JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Lisa Maria Kritzinger

Submitted at
Institute for Software Systems Engineering

Supervisor
Priv.-Doz. Mag. Dr. Rick Rabiser

Co-Supervisor
DI Thomas Krismayer

September 2018

Visualization Support for Requirements Monitoring in Systems of Systems

Master Thesis
to obtain the academic degree of
Diplom-IngenieurIn
in the Master's Program
Computer Sience

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Abstract

Nowadays many software systems are so called systems of systems (SoS). The full behavior of SoS only emerges at runtime and these systems therefore need to be monitored during runtime to determine compliance with requirements. Although there exists many requirements monitoring solutions, only few of them provide visualizations to present monitoring results and details on requirements violations to end users.

The first part of this thesis describes how we extended the existing requirements monitoring framework ReMinds (Requirements Monitoring Infrastructure for Diagnosing Systems of Systems) by developing visualization capabilities motivated by industrial scenarios. The visualizations support a 'drill-down' scenario: starting with a graphical status overview of the monitored SoS, which allows determining violated systems or components, trends and statistics about monitoring results can be viewed and the root cause of problems may be diagnosed by inspecting the dependencies on events and event data that led to a violation.

The second part of this theses describes a study of the usefulness of the developed visualization capabilities. The study consisted of three steps: (1) a walk-through of the tool using the cognitive dimensions of notations framework from the field of human-computer interaction, (2) a user study involving four researchers followed by some refinements of the tool, and (3) another study involving five industrial practitioners. All subjects succeeded in monitoring a real-world automation system for steelplants and in diagnosing violations by using the visualization capabilities. The results show that the visualizations particularly helps to understand the behavior of complex systems. Based on the study results we derive implications, opportunities, and risks of using visualization in requirements monitoring tools.

Kurzfassung

Heutzutage sind viele Software Systeme sogenannte Systems of Systems (SoS). Alle Teilsysteme eines SoS interagieren erst während der Laufzeit, weswegen Laufzeitüberwachung erforderlich ist, um sicherzustellen dass sich das SoS gemäß seiner Anforderungen verhält. Es existieren zwar einige Lösungen zur Anforderungsüberwachung, aber nur wenige davon bieten Visualisierungen um einem End-Benutzer die Resultate der Überwachung und die Details von Anforderungsverletzungen zu präsentieren.

Der erste Teil dieser Arbeit beschreibt die Erweiterung des existierenden Anforderungsüberwachungsframeworks ReMinds durch die Entwicklung von Visualisierungsmöglichkeiten motiviert durch industrielle Szenarien. Die Visualisierungen unterstützen ein 'Drill-Down'-Szenario: angefangen mit einem grafischen Statusüberblick des überwachten SoS, welches die Bestimmung von Systemen oder Komponenten mit Anforderungsverletzungen erlaubt, können Trends und Statistiken über die Monitoring-Resultate betrachtet werden. Durch das Inspizieren von Abhängigkeiten von Ereignissen und Daten, welche zu einer Verletzung geführt haben, kann die Wurzel der Ursache des Problems diagnostiziert werden.

Der zweite Teil dieser Arbeit beschreibt die Durchführung einer Studie über die Zweckmäßigkeit der entwickelten Visualisierungsmöglichkeiten. Die Studie bestand aus drei Teilen: (1) ein Walkthrough des Tools unter Verwendung des Cognitive Dimension of Notations Frameworks aus dem Feld der Human-Computer Interaktion, (2) die Durchführung einer Studie mit vier Forschern gefolgt von einer Überarbeitung des Tools und (3) der Durchführung der selben Studie mit fünf Praktikern aus der Industrie. Alle Teilnehmer konnten erfolgreich ein echtes Automatisierungssystem für Stahlwerke überwachen und Probleme unter Verwendung der Visualisierungen diagnostizieren. Die Resultate haben gezeigt, dass die Visualisierungen vor allem dabei helfen, das Verhalten komplexer Systeme besser zu verstehen. Basierend auf den Studienergebnissen konnten wir schließlich Implikationen, Chancen und Risiken der Verwendung verschiedener Visualisierungen in Anforderungsüberwachungstools ableiten.

Contents

1 Introduction

Many software systems nowadays are so called Systems of Systems (SoS) whose full behavior only emerges at runtime when the systems interact with each other, hardware and/or legacy or third-party systems. System testing thus is insufficient to determine the systems' compliance with their requirements. The involved systems and their interactions must be continuously monitored at runtime to discover unexpected behavior.

Robinson [1] is one example of such an approach, it has defined a four-layer architecture of requirements monitoring that covers support for instrumenting systems, collecting and aggregating events and data, defining and checking behavior (e.g., using temporal logic [2]), and visualizing requirements violations to indicate deviations from the expected behavior. Many other monitoring approaches, supporting observing and checking the behavior of complex systems during operation, are summarized in a survey [3]. However, few of these existing monitoring approaches provide visualizations to present monitoring results to end users. Only 14 of 37 requirements monitoring frameworks identified in a recent systematic literature review [4] provide support for presenting the monitored events and event data. While not all of these frameworks have been developed with the goal to support end users, 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 earlier research the tool-supported ReMinds framework [5], [6] has been developed for requirements monitoring in systems of systems (SoS). The main use case of ReMinds is continuous monitoring, i.e., checking and presenting the behavior of an SoS during its operation. An earlier 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 (developing automation software for metallurgical plants) 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 are needed 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. As first part of this theses we developed such visualization capabilities in cooperation with industrial engineers in an ongoing research project on requirements monitoring in SoS. We also summarized the result in a tool demo paper [9]. The new visualization features considerably complement and extend the existing ReMinds tool suite [10].

Typically, monitoring tools have been evaluated with a focus on their performance overhead and feasibility, while only little empirical research exists on their usefulness for practical environments and industrial users. Assessing usefulness requires qualitatively studying users and their behavior [11]. Only few and rather limited usefulness studies exist for monitoring tools, for instance, Wassermann and Emmerich [12] presented an experiment assessing their Monere tool, Cobleigh et al. [13] reported on an experimental evaluation with end users specifying formal properties with PROPEL and Fittkau et al. [14] conducted a controlled experiments with 29 students to compare two software landscape visualizations and their support for program comprehension. Our own earlier study [7], to the best of our knowledge, so far is the only study involving industrial software engineers to assess the usefulness of a requirements monitoring tool. We thus conclude that the usefulness of visualization capabilities for requirements monitoring, especially for industrial practitioners, so far has not been studied in detail.

Empirical results can lead to increased acceptance of tools in industry, as they help engineers selecting and adapting tool capabilities for their application context. In particular, the usefulness of software engineering tools covers two dimensions: utility, i.e., to what degree the tool's functionality allows its users to do what is needed; and usability, the capability of the tool "to be understood, learned, used and attractive to the user, when used under specified conditions" [15], [16], thus showing how well users can exploit the offered functionality [17], [18]. As second part of this theses we present a study we have conducted with both researchers and industrial practitioners to assess the usefulness of the visualization capabilities as implemented in ReMinds. Specifically, this part provides the following research *contributions*:

- An initial assessment of the realized visualization capabilities' potential usefulness using the cognitive dimensions of notations framework [19] (Section 6.2).
- The design (Section 6.1) and results (Section 6.3 and 6.4) of an in-depth usability study involving subjects from industry and academia.

- A discussion of implications for developers of requirements monitoring tools, particularly with regard to visualization support (Section 7).

This theses is organized as follows: Section 2 explains the ReMinds framework, along with the used Requirements Monitoring Model (RMM) and the tool suite developed based on the RMM, Section 3 summarizes the motivating industrial scenarios and the derived required visualization capabilities, Section 4 explains the developed visualizations in detail, Section 5 shows a few selected implementation details and Section 6 explains the usefulnesses study method, the Cognitive Dimensions of Notations [19] and their assessment, followed by a description of the actual study and the results. Section 7 discusses implications for developers of requirements monitoring tools and Section 8 rounds out the theses with a summary and conclusions.

2 Background and Related Work

The following chapter describes the Requirements Monitoring Model the ReMinds framework is based on this model. Section 2.3 describes the monitoring tool suite. The developed visualization capabilities are an addition to the end user client, the *Monitoring Center* (cf. Section 2.3.1). Some other monitoring solutions and their supported visualizations are described in Section 2.4.

2.1 Requirements Monitoring Model

ReMinds [5] is based on a Requirements Monitoring Model (RMM) comprising the following key elements as defined in a meta-model [20] (cf. Figure 2.1).

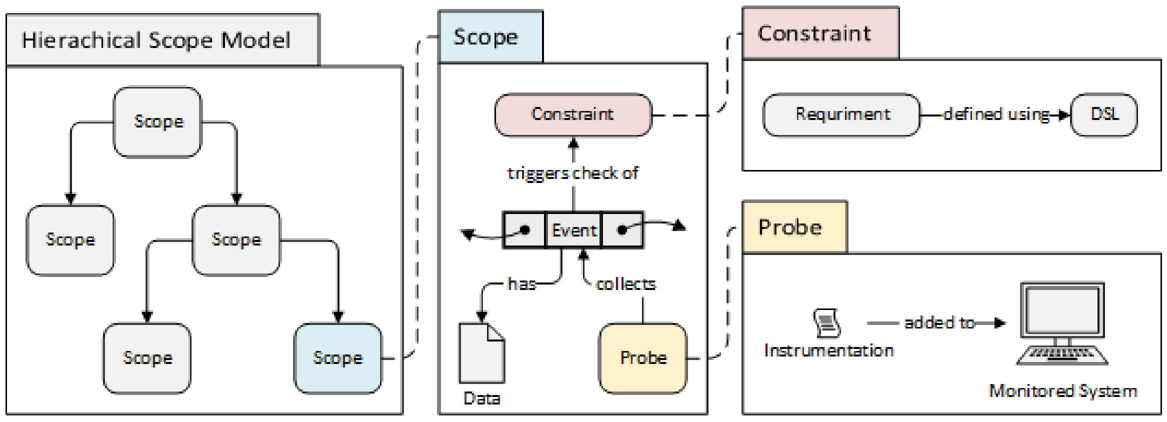


Figure 2.1: Runtime Monitoring Meta-Model.

The RMM comprises *Scopes* which can be arranged hierarchically, thus representing the SoS architecture. A scope may represent a particular system, one or more components, or a connector (such as interfaces or APIs) between different parts of a SoS. Every *Probe* is assigned to the scope which represents the source of this probe, e.g. the system where the source code was instrumented to define this probe. A probe sends *Events* to the monitoring framework. These events may define the occurrence of a relevant system operation or interaction at a certain time. Events can also contain data elements, which are needed for the evaluation (which is triggered as soon as corresponding events happen at runtime) of a certain *Requirement*. Requirements describe functionalities, properties or specific behaviors which shall be monitored at runtime. Typically requirements ensure the order of events (event A before event B) or the correctness of event-data (d>0). In ReMinds requirements are defined as *Constraints* using a domain-specific language (DSL) [6]. 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.

2.2 REMINDS Framework

This RMM is realized as flexible Monitoring-Framework (cf. Figure 2.2) by ReMinds, which consists of five layers: Probing and Instrumentation, Aggregation and Distribution, Processing and Analysis, Views and an additional cross-cutting layer over all other layers, which is responsible for managing variability [21].

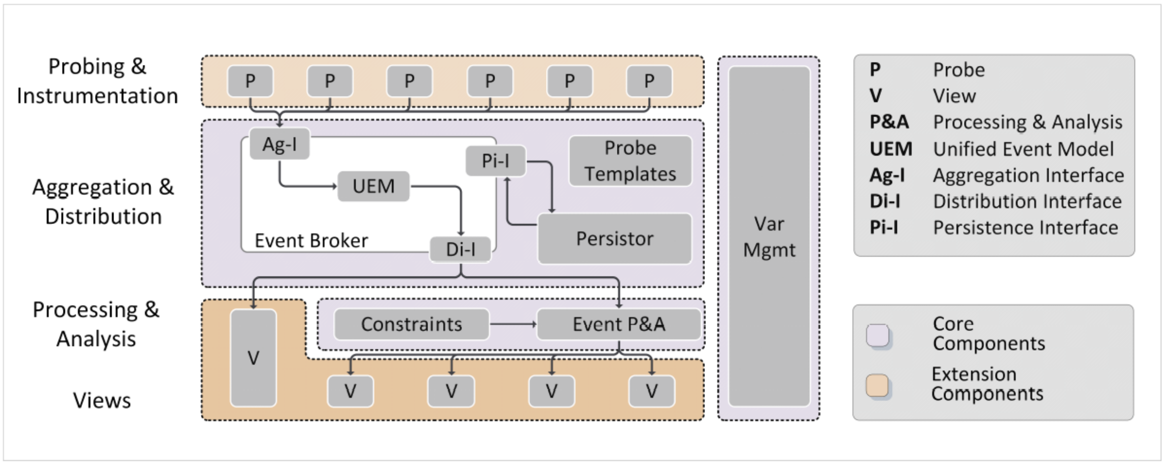


Figure 2.2: ReMinds Framework Architecture [5].

Probing and Instrumentation. The first layer collects events as well as corresponding data from different systems of the SoS using probes intercepting or extracting system information. ReMinds offers templates for probes which are often needed. Depending on the system to be monitored different probes are designed to either extract data directly from the running systems or to analyze the output of these systems.

For example, Java-based system can be instrumented without modification of the source code by using AspectJ (an aspect-oriented extension of Java for the implementation of an event-based system) [22]. For example, AspectJ allows to define that before each call of certain methods an event "MethodX-occurred" is sent. Other technologies like C and C++, require the modification of the source code to implement probes.

Aggregation and Distribution. Monitoring data of different systems of the SoS is collected by this layer by providing an access point for probes. Monitored events are automatically forwarded to all interested clients registered at the distribution server. ReMinds also provides a persistence interface for storing events and their data, monitoring results and informations about the system status. The persistence component executes the archiving of the monitored events and allows to query of past events, data or constraints. Persistent data can also be forwarded to registered clients for visualization and analysis.

Processing and Analysis. To allow the definition of global constraints, which require data from different systems, low-level-event-processing and high-level-constraint-management is separated in ReMinds. The evaluation of the data is thus done on a separate layer. All needed constraints are defined and verified here.

ReMinds provides a domain-specific language (DSL) [23] for constraint definition and a constraint checker. Every constraints starts with the indication of an event, which triggers the evaluation of this constraint, called trigger-event, followed by the specification of the conditional statement (cf. Figure 2.3).

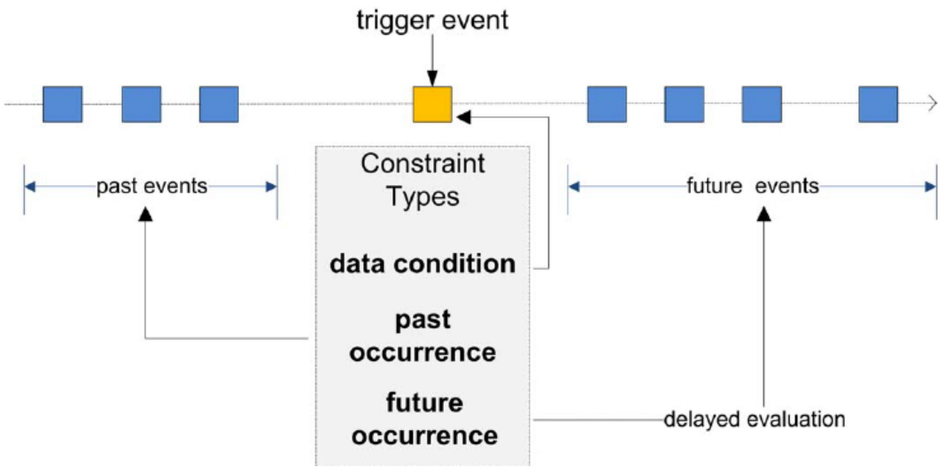


Figure 2.3: Event-flow and Constraint Types of the ReMinds DSL [6].

Constraints can have three types: (1) data constraints allow checking if the event data equals a certain value, (2) past occurrence constraints allow checking which events must have occurred before the trigger event and/or in which order they must have occurred, (3) future occurrence constraints allow checking the occurrence, order and/or timing of events that occur after the trigger-event.

Views. ReMinds provides an end-user application, the monitoring client [10], showing a basic representation of scopes, probes, events and data, and requirements violations. We developed visualization capabilities (Section 4) to complement and extend this tool to make it more suitable for industrial monitoring scenarios.

2.3 REMINDS Tool

ReMinds consists of different tools, which have been developed based on the monitoring framework.

The tools provide support for the layers of the framework and are therefore divided into several categories (cf. Figure 2.4).

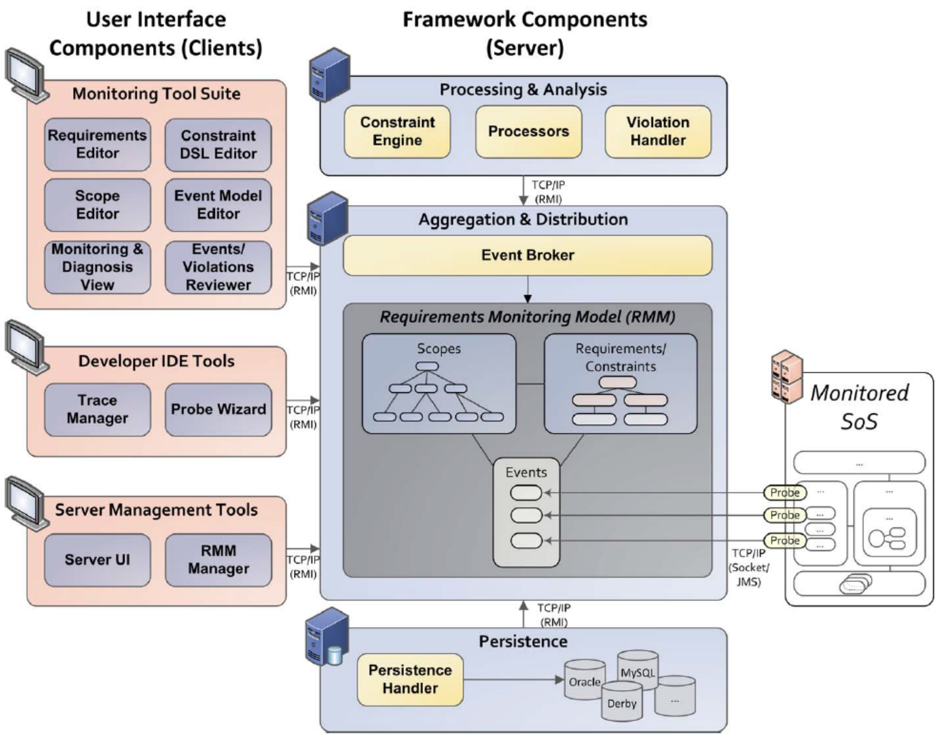


Figure 2.4: ReMinds tool architecture [10].

The framework component consists of the *Processing & Analysis* component containing event processors, the constraint checking engine and a violation handler, the *Aggregation and*

Distribution Component containing an event broker and an implementation of the RMM and a *Persistence* Component providing support for different databases and distributed file systems.

The *User Interface* Component consists of different user interfaces which provides editors and tools for managing the elements of the RMM, like constraints or scope models.

2.3.1 Monitoring Center

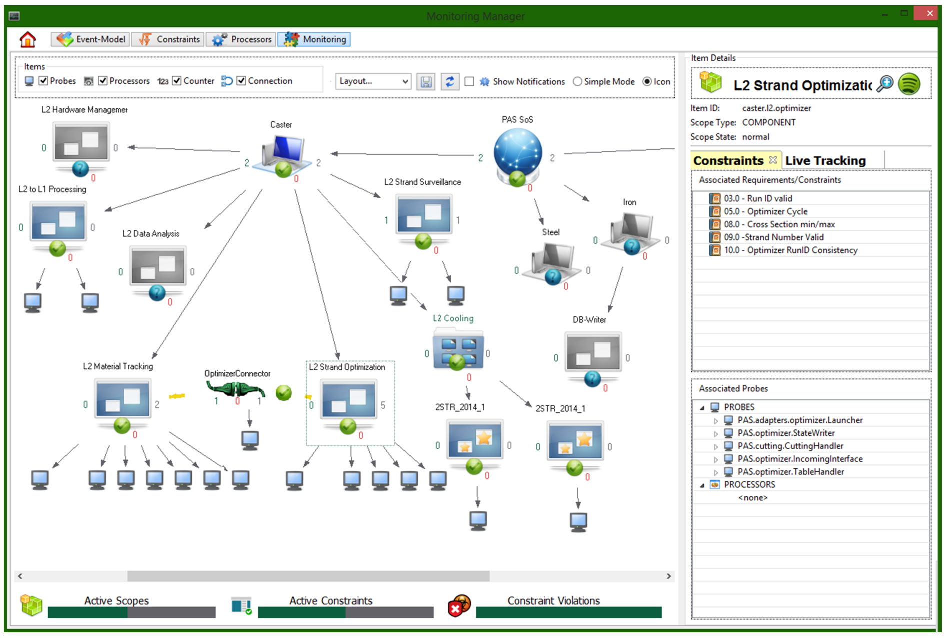


Figure 2.5: Previous version of the REMINDS tool.

The end-user interface, called monitoring center, provides all needed editors to setup a scope/event model, to define constraints and to analyze and diagnose event flows and constraint checks of such models at runtime.

In particular, the *Requirements Editor* allows to define requirements, from which constraints can be defined in a next step in the *Constraint DSL Editor* [6]. This editor is based on the XText [24] framework, it supports auto-completion, syntax-highlighting and is integrated with the monitoring server, so that new constraints are automatically transferred to the server, compiled and activated at runtime. The hierarchical scope and event models can be defined and arranged using the *Scope Editor* and the *Event Model Editor*.

The *Events/Violations Reviewer* provides capabilities for reviewing past violations and recorded event traces. The *Monitoring & Diagnosis* view can finally be used to observe the current system status (active scopes, incoming events, constraint violations). The developed visualization capabilities are an addition to this last mentioned view to improve the possibilities of analyzing and diagnosing system behavior and constraint violations, particularly for end users. Figure 2.5 shows the old version of the Monitoring & Diagnosis view before we added the visualization capabilities and other changes to improve the tools usability.

2.4 Visualization in other Monitoring Tools

Even though most monitoring tools lack support for presenting monitoring results to end users [4], some tools provide visualizations for several purposes.

Monitoring and visualization of service-oriented systems is, for example, supported by ECoWare [25]. Its dashboard supports on-line and off-line violation drill-down analyses when monitoring service-based systems. RuMoR [26] 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 [27] provides three different views that visualize the key performance indicators of monitored processes and services, the human or mechanized resources required for executing processes, and the business objects such as inquiries, orders or claims. These views are supplemented with basic descriptive statistics. In contrast to these tools, ReMinds is not limited to service-based systems. It also offers visualizations providing details about monitored events and violations (cf. Section 4) in addition to logs, trends, and statistics.

Visualization support of the architecture of the monitored systems is supported by some other monitoring solutions like ARAMIS [28] which focuses on real-time visualization of software architectures. A web-based tool builds and visualizes sequence diagrams in real-time. Similarly, the MOSAICO [29] approach to monitoring architectural properties allows to define and visualize property sequence charts using a UML-based notation. The ExplorViz [14] 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, but unlike to the other tools it also allow to 'dig deeper' and investigate events and violations in detail.

Monitoring BPEL processes is purpose of the Monere [12] which provides a hierarchical overview for resource trees of all monitored components and a dependency view displaying dependencies of a selected component. Unlike to ReMinds Monere focuses on (business) process monitoring and provide more details on constraint violations.

The *requirements monitoring* framework ReqMon [1] provides a digital dashboard for monitoring results presenting information on the status of requirements at runtime along with historical data. The ReqMon presenter uses UML diagrams, like message sequence diagrams and activity diagrams, for visualization. Our tool has a different focus (systems of systems) and it 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 [30] or Dynatrace [31] 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 to details about monitored components.

3 Motivating Industrial Scenario

The ReMinds framework is developed in cooperation with Primetals Technologies, a leading engineering and plant-building company in the iron, steel, and aluminum industries [5]. The company's 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 can only 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.

3.1 Running Examples

To explain and illustrate both parts of this thesis we will use two running examples for such components and their requirements.

Optimizer. This 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 its correct operation and performance, i. e., after being triggered by another system or the user, it must perform the correct sequence of events (read material tracking data, calculate optimizations, provide calculation results to other systems) within a certain time period.

Quality Control System (QCS). This system tracks the overall production process through multiple stages, helps to ensure the quality of the produced steel slabs, and allows to detect errors in the process flow. For this purpose the QCS, e.g., checks that different parts of the production process are executed in the correct order and adhere to given time constraints.

Constraints for such requirements can be defined in our DSL [23] as follows: *Constraint 1* defines that whenever the event `Optimizer.feedCyclicData_START` occurs, the indicated sequence of events must also occur within 10 seconds. Other events may also occur between this sequence.

```
trigger = if event "Optimizer.feedCyclicData_START" occurs
condition = events "Optimizer.feedCyclicData_FINISHED",
    "Optimizer.optimize_START",
    "Optimizer.writeOptiData",
    "Optimizer.optimize_FINISHED",
    "Optimizer.getOptimizationResult",
    "Cutting.feedOptimizationResult"
occur ignore others within 10 seconds
```

Listing 3.1: C1 — Order of Events.

Constraint 2 defines that whenever the event `QCS.Strip_Data` occurs, event `QCS.Genea-ogy_Data` must occur as well, maybe along with other events, within 12 seconds and the indicated data elements must be equal to each other.

```
trigger = if event "QCS.Strip_Data" occurs
condition = events "QCS.Genealogy_Data" as "e" where
    "e".data("general", "ID") == data("general", "ID")
occur ignore others within 12 seconds
```

Listing 3.2: C2 — ID valid.

In the context of this thesis, to test and present such scenarios without the need of running them in a real steel plant we built mockups which simulate the system, e.g., they send a default sequence of events based on the real events recorded from Primetals' system earlier. These mockups are built to trigger evaluations, which result in multiple severity levels. In particular, we defined one mock-up probe for each scenario.

Mockup-Probe 1 is associated with the scope `Optimizer` and sends the events indicated in Table 3.1 in an endless loop with a certain timeout between the events. In some randomly picked cases the event `Optimizer.getOptimizationResult` is not sent and in some other randomly picked cases the timeout between `Optimizer.writeOptiData` and `Optimizer.-optimize_FINISHED` increases from one second to five seconds. Therefore constraint *C1 - Order of events* will be violated for two reasons: (1) a timeout will occur because the timeouts sum up to more then 10 seconds, (2) a wrong sequence will be detected when `Cutting.feedOptimizationResult` is sent instead of `Optimizer.getOptimizationResult`.

Table 3.1: Mockup-Probe 1.

Event Type	**Timeout** $\sum$ 7000 or 11000	**Occurrence**
Optimizer.feedCyclicData__START	500	every loop
Optimizer.feedCyclicData__FINISHED	1000	every loop
Steel.StatusUp	500	every loop
Steel.StatusDown	1000	every loop
Optimizer.optimize__START	500	every loop
Optimizer.writeOptiData	1000 or 5000	every loop
Optimizer.optimize__FINISHED	500	every loop
Optimizer.getOptimizationResult	1000	randomly
Cutting.feedOptimizationResult	1000	every loop

Mockup-Probe 2 is associated with a scope named `QCS-Downstream`, a component of the QCS system. It sends the events indicated in Table 3.2 in an endless loop. `QCS.Strip_Data` containts the data element `general.ID` with the value 0 in every loop, while `QCS.Geneaogy_Data` contains randomly either no data element or also `general.ID` where the value is also randomly either 0 or 1. This means the evaluation of constraint *C2 - ID valid* will fail in some cases, because the data element of `QCS.Geneaogy_Data` cannot be found and lead to a violation in some other cases because the IDs of both events are not equal.

Table 3.2: Mockup-Probe 2.

Event Type	**Data Element**	**Data Value**
Caster.SData	general.ID	0
QCS.Strip__Data	*None*	
Caster.Schedule	*None*	
QCS.Genealogy__Data	general.ID or *None*	0 or 1
QCS.Cyclic	*None*	

3.2 Required Visualization Capabilities

When violations of such constraints are recognized at runtime, the engineers need to investigate the cause of the violation: Based on these industrial scenarios and through checking existing monitoring tools [9] [3], we define the following four key visualization capabilities (cf. Figure 3.1) as essential to support detecting and diagnosing violations when monitoring complex software systems such as the PAS:

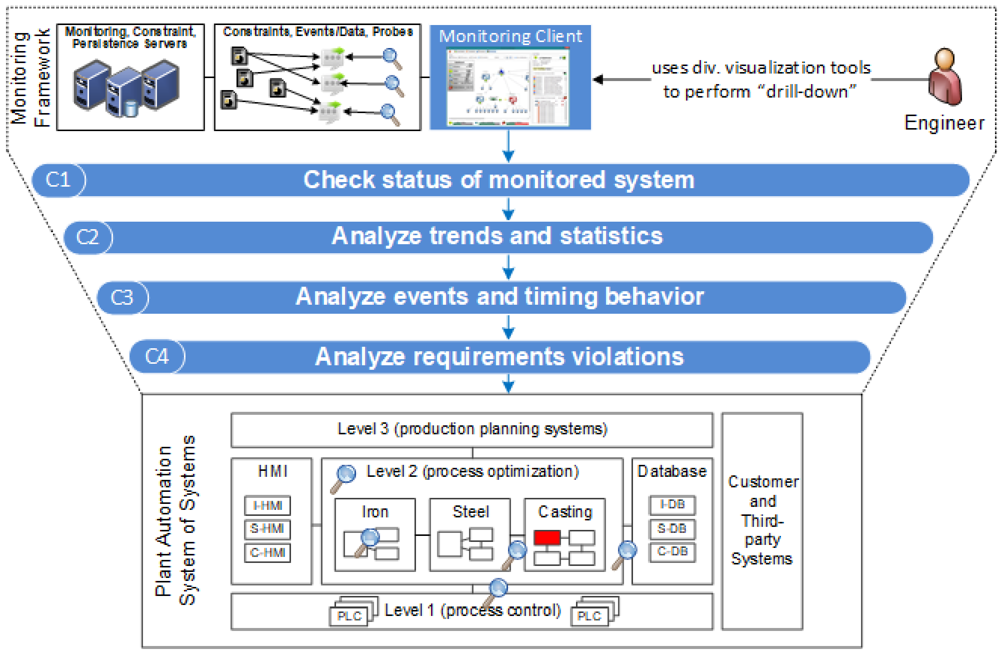

Monitoring Framework
Monitoring, Constraint, Persistence Servers
Constraints, Events/Data, Probes
Monitoring Client
uses div. visualization tools to perform "drill-down"
Engineer
C1 Check status of monitored system
C2 Analyze trends and statistics
C3 Analyze events and timing behavior
C4 Analyze requirements violations
Plant Automation System of Systems
Level 3 (production planning systems)
HMI
I-HMI
S-HMI
C-HMI
Level 2 (process optimization)
Iron
Steel
Casting
Database
I-DB
S-DB
C-DB
Customer and Third-party Systems
PLC
Level 1 (process control)
PLC

Figure 3.1: Monitoring a plant automation system of systems: drill-down scenario.

(C1) Check system status. In case of a violation, engineers first need to check the status of the overall system (e.g., the PAS) to determine for which (sub-)systems (e.g., the QCS) or components violations have been detected. Thus a key capability of monitoring tools is a high-level visualization of the monitored system to understand its architecture and quickly assess the status of its components. For this purpose they need an overview of the monitored SoS and the violations that have occurred. Our tool provides such a status overview visualizing the monitored systems and their relations in a tree and highlighting systems with violations. For example, the status view may highlight the Optimizer system to indicate its error-state.

(C2) Analyze trends and statistics. Engineers then need to investigate which and how many violations already occurred for the monitored system(s), and in which time period. For instance, a pie chart could depict that one specific requirement was checked X times and violated Y times. Also, they may require further details about the change rate of the detected violations. Thus a monitoring tool must allow to compute and visualize trends and statistics on violations. For this purpose we provide views with area chart diagrams and line diagrams. For example, these views can depict that the relative number of violations of one or several QCS constraints increased after a certain point in time.

(C3) Analyze events and timing behavior. Upon the detection of violations engineers need to analyze details of the occurred events and their timing behavior (e.g., which events of the Optimizer occurred in which order). For example, the violations could be the result of missing events or caused by delays in executing the required sequence of events. For this analysis the engineers need details on the order of events preceding the violation to discover behavioral patterns, as provided by our Event Flow view described in Section 4.3. For each violation or performance issue detected, a monitoring tool must allow to check events and timing behavior.

(C4) Analyze violation details. Engineers then *analyze the violations* 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, 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 QCS violation occurred. For example, when analyzing a violation for the QCS, an engineer needs to view the QCS events monitored and their associated data items in relation to constraints and their violations.

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].

4 REMINDS Visualizations

This chapter describes the developed visualizations which extend the ReMinds Monitoring Center (Section 2.3.1). These visualizations were motivated by the earlier conducted study with engineers of Primetals Technologies [7] and the industrial scenario as described in Section 3.

The extension consists of six visualizations in three groups:

1. *Status Overview.* These visualizations provide an overview about the current status of the monitored SoS, the detected constraint violations and how often they are violated (cf. Section 4.1).

2. *Trending and Statistics.* The two corresponding views shows more details about each active constraint. They provide not just the total amount of violations of each constraint but also the number of violations in different time periods and the change rate of the violations (cf. Section 4.2).

3. *Event Behavior.* These views provide details on the monitored events, like the common sequence they are appearing along with indications of outliers and their dependencies to constraint violations (cf. Section 4.3).

These views are developed to allow usage as independent views, but we also provide guidance within the ReMinds tool to support the drill-down scenario (cf. Section 3).

Figure 4.1 shows how these views are implemented in the ReMinds framework. All views are an addition to the *Monitoring & Diagnosis* view of the client, which must be subscribed to the server to receive the needed informations (events, violations). The view is separated into three parts, a tab-folder where *Trending and Statistics* views are added via the provided extensions point (EP), a view which provides the system overview and another tab-folder, in which all *Event Behavior* views are added via EP.

The starting point in ReMinds to begin a drill down is the overview of the monitored SoS, in which the user can choose one particular system or component to diagnose. Using context menus or buttons one can switch among the different trending, statistics and event views in the suggested order. Furthermore, a status bar provides important informations for

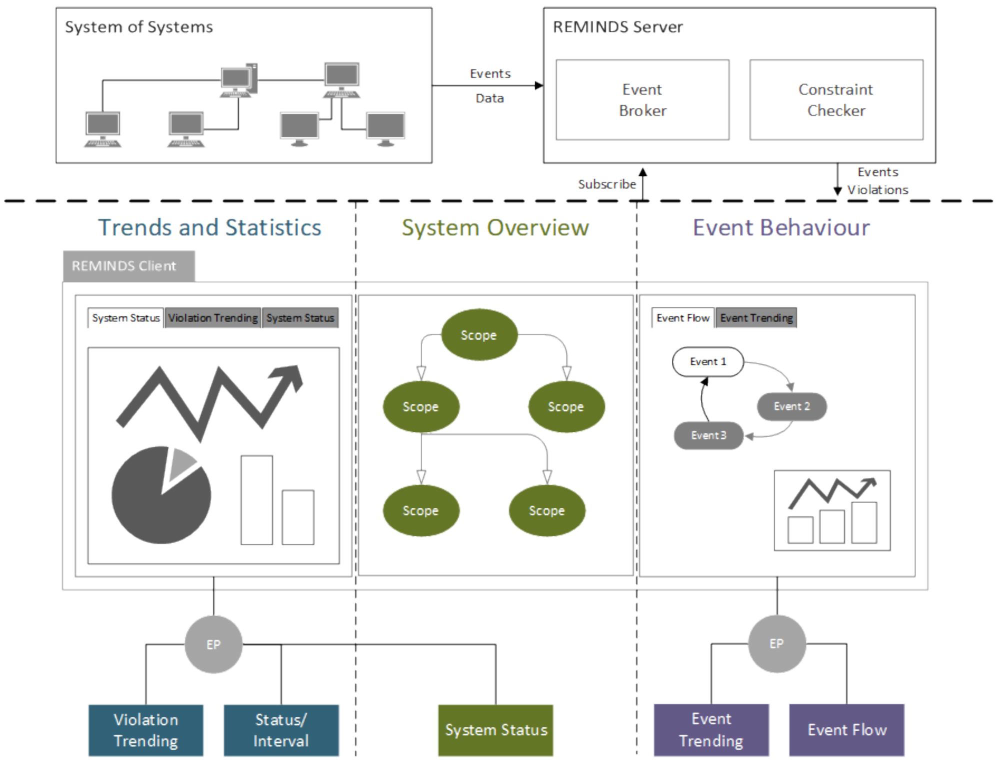


Figure 4.1: Visualization Capabilities extended to the REMINDS Framework.

the visualizations, e.g., the currently selected time interval, which filters all visualizations except of the *Status Overview* views to show just information about this interval. We use the `Optimizer` scenario as a running example throughout this Section and additionally the `QCS` scenario in Section 4.1.

4.1 System Overview

REMINDS provides a tree-based view, which consists of important components and systems of the monitored SoS, which are depicted using icons for different scope types (SoS, Systems, Components or Connectors) [20]. Edges with arrows indicate relations between scopes, e.g., the `Caster` system is part of the `Plant Automation System` and the `Optimizer` component is part of the `Caster` system. Scopes with violated constraints are highlighted by adding an error symbol to the corresponding node and by coloring the nodes icon red when the first violation is detected. Figure 4.2 (right) shows that both active scopes (= a probe has been registered to them) `QCS-Downstream` and `Optimizer` have violated constraints.

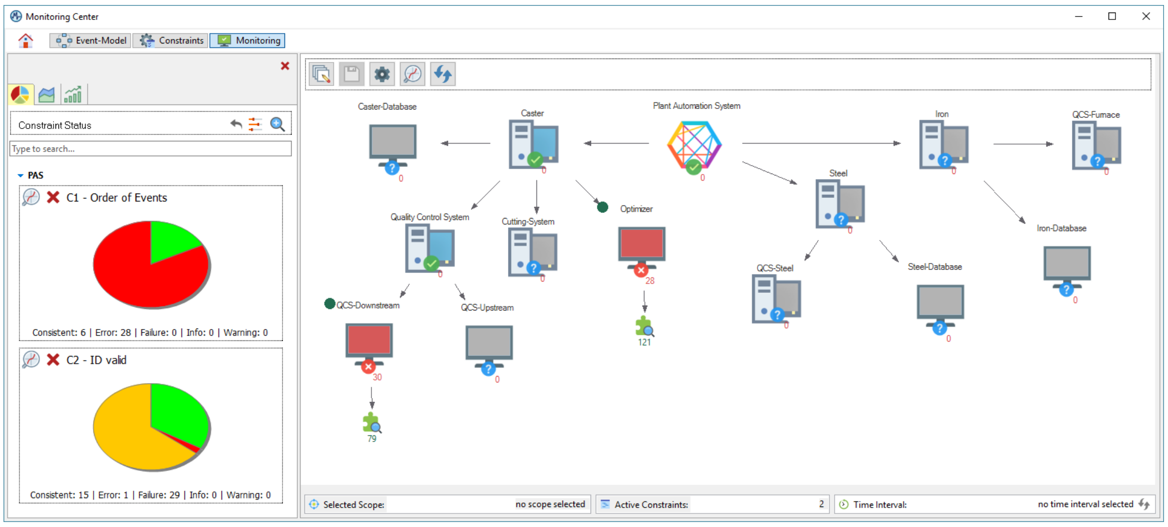


Figure 4.2: System Status Overview.

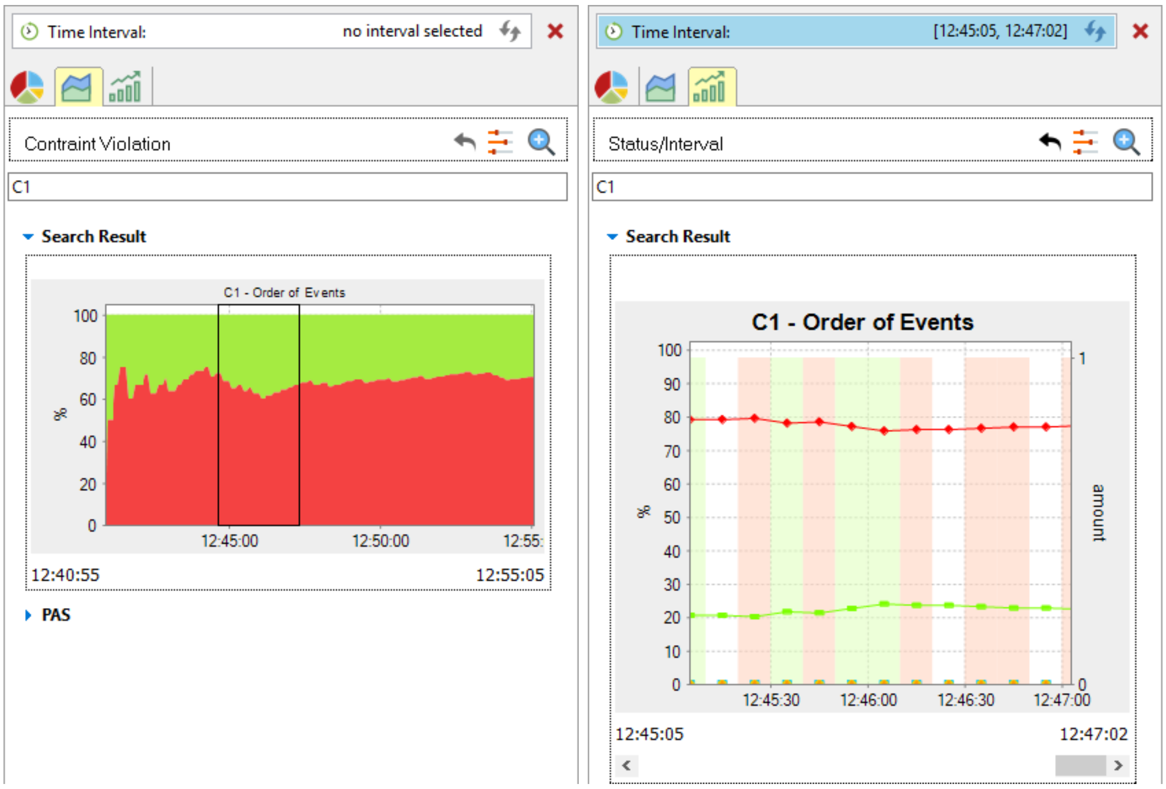
More details about the constraint status are provided via pie charts (cf. Fig. 4.2 (left)). This view shows one pie chart for each active constraint (= not manually deactivated and an evaluation has been triggered) grouped by user-defined constraint groups. Each pie chart shows the percentage of violations detected among all checks performed for this constraint.

Severity levels of violations are indicated using different colors: *green* (consistent = the constraint has been checked and no violation was detected), *orange* (failure = the constraint check has been triggered but could not be evaluated, e.g., due to missing data), *yellow* (warning = a violation with low severity has been detected for this constraint), *red* (error = a critical violation has been detected for this constraint).

In the example depicted in Figure 4.2 (left) the pie charts show that a constraint checking the correct order and timing of `Optimizer` events (`C1 - Order of Events`) was violated in 28 of 34 cases and the evaluation of another constraint checking the `ID` of the `QCS-Downstream` (`C2 - ID Valid`) was triggered 45 times, 29 times the evaluation failed, 15 times it was evaluated without detecting any violation and one evaluation resulted in an error.

4.2 Violation-related Visualizations

The *Trending and Statistics* views provide diagrams for each constraint to allow a more detailed analysis of the system status, using the same colors as the pie charts to indicate constraint status/violation severity. Figure 4.3a shows that area chart diagrams in the *Violation*



(a) Trending area chart diagrams.

(b) Status/Interval diagrams.

Figure 4.3: Trending and Statistics Views.

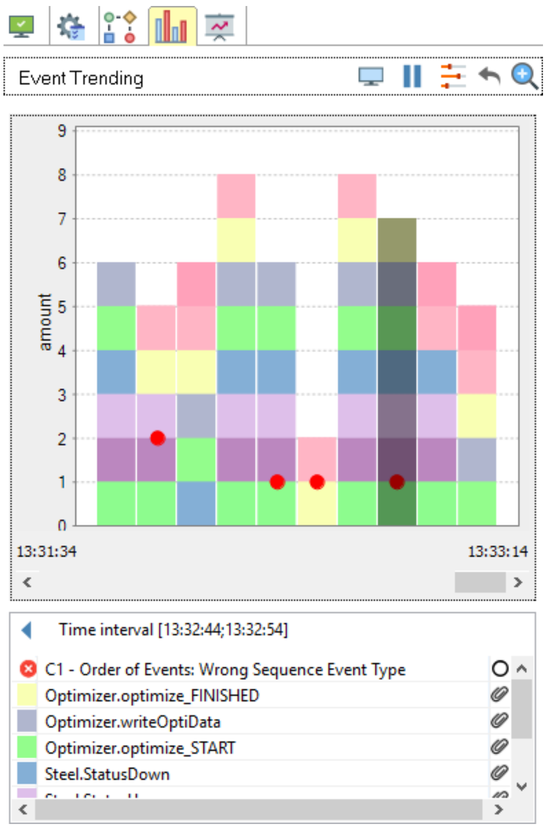
Trending view depict the percentage of violations detected among the performed checks (y-axis) for a certain point in time (x-axis) for each constraint. The diagram is regularly updated with current values.

With these area diagrams it is easy for the user to identify time ranges in which the system behavior changed, e.g., the figure shows that the amount of violations decreases at about 12:45 (the green area is growing) and increases again a few minutes later (the red area is growing), afterwards the violation rate did not change much anymore. The user can select such a time range (cf. the black rectangle in Fig. 4.3a), e. g., the range with changing behavior. The selection of a time range is then used by all other visualizations except of the *System Overview* views until changed again or reseted.

The *Status/Interval* view (cf. Fig. 4.3b) provides another visualization of the status of constraints. It provides a multi-plot diagram for each constraint where the absolute number of evaluations and their status/severity for a certain interval is shown as stacked bars and the percentage of violations detected among the checks performed in the same intervals as lines.

The height of bars refers to the values shown on the right y-axis, i. e., the absolute number of performed evaluations, which resulted in an error, failure, warning or consistent state in the time interval. The left y-axis depicts the percentage referring to the line diagram. The x-axis indicates the time interval, which remains the same information for both plots. This view is updated in the same regular intervals as the area diagrams and shows the last ten entries with a slider allowing to move the time interval to the past (if not currently focused on a particular time range already). In the example this view already shows the information for the selected time interval in Figure 4.3a.

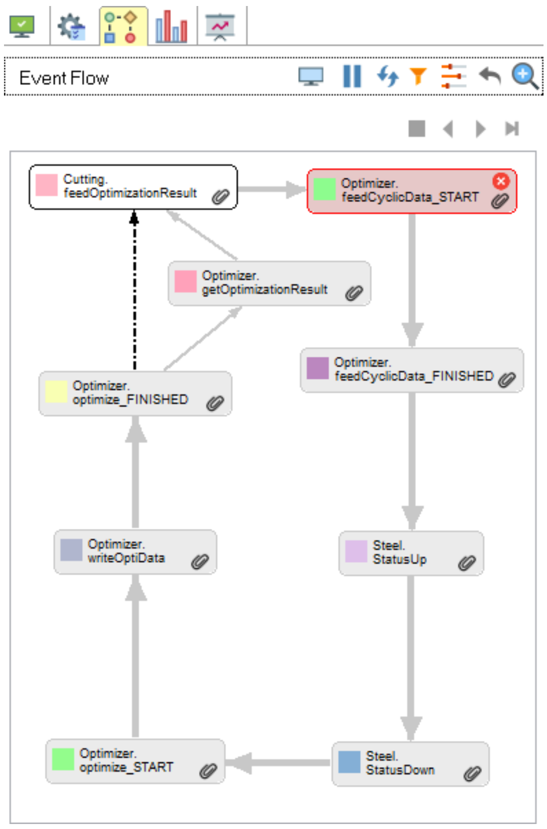
4.3 Event-related Visualizations



Time interval [13:32:44;13:32:54]

C1 - Order of Events: Wrong Sequence Event Type
Optimizer.optimize_FINISHED
Optimizer.writeOptiData
Optimizer.optimize_START
Steel.StatusDown

(a) Event Trending.


Event Flow
Cutting. feedOptimizationResult
Optimizer. feedCyclicData_START
Optimizer. getOptimizationResult
Optimizer. optimize_FINISHED
Optimizer. feedCyclicData_FINISHED
Optimizer. writeOptiData
Steel. StatusUp
Optimizer. optimize_START
Steel. StatusDown

(b) Event Flow.

Figure 4.4: Live Event/Violation Views.

Two *Event-related* views allow the user to observe the behavior of the monitored events and the constraint checks and violations they trigger beside the overall constraint status informations. These views are referring to the currently selected SoS component in the tree-based system overview.

The *Event Trending* view (cf. Fig. 4.4a) shows a stacked bar per predefined time 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. In Fig. 4.4a, for example, the third last stack is selected. This stack contains seven events and all of this events occurred once in the corresponding time interval. Also one violation of the `C1 - Order of Events` constraint occurred within this interval because the wrong event type occurred in the checked sequence of events. Whenever a violation occurs, it is added as a dot in the associated bar with respect to the time the violation was detected and the number of violations which occurred in this interval. The total amount is depicted by the values of the y-axis. For both plots, they indicate the number of event types and the number of violations occurred. Upon selecting a violation, the events involved in the violation are highlighted using darker variants of the same colors.

The *Event Flow* view (cf. Fig. 4.4b) 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 last occurred event type is displayed with a white background (i. e., `Cutting.feed-OptimizationResult`), while all others have a gray or red background (if they triggered a violation). The last connection (from `Optimizer.optimize_FINISHED` to `Cutting.feed-OptimizationResult`) is colored in black and dotted, all others are gray and solid. When a violation occurs, the trigger event is highlighted using a red border (i. e., `Cutting.feed-CyclicData_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, `Opti-mizer.feedCyclicData_START` as the trigger event of the `C1 - Order of Events` constraint led to a violation because the wrong event type occurred in the checked sequence of events. In particular, `Optimizer.getOptimizationResult` did not occur in this sequence loop.

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. 4.4b, the connections from `Optimizer.optimize_FINISHED` to `Cutting.feedOptimizationResult` or `Optimizer.getOptimizationResult` and from `Optimizer.getOptimizationResult` to `Cutting.feedOptimizationResult` did occur less then all other connections, but quite equal often to each other.

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 a sequence until the next violation appears. Also stepping back is possible. 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 or to only show event types which triggered violations.

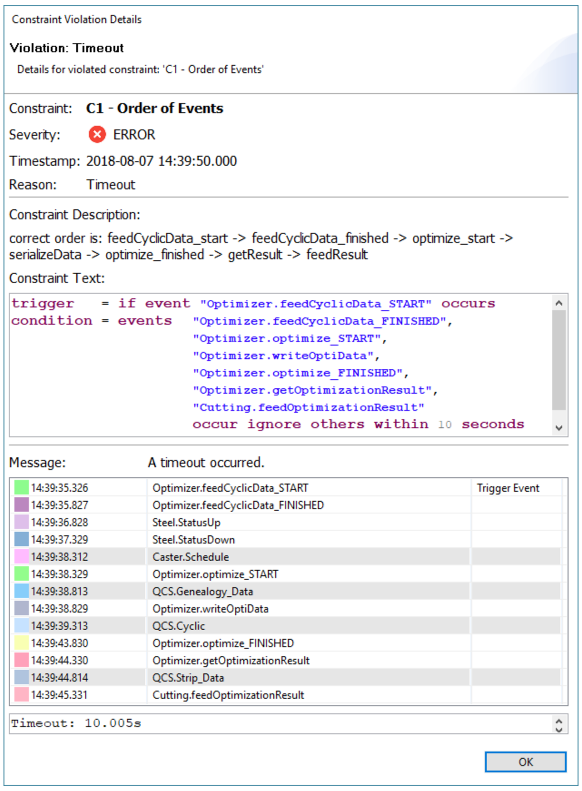
4.4 Diagnosis of Violations

Providing an explanation of detected violations or performance issues is essential for monitoring tools to help users to analyze violation details. Commercial tools like Dynatrace [31] provide detailed error analyses along with AI assistants explaining issues. However, research prototypes such as ReqMon [1] or Monere [12] only report an error, but do not provide detailed information for diagnosing these errors. Thus engineers would need to inspect related source code or other artifacts to investigate the root cause of such errors. For this purpose, the engineer would need interfaces to related engineering tools such IDEs, debugging tools, or issue trackers [8].

In ReMinds we provide an extensive violation details dialog. Figure 4.5a shows the dialog for a violation of the constraint `C1 - Order of Events` and Figure 4.5b for a violation of the constraint `C2 - ID Valid`. Informations about the violation severity, the timestamp, the reason (e.g. timeout, data missing) is given along with the constraint description and definition. The event sequence and events which led to the violation can also be viewed in a table view, which allows users to investigate event or data details by clicking on these entries, followed by a simple textual description of the violation.

4.5 Application to the Industrial Scenario

The drill-down scenario described in Section 3.2 is supported by interrelating the previously described views. The tree-based system status view (Fig. 4.2) provides an overview of the


Constraint Violation Details

Violation: Timeout

Details for violated constraint: 'C1 - Order of Events'

Constraint: **C1 - Order of Events**

Severity: ERROR

Timestamp: 2018-08-07 14:39:50.000

Reason: Timeout

Constraint Description:

correct order is: feedCyclicData_start -> feedCyclicData_finished -> optimize_start -> serializeData -> optimize_finished -> getResult -> feedResult

Constraint Text:

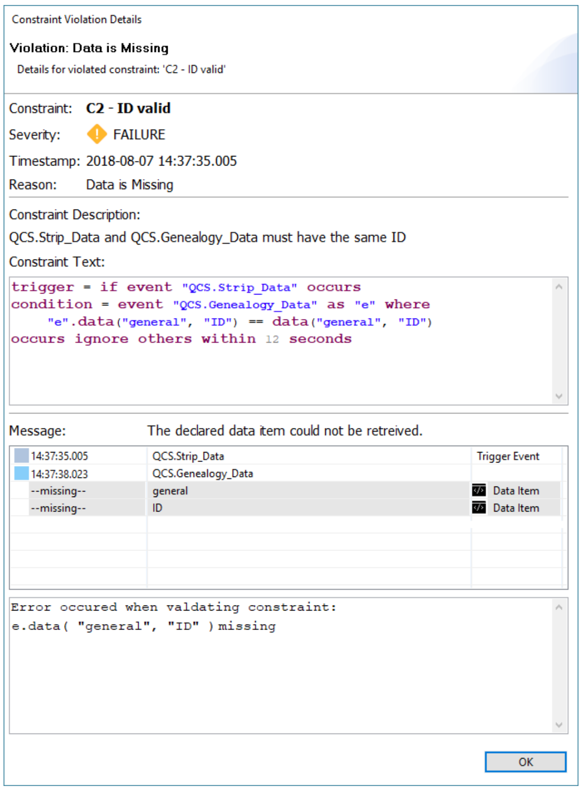
```
trigger   = if event "Optimizer.feedCyclicData_START" occurs
condition = events   "Optimizer.feedCyclicData_FINISHED",
                     "Optimizer.optimize_START",
                     "Optimizer.writeOptiData",
                     "Optimizer.optimize_FINISHED",
                     "Optimizer.getOptimizationResult",
                     "Cutting.feedOptimizationResult"
                     occur ignore others within 10 seconds
```

Message: A timeout occurred.

🟩 14:39:35.326	Optimizer.feedCyclicData_START	Trigger Event
🟪 14:39:35.827	Optimizer.feedCyclicData_FINISHED	
🟪 14:39:36.828	Steel.StatusUp	
🟦 14:39:37.329	Steel.StatusDown	
🟪 14:39:38.312	Caster.Schedule	
🟩 14:39:38.329	Optimizer.optimize_START	
🟦 14:39:38.813	QCS.Genealogy_Data	
🟪 14:39:38.829	Optimizer.writeOptiData	
🟦 14:39:39.313	QCS.Cyclic	
🟨 14:39:43.830	Optimizer.optimize_FINISHED	
🟥 14:39:44.330	Optimizer.getOptimizationResult	
🟦 14:39:44.814	QCS.Strip_Data	
🟥 14:39:45.331	Cutting.feedOptimizationResult	

```
Timeout: 10.005s
```

(a) `C1 - Order of Events`- Error.


Constraint Violation Details

Violation: Data is Missing

Details for violated constraint: 'C2 - ID valid'

Constraint: **C2 - ID valid**

Severity: FAILURE

Timestamp: 2018-08-07 14:37:35.005

Reason: Data is Missing

Constraint Description:

QCS.Strip_Data and QCS.Genealogy_Data must have the same ID

Constraint Text:

```
trigger = if event "QCS.Strip_Data" occurs
condition = event "QCS.Genealogy_Data" as "e" where
    "e".data("general", "ID") == data("general", "ID")
occurs ignore others within 12 seconds
```

Message: The declared data item could not be retreived.

🟦 14:37:35.005	QCS.Strip_Data	Trigger Event
🟦 14:37:38.023	QCS.Genealogy_Data	
--missing--	general	Data Item
--missing--	ID	Data Item

```
Error occured when valdating constraint:
e.data( "general", "ID" ) missing
```

(b) `C2 - ID Valid`- Failure.

Figure 4.5: Violation Dialogs.

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 *Violation Trending* view (the area chart of the same constraint; cf. Fig. 4.3a) to analyze trends regarding a specific violation (cf. scenario steps (2) and (3)). In this view it is easily possible to choose a specific time range, e. g., the period with the most violations which allows to restrict the area to be diagnosed further more. This time interval is then set to other views like the *Status/Interval* view (Fig. 4.3b), 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. 4.4a) from both the *Violation 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 Flow* view. The user can either replay the event-flow of the time range until the violation to be diagnosed occurs or walk through step by step (i. e., event by event). In our example, using the *Event Flow* view the user can check the typical event-flow of the `Optimizer` and then check the `C1 - Order of Events` violation and its relation to particular events using the violation details dialog.

5 Selected Implementation Details

The ReMinds Monitoring Center is developed using Eclipse RCP [32]. All developed visualization capabilities are plugin components added to the main client. The following chapter explains some selected principles of the Eclipse RCP platform and the extension point concept (Section 5.1), some used libraries (Section 5.2) and some details of the visualization plugin (Section 5.3).

5.1 Eclipse RCP

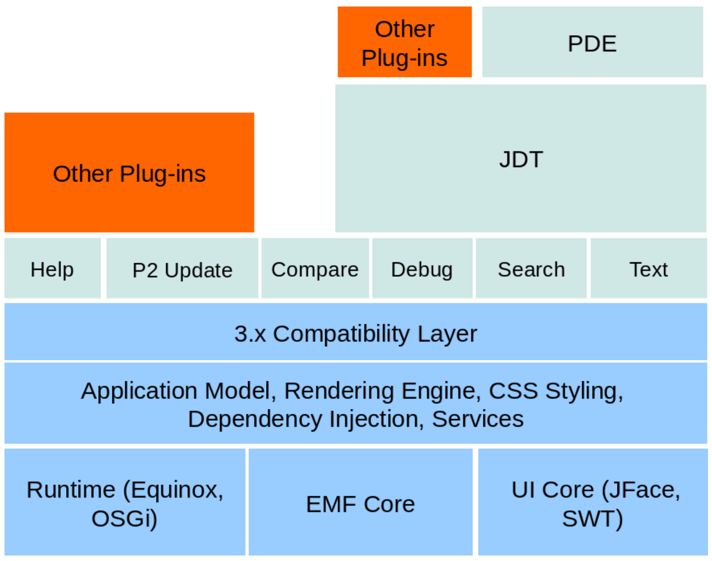


Figure 5.1: Eclipse RCP architecture [32].

The Eclipse Rich Client Platform (RCP) can be used used as a basis to create feature-rich stand-alone applications. Although the Eclipse programming model was simplified in version 4, we are still using version 3.8 for most of the client parts of ReMinds as our industrial partner is working with this standard as well. Applications developed with Eclipse RCP benefit from the existing user interface and the internal framework, and can reuse existing plug-ins and features. As the Eclipse plattform is the basis of one of the most used Java IDE, it is very stable. It provides some default user interface components, which can be extended, changed or removed and allows the development of component-based systems due to its modular approach.

High-level architecture. An Eclipse application consists of individual software components, called plugins. They typically are based on some basic components and other components provide additional functionality on top of them. For example, a new plug-in can create new menu entries or toolbar entries. The Eclipse IDE (cf. Figure 5.1) is for example a special Eclipse application with the focus on supporting software development. The Java development tools (JDT) provide the functionality to develop Java applications.

Extensions and extension points. Eclipse RCP provides a concept of adding extensions to extend functionality to a certain type (defined by a plug-in via an extension point) of API. These extensions can be contributions to one or more plug-ins. Listing 5.1 shows an example of the definition of an extension point and Listing 5.2 shows the implementation of this extension point. In particular, this EP is used to add tabs to a tab-folder. Listing 5.2 describes the implementation of the *Event Flow* tab.

```xml
<?xml version="1.0" encoding="utf-8"?>
<?eclipse version="3.2"?>
<plugin>
<extension-point
    id="scopetab"
    name="scopetab"
    schema="schema/scopetab.exsd" />
</plugin>
```

Listing 5.1: Example of an extension point.

```xml
<plugin>
    <extension point="at.jku.mevss.idetools.runtimeviewer.scopetab">
    <tab
        class="at.jku.mevss.idetools.runtimeviewer.visualization.scope.
            statechart.VisualizationScopeTab"
        icon="IMG_MM_MENU_STATECHART"
        order="1"
        tabTitle="Event Flow">
    </tab>
    </extension>
</plugin>
```

Listing 5.2: Example of an extension.

5.2 Used Libraries

Along with Eclipse RCP we use some libraries to ease the implementation of certain parts. In the scope of this thesis the Zest [33] toolkit is used for the tree-based system overview (Section 4.1) and the event flow view (Section 4.3), and JFreeChart [34] is used for the development of all different chart diagrams (Section 4).

Zest. This visualization toolkit provides a set of visualization components for Eclipse. The whole library is developed in SWT (Standard Widget Toolkit), an open source widget toolkit for Java designed to provide efficient access to the user-interface facilities of the operating systems on which it is implemented. All Zest views conform to the same standards and conventions as existing Eclipse views. An additional graph layout package can be used independently within existing Java applications to provide layout locations for a set of entities and relationships.

JFreeChart. This Java chart library can be used to display professional quality charts in Java applications. It consists of a consistent and well-documented API, supporting a wide range of chart types. Its flexible design is easy to extend, and targets both server-side and client-side applications. Furthermore it provides support for many output types, including Swing and JavaFX components, image files like PNG and JPEG, and vector graphics file formats like PDF, EPS and SVG.

5.3 Visualization Plugin

Our visualization capabilities are developed as a set of three plug-ins. The main plug-in manages all visualizations by providing some higher-level APIs, handlers for global functionalities. It also collects and provides the data needed for the visualization views. The two other plug-ins provide extensions to display the views in tabs. The Violation-related visualizations (Section 4.2) and the Event-related visualizations (Section 4.3) are implemented in each case one of the two plug-ins.

Figure 5.2 shows the lifeline of the visualization plug-in. Plugin can in this graphic be interpreted as the set of all three visualization plug-ins. When the plugin is started the visualization handler is created, followed by all available visualizations, which are then registered to the handler. The handler creates a thread to refresh all registered visualizations in a certain time interval. Some user actions, like setting and resetting the time interval are directly performed by the handler for every registered visualization.

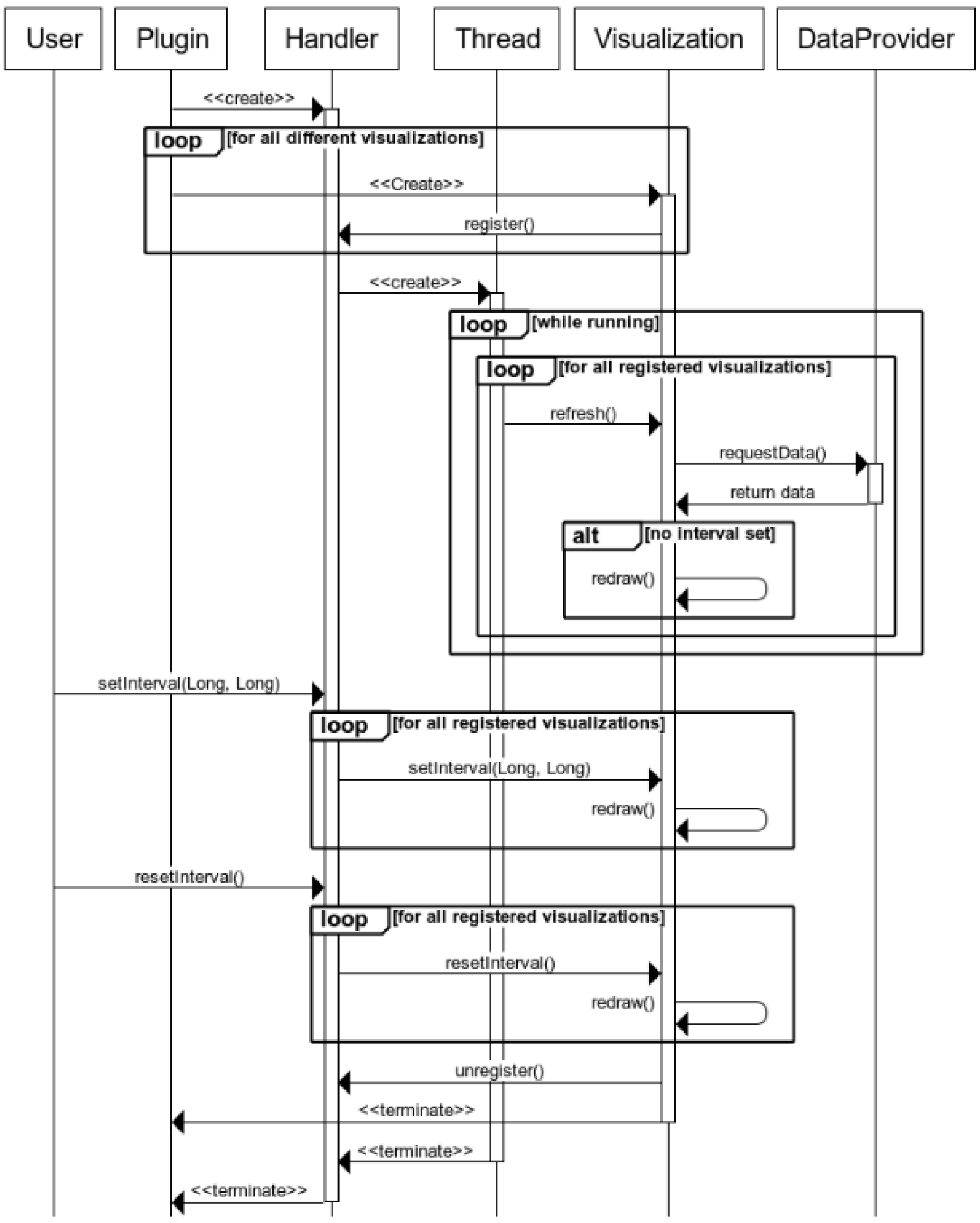


Figure 5.2: Sequence Diagram of the Visualization Plugin.

6 Evaluation

The following sections describes how we conducted the evaluation of this work. The chosen method for the evaluation of this thesis is explained in Section 6.1. In Section 6.2 we describe how we performed a walkthrough of our visualizations using the Cognitive Dimensions of Notations framework [19]. Section 6.3 explains the conducted usefulness study and the results of this study are summarized in Section 6.4.

6.1 Method

The field of Human-Computer Interaction distinguishes inspection-based approaches and test-based techniques to usability evaluation [18]: *Inspection-based* methods aim at improving the usability of an interface design by checking it against some standard such as Nielson's usability attributes [17] or the Cognitive Dimensions (CD) of notations framework [19]. *Test-based* approaches directly involve end users in the evaluation. Given the complexity and high interactivity needed to visualize results of monitoring large software systems, our research approach covers both inspection-based and test-based elements. Specifically, we investigate two research questions on the usefulness of requirements monitoring tools' visualization capabilities using the example of ReMinds: (1) Regarding *usability* we assessed ReMinds's visualization capabilities from the perspective of researchers as well as industrial software engineers, guided by the CD framework [19] and Nielsen's usability attributes [17]. (2) Regarding *utility* we observed users monitoring a real-world industrial system using ReMinds's visualization capabilities.

6.1.1 Identification of Visualization Capabilities

As described in Section 3.2, to identify key requirements monitoring activities that can benefit from visualization capabilities, we analyzed our industrial requirements monitoring scenario to define key user activities and visualization capabilities for end users. We also analyzed the support for these capabilities in existing frameworks and tools [4] [3].

Table 6.1: Cognitive Dimensions of Notation [19].

Viscosity	A viscous system needs many user actions to accomplish one goal.
Example	Setting a time interval for all views could need one interaction per view.
Visibility	Systems that bury information in encapsulations reduce visibility.
Example	REMINDS supports multiple options to complete certain tasks, e.g., a button and a context menu option.
Hidden dependencies	Systems with hidden dependencies have important links between entities, which are not visible.
Example	If setting a time interval in one view affects other view, this change must be made visible to the user.
Role-expressiveness	The purpose of an entity shall be readily inferred.
Example	All visualizations use the same color for the same violation severity.
Abstraction	Abstractions group a number of elements together which shall be treated as one entity.
Example	The terminology of key REMINDS abstractions are chosen to be as understandable as possible (scope, constraint).
Secondary notation	Extra information shall be provided in means other than formal syntax.
Example	The time interval can be set via drag and drop in the *Violation Trending* view and via a settings dialog in all views.
Diffuseness	Notations should not be long-winded, or occupy too much valuable “real-estate” within a display area.
Example	The time indications are in short format (hh:mm:ss.s), the whole date can be viewed via tooltip.
Consistency	Similar semantics are expressed in similar syntactic forms.
Example	An error is always highlighted using red color.
Error-proneness	This systems have notations, which invite mistakes.
Closeness of mapping	The notation shall be closely related to the result it is describing.
Hard mental operations	The demand on cognitive resources shell be as low as possible.
Provisionality	Provisionality allows users to make indicative selections before making definitive choices.
Progressive evaluation	It shall be easy to evaluate and obtain feedback on an incomplete solution.

6.1.2 Cognitive Assessment and Tool Improvement

We assessed ReMinds's visualization capabilities using an inspection-based walkthrough based on the CD framework [19] to reveal usability flaws that could bias a study with end users. The CD framework is well known in the HCI community and has been used successfully to assess software engineering tools in the past. For instance, Wassermann and Emmerich [12] describe an experiment assessing the Monere tool for diagnosing errors in service-based systems. Another example is the study by Cobleigh et al. [13] describing an experimental evaluation with end users specifying formal properties with PROPEL. Fittkau et al. [14] describe controlled experiments with 29 students to compare two software landscape visualizations and their support for program comprehension. In their 2017 article they also write "to the best of our knowledge, we are the first to conduct such a controlled experiment" [14].

Our own earlier study [7] on the usefulness of ReMinds for collecting, aggregating, and analyzing events and event data at runtime, to the best of our knowledge, so far was the only study involving industrial software engineers to assess the usefulness of a requirements monitoring tool.

The CD framework supports evaluating and characterizing capabilities of interactive artifacts, such as software tools, and offers a vocabulary for discussing trade-offs based on a set of basic user activities and cognitive dimensions. The notational dimensions are described in Table 6.1 with examples from the ReMinds visualization capabilities (if applicable).

We addressed any usability issues discovered in the walkthrough and then conducted a pilot study in our lab involving four researchers unfamiliar with the visualization capabilities of the tool as well as a study with industrial software engineers as described below.

6.1.3 User Study

We conducted a usefulness study involving researchers as well as industrial software engineers. We defined the study method based on findings from the CD assessment, following the guidelines for conducting empirical studies described by Runeson and Höst [35].

We selected the study system and the subjects working with the monitoring tool. We also defined the experimental setting, the monitoring tasks to be conducted by subjects, the data collection methods and sources, as well as the data analysis and reporting process (cf. Section 6.3).

Specifically, during the study, ReMinds was used by researchers from our lab and software engineers of our industry partner to monitor a plant automation software system (cf. Section 3). For instance, using the visualizations, the subjects had to investigate the current system status and diagnose constraint violations of different complexity. We asked each subject to “think aloud” [18], i.e., to describe what they were doing and to comment on any concerns. After the working session with the tool, we performed a utility interview [36] and handed out a usability questionnaire [17]. We describe results structured by our two research questions (cf. Section 6.4).

6.1.4 Derivation of Implications for Developers

We use the results recorded during the study to discuss lessons learned and findings for both researchers and practitioners (cf. Section 7). We also discuss the perceived opportunities and risks [36] of using visualization capabilities in practice.

6.2 Cognitive Dimensions Assessment

Here we report the results of our cognitive dimensions assessment, which are additionally summarized in Table 6.2.

6.2.1 Check System Status

The *System Status* is affected by cognitive dimension *abstraction*, as the visualized scope hierarchy is an abstraction of the real monitored system. The same holds true for the different types of violations shown in both the *System Status* and the *Constraint Status* views, e.g., different colors represent different types of constraint check results. While we were confident these abstractions would work for users, we decided to put a special focus on this in our user study.

The cognitive dimension *hidden dependencies* is also relevant: the *System Status* view displays the scope hierarchy and connected probes, but does not show events, data, and constraints, which are shown in separate views/tabs (on the right). Of particular relevancy to the system overview is cognitive dimension *diffuseness*, i.e., our scope visualization might waste too much of the available screen space.

6.2.2 Analyze Trends and Statistics

Selecting a time interval in the *Violation Trending* view affects multiple other views such as the *Event Trending* and *Event Flow* views (cf. Section 4.3). During the assessment we decided that if this is not properly communicated to the user, this could easily lead to problems (show stopper), particularly related with the cognitive dimension *hidden dependencies.* Before the study we thus implemented a status bar that highlights the set time interval. During the study, we wanted to assess whether users still need additional support. Similarly, the selection of the time period to focus on can be done by selecting it in the trending area chart as well as via a settings menu. This is related with cognitive dimensions *secondary notation* and *consistency* and we decided to investigate this in detail in the user study. The cognitive dimension *viscosity* also needs to be studied here, as changing the time is automatically applied to multiple views, *(repetition viscosity)* which might not be immediately obvious to the user.

6.2.3 Analyze Events and Timing Behavior

The *Event Trending* and *Event Flow* visualizations are affected by the cognitive dimension *viscosity*, i.e., these views might focus on a particular time period, which can be set in other views (e.g., the violation trending view). We expected this to be a potential problem for users to investigate in our user study. Also the *Event Trending* view highlights events related with a particular violation by changing colors to a darker tone. Our assessment showed that this negatively affects the cognitive dimension *consistency.* We thus decided to fix this by also highlighting the events using a border. Our CD assessment also revealed that the colors used by the *Event Trending* have another negative effect on consistency, i.e., they were too similar to the colors used to visualize the results of constraint checks and violations. We also fixed this show stopper by using a different set of colors. During our CD assessment we also noted to focus on *abstraction* and *diffuseness* during our user study, i.e., to find out the users' opinion on the abstractions used to represent monitored events in both the *Event Trending* and *Event Flow* views and their perception on whether the views present too much information.

6.2.4 Analyze Requirements Violation

For the violation details dialog our assessment revealed that cognitive dimension *hidden dependencies* is affected, as the dialog in its original version did not include the original constraint definition, but only the violation text. We thus fixed this potential show stopper before the user study by integrating the constraint definition. Furthermore, we decided to study whether users can correctly identify the cause of a violation with the information provided in the dialog. This regards the cognitive dimension *abstraction.* Also, *diffuseness* is relevant here as the textual explanation of violations might be too long-winded.

Table 6.2: Results of the initial cognitive dimensions assessment. For each visualization capability and corresponding tool views, multiple cognitive dimensions were either considered as to be studied in more detail during the user study (STUDY) or show stopper fixed (FIX) before the user study.

Visualization Capability	**Tool Views**	**Cognitive Dimensions** [19]	**Assess-ment**	**Usability (Section 6.4)**
Check System Status	System Overview and Pie charts	Abstraction	STUDY	+
		Hidden dependencies	STUDY	+
		Diffuseness	STUDY	+
Analyze Trends and Statistics	Trending, Status/Interval and Statistics	Hidden dependencies	FIX	+/−
		Secondary Notation	STUDY	−
		Viscosity	STUDY	+/−
		Consistency	STUDY	+/−
Analyze Events and Timing	Event Trending and Event Statechart	Viscosity	STUDY	+
		Consistency	FIX	−
		Abstraction	STUDY	+/−
		Diffuseness	STUDY	−
Analyze Requirement Violation	Violation Details Dialog	Hidden dependencies	FIX	+/−
		Abstraction	STUDY	+
		Diffuseness	STUDY	−
Cross-cutting aspects	All views	Visibility	STUDY	+
		Premature Commitment	FIX	+
		Hidden dependencies	STUDY	+/−
		Abstraction	STUDY	+
		Consistency	STUDY	+/−
		Diffuseness	STUDY	−

6.2.5 Cross-cutting Aspects

Several cognitive dimensions were found to be relevant for all views of ReMinds. For instance, the overall GUI layout (order and arrangement of views) regards the dimension *visibility* and was decided to be explored in detail with users. During the CD assessment we found that *premature commitment* could become a showstopper as the time period to focus on could not be set in all views. We thus fixed this by integrating a settings button in all views. *Hidden dependencies* also affects all views, e.g., each view focuses on particular elements of our model and hides elements or dependencies to other elements. We thus decided to investigate this in detail during the user study. Similarly, all views abstract from the monitored system and we want to investigate during the study whether our *abstractions* really work for users. Furthermore, we decided to investigate *consistency* regarding the use of colors and icons throughout the tool and *diffuseness* regarding unnecessary or unnecessarily complex views.

6.3 Usefulness Study

Based on the CD assessment and the improvement of ReMinds to address the potential showstoppers, our qualitative study investigates the usefulness (usability and utility) of ReMinds's visualizations as perceived by researchers and practitioners. We defined and structured our study based on the guidelines by Runeson and Höst [35]. We also followed the guidelines by Ko et al. [37] to select the tasks to be conducted by our subjects. Specifically, based on the initial assessment of the visualization capabilities we defined several tasks and iteratively refined our initial definition, i.e., we tested the task difficulty ourselves and later with four researchers from our lab, who had never used the visualizations before. Based on their feedback, we made some minor adaptations in the study method, e.g., we removed ambiguities by rephrasing text in the material we later handed out to the industrial engineers. To increase the participation of industrial users, our goal was to ensure that subjects could complete all tasks in less than one hour, while still covering all visualization capabilities. We also considered the results of our initial cognitive dimensions assessment and selected tasks that allow us to provide empirical data on the cognitive dimensions we wanted to investigate in more detail in the study.

6.3.1 Study System and Subjects

With the help of Primetals Technologies we selected five experienced people as subjects for the study in addition to the four researchers from our lab. We already had interviewed the subjects from Primetals Technologies in an earlier study [7]. This ensured that people already knew the context and purpose of monitoring software systems (cf. the inclusion condition defined by Ko et al. [37]). Most of our subjects (8/9) have a Master's degree in computer science, while one has a Bachelor's degree. Two additionally have a PhD degree. The average experience with automation software development of all our subjects was 6.4 years and the average time of the five Primetals Technologies subjects working at Primetals Technologies was 10.6 years. Only two subjects said they have already worked with the QCS before the study. Five subjects had seen ReMinds before and wrote some constraints to be monitored, but none of our subjects has used the visualizations before. This means that while we performed the study with experienced people, they were not familiar with the ReMinds visualizations nor the study system. The whole data collected of our subjects is summarized in Table 6.3.

We used the Quality Control System of Primetals Technologies already described in Section 3.1 as the study system. Probes instrumenting the QCS as well as constraints were developed before the study by an expert from Primetals Technologies. For the study, we selected two

Table 6.3: Some information on our nine subjects.

Average Education	Master CS
Highest Education Level (2 subjects)	PhD
Lowest Education Level (1 subject)	Bachelor CS
Average Experience in Automation Software Development (yrs)	6,44
Average Time Working at Primetals (yrs; only of those 5 working at Primetals)	10,6
Experience with QCS	1 little, 1 yes, 7 no
Experience with ReMinds	5 yes, but only writing constraints, 4 no

constraints: one checking the order and timing of multiple events in the QCS and one checking the order and timing of two events as well as data attached to the two events. We prepared the probes to randomly but frequently produce violations of the two constraints, which subjects then had to diagnose using our visualizations. the first constraint checks that after event `QCS-Schedule` occurred, events `QCS-SData` and `QCS-GData` occur within 12 seconds, while ignoring any other events that might occur in between. For this constraint, we modified the probes such that sometimes it takes longer than 12 seconds for all events to occur, thus leading to a violation of the required timing. For the second constraint – checking that after event `QCS-SData`, event `QCS-GData` occurs within 12 seconds and that the `C-ID` attached to event `QCS-SData` equals the `C3-ID` attached to event `QCS-GData` – we modified the probes to sometimes not provide the data at all (leading The constraint check to fail) and sometimes to send a wrong id for one of the events (leading to a data-check violation).

6.3.2 Study Process and Data Collection

We conducted the following process taking 56 minutes on average (min. 45 and max. 75) for each subject:

Preparatory steps. We started ReMinds 10 minutes before each session to ensure a considerable amount of events and data had already been monitored and visualizations were thus already filled with information, before the subject started.

Briefing. The moderator first explained the goals of the study to the subject and asked for consent to publish the findings [37]. The moderator also introduced ReMinds, i.e., explained the main views as well as the key concepts: scope, constraint, event. The moderator also explained the different possible results of a constraint check. Also, the moderator provided the subject with a printed document explaining the selected constraints, the events and data checked by these constraints, and a list of **tasks** to be completed (cf. capabilities):

1. *Check status.* Find out which system/component has violations, which constraints did violate, and how often they violated.
2. *Analyze violation trends.* Find out for each constraint with violations, in which time window most violations occurred and set this time period as the period to focus on.
3. *Analyze details about the violation.* Review the events/data and/or event timing behavior that led to each violation, i.e., review event trending and event flow.
4. *Diagnose and find the reason for the violation.* Possible reasons: data outside defined range, event missing, timeout occurred, event sequence occurred in unexpected/wrong order, or constraint check failed.

We asked each subject to "think aloud" [18], i.e., to describe what s/he was doing and to comment on any concerns. One scribe documented the think-aloud statements. Another scribe watched the subject and took additional notes on interesting observations beyond the think-aloud protocol. The moderator also took notes.

Data Collection. After the monitoring phase, the moderator performed semi-structured interviews on utility and usability with each subject, covering questions on the results of the cognitive dimensions assessment (cf. Section 6.2) and the utility questionnaire by Davis [36]. More specifically, regarding usability we asked questions such as "How did you like the capabilities to define time intervals to focus on and did you experience any problems with that"? or "What did you like/dislike about the tool"? Regarding utility, we asked questions such as "What opportunities and risks do you see for Primetals Technologies when using this tool in daily business?" We also used the interview to collect demographic information. After the interview, the moderator handed out a usability questionnaire, which was followed by a debriefing. The questionnaire is based on Nielsen's usability attributes [17] and covers the different views of ReMinds. We phrased the attributes as questions, e.g., "how easy was it to learn working with the visualizations"? and "were the visualizations pleasant to use"?.

The templates we used for writing think-aloud and observer protocols, the usability questionnaire, as well as the interview questions can be downloaded from https://tinyurl.com/REMINDSVisStudy

6.3.3 Data Analysis and Reporting

Three researchers carefully transcribed the think-aloud protocols and observer notes using an open coding technique [11] and related all statements to the visualization capabilities (cf. Section 3). A total of 122 think-aloud statements (14 per subject on average) and 180 observations were recorded by the scribes. We cannot reveal all of them here, but focus on the derived findings instead. We quote several think-aloud statements as well as observations throughout the next section. Table 6.4 depicts three examples of such think-aloud statements.

Table 6.4: Examples for think-aloud statements.

Task	**Think Aloud Protocol**
Details about the Violations	The *Event Trending* should show only full integers on y-axis.
Violation Trends and Interval	It should not be possible to zoom the x-axis.
Violation Trends and Interval	Zooming would be great for *Status/Interval* too.

The coding technique [11] allowed us to relate many think-aloud statements with the cognitive dimensions as discussed in Section 6.2. We discussed the interpretation of all think-aloud statements and observer notes as well as the answers given by study subjects in the interviews to derive implications on usability and utility (Section 6.4) and also general implications (Section 7). As we related interview questions regarding usability with activities and cognitive dimensions, we can discuss the subjects' answers in the light of the CD framework [19].

6.4 Results

In this Section we discuss the results of our study regarding the usability and utility of the ReMinds visualizations.

6.4.1 Usability

Table 6.5 depicts the results of the usability questionnaire following Nielsen's [17] attributes of usability we handed out to our nine subjects. In our discussion the subject keys S1-S4 refer to the researchers, while S5-S9 refer to the practitioners. We do not report results about memorability here as users were all novices in using the visualizations and were only available for a single session. Most subjects found the tool easy to learn. Most found the system overview and status visualizations as very easy to learn and the violation trends and event-based visualizations as easy to learn. Only three subjects (S1, S2, S6) gave negative ratings regarding learnability of these visualizations. Regarding efficiency, all subjects found that once learned, they could efficiently use our tool. While some errors occurred in the study, only one subject (S5) found that too many errors occurred. Unfortunately, an unexpected crash occurred during the session with this subject. Regarding subjective satisfaction, most subjects liked our tool, only two negative comments (by S1 and S5) were made on the violation

Table 6.5: Results of the usability questionnaires. We used a four-point scale (-2, -1, 1, 2), cf. the online material.

Nielsen's Attributes [17]	S1	S2	S3	S4	S5	S6	S7	S8	S9	Avg
Learnability *(very easy (2), easy (1), hard (-1), very hard (-2))*	1	1	2	2	1	1	2	1	1	1
· System Overview/Status	2	2	2	2	2	2	2	1	1	2
· Violation Trends/Interval	-1	1	2	2	1	-1	2	1	1	1
· Event Trends/Flow	1	-1	2	1	1	1	1	1	1	1
Efficiency *(very eff. (2), eff. (1), ineff. (-1), very ineff. (-2))*	2	2	2	2	1	1	1	2	2	2
Accuracy *(too many (-2), many (-1), some (1), no errors (2))*	2	1	2	2	-2	1	2	1	1	1
· System Overview/Status	2	2	1	2	1	2	2	1	2	2
· Violation Trends/Interval	1	-1	2	2	-2	1	2	1	1	1
· Event Trends/Flow	2	2	2	1	-2	1	1	1	1	1
Subject Satisfaction *(very good (2), good (1), bad (-1), very bad (-2))*	1	2	2	2	1	1	2	1	2	2
· System Overview/Status	2	2	2	2	1	2	1	1	2	2
· Violation Trends/Interval	1	2	2	2	-1	1	2	1	2	2
· Event Trends/Flow	-1	1	2	1	1	1	2	1	1	1

trends and on the event-related views.

The think-aloud statements and the observations as well as the answers to the interview questions allow us to discuss the usability of the ReMinds visualizations in detail. We first discuss some general comments (cf Table 6.6) and then analyze results for each of the four visualization capabilities in detail. We conclude by discussing cross-cutting results regarding multiple capabilities or visualizations (mostly resulting from the interview). Tables 6.7-6.15 summarize the results regarding usability for each capabilities and list examples of think-aloud statements and observations.

Table 6.6: Examples for general comments.

General comment S3:	"Progress bars should be shown when diagrams are built after start"
General comment S1:	"Notifications (e.g., similar to Microsoft Windows notifications) could be helpful for users to understand key changes/settings they make (and to explain consequences), e.g., selected time to focus on, applied search/filter, etc."

6.4.1.1 Check System Status

Our general observation is that the subjects had no problems to complete this task, i.e., the *System Status* and the *Constraint Status* views allowed them to easily spot violating constraints together with the frequency and severity of the violations. S5, however, commented that "for a high number of constraints, it could lead to problems that all constraints are

Table 6.7: *Check System Status* results.

System Overview, Constraint Status	
Abstraction	+
Hidden Dependencies	+
Diffuseness	+

shown all the time" and that the legend of the pie charts is missing some details, e.g., the number of consistent constraint checks. S2 wondered about the order of constraint-related views, i.e., whether the trending visualizations (one per constraint) are sorted by constraint name. In fact they are, and they can also be filtered dynamically using a text box just above the visualizations. Overall, we can, however, conclude that apart from minor issues, *abstraction*, *hidden dependencies*, and *diffuseness* are well supported in ReMinds when it comes to checking the system status.

Table 6.8: Examples for observations and think-aloud statements.

Observation on S7	Subject could easily find out what constraints violated how often and with what severity
General comment S1:	"Notifications (e.g., similar to Microsoft Windows notifications) could be helpful for users to understand key changes/settings they make (and to explain consequences), e.g., selected time to focus on, applied search/filter, etc."
Think aloud comment of S5	"It could lead to problems that all constraints are shown all the time for a high number of constraints"
Observation on S2	User wondered about order of constraints (why not sorted by name)

6.4.1.2 Analyze Trends and Statistics

Table 6.9: *Analyze Trends and Statistics* results.

Trending area charts; Status/Interval diagrams; Statistics view	
Hidden Dependencies	+/-
Secondary Notation	-
Viscosity	+/-
Consistency	+/-

Think-aloud comments and observations confirm that subjects found the trending visualizations, as S9 put it, "useful for easily finding and selecting the time period with most violations". Regarding *hidden dependencies* we received a comment by S5 that there should be a warning if the selected time interval does not contain any data. Also regarding hidden dependencies, but also *viscosity*, we observed that multiple subjects wondered if setting a time to focus on has effects on all views. S3 even wondered whether setting a time interval has effects on the shown scopes in the system overview (it does not) and should not as this view does not depend on the selected time).

Table 6.10: Examples for observations and think-aloud statements.

Observation on S7	Subject could easily select a time interval to focus on
Observation on S7	like many other subjects, this subject, when asked to select an "interesting" time period, immediately selected the biggest spike in the diagram
Observation on S8	Subject is confused by two types of diagrams in one (the status/interval violation diagram)
Think aloud comment by S5 and S7	"Trending, it should not be possible to zoom x-axis"
Observation on S5	Subject prefers to set an interval to focus on by using a settings dialog than the graphical (draw a rectangle with the mouse) option
Observation on S4	User wonders if setting a time to focus on has effects on all views
Think aloud comment by S3	"setting a time interval does not have effects on the shown scopes in the system overview . . . should it?"

As we knew this was a critical capability, to further investigate the usability of the time interval definition features, we also asked a question during the follow-up interview to get feedback on the feature for defining time intervals.

How did you like the capabilities to define time intervals to focus on and did you experience any problems with that (viscosity)?

Three subjects (S3, S7, S9) said they did not experience any problems and four subjects (S2, S3, S4, S8) said they liked the features. S2 and S8, however, also commented that it was not clearly visible to them at first what time they had selected. Three subjects (S4, S5, S6) found the feature "a bit buggy". Our observations reveal that selecting a time period in the trending area chart sometimes can indeed lead to bugs, if users move the mouse outside the area chart too fast. Two subjects (S1 and S8) also commented that views should be better linked, i.e., it should be made explicit what views are affected based on the time definition. We now added a specific icon to all views to indicate that. Overall, we think selecting a time interval works nice now, but only after several rounds of improvements and bugfixes.

In addition to the time-setting feature, S8 was confused by the two different types of diagrams (trending area chart vs. status/interval diagram) visualizing similar information in different ways (cf. cognitive dimension *secondary notation*). Our observations confirm the confusion. A key observation we made with multiple subjects was that the violation status/interval view was perceived as overwhelming, in particular as two visualizations are shown at the same time in one diagram (interval and trend). We thus conclude that ReMinds should be improved with regard to secondary notations.

Regarding *consistency*, while most subjects identified issues that regard all views (cf. below under cross-cutting results), the subjects S5 and S7 specifically commented on a consistency issue of the violation trends: they would have expected a feature to select a time period to focus on to work similar to other diagrams in other tools they knew, i.e., to not only allow selecting a time period (by drawing a rectangle from top-left to bottom-right) but to allow resetting the selected time interval by selecting a rectangle from bottom-right to top-left, similar to other tools they knew. S3 and S8 also expected that the selected time window could be "moved" as a whole to change both start and end time of the interval simultaneously back and forth in time, as they knew this behavior from other diagrams. In most cases, however, subjects found the trending views are similar to time(interval)-based views they already knew. We thus conclude consistency is acceptable but could be improved.

6.4.1.3 Analyze Events and Timing Behavior

Table 6.11: *Analyze Events and Timing Behavior* results.

Event Trending; Event State chart	
Viscosity	+
Diffuseness	-
Consistency	-
Abstraction	+/-

When observing subjects, we noticed that they liked the feature allowing them to step through the events in the state chart view until reaching the first violation (in this interval). However, the subjects S2 and S4 also commented that it would make sense to show the current time (when stepping through events) in the violation trending views to show the progress in stepping through the events. S5 also would have wished support to go to the first violation. This is related with cognitive dimension *viscosity*.

A key observation we made with multiple subjects was that the event trending view was not self-explanatory and overwhelming for users at first (cognitive dimension *diffuseness*). Also, and this regards cognitive dimension *consistency*, users found the colors of events confusing at first because of their similarity to the colors used to represent constraint check results. Also, regarding *abstraction*, it was not clear to three subjects (S3, S5 and S9) in the event-related views, "which events belong to a specific violation". In this context, three subjects (S2, S5, and S7) also suggested a filter based on concrete violations and severity (only show events related with a certain violation or violation type) to facilitate the highlighting of the violated nodes. S4 also commented that both, the current number of events focused on (through the selected time interval) as well as the total number of events (that could be displayed) should be shown in the event-related views, so that a user knows that "there are more events available".

Table 6.12: Examples for observations and think-aloud statements.

Think aloud comment by S5 and S9	"In the event interval it is not clear which events belong to a specific violation"
Think aloud comment by S5	"Violations list, filter for severity would be great"
Think aloud comment by S5	" live tracking should be deactivated when interval is set"
Think aloud comment by S5	State Chart: violated nodes should be highlighted better
Observation on S5	Subject prefers to set an interval to focus on by using a settings dialog than the graphical (draw a rectangle with the mouse) option
Observation on S3	Make it more obvious which events should be checked for a certain violation
Think aloud comment by S2	Violations should also be shown in live tracking [added after in-lab study]

6.4.1.4 Analyze Violation Details

Table 6.13: *Analyze Violation Details* results.

Violation details dialog	
Hidden dependencies	+/-
Abstraction	+
Consistency	-
Diffuseness	-

The violation details dialog overall was perceived as usable by all subjects and we received positive comments such as "a data mismatch is easy to discover due to highlighting of differences in value comparisons" (S5 and S7) and "it is clear that a time-out was the reason for this violation" (*abstraction*). One subject (S1) even commented that the information provided in the violation dialog would be enough for him/her to diagnose most violations, without also having to check the other visualizations (no *hidden dependencies* perceived by this subject).

Six subjects, however, also experienced some issues: both S6 and S8 found it hard to find the "missing events" in the displayed sequence of events, due to the long list of quite long event names. This regards *diffuseness*. Both S7 and S8 had difficulties parsing the error message due to too much text, again related with diffuseness. Subject S3 would have liked to also see the full definition of the constraint in the violation details dialog to better understand the violation (*hidden dependencies*). The subjects S2, S3, and S4 missed details in the description of the violations, which is not surprising, as these were subjects not familiar with the study system QCS.

Table 6.14: Examples for observations and think-aloud statements.

Think aloud comment by S5 and S7	"Data mismatch is easy to discover due to highlighting of differences in value comparison"
Observation on S7	Subject could easily diagnose both constraints and perfectly understood the violation reason
Observation on S8	Subject has a hard time finding the "missing events" in checked sequence of events in the violation dialog
Observation on S7 and S8	Subject has difficulties parsing the error message in the violation dialog (too much text)
Observation on S6	Violation Details Dialog, missing event should be indicated in sequence of events, subject would be much faster
Think aloud comment by S2	"Violation View: Time since last violation would be very interesting information (currently missing)"

6.4.1.5 Cross-Cutting Results

Table 6.15: *Cross-cutting* results.

Cross-Cutting Results	
Visibility	+
Premature commitment	+
Hidden dependencies	+/-
Abstraction	+
Consistency	+/-
Diffuseness	-

Regarding cognitive dimension *visibility*, we asked users during the interview: *Did you use multiple visualizations at the same time or did you rather use them one after the other?* The subjects' answers as well as observations showed an unclear picture on this issue: Five subjects (S2, S3, S4, S5, and S9) prefer parallel/side-by-side visualizations and want to be able to compare information presented in different views (e.g., violation-related trends and event-related timing behavior). Four subjects said they rather prefer to view one visualization at a time and switch back and forth. We thus developed the possibility to detach views from the main GUI (as dialogs), this allows to view all at the same time and also the possibility to close different types of views, which allows to view one visualization at a time.

Regarding visibility we also asked *Did you find the visualizations you needed [to complete your tasks] easily?* Six subjects (S2-S6 and S9) commented they could find everything easily and S1 even commented that s/he "found the icons/buttons [representing views and allowing to navigate between views] useful". Subject S8 would have preferred a different grouping of views, i.e., all views affected by time interval changes should be in one, visually separated group. S7 commented "this is a complex tool and some learning will always be required for such tools, but after minimal learning, I think one can easily use this tool." Overall, we thus concluded that the dimension visibility is well supported.

Regarding the cognitive dimension *premature commitment*, we asked users how they like the feature to change the time interval.

Did you modify any settings? If yes, how did you like this feature? If no, why not?.

All subjects defined one or multiple time intervals to focus on (not surprisingly as this was required as part of task 2). While all subjects liked the feature, four subjects had issues as also described above under viscosity with the effects of selecting a time interval. After they understood these effects, our observation was that all subjects liked this feature, especially as the time interval can be reset at any time.

Regarding *hidden dependencies*, we also asked subjects if they were aware of the currently selected time period. Three subjects (S1, S5, S7) commented that this was not clear to them and that it should be made explicit, e.g., by showing better highlighting which views are filtered (which we now do with an explicit icon). Three subjects (S3, S8, S9) said they were aware, but our observations reveal they were also briefly uncertain before they found the information in the status bar. Two subjects (S2 and S6) commented that the status bar showing the selected time interval should be visually closer to the filtered views. We changed the layout of the tool to achieve that. We conclude that, after our improvements, hidden dependencies should not be a big issue anymore.

Regarding *abstraction* we asked about problems with the used terminology, i.e., naming of displayed content and/or icons used?. Subject S9 commented that s/he found legends of diagrams confusing due to the use of abbreviations (e.g., E for Error, W for Warning). S8 commented that some buttons have icons, while others have not, which s/he found confusing. S7 on the other hand explicitly said s/he liked these icons. Six subjects (S1, S2, S4-S7) said they found the used abstractions were clear and natural. S5 even commented "the terminology is natural, especially for someone with computer science background". We thus rate abstraction as very positive for ReMinds.

When asking users if they experienced any inconsistencies three subjects (S6, S7, S9) said that they found the visualization features consistent. S8 commented that there are some rounding errors in diagrams and that especially the axes should be labeled consistently. The subjects S1, S3 and S4 all found the similarity of event colors in the event trending vs. the violation colors in all views confusing (as we had expected, cf. CD assessment). S3 also commented on different font sizes used in different views. S1 and S2 found that "clicks do not always lead to the same result in different views" (e.g., left click selects/highlights, double-click opens dialog, right-click displays context menu with other options)". We now fixed these issues. We conclude that consistency was good, but not ideal.

Regarding *diffuseness*, we asked users if they perceived any items or views as unnecessary or unnecessarily complex. S1, S8 and S9 found the event trending view too colorful and presenting too much information (events in multiple intervals, all related violations and data elements). S1 found the amount of information presented in all views hard to understand at first. S4 generally did not like the interval-based views as she found the other views sufficient to complete his/her tasks. Five subjects (S2, S3, S5, S6, S7) said they did not find the views unnecessarily complex, but two of them also suggested simplifications do better deal with many/events violations: S3 suggested to provide more filtering capabilities, e.g., filtering all views based on severity or event type, and S6 suggested to remove unnecessary information. The latter needs to be done very carefully, to not also negatively affect other cognitive dimensions such as abstraction and hidden dependencies.

At the end of the interview, we also asked subjects what they **liked** or **disliked** about the tool. In their responses subjects liked the overall intuitive GUI, the system overview, the violation trending, the event state chart, the feature allowing to step through events until reaching violations, and the violation dialog. S6 commented that "complexity results from the need to understand quite complex constraints/system behavior [...] and ReMinds helps a lot with that". Four subjects (S1, S5, S6, S7) did not like the interval-based visualizations, especially the event trending view, as they found them hard to understand at first and thought the other visualizations would be sufficient to complete the tasks. These visualizations indeed only make real sense when a large number of events and violations needs to be diagnosed over a longer period of time. Other than that, subjects commented that there were some bugs and inconsistencies but nothing really bad.

Overall 100 minor **usability issues** (mostly improvements but also some bugs) resulted from comments by participants of the study. For example, several subjects commented on missing tooltips, legends and axis labels in diagrams. Addressing such issues contributes a lot to improving a tool's usability.

6.4.2 Utility

Regarding the utility of ReMinds's visualizations, we first asked subjects whether they liked the workflow of using visualizations we envisioned, i.e., check status first, then trends and statistics, then event details and timing, and finally violation details. Specifically, we asked them whether they found that order natural in the user study or whether they would prefer any other order. Six subjects found the order natural, e.g., S6 commented: "Yes, I think starting with the constraints (their violations), then selecting the time interval most interesting, and then looking at event details is a very natural way for finding out how the monitored system behaves". However, three of these six subjects and also three other subjects commented that the arrangement of visualizations in the GUI did not naturally support this envisioned order or at least confused them at first. Specifically, while the user first is presented with the system overview on the left and the status and trending visualizations (regarding all systems) on the right, selecting a particular system with a violation opens a set of system-related views (with event details) directly to the right of the system overview (and thus left of the status and trending views). Re-arranging the views in the GUI (cf. Fig. 4.1; status regarding all systems left and event details regarding only the currently selected system right) – helps users to follow the envisioned order, while they still are free to switch to any visualization at any time. As commented by S9: "It's good that the user has the freedom to choose, but that a certain order can be followed too."

We also asked users whether they missed a specific tool feature and whether they had any other comments. Six subjects provided us with feedback. For instance, S7 commented that the violation trends and the event state chart should be more closely related, i.e., when stepping through events in the event state chart, the violation trending area diagram should display the current position in time in parallel. In general, better linking multiple visualization is a very important improvement we learned from the study. S5 commented that s/he would want to (visually) compare multiple data items attached to (multiple) events and would expect more capabilities to filter visualizations based on, e.g., violation types and severity levels. S3 found that it would make sense to highlight affected systems (in the system overview), when focusing on particular constraints/their violations. S2 suggested the event state chart to explicitly show constraint violations as elements, instead of (or in addition to) just highlighting events that caused violations. Other comments were rather minor and, e.g., regarded the need to display the "time since last violation" somewhere.

To further assess utility, we asked the subjects from Primetals Technologies what are the opportunities and risks of using ReMinds in daily business.

Opportunities

- Could be very useful for commissioning and for acceptance tests (3 subjects).
- Faster identification of issues than without the tool, i.e., visualizations point the user to the right time/location.
- We could use the tool to analyze view model behavior in our human-machine interface.
- For large, distributed, systems of systems the tool provides a very good overview of what is happening/has happened.

Risks

- Tool not used due to too much maintenance or configuration effort (4 subjects).
- Engineers don't see "their" concrete benefit, e.g., if not finding violations quicker with than without the tool.
- If there are too many violations (visualized), at some point people start ignoring them/no longer diagnose them.
- The GUI might be overwhelming (too much information visualized) for some engineers.

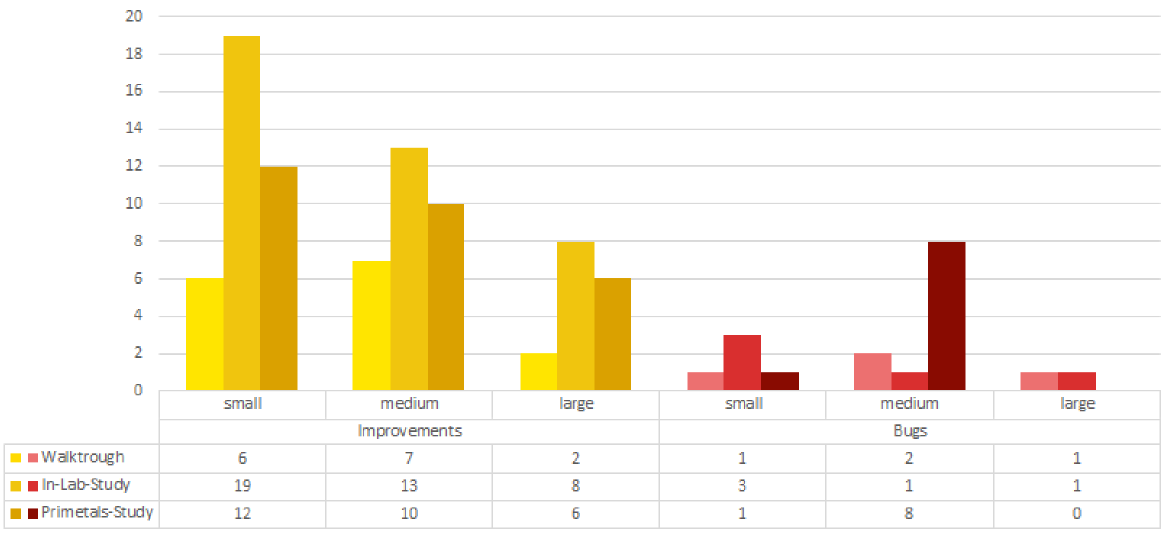
6.4.3 Summary and Threats to Validity

Figure 6.1 shows improvements and bugs found during the three phases of the study (walk-through, In-Lab-Study and Primetals-Study). Overall we discovered 37 minor, 30 medium and 16 major improvements and 5 minor bugs, 11 medium and 2 major. We define an improvement as a change to the user interface that changes the way users can interact with it in some way and a bug as a change to the user interface that corrects an error, flaw, failure or fault that causes the interface to produce an incorrect or unexpected result. Table 6.16 shows examples for small and major bugs and improvements.

We conclude that the usability and utility of the ReMinds visualizations are very good. All subjects liked the tool and could complete their tasks in a rather short time, especially

Table 6.16: Examples for major/small bugs/improvements.

Type	**Example**
Bug, small	The dialog to display event data has two scrollbars.
Bug, major	Using drag and drop to select a time interval lead to an error.
Improvement, small	List of constraints shall be sorted by name.
Improvement, major	It should be possible to switch between percent and absolute values in the *Violation Trending* view.



	Improvements			Bugs		
	small	medium	large	small	medium	large
■ ■ Walktrough	6	7	2	1	2	1
■ ■ In-Lab-Study	19	13	8	3	1	1
■ ■ Primetals-Study	12	10	6	1	8	0

Figure 6.1: Improvements and Bugs found during the Study.

considering that they did not use the visualizations before. While one could claim this is due to the simplicity of tasks, the constraints we selected for subjects to diagnose are real constraints from a real automation system currently running in a plant in China. Subjects' comments have and will help us to further improve our tool. Particularly the opportunities and risks will help us to focus further developments and motivate further research into the automation of certain user tasks and the simplification of the overall GUI.

A threat to construct validity is the potential bias caused by the system we selected for the study. We selected the QCS as it allows working with all visualization capabilities described and is easy to explain. Also, all industry subjects have at least heard of it and know what it does. A further threat is that our study assumes the QCS probes to be correct and providing the correct events, a fact that has been confirmed by industry experts, but is hard to verify.

There are also threats to internal validity, i.e., the results might have been influenced by our treatment. Subjects were selected together with our industry partner and based on their availability. However, this was done based on criteria such as domain experience and their participation in earlier studies, which provided the context for this work. Also, the number of subjects may seem relatively small. However, subjects have a lot of experience in the development and testing of (automation) software systems (average experience of over six years) and are well qualified for the study. While additional experienced industry participants may have added further value and insights to the study, we feel that the results we discovered nicely converged and there were many common insights.

Regarding conclusion validity, there is a threat that the results are based on qualitative data [11]. However, given that an aim of the study was to study the behavior and opinions of users of a tool, qualitative research methods are well suited. The analysis of the collected data still depends on our interpretation. The work was performed by three researchers and all results were also checked by another senior researcher.

With respect to external validity, we selected a system representative for the automation software domain and asked experienced industry people as well as researchers to be part of the study. Although the derived implications depend on our experiences of using and implementing ReMinds, the capabilities are common in other monitoring tools, as discussed in Section 3.2. The mapping to the CD framework also relates our results to HCI knowledge.

7 Discussion

Besides important, yet not very surprising findings such as that the consistent use of colors is essential and too similar colors should not be used for different views/things, we discuss implications for researchers and practitioners designing software monitoring tools.

Support both guidance and freedom in navigation. While some users preferred step-by-step guidance through the different views, others wanted to choose freely when switching between different visualizations. We conclude that it is essential, especially when offering multiple visualizations, to offer both a certain order guiding users, but still allow them to freely choose among available views. Detailed guidance can support a “drill-down” approach to diagnose violations by starting with a high-level system overview, then focusing on one system in a certain time interval, to finally analyze event and violation details. At the same time, depending on the monitoring task, it may be preferable to freely choose among the available views. This finding is rooted in the cognitive dimensions premature commitment and hidden dependencies and confirms results presented in an earlier study regarding “branching and navigation” vs. “freedom in navigation” and is related with.

Allow combinations of multiple interrelated views. Interestingly, some users preferred to see and compare multiple visualization at the same time, while others only selected a single visualization for the task. This calls for support to detach and/or close views. However, when using multiple visualizations concurrently it becomes particularly important to keep them consistent with each other. For instance, users liked the feature allowing them to step through the events to understand system behavior and to diagnose violations in a certain period. However, due to the potentially high number of events, this feature relies on the feature to set a certain time interval. When multiple views are involved it is important to consider the cognitive dimension viscosity, i.e., to make the effects of a setting on different views explicit, e.g., highlighting which views are filtered based on time.

Several subjects in our study commented that views should be better linked, e.g., the current position when stepping though events should in parallel be shown in the violation trending visualization (with both views being based on the same time scale). Especially when supporting multiple views in parallel, *explicitly linking multiple visualizations is essential to guide users*, especially when diagnosing a monitored system by stepping back and forth in time. This mainly regards cognitive dimension hidden dependencies.

This is also related with cognitive dimension diffuseness. Talking of this feature – setting a time interval – all subjects liked this feature, especially as the selected time interval can be reset at any time.

Ease finding unusual behavior. We found that users typically tried to find and select violation peaks to reveal unusual behavior, e.g., based on the trending visualizations. A low-hanging fruit is to support users by detecting such peaks automatically and point them to the most relevant time periods.

Keep visualizations comprehensible. A key observation we also made with multiple subjects – independent of their experience – was that visualizations can easily become overwhelming (cf. cognitive dimensions diffuseness and abstraction). Particularly, the violation status/interval and the event trending views – especially the fact that there are 2 visualizations shown at the same time in one diagram (two y-axes, interval and trend) – were hard to grasp for several users. This suggests an incremental visualization approach showing first only the visualizations essential for understand system behavior and adding more complex visualizations only if requested. This, however, needs to be done carefully, to avoid possible negative effects on the dimensions abstraction and hidden dependencies. Complex visualizations can be made more comprehensible by providing searching and filtering capabilities.

Provide explanations of behavior. Analyzing system behavior can be very challenging, depending on the complexity of the checked constraints and the detected violations. Visualizations support users in understanding system behavior. However, the success of a monitoring tool depends also on providing proper explanations of each detected violation in a textual form (cf. our violation details dialog). Such explanations must be brief and clearly describe what happened. Features such as highlighting of differences and explicit listing of missing information (events) can really help users, as the subjects in our study confirmed. This is again related with both diffuseness and abstraction.

Automate initial set-up and maintenance. Regarding the utility of a tool like ReMinds, it is essential to keep the effort for the initial set-up as well as frequently required maintenance as low as possible. This can be achieved, for instance, by automating the definition and maintenance of constraints, e.g., using mining approaches such as specification mining [38] or process mining [39]. Especially for large, distributed, systems of systems a tool like ReMinds can provide a very good overview of what is currently happening/has happened.

8 Conclusion

In the first part of this thesis 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.

As a second part we evaluated the usefulness of our visualizations in detail, e. g., by performing a user study with our industry partner and some researchers. Specifically, we presented the results of assessing the usefulness of visualization capabilities for requirements monitoring as implemented in our own tool ReMinds using the cognitive dimensions of notations framework and the user study involving researchers and industrial practitioners. We used the results of the assessment and the user study to discuss both specific implications for ReMinds as well as general implications for researchers and practitioners working in the area of software monitoring. Explaining monitored system behavior is hard, but (the right) visualizations can really help. In our future work we plan to further improve our tool, especially lower the effort for initial set-up and maintenance, and conduct experiments with other systems and users.

List of Figures

List of Tables

Bibliography

[1] W. Robinson, “A requirements monitoring framework for enterprise systems”, *Requirements Engineering*, vol. 11, no. 1, pp. 17–41, 2006.

[2] N. Delgado, A. Q. Gates, and S. Roach, “A taxonomy and catalog of runtime software-fault monitoring tools”, *IEEE Transactions on software Engineering*, vol. 30, no. 12, pp. 859–872, 2004.

[3] R. Rabiser, S. Guinea, M. Vierhauser, L. Baresi, and P. Grünbacher, “A Comparison Framework for Runtime Monitoring Approaches”, *Journal of Systems and Software*, vol. 125, pp. 309–321, 2017.

[4] M. Vierhauser, R. Rabiser, and Grünbacher, “Requirements Monitoring Frameworks: A Systematic Review”, *Information and Software Technology*, vol. 80, pp. 89–109, 2016.

[5] M. Vierhauser, R. Rabiser, P. Grünbacher, K. Seyerlehner, S. Wallner, and H. Zeisel, “ReMinds: A Flexible Runtime Monitoring Framework for Systems of Systems”, *Journal of Systems and Software*, vol. 112, pp. 123–136, 2016.

[6] M. Vierhauser, R. Rabiser, Grünbacher, and A. Egyed, “Developing a DSL-Based Approach for Event-Based Monitoring of Systems of Systems: Experiences and Lessons Learned”, in *30th IEEE/ACM Int’l Conf. on Automated Software Engineering*, IEEE, 2015, pp. 715–725.

[7] R. Rabiser, M. Vierhauser, and P. Grünbacher, “Assessing the Usefulness of a Requirements Monitoring Tool: A Study Involving Industrial Software Engineers”, in *38th Int’l Conf. on Software Engineering, ICSE Companion*, ACM, 2016, pp. 122–131.

[8] M. Vierhauser, R. Rabiser, and J. Cleland-Huang, “From Requirements Monitoring to Diagnosis Support in System of Systems”, in *23rd Int’l Working Conf. on Requirements Engineering: Foundation for Software Quality*, Springer, 2017, pp. 181–187.

[9] L. M. Kritzinger, T. Krismayer, M. Vierhauser, R. Rabiser, and P. Grünbacher, “Visualization Support for Requirements Monitoring in Systems of Systems, BookTitle = 32th IEEE/ACM Int’l Conf. on Automated Software Engineering”, IEEE, 2017, pp. 889–894.

[10] M. Vierhauser, R. Rabiser, Grünbacher, and J. Thanhofer-Pilisch, "The REMINDS Tool Suite for Runtime Monitoring of Systems of Systems", in *30th IEEE/ACM Int'l Conf. on Automated Software Engineering*, IEEE, 2015, pp. 777–781.

[11] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering", *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[12] B. Wassermann and W. Emmerich, "Monere: Monitoring of service compositions for failure diagnosis", in *Int'l Conf. on Service-Oriented Computing*, Springer, 2011, pp. 344–358.

[13] R. L. Cobleigh, G. S. Avrunin, and L. A. Clarke, "User guidance for creating precise and accessible property specifications", in *14th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering*, ACM, 2006, pp. 208–218.

[14] F. Fittkau, A. Krause, and W. Hasselbring, "Software landscape and application visualization for system comprehension with ExplorViz", *Information and Software Technology*, vol. 87, pp. 259–277, 2017.

[15] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, "Usability Meanings and Interpretations in ISO Standards", *Software Quality Journal*, vol. 11, no. 4, pp. 325–338, 2003.

[16] "ISO/IEC 9126. Software engineering – Product quality", Tech. Rep., 2001.

[17] J. Nielsen, *Usability Engineering.* Morgan Kaufmann, 1994.

[18] A. Holzinger, "Usability engineering methods for software developers", *Communications of the ACM*, vol. 48, no. 1, pp. 71–74, 2005.

[19] A. Blackwell and T. Green, "Notational Systems – the Cognitive Dimensions of Notations framework", in *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*, J. Carroll, Ed., San Francisco: Morgan Kaufmann, 2003, pp. 103–134.

[20] M. Vierhauser, R. Rabiser, P. Grünbacher, and B. Aumayr, "A Requirements Monitoring Model for Systems of Systems", in *23rd IEEE Int'l Requirements Engineering Conf.*, IEEE, 2015, pp. 96–105.

[21] R. Rabiser, M. Vierhauser, and P. Grünbacher, "Variability Management for a Runtime Monitoring Infrastructure", in *9th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2015)*, Hildesheim, Germany: ACM, 2015, pp. 35–42.

[22] *AspectJ*, `https://www.eclipse.org/aspectj`, Accessed: 2018-07-31.

[23] R. Rabiser, J. Thanhofer-Pilisch, M. Vierhauser, P. Grünbacher, and A. Egyed, "Developing and Evolving a DSL-Based Approach for Runtime Monitoring of Systems of Systems", *Automated Software Engineering*, 2018. DOI: 10.1007/s10515-018-0241-x.

[24] *XText*, http://www.eclipse.org/Xtext, Accessed: 2018-07-31.

[25] L. Baresi and S. Guinea, "Event-based multi-level service monitoring", in *IEEE 20th Int'l Conf. on Web Services*, IEEE, 2013, pp. 83–90.

[26] J. Simmonds, S. Ben-David, and M. Chechik, "Monitoring and recovery for web service applications", *Computing*, vol. 95, no. 3, pp. 223–267, 2013.

[27] A. Mos, C. Pedrinaci, G. A. Rey, J. M. Gomez, D. Liu, G. Vaudaux-Ruth, and S. Quaireau, "Multi-level monitoring and analysis of web-scale service based applications", in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, Springer, 2010, pp. 269–282.

[28] A. Dragomir and H. Lichter, "Run-time monitoring and real-time visualization of software architectures", in *20th Asia-Pacific Software Engineering Conf.*, IEEE, 2013, pp. 396–403.

[29] H. Muccini, A. Polini, F. Ricci, and A. Bertolino, "Monitoring architectural properties in dynamic component-based systems", in *Int'l Symp. on Component-Based Software Engineering*, Springer, 2007, pp. 124–139.

[30] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis", in *3rd joint ACM/SPEC Int'l Conf. on Performance Engineering*, ACM, 2012, pp. 247–248.

[31] *Dynatrace*, http://www.dynatrace.com, Accessed: 2018-07-31.

[32] *Eclipse RCP (Rich Client Plattform)*, http://www.vogella.com/tutorials/EclipseRCP/article.html, Accessed: 2018-08-20.

[33] *Eclipse Zest*, http://www.vogella.com/tutorials/EclipseZest/article.html, Accessed: 2018-08-20.

[34] *JFreeChart*, http://www.vogella.com/tutorials/JFreeChart/article.html, Accessed: 2018-08-20.

[35] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering", *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[36] F. D. Davis, "Perceived usefulness, perceived ease of use and user acceptance of information technology", *MIS Quarterly*, vol. 13, pp. 319–340, 1989.

[37] A. J. Ko, T. D. LaToza, and M. M. Burnett, “A practical guide to controlled experiments of software engineering tools with human participants”, *Empirical Software Engineering*, vol. 20, no. 1, pp. 110–141, 2015.

[38] D. Lo, S.-C. Khoo, J. Han, and C. Liu, *Mining Software Specifications: Methodologies and Applications*. CRC Press, 2011.

[39] A. R. C. Maita, L. C. Martins, C. R. L. Paz, L. Rafferty, P. C. K. Hung, S. M. Peres, and M. Fantinato, “A systematic mapping study of process mining”, *Enterprise Information Systems*, pp. 1–45, 2017. [Online]. Available: `https://doi.org/10.1080/17517575.2017.1402371`.

Lisa Maria Kritzinger

Curriculum Vitae

PERSONAL DETAILS

Birth	April 13, 1992
Address	Rilkestraße 20/17/118, 4020 Linz
Phone	0650 883 12 55
Mail	`kritzinger@gmx.net`

EDUCATION

Master of Science 2016-present
Johannes Kepler University Linz, Austria
Software Engineering

Bachelor of Science 2012-2016
Johannes Kepler University Linz, Austria

High School Diploma (German: Reife- und Diplomprüfung) 2006-2012
Bundeshandelsakademie Vöcklabruck
Informationstechnologie und -management

WORK EXPERIENCE

Student Researcher 2016-present
Johannes Kepler University Linz, Austria
Christian Doppler Laboratory
Monitoring and Evolution of Very-Large-Scale Software Systems

Software Tester 2015-2016
fmade GmbH

Intern July 2011
Gemeinde Neukirchen an der Vöckla

Trainee 2009-2010
Ing. Philipp GmbH & Co.KG
Office clerk

Intern August 2009
Gemeinde Neukirchen an der Vöckla

SKILLS

Languages	German (mother tongue)
	English (fluent)
Computer Skills	Programming Skills: Java, C#, C++, C
	Knowledge of Operating System: Windows, Linux
Driving License	Category B

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, am 5. September 2018

Lisa Maria Kritzinger, BSc