Visualization Support for Requirements Monitoring in Systems of Systems

Lisa Maria Kritzinger*, Thomas Krismayer*, Michael Vierhauser†, Rick Rabiser*, and Paul Grünbacher*

*Christian Doppler Lab MEVSS, ISSE, Johannes Kepler University Linz, Austria
rick.rabiser@jku.at
†Computer Science and Engineering, University of Notre Dame, IN, USA
mvierhau@nd.edu

Abstract—Industrial software systems are often systems of systems (SoS) whose full behavior only emerges at runtime. The systems and their interactions thus need to be continuously monitored and checked during operation to determine compliance with requirements. Many requirements monitoring approaches have been proposed. However, only few of these come with tools that present and visualize monitoring results and details on requirements violations to end users such as industrial engineers. In this tool demo paper we present visualization capabilities we have been developing motivated by industrial scenarios. Our tool complements ReMINDS, an existing requirements monitoring framework, which supports collecting, aggregating, and analyzing events and event data in architecturally heterogeneous SoS. Our visualizations support a ‘drill-down’ scenario for monitoring and diagnosis: starting from a graphical status overview of the monitored systems and their relations, engineers can view trends and statistics about performed analyses and diagnose the root cause of problems by inspecting the events and event data that led to a specific violation. Initial industry feedback we received confirms the usefulness of our tool support. Demo video: https://youtu.be/iv7kWzeNkdk.

Index Terms—Requirements monitoring, visualization, systems of systems.

I. INTRODUCTION

Today’s software systems are often developed as systems of systems comprising heterogeneous yet interrelated architectural elements. Common properties of SoS are decentralized control; support for multiple platforms; inherently volatile and conflicting requirements; continuous evolution and deployment; as well as heterogeneous, inconsistent, and changing elements [1]. As the full behavior of SoS emerges during operation only, system testing is insufficient to determine the systems’ compliance with their requirements. Thus, the involved systems and their interactions need to be continuously monitored during operation to discover unexpected behavior. Many monitoring approaches exist for this purpose as summarized in recent surveys [2], [3].

Robinson’s four-layer architecture of requirements monitoring [4] covers support for instrumenting systems, collecting and aggregating events and data, defining and checking behavior (e.g., using temporal logic), and visualizing requirements violations indicating deviations from the expected behavior. The presentation and visualization of monitoring results, however, is often insufficient in existing approaches. For instance, only 14 of 37 requirements monitoring frameworks identified in a recent systematic literature review [2] provide support for presenting the monitored events and event data to end users. While not all these frameworks have been developed with this goal, 38% is still a rather low number, particularly when considering that “most approaches do not provide a proper validation” with real-world systems and users, who would need or at least benefit from visualizations [3].

As part of our earlier research we have developed the tool-supported ReMINDS framework [5], [6] for requirements monitoring in SoS. The main use case of ReMINDS is continuous monitoring, i.e., checking and presenting the behavior of an SoS during its operation. An assessment of the usefulness of ReMINDS for collecting, aggregating, and analyzing events and event data at runtime in a study with industrial engineers [7] and further feedback from our industry partner suggested more advanced tool support for analyzing and visualizing monitoring results in real-world scenarios. For instance, the study showed that engineers from industry frequently require details on the monitored events and associated event data that led to requirements violations when diagnosing problems [8]. Furthermore, trends and statistics need to be computed allowing practitioners to understand which (kind of) requirements are violated during a particular time period, or if the frequency of violations is changing for specific systems. Particularly in systems of systems, visualizations need to exist for different levels of granularity, e.g., to depict the status and trends or details about specific violations for the overall SoS as well as individual systems.

In this tool demo paper, we present such visualization capabilities, developed in cooperation with industrial engineers in an ongoing research project on requirements monitoring in SoS. The new visualization features considerably complement and extend the existing ReMINDS tool suite [9].

II. BACKGROUND

ReMINDS [5] is based on a Requirements Monitoring Model (RMM) comprising the following key elements as defined in a meta-model [10]: the Requirements to be monitored are represented as Constraints, which can be defined using a domain-specific language (DSL) [6]. Constraint checks are triggered as soon as corresponding Events happen at runtime. The approach can check constraints on the occurrence, order, and timing of events and on the associated event
data. Events and data are collected by Probes instrumenting the monitored software. The RMM also comprises Scopes defining the provenance of events in the SoS. Scopes can be arranged hierarchically, thus representing the SoS architecture: for instance, a scope may represent a particular system, one or more components, or a connector (such as interfaces or APIs) between different parts of an SoS.

The RMM links requirements and scopes for diagnosing the violated constraints and tracing violations to their originating elements in the SoS architecture. Also, each constraint in the RMM is related with one or more events, e.g., the events involved in a particular sequence checked by the constraint. Further, for each event that led to a constraint violation, the related scope elements of the SoS architecture can be determined based on the probes when diagnosing violations.

The RE:MINDS framework [9] provides tools to define constraints using the DSL and to instrument an SoS using probes, e.g., using Aspect-Oriented Programming or direct code instrumentation [5]. Engineers can define scopes, types of events, and relations between the elements of the RMM. Probes send events and data from the monitored software to the RE:MINDS monitoring server, which aggregates and stores them. Applications can subscribe to this server to receive events and data. One application is the constraint checker [6], which continuously analyzes the stream of events to inform users about requirements violations. An end-user application is the monitoring client [9] providing a basic representation of scopes, probes, events and data, and requirements violations. Our tool demonstration paper describes the visualization capabilities we developed to complement and extend this tool to make it more suitable for industrial monitoring scenarios.

III. INDUSTRIAL MONITORING SCENARIO

Primetals Technologies is a leading engineering and plant-building company in the iron, steel, and aluminum industries. Its plant automation SoS (PAS) tracks and optimizes different stages of the metallurgical production process. It comprises systems for process automation of melting iron ore and raw materials, refining liquid iron to produce steel, and casting liquid steel. The correct interplay between these independently developed software and hardware systems is crucial to guarantee continuous, uninterrupted production and high-quality products.

Engineers at Primetals Technologies thus monitor if and how the PAS meets its requirements at runtime. Even though the basic capabilities essential for running the PAS and ensuring its safety are checked and guaranteed already on the machine automation layer, the full behavior of the process optimization and automation systems only can be checked at runtime when they interact with each other, the hardware, and third-party systems. For instance, engineers need to check the components’ correct timing or measure performance and resource consumption. For example, the Optimizer system optimizes the cutting of steel slabs to minimize scrap and maximize the resulting quality of the slabs. A key requirement for the Optimizer regards it correct operation and performance, i.e., after being triggered by another system or the user, it must perform the correct sequence of events (read tracking data, calculate optimizations, provide calculation results to other systems) within a certain time period. When a monitoring tool detects a violation of such a PAS requirement at runtime an engineer needs to investigate the cause of the violation (cf. Fig. 1):

1. The engineer checks the status of the overall plant automation SoS to determine for which systems or components violations have been detected. For this purpose she needs an overview of the monitored SoS and the violations that have occurred. Our tool provides such a status overview visualizing the monitored scopes and their relations in tree and highlighting scopes with violations as described in Section IV-A. For example, the status view may highlight the Optimizer system to indicate its error-state.

2. The engineer then also investigates how many violations already occurred for the different requirements. For instance, a pie chart could depict that the Optimizer requirement was checked X times and violated Y times (cf. Section IV-A). The engineer may require further details about the time period and the change rate of the detected violations. A monitoring tool must thus compute and visualize trends and statistics on violations. We provide views with area chart diagrams and line diagrams for this purpose (cf. Section IV-B). For example, these views can depict that the relative number of violations of the Optimizer requirement increase after a certain point in time.

3. The engineer then analyzes the requirements violation in detail, e.g., by reviewing details on the event time and event data recorded when the violation occurred. We provide a visualization for this purpose (cf. Section IV-C), which allows users to view the events related to a violation and inspect the data associated with these events, e.g., the events monitored when the Optimizer violation occurred.

4. The engineer then investigates the context of the violation by analyzing events and timing behavior. For example, the
violation of the Optimizer requirement could be the result of missing events or caused by delays in executing the required sequence of events. For this analysis the engineer needs details on the order of events preceding the violation to discover behavioral patterns, as provided by our Event Statechart view described in Section IV-C.

Eventually, to investigate the root cause of (missing) events that led to the violation, the engineer will inspect related source code or other artifacts. For this purpose, the engineer will use other engineering tools such IDEs, debugging tools, or issue trackers [8].

IV. VISUALIZATION SUPPORT

The industrial scenario and the study we conducted with engineers [7] motivated us to extend RE Minds with different tool views. Specifically, we developed six visualizations: (i) two status visualizations provide an overview of the monitored SoS and the detected constraint violations, (ii) two trending and statistics visualizations show details about each constraint (e.g., the number of violations in different time periods and the change rate of the violations), and (iii) two event visualizations provide details on the monitored events, including relations to other events and constraint violations.

We integrated the different views into the RE Minds monitoring tool. We also guide users who need to switch between views during diagnosis. Such guidance is essential in complex tools with multiple views to reduce the high cognitive load caused by many hidden dependencies not obvious for users unfamiliar with the details of the underlying approach. Hidden dependencies and hard mental operations are also two important dimensions in the cognitive dimensions of notations framework [11], a well-known method in the HCI community to analyze usability trade-offs we had also used earlier to assess the RE Minds tool [7].

The starting point to begin a drill down in the RE Minds tool is a status view from which the user can choose one particular system component. Using context menus she can activate the different trending, statistics and event views in the suggested order. Furthermore, a toolbar allows users to select an arbitrary view at any time. We use the Optimizer scenario as a running example throughout this section.

A. System Status Views

The RE Minds tool provides a tree-based view of the monitored SoS allowing the engineer to start investigating violations. Monitored components are depicted using icons differing by scope type [10]. Relations between scopes – e.g., the Caster system is part of the PAS SoS and the Optimizer is part of the Caster system – are depicted using edges with arrows. Components with violated constraints are highlighted by adding an error symbol to the corresponding node (cf. Fig. 2-(1)) when a violation is detected.

More detailed information about the status of the monitored SoS is provided via pie charts (cf. Fig. 2-(2)) for all active constraints, grouped by scope. Each pie chart shows the percentage of violations detected for one constraint among all checks performed for the constraint. Colors are used to indicate the status and severity: green (consistent: constraint checked but no violation detected), yellow (warning: violation with low severity), orange (failure: check went wrong, e.g., due to missing data), and red (error: critical violation). In the example depicted in Fig. 2-(2) the pie charts show that a constraint checking the RunID of the Optimizer (Run ID valid) has failed in all 28 cases and another constraint checking the correct order and timing of Optimizer events (Optimizer Cycle) was violated in 8 of 28 cases.

B. Trending and Statistics Views

For a more detailed analysis of the system status we provide diagrams for each constraint in Trending and Statistics views, using the same colors as the pie charts to indicate constraint status/violation severity. As depicted in Fig. 3a, area chart...
diagrams in the *Trending* view depict for each constraint the percentage of violations detected among the performed checks (y-axis) for a certain point in time (x-axis). The diagram is regularly updated with current values.

Through this view, it is easy for the user to identify a time range in which system behavior deviated from the usual behavior. In the example, we can see that in the last third of the monitored period, constraint checks started to lead to violations, which can be seen in the growing red area. The user can select a particular time range (cf. the black rectangle in the right half of Fig. 3a), e.g., the range with changing behavior. This time range is then used by all other visualizations until changed again, e.g., via the settings menu.

The *Status/Interval* view (cf. Fig. 3b) provides another visualization of the status of constraints. It depicts the absolute number of constraint checks and their status/severity for a certain interval as stacked bars. The bars’ height refers to the values shown on the right y-axis, i.e., the absolute number of performed checks which resulted in an error, failure, warning or consistent state in the time interval (x-axis). In the same diagram additional line plots depict the percentage of violations detected among the checks performed in the same intervals. The percentage is shown on the left y-axis. The information of the x-axis remains the same as for the stacked bar chart plot (time interval). The Status/Interval view is also regularly updated and shows the latest ten entries. A slider allows to move the time interval to the past (if not currently focused on a particular time range already).

### C. Live Event/Violation Views

Besides the constraints’ status, it is also useful to observe the behavior of the monitored events and the constraint checks and violations they trigger. For this purpose two views exist, both referring to the currently selected monitored SoS component.

The *Event Trending* view (cf. Fig. 4a) shows a stacked bar per ten second interval. Again, selecting a certain time range is possible. The behavior of this diagram equals the behavior of the Interval/Status diagrams, i.e., per default the latest ten entries (bars) are displayed and a slider allows to move the view to the past.

Whenever an event occurs (i.e., an Optimizer event), the stacked bar element is updated. Events are represented by colored rectangles added to the stack. Different colors represent different event types. Details such as event name, timestamp, and related event data can be displayed on demand when selecting a stacked bar element. Whenever a violation occurs, it is added as a dot in the associated bar with respect to the time the violation was detected. This relates the violation with the ‘event stack’ monitored when the violation occurred. Upon selecting a violation, the events involved in the violation are highlighted using darker variants of the same colors. Details such as event name, timestamp, and related event data can be displayed on demand when selecting a stacked bar element. Whenever a violation occurs, it is added as a dot in the associated bar with respect to the time the violation was detected. This relates the violation with the ‘event stack’ monitored when the violation occurred. Upon selecting a violation, the events involved in the violation are highlighted using darker variants of the same colors. In Fig. 4a, for example, the third stack (highlighted with a red border) is selected. This stack contains four events that led to a violation of the Optimizer Cycle constraint because the wrong event type occurred in the checked sequence of events. In particular, the constraint is triggered whenever the event type Optimizer.feedCyclicData_START occurs. Optimizer.feedCyclicData_FINISH is expected afterwards, but in the shown sequence Cutting.feedOptimizationResult did occur instead meaning that recent tracking data needed for the optimizations might not have been available. The user can display details about the violation in a separate dialog, e.g., the original constraint definition and an explanation.

The *Event Statechart* view (cf. Fig. 4b) shows the standard sequence of the occurrence of events as a graph (in this case the events monitored from the Optimizer). This graph does not depict every single event but the different event types. It is automatically generated based on the events monitored by REMINDS: when an event of a new type occurs, a new node and a connection from the previous event type (if any) are added to the graph. In case the event type already exists, either
a new connection is generated (if no connection existed before) or the connection line weight is updated, if the connection already existed.

The current event (type) is displayed with a white background (i.e., Optimizer.getOptimizationResult), while all others have a gray or red background. The current connection (from Optimizer.optimize_FINISHED to Optimizer.getOptimizationResult) is colored in black, all others in gray. When a violation occurs, the trigger event is highlighted using a red border (i.e., Cutting.feedOptimizationResult and Optimizer.feedCyclicData_START). It is also possible to show all events related with a certain violation by selecting the radio button at the top right of the trigger event node. For instance, Cutting.feedOptimizationResult is the trigger event of the constraint RunId valid. It led to a violation because the expected data could not be found. Optimizer.feedCyclicData_START as the trigger event of the Optimizer Cycle constraint led to a violation because the wrong event type occurred in the checked sequence of events as described above.

The weight of the connection lines provides information on how often this connection has occurred compared to all other connections, i.e., how often a particular event (type) was followed by a particular other event (type). This makes the normal flow of events and alternative courses visible to the user. As depicted in Fig. 4b, for instance, some ‘transitions’ between events in our Optimizer example occurred more often than others, i.e., 23 times (the thicker lines) vs. 1 time (the thinner lines). The absolute number can be shown using the tooltips of the connections.

It is also possible to ‘replay’ a past time period by visualizing recorded events and violations. The user can replay the event flow of a specific time period either step by step or by re-playing an entire time period at once. In both cases, the user can pause the event-flow at any time, for example, to investigate violations. The user can also define filters, e.g., to only show events of certain types.

D. Application to the Industrial Scenario

The drill-down scenario described in Section III is supported by interrelating the previously described views. The tree-based system status view (Fig. 2-(1)) provides an overview of the monitored system (cf. scenario step (1)) as the starting point. The user can either display trends and statistics for the overall system (cf. scenario step (2)) or use the context menu of a node of the tree-based status view to directly navigate to the violated constraints of that particular system component (cf. scenario step (3)), e.g., the Optimizer.

From each pie chart diagram, again using the context menu, the user can open the Trending view (the area chart of the same constraint; cf. Fig. 3a) to analyze trends regarding a specific violation (cf. scenario steps (2) and (3)). As already mentioned, it is possible to choose a specific time range in this view, e.g., the period with the most violations. If the user selects such a period, the view automatically switches to the Status/Interval view (Fig. 3b), depicting the bar chart and line plot diagram providing details for the selected time period.

The user can navigate to the Event Trending view (Fig. 4a) from both the Trending as well as the Status/Interval view using the context menu. If a time range was selected before, it is also used in the Event Trending view. The engineer can then investigate which events have occurred in this time range and which have led to violations (cf. scenario step (4)). Further details regarding the events can be investigated by switching to the Event Statechart view. The user can either replay the entire event-flow of the time range or walk through step by step (i.e., event by event). In our example, using the Event Statechart view the user can check the typical event-flow of the Optimizer and then check the Optimizer Cycle violation and its relation to particular events.

V. RELATED WORK

Even though most monitoring tools lack support for presenting monitoring results to end users [2], there are still some tools providing visualizations.

Several tools support monitoring and visualization of service-oriented systems. For example, the ECoWare [12] dashboard supports on-line and off-line violation drill-down analyses when monitoring service-based systems. RuMoR [13] supports monitoring and recovery of Web service applications, directly integrated in the IBM WebSphere environment. A violation reporter generates a web page with information about detected violations, as well as a form for selecting a recovery plan. The SOA4All framework [14] provides three different views: the Process View shows key performance indicators of monitored processes and services, the Resource View shows human or mechanized resources required for executing processes, and the Object View visualizes business objects such as inquiries, orders or claims. These views are supplemented with basic descriptive statistics. In contrast to these tools, we are not limited to service-based systems. We also offer visualizations providing details about monitored events and violations (cf. Section IV-C) in addition to logs, trends, and statistics.

Some monitoring tools particularly focus on visualization support of the architecture of the monitored systems. For example, ARAMIS [15] focuses on real-time visualization of software architectures. A web-based tool builds and visualizes sequence diagrams in real-time. Similarly, the MO-SAICO [16] approach to monitoring architectural properties allows to define and visualize property sequence charts using a UML-based notation. The ExplorViz [17] approach offers hierarchical visualization and multi-layer monitoring from landscape level to application level, e.g., for cloud-based applications. Our visualization of scopes and their relations also provides a higher-level visualization of the architecture of a monitored SoS as a starting point. Unlike the other tools, our visualizations allow to ‘dig deeper’ and investigate events and violations in detail.

Monere [18], an approach to monitor BPEL processes, comes with a resource tree view providing a hierarchical
overview of all monitored components and a dependency view displaying dependencies of a selected component. There are panels to display activities, log messages and metrics of monitored BPEL processes. Values are updated in real-time. Other views show charts of historical data, which supports users in identifying trends and anomalies. In contrast to tools like Monere, we are not focusing on (business) process monitoring and provide more details on constraint violations.

The requirements monitoring framework ReqMon [4] provides real-time feedback on requirement satisfaction using a digital dashboard for monitoring results presenting information on the status of requirements at runtime as well as historical data. The ReqMon presenter uses UML diagrams, such as message sequence diagrams and activity diagrams, for visualization. Our tool, apart from its different focus (systems of systems), also visualizes the relations of events and the violations they caused in more detail allowing to observe the event flow and visually recognize deviations from the normal course.

(Application) performance monitoring tools, such as Kieker [19] or Dynatrace (http://www.dynatrace.com) provide diverse sophisticated visualizations of a monitored system’s performance, but do not focus on monitoring requirements. Similar to our tool, they support a drill down from higher-level visualizations (performance or system health) to details about monitored components.

VI. CONCLUSION

In this tool demo paper we presented different visualizations we developed based on an industrial scenario and integrated into the REMINDS requirements monitoring tool for systems of systems. Specifically, different views in the tool depict the system status, detected requirements violations, violation trends and statistics, and the events leading to violations. REMINDS is currently used by our industry partner Primetals Technologies to capture and analyze events and data from particular systems in their plant automation SoS at runtime. Initial feedback we received from them confirms the usefulness of our tool support.

In our future work we plan to integrate further statistics views and want to implement warnings informing users about trends such as an increase of violations for a particular constraint. We also will evaluate the usefulness of our visualizations in detail, e.g., by performing a user study with our industry partner providing empirical data on the usability as well as utility of our tool.

ACKNOWLEDGMENTS

This work has been supported by the Christian Doppler Forschungsgesellschaft Austria and Primetals Technologies. Michael Vierhauser’s work was funded by the Austrian Science Fund (FWF) under Grant No. J3998-N31.

REFERENCES


