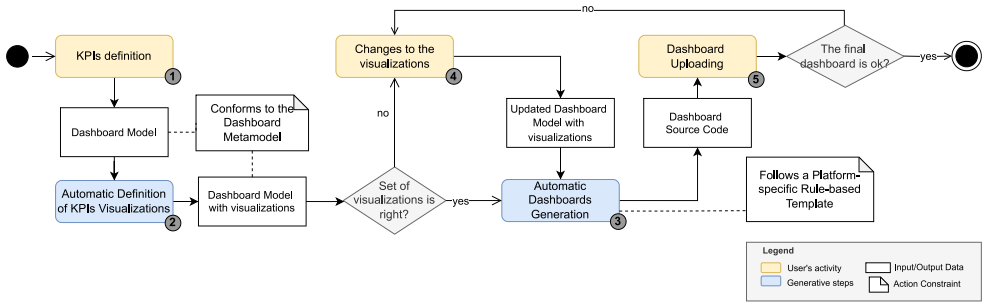

Fig. 2. Model-driven generation of dashboards.

the generated dashboard by updating the *dashboard model* and the *dashboard model with visualizations* respectively, and (iii) *generate the source code* of dashboard ready to be uploaded on the choosen target platform depending on the used *platform-specific rule-based template*.

In the next subsections, we describe the *dashboard metamodel* (Section 5.1) and then present each main generative step of the approach in detail (Sections 5.2 to 5.4). To facilitate the presentation, we use a running example based on the simple case of a service that continuously monitors the resources within a cloud environment, (e.g., virtual servers, databases, storage services). The service collects a wide range of performance metrics, such as CPU utilization, memory usage, network latency, service response speed and more, from all the monitored resources. In this context, we assume that such a service collects the *Estimated Error* in network latency for each monitored resource. This KPI collects the current value for the metric and compares it with its expected value by measuring how much they differ. Moreover, the considered service monitors another KPI, *Maximum Error*, that checks whether the network latency exceeds a predefined tolerable value. If the error exceeds this threshold, an alarm is triggered to alert the system administrators.

5.1. The dashboard metamodel

Palpanas et al. [23] and Erazo et al. [8] proposed metamodels to describe dashboards and facilitate their transformation into actual dashboards. However, these approaches do not allow the direct translation of dashboards into source code for dashboard platforms, nor they capture the separation between a technology-agnostic representation of a dashboard, and its concrete, technology-dependent, implementation. Having a meta-model that distinguishes these key concepts is of fundamental importance to support multiple platforms and allow the seamless collaboration of operations working at different abstract levels (e.g., less experienced operators working on the models with expert operators working directly on the platform-specific code). For these reasons, we designed a meta-model that can fully support the model-driven approach shown in Fig. 2, achieving the required degree of separation between the various concepts and abstractions.

Fig. 3 shows the proposed *dashboard metamodel*, which defines the concepts useful to represent a dashboard, such as, the visualizations, the KPIs and their categories, the queries to retrieve data from data sources, and the targets related to a KPI.

In details, when defining a `Dashboard`, the operator can choose its `structure`, that is, the criterion according to which organize its content (i.e., the visualizations). Specifically, the `StructureType` of a dashboard can be `categorical`, `targetOriented` and `custom`. The first two options allow for the automatic arrangement of the `Visualizations` in the dashboard with respect to the KPIs' `Category` and the `Target`, respectively. The category refers to the usual dimension used to group metrics according to the measured aspect (e.g., reliability, performance), while target refers to the resource or system component whose monitoring data are collected. Meanwhile, if the operator chooses a custom structure, she can create dashboards with visualizations of any `VisualizationType` (e.g., `barchart`, `timeseries`) and arrange them with respect to a custom set of `Groups`.

A `Visualization` is a visual representation of one or more coherent KPIs (e.g., all the KPIs about memory consumption collected from a given target). Visualizations are characterized by their type of representation and a unit of measure for the visualized KPIs.

A dashboard is connected to a `Datasource`, which has a `uid` and a `type` (e.g., `prometheus` or `graphite`). All the KPIs presented in a same dashboard are produced by a same datasource. For the sake of simplicity, we assume to use only one datasource per dashboard. We acknowledge that some platforms allow the definition of multiple datasources within the same dashboard. Our metamodel can be easily extended to support multiple data sources within the same dashboard, for instance, by associating the datasource with the visualizations or directly with the KPIs.

The operator can specify the list of `KPIs` of interest that must be represented within a dashboard by indicating their `names` and `unit of measures`. Each KPI is also associated with a `Query` that defines how to access the data source to retrieve the KPI values. In addition to the query, the operator can specify the `Dimension` of the collected data (i.e., `temporal`, `string`, `numeric`) that influences the type of the visualization selected during the automatic dashboard generation process.

The metamodel separates the concerns about KPIs definitions and dashboard definitions, to allow for both the definition of multiple dashboards in a same model and the reuse of pieces of code and/or model in other dashboard models. For instance, a set of KPIs can be reused across models by simply copy-pasting their definitions, as well as the same could be done for dashboard and visualizations.

5.2. KPIs definition

The first step of our approach requires the operator to enter the list of the KPIs that must be visualized in the *dashboard model*. In addition the operator chooses the arrangement strategy of the visualizations that must be generated in the next step, and specifies the queries to extract the KPI values from the connected datasource.

Fig. 4 (only entities with a white background) shows an example dashboard model that defines two KPIs that must be visualized and that are stored in a Prometheus *node exporter*: the Estimated Error and Maximum Error KPIs. Another relevant information is that, in a model-driven perspective, the list of the KPIs to be included in a dashboard must not be necessarily entered from scratch. In fact, a KPI knowledge base might be created as a repository of models from which the operator can reuse model elements (i.e., copy-paste KPI definitions).

In the example model in Fig. 4 (only entities with a white background), we defined an instance of the metamodel presented in Section 5.1. Indeed, we instantiated the concept of a dashboard that has to organize the visualizations per category, in fact the dashboard's structure type is defined as *categorical*. The KPIs shown in these visualizations are stored in the datasource instantiated with `uid` *cc893e83* and type *prometheus*.

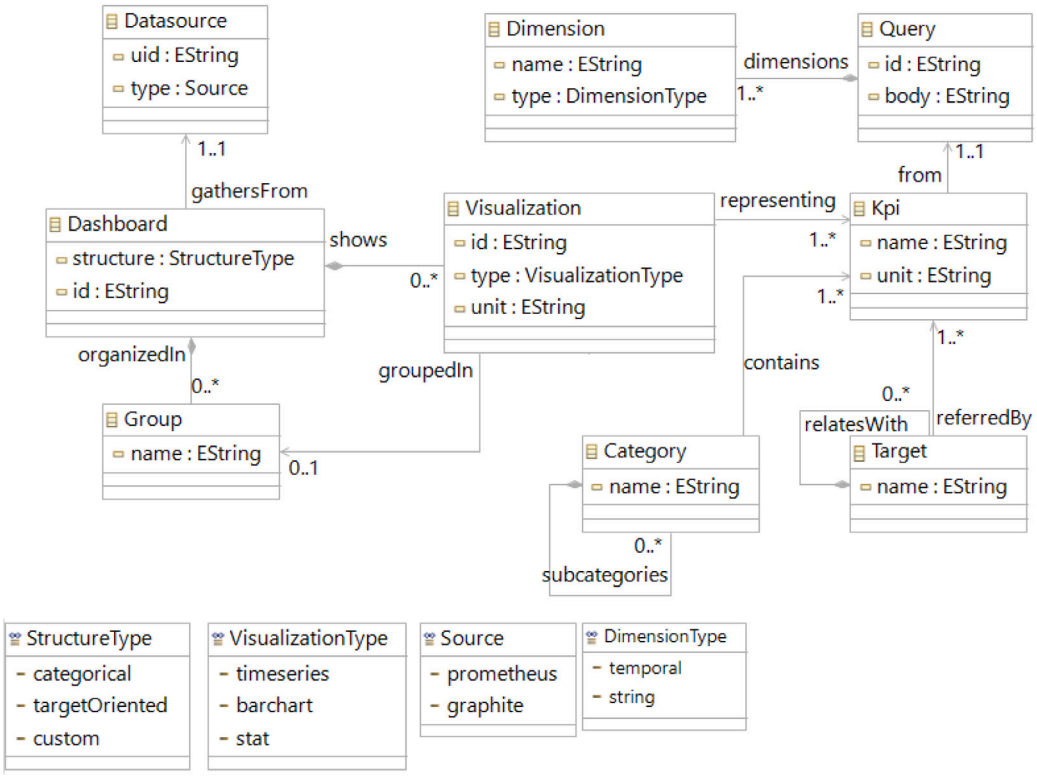

Fig. 3. The dashboard metamodel.

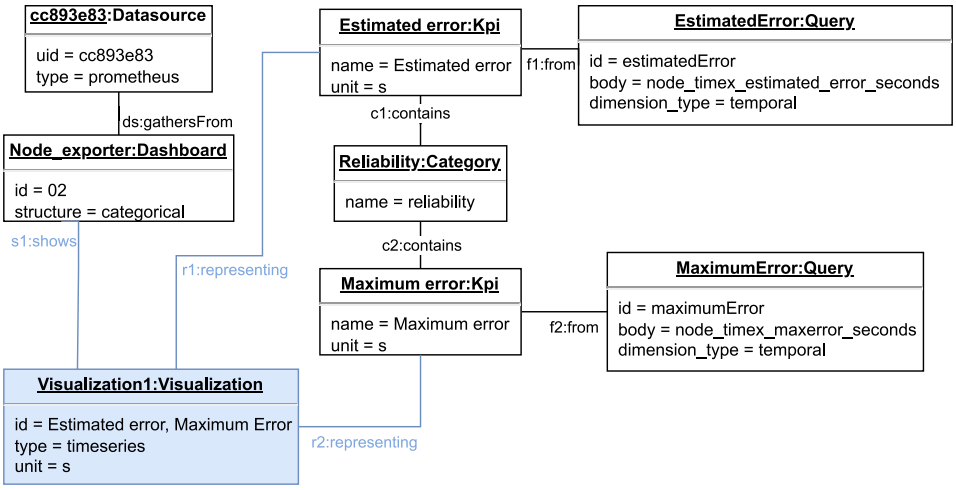

Fig. 4. Example of a dashboard model. The blue entities and associations are added by the Automatic Definition of KPIs Visualizations step, and are not part of the initial dashboard model.

Concerning the KPIs to be shown, the model has to arrange two KPIs, namely, *Estimated error* and *Maximum error*, both with the unit measure *s* that stands for seconds. Each KPI is associated with its corresponding instance of a query that can be executed on the datasource to retrieve it. The `type` attribute of the query indicates the dimension to use to visualize it. In Fig. 4, in the body of the query we only indicate the returned metric. In this field, as anticipated, the operator must write the query in the format specific to the target platform. For instance, if the target platform is Grafana, the query should be defined accordingly. For instance, the following query extracts the estimated system clock error in seconds from the Node Exporter: `node_timex_estimated_error_seconds{instance=''localhost'', job=''node-exporter''}`

In the case shown in Fig. 4, the queries are of type temporal, which means that the dashboard will show their evolution over time. We also support the type string, which implies that the KPI values will consist of labels that may have various meanings, for instance, the names of the components in a system, or the names of the types of observed errors. Finally, in the running example, the two selected KPIs belong to the same KPI category, which is *Reliability*.

We remark that this dashboard model can be quickly assembled by an engineer with knowledge of model-driven design principles, as confirmed by the experiments reported in this paper. Starting from this information provided by the operator, the next steps show how to generate and deploy the actual dashboard automatically.

5.3. Automatic definition of KPIs visualizations

The automatic definition of KPI visualizations is grounded on established principles from the information visualization literature, which consider data semantics, perceptual effectiveness, and cognitive load reduction. In particular, existing guidelines recommend grouping related measures, preserving unit consistency within the same visualization, and selecting visualization types according to the nature of the underlying data (e.g., temporal vs. categorical) to support correct interpretation and comparison [21,22]. Following these principles, our approach deliberately adopts lightweight and transparent heuristics, prioritizing interpretability and controllability over complex optimization strategies.

This step takes as input a dashboard model and produces in output the same model where new *Visualization* instances are added. These visualizations define the arrangement of the KPIs specified by the operators in the previous step within the dashboard. The resulting model can be then used, with the next step, to automatically generate the actual dashboard code. The definition of the dashboard with the visualizations is automatic on the first place, but the operator can modify it, if needed.

In practice, the KPIs defined in the model are first split with respect to the KPIs grouping criterion chosen by the operator in the definition of the model. If the *category* criterion is defined, the KPIs belonging to the same category (i.e., measuring a similar aspect) are placed within the same group. If the criterion *target* is specified, the KPIs collected from the same target belong to the same group.

The KPIs in each group are then divided into sub-groups depending on their unit of measure, that is, KPIs that are in the same group and use the same unit of measure are assigned to the same sub-group, following well established guidelines in information visualization.

Once KPIs are assigned to a sub-group, the actual visualizations are defined. Indeed, the final number of visualizations corresponds to the number of the identified sub-groups. This is achieved by first creating and associating an empty visualization to the dashboard. All the KPIs in the same sub-group are then added to the same visualization.

Finally, depending on the types of the KPIs, the type of visualization is defined. If there is at least a KPI in the visualization whose values are of type `temporal`, then, the type of the visualization is a *timeseries*,[15] since the temporal progression of the data is likely to be a relevant factor in the visualization. Otherwise, if there is at least a KPI whose values are type `string`, then, the visualization is of type *barchart*,[16] so that the various categorical values could be properly represented. In all the other cases, a visualization of type *stat*[17] is selected, which is a fairly general representation that can suitably cope with any information extracted.

Fig. 4 (including blue elements) shows the model after the automatic definition of KPIs visualizations.

Since we only have two KPIs of the same category using the same unit of measure, and the structure of the requested dashboard is `categorical`, a single visualization with the two KPIs is added to the model. The new visualization has an automatically generated id. Since both queries are of type `temporal`, the selected visualization is a *timeseries*.

While the running example shows the simple case of a single visualization automatically added to the model, the same approach scales to the case of dozens or hundreds of KPIs listed by the operator (e.g., by copy-pasting or dragging and dropping KPI definitions from a knowledge base, or an existing model) that must be automatically arranged in several visualizations. In such a case, multiple visualizations organized according to their grouping and type would be automatically created.

The generated model is available with the editing environment, and the operator can conveniently modify the result by changing/ removing/ adding both the visualizations and the links between KPIs and visualizations, before generating the actual dashboard in the next step. This possibility provides a useful trade-off between automation and the possible need of customizing the obtained result.

Rather than aiming at optimal visualization selection through complex optimization or learning-based techniques, the proposed heuristics are intentionally simple, transparent, and explainable, as well as rooted in the information visualization principles [21,22]. This design choice aligns with the goals of MDE, where traceability between design decisions and their effects is crucial. Moreover, simplicity supports manual refinement by operators, ensuring that automatic generation does not hinder customization but instead provides a reasonable and theoretically grounded baseline that can be adapted when needed.

5.4. Automatic dashboards generation

This last step takes in input the complete dashboard model generated in the second step, and produces in output the dashboard implementation (i.e., the code) that can be directly imported into the target dashboard platform. To define a process that is as agnostic as possible from the selected target platform, all the previous steps work uniquely at the model level, and this is the only step where the transformation takes technology into account. Anyway, to enable the support for a plethora of platforms, this step is implemented as a set of template rules that transform the entities in the dashboard model into actual code. The sets of rules are derived manually according to the code format required by the target platform. In particular, we define a rule for each model's element. The rule translates the model's element into its corresponding piece of source code. This method requires an initial investment of effort, but it is a one-time task that enables the creation of reusable templates, thereby ensuring automation and consistency across transformations.

Specifically, we set up the dashboard generation step in such a way that we have for every considered target platform a hierarchy of template rules. We provide a main template that defines the panels that must be created and invokes a dedicated template for each visualization that must be rendered inside a panel. Depending on the selected platform, these template rules might be richer, for instance using colors, labels, etc., or simpler, not using elements that are not supported in the target platform. This maintains the independence between the definition of the desired dashboard, which is the output of the previous step, and the best dashboard that can be achieved with the target platform technology, which is the output of this step.

To demonstrate the capability of this step, we defined the set of template rules necessary to obtain a fully working and deployable dashboards on Grafana and Dynatrace.[18] Following the defined template for Grafana, the visualizations are arranged automatically in rows according to the created group (i.e., category, target, custom group). In particular, a template that creates the header of a new row is first loaded. Then visualizations are arranged inside a row according to their types and total number of panels that have to fit inside a row.

For each different type of panel (e.g., timeseries, barchart), dedicated templates are invoked to render the panel appropriately.

Running Example. Fig. 5 shows the dashboard resulting from the application of the template rules to the running example. The left part of the figure shows an excerpt of the generated dashboard code in JSON format.

[15] A timeseries visualization shows trends of numerical values over time.

[16] A barchart visualization shows multiple bars depending on the frequency of labels.

[17] A stat visualization lists one or more numerical values in the foreground, with trends of values in the background, if applicable.

[18] https://wkf10640.apps.dynatrace.com/ui/apps/dynatrace.launcher/launchpad/99583c94-6c7c-4a5d-9c23-1432e4e1746c.

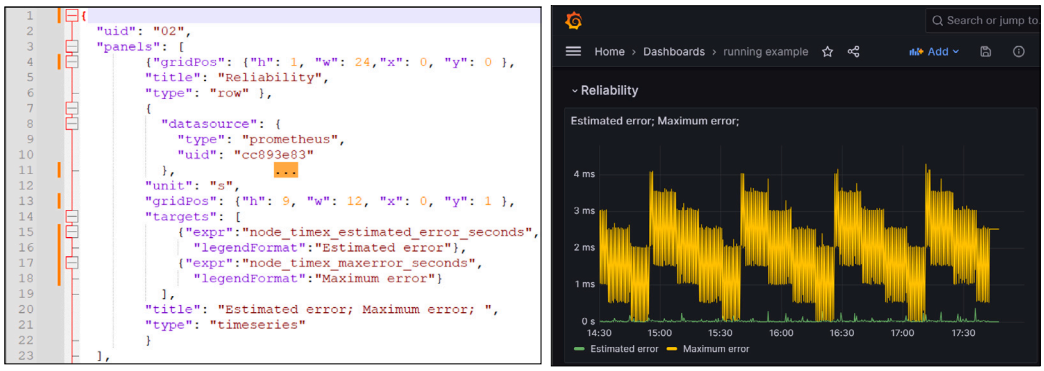

Fig. 5. Final output of the dashboards' generation process for our running example. On the left, the code of the generated dashboard. On the right, the graphical representation of the generated dashboard on Grafana.

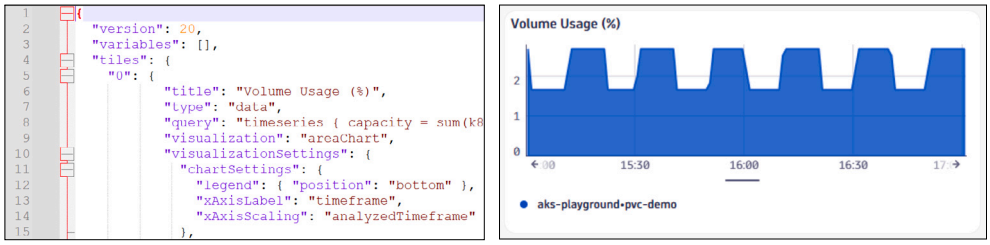

Fig. 6. Final output of the dashboards' generation process for our running example. On the left, the code of the generated dashboard. On the right, the graphical representation of the generated dashboard on Dynatrace.

Specifically, we can see at line 2 that the `uid` is the id that we defined in our dashboard model. Then, from line 3 to line 23, the many panels that compose the dashboard have been created. Namely, the row contains the visualizations that refer to a same category (lines 4 – 6); and the panel contains the generated visualization (lines 7 – 22). In this panel, we can find the declaration of the datasource (lines 8 – 11) with its attributes coming from the dashboard model, the title (line 20) and the type (line 21) of the visualization. The KPIs to be represented are reported from line 14 to 19 as *targets* together with their names (lines 16,18) and queries (lines 15,17).

The right part of Fig. 5 shows how the dashboard is finally rendered in Grafana. Here, we can see that we have a row called *Reliability* in which there is a chart representing the two time series, one for the *Estimated error* as a green line, and the other for the *Maximum error* as a yellow line.

In Fig. 6, we show the dashboard that is automatically obtained from the same model for Dynatrace using its generation template. On the left part we report an excerpt of the generated dashboard code in which the visualizations are generated as `tiles` (line 4) differently from Grafana where they are defined as panels. Further, we can see that in this case the timeseries visualization has been translated into an `areaChart` (line 9) in order to be compliant to Dynatrace visualization types. The right side of Fig. 6 shows how the corresponding chart is rendered on Dynatrace.

Dashboard Adaptability and Generality. Any change to the resulting dashboard, if needed, can be quickly implemented by modifying the dashboard model and automatically regenerating the dashboard code, which can be immediately imported on the target platform. The generation process completely rewrites the generated artifacts. Note that this approach allows addressing the dashboard definition and evolution with a practical model-driven paradigm that lets operators quickly adjust their dashboards, even if they are not experts in the specific target dashboard technology they are working with.

From the reported example generated for Dynatrace, it emerges that only the final template rules need to be extended to support additional dashboard platforms. This ensures that new platforms, such as Kibana, can be supported by providing new template rules, with no changes to the modeling workflow or the metamodel.

In a nutshell, developers contribute to template rules to increase the number of supported dashboard platforms. While users contribute to the definition of the KPIs and the adaptation of the generated visualizations, during the dashboard generation process.

5.5. Implementation details

The implementation of the approach is based on the Eclipse Modeling Framework (EMF).[19] In particular, we exploited both EMF and Eclipse to define the modeling language for dashboards and KPI definitions, and to generate the corresponding code. The plugin for the automatic generation of the visualizations uses the Epsilon Object Language (EOL),[20] which is a language devoted to model management built on top of EMF. We exploited EOL to implement the models' querying and in-line transformations.

For the automatic generation of the dashboard step, we exploited model-to-text transformations implemented with the Epsilon Generation Language (EGL).[21] In particular, we exploited its template-based nature to define a set of templates for each target platform. A main template is devoted to the generation of the main dashboard source code, and then specific templates defined for each type of panel are invoked to obtain the individual visualizations. Our implementation is publicly available [12].

A key benefit of employing this approach within the Eclipse IDE is the ability to utilize its integrated mechanisms, including its persistence manager and version control integrations, to monitor modifications in the managed dashboards.

[19] https://projects.eclipse.org/projects/modeling.emf.emf.
[20] https://eclipse.dev/epsilon/doc/eol/.
[21] https://www.eclipse.org/epsilon/doc/egl/.

6. Empirical evaluation

To evaluate our model-driven approach, we investigate the two research questions stated in Section 3 about the *comprehensibility* and *flexibility* of the generated dashboards compared to manually-configured dashboards. Specifically, to answer RQ1 we involved professionals and researchers in two user studies. The first study involved 20 professionals who have been asked to interpret the content of both manually implemented and automatically generated dashboards considering as target platform Grafana. The second study involved 10 researchers who had to compare the usability of automatically generated dashboards and Dynatrace dashboards selected from the Dynatrace playground.

To answer RQ2, we measure and compare the time needed to complete five different adaptation tasks with Grafana dashboards of different levels of complexity.

6.1. RQ1 - comprehensibility

To answer this research question, we design two user studies with subjects involved in the comprehension of the content of both dashboards manually implemented by experts and the same dashboards generated automatically with the proposed model-driven approach for two target platforms: Grafana and Dynatrace.

The two studies are complemental in the target subjects, the type of interaction with dashboards, and the considered dashboard platform. The first study, namely the *Grafana study*, targets professionals who have to interpret the content of the original and the automatically generated Grafana dashboards, without making any one-to-one comparison between dashboards. While the second study, namely the *Dynatrace study*, targets researchers who interpret the content and provide feedback on the one-to-one comparison of Dynatrace dashboards.

In Sections 6.1.1, 6.1.2, and 6.1.3 we describe our experimental plan, that is, how we select the Grafana and Dynatrace dashboards, how we generate the dashboards with our model-driven approach starting from the selected dashboards, and how we design the two user studies, respectively. Then, we describe in Section 6.1.4 how we conducted the experiment with the human subjects and the structure of the provided questionnaire. Section 6.1.5 explains data analysis. Finally, Sections 6.1.6 and 6.1.7 report the results for the Grafana and Dynatrace studies, respectively.

6.1.1. Dashboards selection

As objects of the first study, we selected 10 public dashboards from the Grafana Labs collection,[22] which is a repository containing ready-to-use dashboards published by community members and by Grafana Labs itself that can be used to visualize KPIs collected from a wide range of systems (e.g., web servers, databases, container platforms, and operative systems).

To select the dashboards for the study, we first identified the ones that have at least five reviews[23] and a minimum rating of four out of five, to focus on popular and well-designed dashboards. Since we need to connect the dashboards to a live instance of a time series database, we selected the ones that use Prometheus[24] as data source, which is the default choice for Grafana dashboards.

We ended up with a first selection of 18 dashboards. We carefully inspected these dashboards and discarded those that (i) use data produced by specific hardware, such as routers (2 cases); (ii) require the development of ad-hoc probes and/or expensive configuration of several components (5 cases); are redundant with respect to already selected dashboards (6 cases).[25]

Table 2
Selected Grafana dashboards.

Name	URL	# panels
JVM dashboard	https://bit.ly/3RktVDA	15
Cluster Monitoring for Kubernetes	https://bit.ly/4ci8Emd	13
Windows Exporter dashboard	https://bit.ly/3VgyfoC	22
Node Exporter Full-Quick basic	https://bit.ly/4b20ixH	15
Node Exporter Full-CPU, Memory, Network, Disk	https://bit.ly/4b20ixH	8
Node Exporter Full-Memory Meminfo	https://bit.ly/4b20ixH	15
Node Exporter Full-Memory Vmstat	https://bit.ly/4b20ixH	4
Node Exporter Full-System Timesync	https://bit.ly/4b20ixH	4
Node Exporter Full-System Processes	https://bit.ly/4b20ixH	7
Node Exporter Full-System Misc	https://bit.ly/4b20ixH	7

Table 3
Selected Dynatrace dashboards.

Name	URL	# panels
Container vulnerability findings	https://bit.ly/463mJmM	17
Kubernetes Cluster	https://bit.ly/4bR4bd4	20
Kubernetes Node - Pods	https://bit.ly/3MFiJlM	11
Kubernetes Monitoring Statistics	https://bit.ly/3MA7Mlz	6
Extension Data Consumption	https://bit.ly/3MqhdDW	4
Generic network overview	https://bit.ly/4adQQu9	10
Synthetic Network Monitors Health & Performance	https://bit.ly/4aM93PC	45
Juniper Overview	https://bit.ly/3O9PjwF	16
OpenTelemetry K8s Cluster	https://bit.ly/4aqBa5E	26
OpenTelemetry K8s Persistent Volumes	https://bit.ly/3OoRkoJ	5

Table 2 shows the name, URL, and size in number of panels of each selected dashboard. Since the Node Exporter Full dashboard includes multiple sections with several panels each, we treat each row as a dashboard itself.

For the second user study, we selected 10 dashboards from the Dynatrace playground.[26] We identified the dashboards among the ones in the *ready-made* examples[27] that are provided with ready-to-use data. Table 3 shows the name, URL, and size in number of panels of each selected dashboards.

6.1.2. Generation of the automatic model-driven dashboards

To obtain the automatic dashboards that visualize the same KPIs shown in the selected Grafana and Dynatrace dashboards, we proceeded as follow. For each dashboard, we identified the set of represented KPIs, we defined the dashboard model that lists exactly the same KPIs using our DSL, and we generated the automatic dashboard with our tool, finally obtaining an automatic dashboard that shows the same KPIs presented in the corresponding selected dashboard.

To create an automatic arrangement of the visualizations that is unbiased, we used the *monitoring goals* presented in [20] and inspired by the ISO/IEC 25010:2011 and ISO/IEC TS 25011:2017 standards as categories for our KPIs. The work by Shatnawi et al. [20] and the related ISO standards offer a way to categorize KPIs hierarchically, according to their goal, which is a reasonable arrangement also for a dashboard. For instance, the example KPIs reported in Section 5 are

[22] https://grafana.com/grafana/dashboards.

[23] Dashboard reviews on Grafana Labs are submitted by users as feedback comments about the quality of the dashboards.

[24] https://prometheus.io.

[25] We always select the dashboard with the highest number of stars (alternatively the highest number of downloads if the number of stars were the same) within a set of dashboards showing the same data.

[26] https://wkf10640.apps.dynatrace.com/ui/apps/dynatrace.launcher/launchpad/99583c94-6c7c-4a5d-9c23-1432e4e1746c.

[27] https://wkf10640.apps.dynatrace.com/ui/apps/dynatrace.dashboards/dashboards.

both associated with the category *Reliability* corresponding to one of the monitoring goals described in [20].

Note that a final automatic dashboard, although representing the same set of KPIs of the corresponding dashboard selected online, shows the KPIs in a potentially totally different way, that is, it might use a different number of visualizations, different visualization types, and it might group KPIs differently across visualizations, when compared to the original dashboard. In fact, the goal of this research question is to establish if the automatic dashboards have a comprehensibility similar to the ones implemented manually, although resulting from an automated process.

Eventually, we obtained multiple sets of dashboards: the set of ten dashboards selected from Grafana Labs, the ten dashboards selected from the Dynatrace Playground, and the *Model-Driven dashboards*, that is, the same set of twenty dashboards (ten Grafana dashboard and ten Dynatrace dashboards) generated with our model-driven approach from the same set of KPIs visualized in the dashboards. Both sets of dashboards are populated using the same data sources, so that all dashboards work with the same set of data to be visualized.

6.1.3. Design of the user study

To answer this research question and compare the comprehensibility of manually curated dashboards and automatically generated Model-Driven dashboards, we designed two studies, complemental in the nature of the involved subjects, the chosen design of the study, and the used dashboard platform.

The first study, the Grafana study, is a study with 20 professionals from 12 companies and 3 research centers working as software engineers, DevOps engineers, and data scientists. We opted for a *within-subjects* design, where each participant is exposed to multiple conditions rather than being assigned to a single experimental condition. This approach allows for comparisons within the same group of participants, reducing variability due to individual differences. Indeed, we split the participants into two groups of equal size, with each group working with both the Grafana Labs and Model-Driven dashboards, to mitigate any side-effect that might be introduced by specific participants working with one class of dashboards only. Consequently, our study does not involve traditional test and control groups, but rather ensures that each participant is exposed to a balanced set of dashboards.

Since we have two comparable versions of each dashboard, and we do not want a subject to inspect both versions of a same dashboard due to a severe learning effect that might be introduced (i.e., the task performed on the first dashboard would drastically simplify the same task performed on the alternative version of the same dashboard), we split the set of dashboards into two sets. Each set includes ten dashboards, 5 Grafana Labs dashboards and 5 Model-Driven dashboards, with no overlap, that is, the two versions of a same dashboard always belong to two different sets.

Each participant assesses all the ten dashboards in one set. Dashboards are presented in a random order to avoid any bias that may be introduced by a specific order (e.g., by alternating Grafana Labs dashboards with Model-driven dashboards). The assessment task consists of inspecting each dashboard and answering four questions about the values of the KPIs presented. The same questions are asked for the two versions of a same dashboard.

The four questions were formulated as multiple-choice questions about KPI values and trends for each pair of dashboards to be assessed in a way that precludes the introduction of subjective judgments and biases. In particular, we selected two KPIs that are rendered in visualizations of the same type (e.g., two barcharts) in both the Grafana Labs dashboard and the corresponding Model-Driven dashboard, and two KPIs that are rendered in visualizations of different types (e.g., a timeseries and a barchart). The two questions are always about identifying the exact value of a KPI at a given point in time, and recognizing the trend of another KPI (ascending, descending or constant). For two out of ten dashboards, we did not have KPIs rendered in visualizations of the same types, so we formulated two questions only.

Table 4
Summary of participants' job titles.

Job title	Software Engineer	DevOps Engineer	Data Scientist/Engineer	Other
Grafana study	7	5	4	4
Dynatrace study	8	0	0	2

The second study, the Dynatrace study, is a study with 10 researchers from 4 universities and research centers working as software engineers. This time we opted for a *between-subjects* study to involve participants in a direct and explicit one-to-one comparison of the generated dashboards.

Each participant had to first complete a task designed as in the Grafana study in one of the dashboards, and then open the alternative version of the same dashboard and decide if one dashboard is presenting data better than the other, or they are equally good. We do not ask the subject to complete the same task on the second dashboard again, since it would be trivial to complete by reusing the knowledge just acquired with the former dashboard. To balance the experiment, we randomly split the participants in two groups. The first group has five manually curated Dynatrace dashboards and five automatically generated dashboards as fist dashboard to inspect. The second group has each pair of dashboard presented in the opposite order.

In additional to collecting data about the correctness of the task completed on the dashboard and on the judgment about the direct one-to-one comparison of the dashboards, the participants completed the usability questionnaire based on the NASA-Task Load Index[28] (TLX) that provides additional information of the usability of the two types of dashboards by including questions about the mental, physical and temporal demand and the levels of performance, effort and frustration experienced. For each question, the values are collected on a 5-point scale that goes from 1 ("Very low") to 5 ("Very high").

6.1.4. Questionnaire and subjects selection

We structured the survey questionnaire into three sections. The list of the survey's questions are reported in [24].

- *Preliminary Information* - participants are asked to answer questions about their job title (i.e., *Software Engineer*, *DevOps Engineer*, *Data Scientist/Engineer*, and *Other*) and experience with dashboards. In particular, we asked about their level of expertise with dashboards in general (i.e., *never used*, *used once or twice*, *used sometimes*, *often used but never configured*, *often used and configured*), how often they use dashboard platforms in their work activities (i.e., *never*, *once in a while*, *weekly*, *daily*), and their level of expertise with Grafana or Dynatrace (i.e., *never used*, *used once or twice for visualization only*, *often used for visualization but never for creation*, *often used for both visualization and creation*).
- *Dashboard Usage* - each participant accesses Grafana, Dynatrace, and Model-Driven dashboards consistently with the design of the study. The accessed dashboard is a live instance of the dashboard that the participant can actually use and interact with. Each dashboard is associated with questions about KPI values and trends, as described above. Dashboards are evaluated sequentially. Participants to the Dynatrace study also complete the NASA-Task Load Index questionnaire.
- *Feedback and Comments* - participants are finally asked to provide feedback about the level of clarity of the survey and additional feedback about the dashboards, if any, in a free text field.

[28] https://humansystems.arc.nasa.gov/groups/tlx/downloads/TLXScale.pdf.

Table 4 presents a summary of the job titles of the participants. In the Grafana study, the data indicates that the majority of participants are software engineers (7/20), followed by DevOps engineers (5/20), data scientists/engineers (4/20), and other job titles (4/20). In the Dynatrace study, the vast majority of the participants are software engineer (8/10), with few other job titles (2/10).

From the answers to the questions concerning the expertise of the participants in the Grafana study, we observed that all the participants have used dashboards at least once. In terms of frequency of use, only two participants utilize dashboards on a daily basis, with the majority of participants using dashboards once in a while (14/20) or weekly (4/20). Finally, the majority of participants (12/20) indicated that they had never used the Grafana dashboard tool or only used it for visualization purposes (4/20). Although 4 out of 20 participants reported that they were regular users who often visualized and created Grafana dashboards. Concerning the expertise of the participants in the Dynatrace study, only one participant has never used a dashboard, all the others used it at least once and (4/10) often use dashboards without configuring them. Regarding the frequency of use only one participant use dashboards on a daily basis and the majority uses them once in a while (4/10). One participant already used Dynatrace once.

It should be noted that the study has been approved by the ethics committee of our academic institution and that all participants in this study are adults who have provided their explicit consent to participate.

6.1.5. Data analysis

We analyze the data collected through the user studies to determine the level of success of the subjects who used the Grafana Labs or Dynatrace dashboards in comparison to the Model-Driven dashboards and we computed two *metrics*. In particular, for every question q submitted about the twenty dashboards, we computed the accuracy of the collected responses as follows:

$$Accuracy_q = \frac{Correct\ answers}{Total\ answers} \tag{1}$$

The accuracy of the responses about a dashboard d is obtained as the average of the accuracy of questions about its KPIs. Consequently, we derive the comprehensibility C of a dashboard d as follows:

$$C_d = \frac{\sum_{q=1}^{Q}(Accuracy_q)}{Q} \tag{2}$$

where Q is total number of questions posed for the dashboard d. This way, we can compare the comprehensibility of Model-Driven and manually curated dashboards.

In the Dynatrace study, we also report the results about the direct one-to-one comparison of the dashboards and the answers to the NASA-Task Load Index questionnaire, when considering the model-driven and manually curated dashboards.

The experimental material, that is, a copy of the survey, the two dashboard sets, the data used to populate the dashboard panels, and the collected results are publicly available [12].

6.1.6. Results for the Grafana study

Fig. 7 depicts the mean accuracy (in percentage form) of the answers to the individual questions posed to the participants with respect to the two types of dashboards (i.e., Grafana Labs and Model-Driven dashboards). The questions on the x-axis are presented in ascending order with respect to the accuracy of Model-Driven dashboards.

Overall, the results appear to be comparable. However, there are some KPIs that are more effectively represented in Grafana Labs dashboards, while others that are more accurately represented in the Model-Driven dashboards. This discrepancy can be attributed to the efficacy of the heuristics employed to place the KPIs and their associated visualizations. In some instances, the heuristics align well with the intended semantics of the KPIs, whereas in others, they are less effective.

It is reasonable to expect that the Grafana Labs dashboards exhibit better performance when compared to Model-Driven dashboards. This is due to the fact that the former are manually curated, while the latter are automatically generated. Despite this fact, it is noteworthy that for eight KPIs, the accuracy of the Model-Driven dashboard has been superior to that of the Grafana Labs dashboards, and for 12 KPIs the accuracy of the Model-Driven and Grafana Labs dashboards has been the same. Further, half of the KPIs represented with the Model-Driven dashboard have an accuracy higher than 75%, which is a promising result.

We calculated the independent Pearson's Chi-Squared Test [25] to assess whether participants' responses to each survey question significantly differed between the two approaches (i.e., Model-Driven, Grafana Labs). We have not applied any correction (e.g., Bonferroni's correction), since they are not recommended for small-sized exploratory studies such as ours [26–28]. Only a few questions showed statistically significant or suggestive differences in the distribution of correct answers across the two approaches. Specifically, *Q27, Q9, and Q11* (highlighted in orange in Fig. 7) showed statistically significant differences ($p < 0.05$), indicating that the responses to these questions were dependent on the approach used. Instead, *Q23 and Q25* (highlighted in orange in Fig. 7) showed marginal significance ($p < 0.1$), suggesting a potential relationship between the approach and the responses. Overall, according to the statistical tests, the two approaches produce similar results in most of the cases.

To dig into the data more deeply, we categorized the questions with respect to the differences between the accuracy percentages, considering four intervals:

- **No Difference**, which is the case where the accuracy of questions for the Grafana Lab and the Model-Driven dashboards are the same;
- **Negligible Differences**, which is the case of a delta in accuracy between 0 and 15%.
- **Moderate Differences**, which corresponds to a delta in the accuracy between 15% and 30%.
- **Large Differences**, which is the case of deltas in accuracy larger than 30%.

Fig. 8 shows the number of questions in each category, with colors distinguishing the Grafana Labs and Model-Driven dashboards. Again the largest class of questions belong to the category showing no differences, and only a few questions reported moderate and large differences in accuracy. In particular, there are 11 KPIs with moderate accuracy differences about their comprehensibility (8 in favor of the Grafana Lab dashboards and 3 in favor of the Model-Driven dashboards) and 8 KPIs with large accuracy differences about their comprehensibility (4 in favor of the Grafana Lab dashboards and 4 in favor of the Model-Driven dashboards). This last group includes all the cases with significant and suggestive differences reported by the statistical test.

We analyze the reasons for moderate and large differences in accuracy and discover these influencing factors.

- **Peak values** The presence of peaks (i.e., outliers) in constant trends sometimes created confusion (four large differences) for both Grafana Labs and Model-Driven dashboards. This has led participants to misinterpret the trend as constant because they were unable to identify whether the trend to be considered was the entire visualized time series or a specific portion of it. We noticed this misinterpretation to occur slightly more frequently for the MDE dashboards, although it is an issue shared by both of them.
- **Captions may bias the operators**. The captions that report summary values (i.e., min, max, avg) in the visualizations of Grafana Labs dashboards may significantly influence the results, both negatively and positively (model-driven dashboards do not use summary values in captions). In one of the cases (one large difference), this information created confusion penalizing the accuracy, since the participant only considered the caption, overlooking at

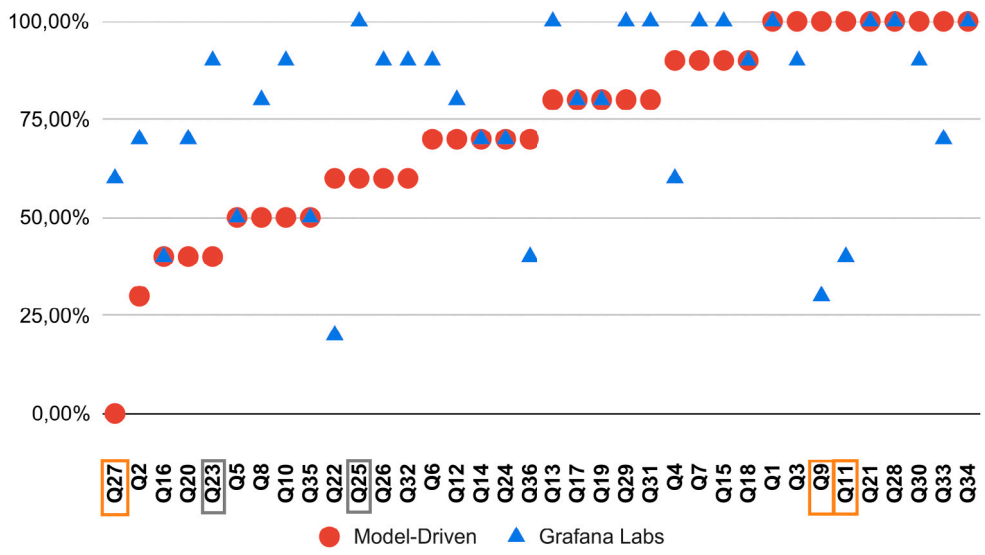

Fig. 7. Distribution of the answer accuracy for each of the questions.

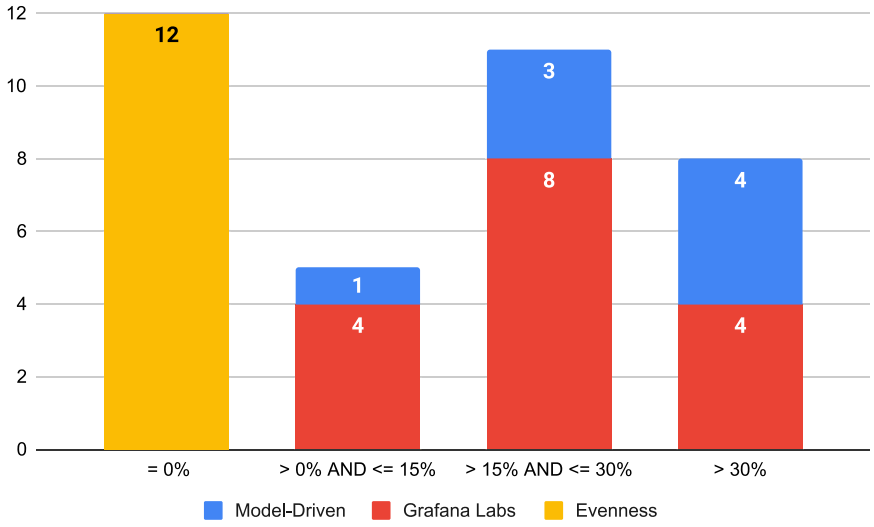

Fig. 8. Distribution of the categories of the differences between the accuracy percentages.

the content of the visualization. Instead, in two cases (two large differences), it supported a higher accuracy of the Grafana Labs dashboard since the participants were facilitated in obtaining the maximum value of a KPI by just looking at the caption, without the need of interpreting the visualization.

- **Too many KPIs in a single visualization** The presence of multiple KPIs in a single visualization in the Model-Driven dashboards penalized the answers' accuracy in multiple cases (one large difference and eight moderate differences). This phenomenon can be attributed to the ambiguity of the visualization title rather than the clarity of the visualization itself. In fact, the visualization title of Model-Driven dashboards is generated as the concatenation of the shown KPI names, creating unreadable titles that in some cases made difficult to identify the right visualization that displays the information useful to answer the question.
- **Simple visualizations sometime better than visually sophisticated visualizations** In three cases (three moderate differences), the model-driven dashboards used simpler visualizations than the fancy visualizations used in Grafana Labs dashboards. Although less appealing on a visual perspective, they resulted more practical for the operators who have to interpret their content.

Although sometimes the automatically generated dashboards may miss some details that slightly decrease their usability, such as a richer caption or a fully self-explanatory title, they still consist of adequate visualizations well representing the data and trends.

For example, Fig. 9 shows two representations of the same visualization, namely the one coming from the Grafana Lab (on the left) and the corresponding generated one (on the right). Even if the specific title and the presence of the summary values in the caption for the Grafana Lab makes easier to extract the main information about the KPIs, the MDE dashboard is still using a visualization of the same type with the same data to plot the values. This is an evidence of how even sub-optimal decisions taken during the automatic generation process are still well-aligned with the design decisions taken by the operators who designed the dashboards used in our study.

Finally, we report in Fig. 10 the comprehensibility of each dashboard. We can observe that the performance of the two compared approaches is similar. The biggest gap is observed for dashboards *node_exporter_06* and *node_exporter_07*. For both dashboards, the difference between the comprehensibility values is mainly due to misleading labels generated by the model-driven approach for some visualization titles. It is however interesting to remark how, although some

Fig. 9. Two representations of the same visualization. On the left, the visualization coming from the Grafana Lab decorated with a caption reporting summary values (i.e., min, max, avg). On the right, the corresponding generated Model-Driven visualization.

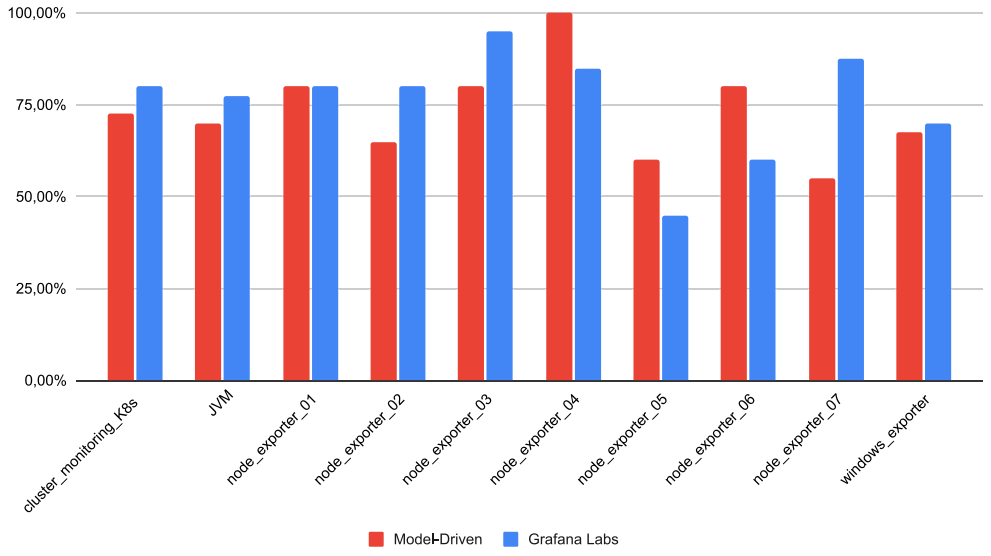

Fig. 10. Comprehensibility of the two proposed representations of the same dashboard.

comprehensibility differences have been observed for individual KPIs, the performance of the two approaches is close dashboard-wise.

6.1.7. Results for the dynatrace study

Fig. 11 show the mean accuracy of the answers, when the model-driven and the manually curated dashboards are presented first, respectively. When model-driven dashboards are presented first, out of ten dashboards, in six cases the accuracy is the same, and in the remaining cases the participants produced more accurated answers with the manually curated dashboard. Conversely, when the manually-curated dashboards are presented first, out of ten cases, the accuracy of the answers is the same in seven cases, while in two cases the participants produced more accurated answers with the manually curated dashboard, and in the remaining case they produced more accurate answers with the model-driven dashboard.

Overall, we can see that the accuracy of the answers for the generated dashboards is not overly influenced by the order. Results-wise, the model-driven dashboard allowed subjects to produce as accurate responses as the original dashboard in more than half of the cases (13 out of 20 cases considered), being even better in one case. Although the representation was less clear in a few cases, the model-driven dashboards provide an interesting tradeoff in terms of cost of generation, which is largely automatic, and accuracy of the responses, which are close to the ones obtained with the original dashboards.

Fig. 12 shows the answers that participants gave to the TLX questionnaire for both the manually curated and model-driven dashboards. We can notice that the two types of dashboards performed similarly, with subjects feeling slightly more in a hurry when using model-driven dashboards. Yet, all values are always below three, indicating a positive assessment of all the considered factors.

Finally, we discuss the feedback returned when comparing the manually curated dashboard to the model-driven dashboard, distinguishing between the cases where participants watched the model-driven dashboard first (Fig. 13) and the manually curated dashboard first (Fig. 14).

We can notice that, although the model-driven dashboard cannot perform exactly the same than a manually curated dashboard (in both plots the manually curated-dashboard is preferred 27 out of 50 times), it is often equally effective or better (23 cases out of 50 in both cases). This is a good result since it confirms that an effective representation of the data can be obtained inexpensively with our model-driven approach.

> **Answer to RQ1**. The results reported for RQ1 show how the model-driven dashboard obtained in an automatic way has a comprehensibility that is close to the original dashboard designed by experts, thus representing a valid option to quickly obtain a usable representation of non-trivial sets of KPIs.

6.2. RQ2 - flexibility

This research question investigates how effectively the model-driven paradigm can be used to generate and adapt the results produced by the automatic generation of the dashboard. We report in Section 6.2.1 our

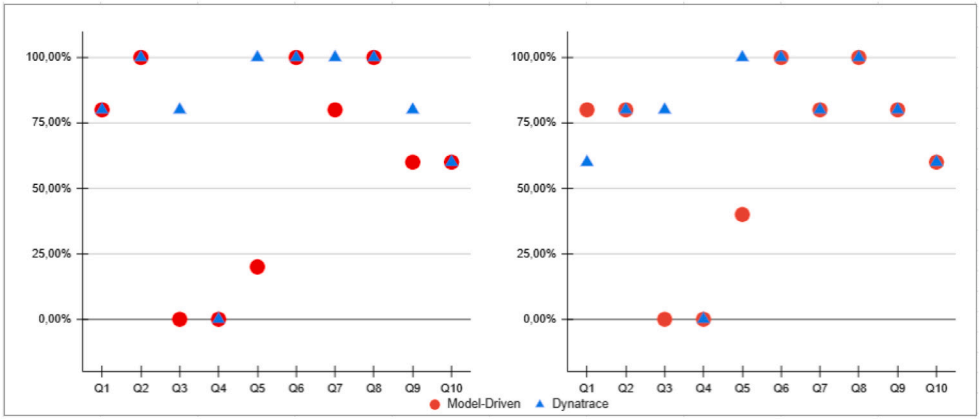

Fig. 11. Distribution of the answer accuracy for each question. (1) On the left, answer accuracy when the Model-Driven dashboards are presented first. (2) On the right, answer accuracy when the manually-curated dashboards are presented first.

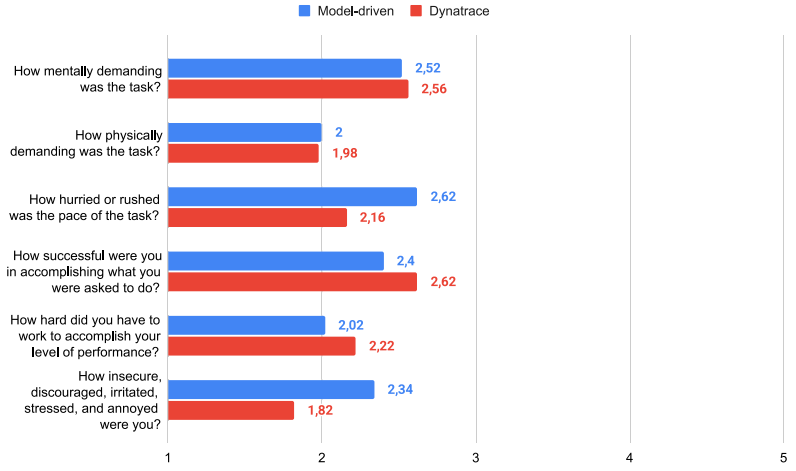

Fig. 12. Task load index.

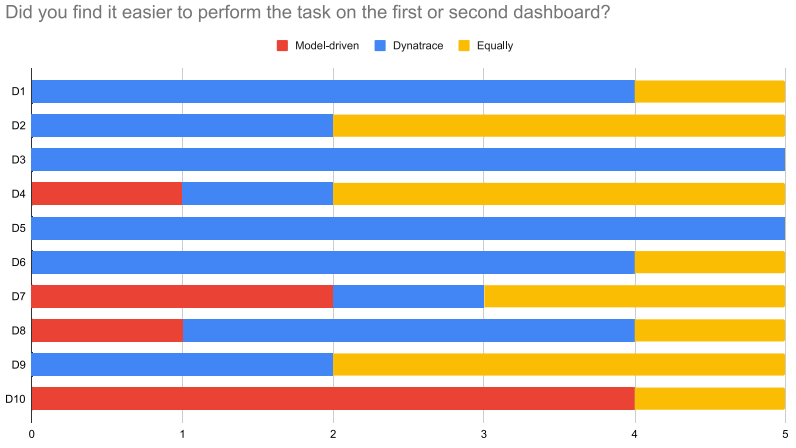

Fig. 13. Preferred dashboard when the model-driven dashboard is accessed first.

experimental plan and how we conduct the experiment to collect the empirical data. Then, we present the obtained results in Section 6.2.2.

6.2.1. Methodology

To answer RQ2, we compare the effort necessary to execute a set of tasks by three different expert profiles, namely a *non-expert user*, who uses the features of the Grafana GUI to configure dashboards, a *Grafana expert*, who configures dashboards implementing code and configuration files and not only using the GUI, and a *MDE expert*, who uses our approach to configure the dashboard. To identify relevant tasks that can be used for the assessment, we inspected the Grafana success stories,[29] which are descriptions of successful and relevant usages of the Grafana technology, and we distilled a total of five tasks

[29] https://grafana.com/success/.

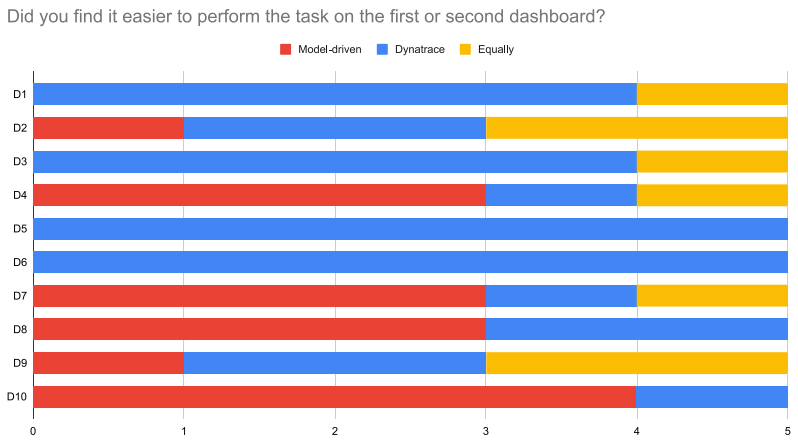

Fig. 14. Preferred dashboard when the manually curated dashboard is accessed first.

Table 5
The management tasks collected by Grafana as success stories.

ID	Task	Task type	Link
1_Create	Create a dashboard from scratch	Generation	https://grafana.com/success/kushki/
2_Split	Split a KPI in two metrics in another visualization	Adaptation	https://bit.ly/47q74fk
3_Merge	Merge two metrics in one KPI in another visualization	Adaptation	https://bit.ly/47q74fk
4_Union	Unify panels from different dashboard into a unique single one	Adaptation	https://grafana.com/success/lightbend/
5_Source	Change the dashboard s data source	Adaptation	https://grafana.com/success/wix/

that are explicitly described as part of these stories. In particular, we obtained five typical dashboard management tasks, one of which is a *generation* task and four of which are *adaptation* tasks. We describe the tasks and report the links to their definitions in Table 5.

To mitigate the presence of bias that might be introduced by the experience of the subjects performing the task, we measure the effort necessary to complete the management tasks when an *informed operator* performs them. More in details, we consider the end-to-end time needed to complete the tasks with no mistakes as the *metric* for this *question*.

To avoid any potential issue related to collecting a single measure by a single informed operator, two authors of this paper independently performed the tasks and collected the measures for the experimentation. Specifically, one author, a Grafana expert with extensive experience on the platform, executed the selected tasks using the Grafana GUI. The other author, who has solid expertise with the Eclipse IDE and basic familiarity with Grafana, performed the tasks both using the Grafana GUI as a non-expert and with the model-driven approach, leveraging their MDE expertise. Further, we involve in the experimentation a professional with knowledge of both MDE and Grafana. The professional works in a software development company and independently performed the tasks, once following the protocol of the Grafana expert and another time following the protocol of the MDE expert. The presence of the professional mitigates any possible implicit bias introduced by the authors when executing the tasks.

The five tasks were repeated for each of three dashboards with a different number of panels, namely *Small*, *Medium*, and *Large* dashboards, corresponding to the Node Exporter Full-System Timesync,[30] the Cluster Monitoring for Kubernetes,[31] and the Windows Exporter[32] dashboards, respectively. Participants to the study acted according to the following protocol. The *non-expert* only uses the graphical features (i.e., buttons, menus, etc.) available in Grafana to complete the ask. The *Grafana expert* uses the coding features (e.g., the Json definition of visualization) available in Grafana to complete the task. Finally, the *MDE Expert* uses our model-driven tool to complete the task. The definition of the event marking the starting and ending of each task, and the specification of the metrics, KPIs and visualizations changed in each tasks were shared among participants. In the results, we report the average execution time of each task, discriminating among the three different expert profiles. The average is calculated among participants and among the executions over the three dashboards of different sizes.

We recorded the tasks completed by the authors. The videos are available in our repository [12], for future inspection and reproduction.

Table 6
Execution times in minutes (′) and seconds (″) of every task for the three profiles.

Task ID	Non-expert	Grafana expert	MDE expert
1_Create	21′51″	**20′43″**	23′45″
2_Split	2′18″	**1′55″**	1′56″
3_Merge	1′19″	**1′18″**	1′50″
4_Union	25′19″	**3′5″**	5′62″
5_Source	5′0″	2′22″	**1′41″**

6.2.2. Results

Table 6 reports the average execution time of each task, with the best result achieved for each task shown in bold. Not surprisingly, the Grafana expert profile is the best profile in four out of five tasks. The MDE expert is the best profile in one case, and performs close to the Grafana expert in several tasks (e.g., the split and merge tasks). The non-expert is never the fastest profile, sometime with large differences with respect to the other profiles (e.g., the union and source tasks).

Fig. 15 shows the percentage of reduction time achieved when using the MDE approach compared to a non-expert user who uses the Grafana GUI.

Modifying the dashboard using the model-driven approach is often faster than using the Grafana GUI for users with a background in MDE, especially when compared to non-expert users. The reduction is more significant when the tasks are performed on dashboards of large size and for tasks *4_Union* and *5_Source*, which can be quickly completed by small changes in the model but require a large number of GUI interactions on the Grafana dashboard. For expert Grafana users familiar with scripting or advanced configuration, the time advantage of the

[30] https://bit.ly/4b20ixH.
[31] https://bit.ly/4ci8Emd.
[32] https://bit.ly/3VgyfoC.

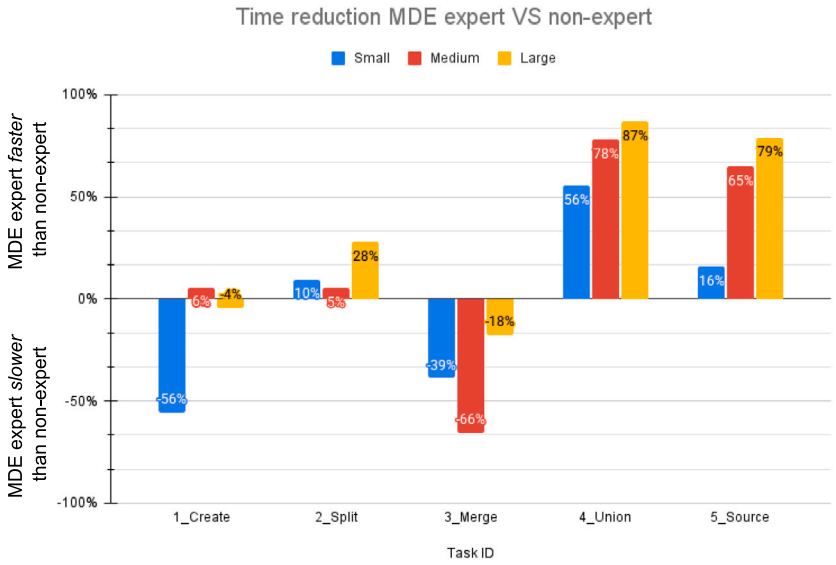

Fig. 15. Percentage reduction of the execution times of the five tasks with respect to the three subject dashboards of the MDE expert compared to the non-expert.

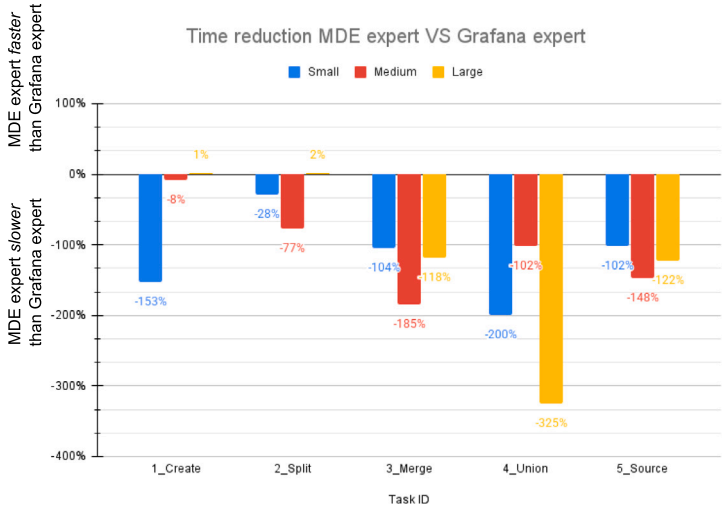

Fig. 16. Percentage reduction of the execution times of the five tasks with respect to the three subject dashboards of the MDE expert compared to the Grafana expert.

model-driven approach is smaller and depends on the specific task. This result suggests that dashboard users who have a background in MDE should prefer the approach presented in this work to cost-effectively modify a dashboard, especially compared to GUI-based modifications by non-experts.

Fig. 16 compares the MDE approach to the performance of Grafana experts. In this case, the results are mostly in favor of Grafana experts using advanced features. This is expected since coding the dashboard is faster than modifying entities within a model. There are some exceptions to this trend. In fact, for time-consuming tasks such as task *1_Create* that implies the creation of the dashboard from scratch, the performance are similar among the two approaches. This is mainly due to the fact that Grafana does not support the automatic arrangement of visualizations in a dashboard. For this reason, every time a user creates a visualization, then it must be placed manually using GUI features. Instead, MDE supports the definition of the arrangement of the visualizations. The same happens for the dashboard large during the execution of the task *2_Split* since also this task requires the creation and positioning of visualizations.

Shorter tasks that can be largely handled with coding and scripting, such as *3_Merge* and *5_Source*, are largely in favor of Grafana experts.

Answer to RQ2.
Results show that the proposed MDE approach enables more efficient dashboard generation and adaptation than Grafana's GUI, especially for non-experts or those less confident with coding. For certain tasks, its performance matches that of advanced users who use scripting. Thus, its efficiency gains are greatest for simplifying complex GUI interactions for non-experts and tasks suited to model-level manipulation, while for experts, the advantage significantly reduces and it is task-dependent.

6.3. Main findings

In conclusion, we hereby summarize the benefits of employing the proposed approach for dashboard generation with respect to the challenges outlined in Section 3.

- `C1` - **Task Repetitiveness**: We have described and empirically evaluated how our approach addresses the problem of repetitive

tasks by automating the definition and configuration of dashboards. Specifically, it allows for the modeling of dashboards at an abstract level, thereby eliminating the need for operators to perform repetitive tasks such as the creation and placement of numerous panels and rows, or the modification of the data source. This approach is particularly valuable for non-expert operators, since it reduces the time required to manage large dashboards up to 87% in some tasks (i.e., unifying panels).

- `C2` **- Evolution & Adaptation**: Our approach enables the operator to execute modifications to the dashboards by modifying values within the model, independently of the GUI of the dashboard platform. The approach has been meticulously designed to permit navigation through the various phases in an iterative manner. For instance, when a query for a KPI is altered, the operator has the capability to update it in the enriched dashboard model and regenerate the source code for the dashboard. This accelerates the update process (e.g., splitting visualizations by up to 29% for non-experts) and enhances flexibility in dashboard management.
- `C3` **- Levels of Complexity**: The proposed approach has been shown to reduce the technical complexity required to create and manage dashboards, especially for non-expert users. This is achieved by allowing operators to manipulate the dashboard without interacting with the dashboard platform. Consequently, even less experienced operators are able to contribute to the management of dashboards without requiring in-depth knowledge of platform functionalities. Also, more experienced operators are able to focus on advanced aspects such as customization and query syntax of dashboards. This, in turn, improves the overall efficiency of the process.
- `C4` **- Logical Organization**: The proposed approach has the potential to ensure consistent and optimized arrangement of panels by automating the logical organization of dashboards, thereby avoiding inconsistencies resulting from manual positioning. Furthermore, by eliminating the need for manual visual design, it facilitates the creation of more readable and accessible dashboards by even less experienced operators, thus guaranteeing the overall clarity of the information presented as demonstrated by the comprehensibility results reported for RQ1.

6.4. Limitations

Despite the benefits discussed above, we are aware of the limitations of the proposed approach. First, the metamodel deliberately adopts a simplified abstraction of dashboards, focused on KPIs, visualizations, and structural organization. From one hand this abstraction enables automation and technology-independence, from the other it necessarily constrains expressiveness. Advanced platform-specific features (e.g., highly customized visual encodings, interactive drill-down mechanisms, or complex cross-panel interactions) are not explicitly captured at the metamodel level. This reflects a design trade-off between generality and completeness, as it is the metamodel prioritizes portability and systematic generation over exhaustive coverage of platform-specific capabilities.

Second, the current version of the metamodel supports the association of a single data source per dashboard. While this choice simplifies the generation and validation processes, it may limit flexibility in scenarios where data from multiple heterogeneous sources need to be integrated. However, the metamodel has been designed in the perspective to be easily adapted and extended to support multiple data sources in future iterations.

Third, although the approach reduces the complexity of dashboard configuration, it assumes a certain familiarity with the abstract modeling language, which may still require a minimal learning curve for novice users.

Fourth, while the automatic organization of panels improves clarity and consistency, it may not always satisfy advanced customization needs or personal preferences in layout design. These aspects represent promising directions for future work aimed at improving both usability and expressiveness of the approach.

Regarding generalization, the approach has been validated in the context of cloud monitoring dashboards (e.g., Grafana, Dynatrace). Although the underlying principles (namely, explicit modeling of KPIs, separation between abstract dashboard specification and platform-specific generation, and model-level customization) are domain-agnostic, their applicability to other domains (e.g., business intelligence, IoT ecosystems) may require extending the metamodel with domain-specific concepts. Therefore, the current contribution should be interpreted as a domain-grounded instantiation of a broader model-driven pattern for dashboard engineering.

From an empirical perspective, the validation of the approach is based on a relatively modest sample size. This is justified by the nature of our evaluation, which is conceived as an exploratory and structured empirical assessment of an engineering artifact rather than a study aimed at deriving universal cognitive or usability rules. Yet the design incorporates multiple platforms, diverse participant profiles, standardized workload assessment instruments (e.g., NASA-TLX), and publicly available materials to enhance transparency.

Concerning technology-independence, it is worth specifying that the proposed metamodel is technology-agnostic at the implementation level (i.e., independent from specific dashboard platforms), but it is not ontologically neutral. The abstraction deliberately reflects the dominant conceptual paradigm of monitoring dashboards, including notions such as KPIs, visualizations, grouping structures, and data sources. The metamodel aims to capture a stable and widely adopted abstraction layer within the cloud monitoring domain. We are aware that this design choice may limit the expressiveness of models. The approach should therefore be interpreted as a domain abstraction tailored to operational monitoring scenarios, rather than as a universal visualization theory.

Another aspect that can be improved regards the reuse of modeling artifacts defined and generated during the dashboard generation process. We acknowledge the potential of a model reuse strategy in this context, where KPI definitions might be shared and reused by different stakeholders. Specifically, a concern-oriented strategy [29] or importable model libraries [30] could improve the usability of the approach by enabling the selection of already defined set of KPIs.

6.5. Threats to validity

We describe here the four basic threats to the validity according to the guideline by Wohlin et al. [31], and we discuss how we mitigated them.

Construct validity. The preparation of the survey and its usage to evaluate the expressiveness of our the generated dashboards may represent a threat to *construct validity*. This is due to the fact that the experts involved in our study might be influenced by our expectations, which could inadvertently affect the answers. To tackle this issue, we took proactive measures by first running a pilot of the study with our colleagues. Their feedback led to several revisions in the design of the questions, ensuring it was as unbiased as possible and independent from our initial expectations. Further, the experimental protocol and also the survey have been approved by the ethics committee of our academic institution.

Concerning the second research question, a treat to construct validity might be related with the inadequate tasks definition. If tasks are not clearly defined or are not representative of actual activities, differences in execution times may not accurately reflect the performance of the process. To mitigate this threat, we considered tasks derived from Grafana success stories, thus likely representative of relevant and useful tasks.

Internal validity. The survey of the first experiment could suffer from the threat of internal validity related to subjects guessing the generated and original dashboards during the experiment. To avoid this, we randomized the order dashboards are presented to subjects.

The second experiment included the authors among the participants, who performed the tasks and collected the time figures. This could represent another potential threat to validity due to possible expectation bias that could have unintentionally altered both execution and time recording. To mitigate this threat, an independent professional also participated in the study; further we publicly disclosed the recorded videos of the performed activity. Finally, the use of different environments and tools for each approach (i.e., GUI, manual configuration, and IDE for the MDE approach) leads to inherent variations in response times and operational processes, further complicating a direct and unbiased comparison between the different methodologies. To address this threat, we focused the comparison on the end-to-end time required to complete tasks, avoiding any fine-grained comparison of the activity performed within each task.

External validity. The limited number of participants in the study poses a threat to external validity. However, in the selection process, we made efforts to reach out to potential participants with diverse perspectives, such as data scientists/engineers, DevOps engineers, and software engineers. To mitigate this threat we also purposely avoided involving any student in the study, involving professionals only.

As regards the second research question, a threat to the external validity might result from the authors implementing the management tasks. While absolute timing figures might not be necessarily representative of the time needed by any user to complete their management tasks (e.g., some users might take more time than an informed operator), our design guaranteed a fair comparison among the approaches.

7. Conclusions

Dashboards are commonly used to visualize the data collected while monitoring large and complex software systems, such as cloud systems and systems of systems. However, the larger and the more complex the monitored systems are, the more difficult and time-consuming the configuration of a dashboard is. Identifying the visualizations that must be Incorporated in a dashboard, distributing the KPIs among the visualization, placing the visualizations in a coherent way, and configuration every element appropriately can be cumbersome.

In this paper, we addressed this problem proposing a model-driven approach that can automate the generation of a dashboard starting from a list of KPIs that are collected and the queries necessary to retrieve the values of these KPIs. Interestingly, the outcome of the dashboard generation process is represented as a model that can be further modified before generating the actual dashboard for the target dashboard technology,

The empirical results with Grafana Labs and Dynatrace dashboards show that the interpretability of the dashboards generated automatically is similar to the one of the manually configured dashboards. Moreover, the model-driven customization of the dashboard allows non-expert operators to act efficiently, sometime as efficient as expert dashboard users.

Future work concerns with extending dashboard generation capabilities with the possibility to add rules for the generation of alerts and the actuation of countermeasures.

Furthermore, although our work focuses on monitoring dashboards for cloud systems, some of the challenges we address, such as the dynamic nature of the monitored system and the difficulty in defining and measuring quality attributes, are also relevant in the broader context of Systems-of-Systems (SoS). Previous studies, such as the work by Ferreira et al. [32], have investigated reliability concerns in software-intensive SoS, highlighting the importance of effective monitoring mechanisms. Exploring how a model-driven dashboard approach could be extended to support SoS monitoring remains interesting for future work.

CRediT authorship contribution statement

Maria Teresa Rossi: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **Alessandro Tundo:** Writing – review & editing, Writing – original draft, Validation, Resources, Methodology, Conceptualization. **Leonardo Mariani:** Writing – review & editing, Supervision, Project administration, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been supported by the Centro Nazionale HPC, Big Data e Quantum Computing (PNRR CN1 spoke 9 Digital Society & Smart Cities); the Engineered MachinE Learning-intensive IoT systems (EMELIOT) national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY); and the COmmunity-Based Organized Littering (COBOL) national research project, which has been funded by the MUR under the PRIN 2022 PNRR program (Contract P20224K9EK).

Data availability

We have shared the link to our data/code in the article.

References

[1] J.D. Pereira, R. Silva, N. Antunes, J.L.M. Silva, B. de França, R. Moraes, M. Vieira, A Platform to Enable Self-Adaptive Cloud Applications Using Trustworthiness Properties, SEAMS '20, 2020, pp. 71–77, http://dx.doi.org/10.1145/3387939.3391608.

[2] A. Ali-Eldin, J. Tordsson, E. Elmroth, An adaptive hybrid elasticity controller for cloud infrastructures, in: 2012 IEEE Network Operations and Management Symposium, 2012, pp. 204–212, http://dx.doi.org/10.1109/NOMS.2012.6211900.

[3] A. Tundo, C. Castelnovo, M. Mobilio, O. Riganelli, L. Mariani, Declarative dashboard generation, in: IEEE International Symposium on Software Reliability Engineering Workshops, 2020.

[4] M. Kintz, M. Kochanowski, F. Koetter, Creating user-specific business process monitoring dashboards with a model-driven approach, in: Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, in: MODELSWARD 2017, 2017, pp. 353–361.

[5] A. Vázquez-Ingelmo, F. García-Peñalvo, R. Therón, D. Amo-Filva, D. Fonseca, Connecting domain-specific features to source code: Towards the automatization of dashboard generation, Clust. Comput. 23 (2020).

[6] A. Tundo, M. Mobilio, M. Orrù, O. Riganelli, M. Guzmàn, L. Mariani, VARYS: An Agnostic Model-Driven Monitoring-as-a-Service Framework for the Cloud, in: ESEC/FSE 2019, 2019, pp. 1085–1089.

[7] B. Mayer, R. Weinreich, A dashboard for microservice monitoring and management, in: 2017 IEEE International Conference on Software Architecture Workshops, ICSAW, 2017, pp. 66–69.

[8] L. Erazo-Garzon, K. Quinde, A. Bermeo, P. Cedillo, A domain-specific language and model-based engine for implementing IoT dashboard web applications, in: J. Maldonado-Mahauad, J. Herrera-Tapia, J.L. Zambrano-Martínez, S. Berrezueta (Eds.), Information and Communication Technologies, 2023, pp. 412–428.

[9] O. Belo, P. Rodrigues, R. Barros, H. Correia, Restructuring dynamically analytical dashboards based on usage profiles, in: International Symposium on Foundations of Intelligent Systems, 2014.

[10] H. Santos, V. Dantas, V. Furtado, P. Pinheiro, D.L. McGuinness, From data to city indicators: A knowledge graph for supporting automatic generation of dashboards, in: Extended Semantic Web Conference, 2017.

[11] M.T. Rossi, A. Tundo, L. Mariani, Towards model-driven dashboard generation for systems-of-systems, in: Proceedings of the 12th ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems, SESoS '24, 2024, pp. 9–12, http://dx.doi.org/10.1145/3643655.3643876.

[12] M.T. Rossi, A. Tundo, L. Mariani, GitLab repository, 2024, URL https://gitlab.com/learnERC/model-driven-adg. (Accessed 08 October 2025).
[13] S. Da Col, R. Ciucanu, M. Soare, N. Bouarour, S. Amer-Yahia, DashBot: An ML-guided dashboard generation system, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, CIKM '21, 2021, pp. 4696–4700.
[14] I. Dabbebi, S. Iksal, J. Gilliot, M. May, S. Garlatti, Towards adaptive dashboards for learning analytic: An approach for conceptual design and implementation, in: International Conference on Computer Supported Education, 2017.
[15] D. Deng, A. Wu, H. Qu, Y. Wu, DashBot: Insight-driven dashboard generation based on deep reinforcement learning, IEEE Trans. Vis. Comput. Graphics 29 (1) (2023) 690–700.
[16] L. Jiang, N.K. Tran, M. Ali Babar, Mod2Dash: A framework for model-driven dashboards generation, Proc. ACM Hum. Comput. Interact. 6 (EICS) (2022) 1–28.
[17] I. Alfonso, A. Conrardy, A. Sulejmani, A. Nirumand, F. Ul Haq, M. Gomez-Vazquez, J.-S. Sottet, J. Cabot, Building besser: An open-source low-code platform, in: International Conference on Business Process Modeling, Development and Support, 2024, pp. 203–212.
[18] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, M. Wimmer, Low-code development and model-driven engineering: Two sides of the same coin? Softw. Syst. Model. 21 (2) (2022) 437–446.
[19] A. Tundo, M. Mobilio, O. Riganelli, L. Mariani, Monitoring probe deployment patterns for cloud-native applications: Definition and empirical assessment, IEEE Trans. Serv. Comput. 17 (4) (2024) 1636–1654, http://dx.doi.org/10.1109/TSC.2024.3349648.
[20] A. Shatnawi, M. Orrù, M. Mobilio, O. Riganelli, L. Mariani, CloudHealth: A model-driven approach to watch the health of cloud services, 2018.
[21] S. Few, Now You See It: Simple Visualization Techniques for Quantitative Analysis, first ed., Analytics Press, Oakland, CA, USA, 2009.
[22] L. Wilkinson, The Grammar of Graphics, Springer-Verlag, Berlin, Heidelberg, 1999.
[23] T. Palpanas, P. Chowdhary, G. Mihaila, F. Pinel, Integrated model-driven dashboard development, Inf. Syst. Front. 9 (2007) 195–208.
[24] M.T. Rossi, A. Tundo, L. Mariani, Survey sample, 2024, URL https://drive.google.com/file/d/1gTDm54NcZDaEeGh5GURVUes2Tv9oSheY/view?usp=sharing. (Accessed 08 October 2025).
[25] P.E. Greenwood, M.S. Nikulin, A Guide to Chi-Squared Testing, John Wiley & Sons, 1996.
[26] Faster Capital, Bonferroni correction: Refining accuracy: The Bonferroni correction in multiple testing, 2026, URL https://bit.ly/4lo2jLE.
[27] T.V. Perneger, What's wrong with Bonferroni adjustments, BMJ 316 (7139) (1998) 1236–1238, http://dx.doi.org/10.1136/bmj.316.7139.1236, arXiv:https://www.bmj.com/content. URL https://www.bmj.com/content/316/7139/1236.
[28] S. Nakagawa, A farewell to Bonferroni: The problems of low statistical power and publication bias, Behav. Ecol. 15 (2004) http://dx.doi.org/10.1093/beheco/arh107.
[29] G. Mussbacher, J. Kienzle, A vision for generic concern-oriented requirements reusere@21, in: 2013 21st IEEE International Requirements Engineering Conference, RE, 2013, pp. 238–249, http://dx.doi.org/10.1109/RE.2013.6636724.
[30] A. Gerasimov, N. Jansen, J. Michael, B. Rumpe, S. Will, A model-driven approach to design, generation, and deployment of GUI component libraries, in: Proceedings of the 18th ACM SIGPLAN International Conference on Software Language Engineering, SLE '25, 2025, pp. 57–70, http://dx.doi.org/10.1145/3732771.3742713.
[31] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell, A. Wessln, Experimentation in software engineering, 2012.
[32] F.H.C. Ferreira, E.Y. Nakagawa, R.P. dos Santos, Towards an understanding of reliability of software-intensive systems-of-systems, Inf. Softw. Technol. 158 (2023) 107186, http://dx.doi.org/10.1016/j.infsof.2023.107186.