Towards a Dependability Control Center for Large Software Landscapes

Florian Fittkau
Software Engineering Group
Kiel University, Kiel, Germany
Email: ffi@informatik.uni-kiel.de
Phone / Fax: +49 431 880 4467 / 7617

André van Hoorn
Reliable Software Systems Group
University of Stuttgart, Stuttgart, Germany
Email: van.hoorn@informatik.uni-stuttgart.de
Phone / Fax: +49 711 685 88 252 / 472

Wilhelm Hasselbring
Software Engineering Group
Kiel University, Kiel, Germany
Email: wha@informatik.uni-kiel.de
Phone / Fax: +49 431 880 3734 / 7617

***Abstract*—Manual management of dependability while operating large software systems — including failure detection, diagnosis, repair, and prevention activities — is time-consuming and error-prone. Various automatic approaches supporting these activities have been proposed, e.g., to detect and diagnose performance degradation effects caused by software aging and to execute reactive or proactive rejuvenation actions. However, users often mistrust fully-automatic dependability management approaches due to a lack of control over the change actions conducted to the business-critical software landscape. Building trust for automatic systems is challenging.**

In this paper, we present our envisioned control center for a semi-automatic management of large software landscapes, featured by a graphical user interface including interactive system visualizations. The control center will provide a reusable platform for integrating techniques for dependability management, including monitoring, and analyzing a system's dependability during production as well as for planning and executing reactive or proactive change actions to the software landscape.

***Keywords*—Control center, software visualization, semi-automatic approach, dependability management, MAPE-K.**

I. Introduction

To name just one example for a threat to system dependability, software aging is a well-known problem when operating software systems. Example causes for software aging include memory leaking or unterminated threads. Software rejuvenation approaches are aimed at removing those effects before failures occur. [1]. However, it can be tedious to manually inspect the threatening failures in large software landscapes. Hence, automatic techniques have been proposed to diagnose, and resolve problems at runtime. They can be roughly grouped into the four phases of the MAPE-K [2] control loop from autonomic computing and self-adaptive software, i.e., monitoring, analysis, planning, and execution. Typically, the overall strategy is to fulfill high-level goals — e.g., with respect to avoiding violations of SLAs — by continuously monitoring measures of interest and to react to present or predicted violations. From our experience, users often mistrust such fully-automatic systems. Building trust is an open research challenge [2] in this area. Users are missing the control of the automatic changes conducted to the software landscape.

To tackle this problem, we envision a semi-automatic dependability control center for large software landscapes focusing on business critical systems. Our control center will provide a platform that features a reusable GUI to integrate tools and techniques that take part in the MAPE-K control cycle in a semi-automatic fashion. The control center is designed for extensibility by providing a plug-in architecture enabling the integration into visualization perspectives. With respect to visualization of software runtime behavior, our previous work has focused on providing different statically generated levels of visualization ([3], [4]). This can become tedious, if the software landscape provides thousands of those levels. Therefore, our envisioned control center will provide interactive live visualization of the system topology, including QoS-relevant information, e.g., useful for detecting, diagnosing, and resolving failures. The control center will be part of our ExplorViz [5] approach for comprehending large software landscapes based on live trace visualization.

The contribution of this paper is the envisioned semi-automatic control center enabling a visual integration of tools and techniques supporting the full MAPE-K cycle. The control center will be presented based on a running software aging and rejuvenation usage scenario. The plug-in architecture will be sketched including the description of example plug-ins for dependability management from our previous work.

The remainder of this paper is organized as follows. Section II provides a brief overview of the ExplorViz approach. We will present the envisioned control center in Section III. Related work will be discussed in Section IV. Section V draws the conclusions and outlines future work.

II. ExplorViz

Our ExplorViz [5] approach,[1] which builds the foundation of our control center, enables live trace visualization for comprehension of large software landscapes. In ExplorViz, every application in the software landscape is monitored. The monitored data is processed using a master-worker pattern. A model representation of the software landscape is created and updated from these traces. Afterwards, the landscape model is used to visualize the software landscape in different views.

Since it is designed for large software landscapes, it focuses on providing an overview, showing details on demand through interaction. Furthermore, it features a 2D landscape-level view where each application in the software landscape is visualized. For each application, a 3D component-level view utilizing the city metaphor [6] is provided. For further details about our approach, we refer to [5].

[1] http://www.explorviz.net

III. Control Center

This section describes our envisioned semi-automatic control center for large software landscapes by demonstrating its use based on a common scenario (Section III-A), sketching the plug-in architecture enabling custom extensions (Section III-B), and exemplifying this extensions mechanism based on approaches from our previous work (Section III-C).

A. Usage Scenario

This section aims to introduce our semi-automatic control center concept based on a common threat and management strategy for system dependability: software aging and rejuvenation. Software aging denotes the phenomenon that software components, when executed over a longer period of time, tend to show degradation effects [7]. These can manifest themselves in slightly increasing response times or memory consumption. Possible causes are software bugs, e.g., unreleased resources, whose impact accumulates over time. A common reactive or proactive resolution strategy, known as software rejuvenation, is the restart of selected system components.

As an example software landscape, we utilize a system that provides publication workflows for scientific data, called PubFlow.[2] It comprises a heterogeneous architecture of distributed applications and hence provides a well-fitting example system for our usage scenario. We assume that the response times of a PubFlow service increase over time and threaten the fulfillment of SLAs. The causing application needs to be determined, and a restart of this application needs to be planned and executed. In the remainder of this section, we describe our envisioned control center according to the activities involved in maintaining the dependability of this software landscape, i.e., viewing the symptoms (*symptoms perspective*), determining the root cause (*diagnosis perspective*), planning a countermeasure (*planning perspective*), and executing it (*execution perspective*).

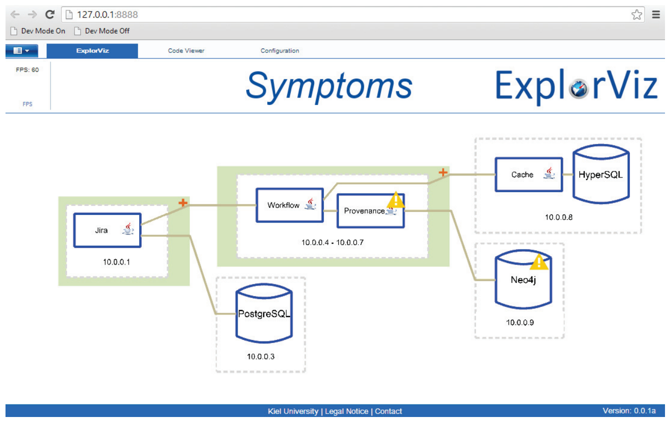


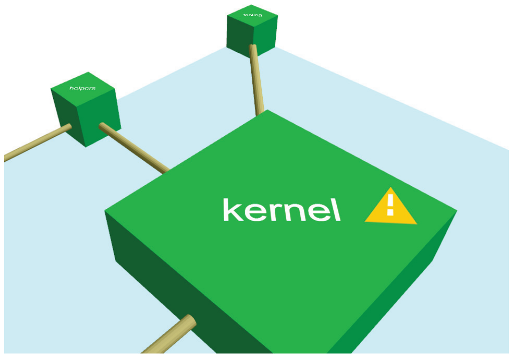
Fig. 1. Mockup of the symptoms perspective visualizing PubFlow. The Neo4J and Provenance applications are marked with a warning symbol.

1) Detecting Problems: The *symptoms perspective* is the control center's starting perspective. It provides an overall view on the software landscape, including notifications provided by control center plug-ins. The user can observe the execution of the monitored applications and monitor their status with respect to dependability. Using ExplorViz's interactive visualization concepts, it is possible to dive into single applications to get a component-level view. This will be exemplified for the following diagnosis perspective.

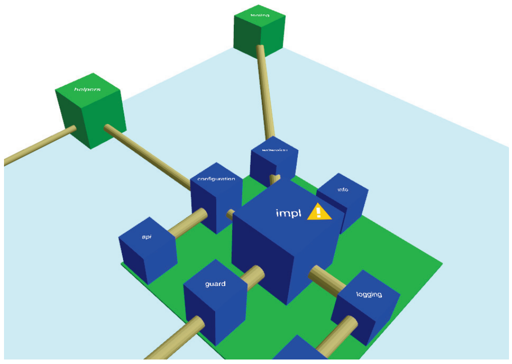
[2]http://www.pubflow.de

Figure 1 shows a mockup of the symptoms perspective including a visualization of the PubFlow system. When an abnormal state is detected or predicted by a plug-in, the corresponding application is marked with a warning symbol. In this case, we perceive that both the Neo4J and the Provenance application are marked with a warning sign as a hint for further investigation. After hovering over one of the warning icons, the message *"Warning: The response times are abnormal for Mondays at this time."* is displayed.

2) Diagnosing the Root Cause: As we want to find the root cause of this warning, we open the control center's *diagnosis perspective*. Here, the user is guided by automatic tools to find the root cause of a dependability problem — as opposed to the previously described perspective showing only its symptoms. Similar to the symptoms perspective, notifications are provided by respective plug-ins.



(a) Component-level view of Neo4J with warning sign on component `kernel`



(b) Opened `kernel` component with warning sign on component `impl`

Fig. 2. Mockup of component-level views of Neo4J

In our scenario, a diagnosis tool marks the Neo4J application as the root cause of the abnormal behavior. We want to further investigate this circumstance and thus jump into the application to get a detailed view which component is causing the abnormal response times. Figure 2 shows a mockup of the component-level view of Neo4J. The component `kernel` is marked with a warning sign. By jumping further into the `kernel`, we can see that the subcomponent `impl` is responsible for the warning.

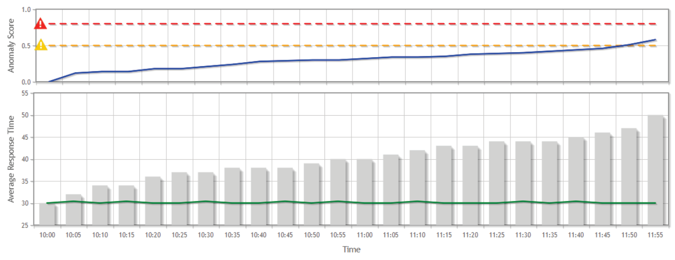


Fig. 3. Mockup of average response times (bars) and corresponding anomaly scores (upper part) for the `impl` component

To get further insights, we analyze the average response time of the `impl` component. The average response time and its corresponding anomaly score are sketched in Figure 3. The gray bars represent the average response times in time windows of five minutes. The green series in the plot represents the predicted response time basing on past behavior. The anomaly score is shown by a blue line chart above the bar chart. The thresholds for a warning and for an error are displayed in yellow and red. From the current rising response times and the normal nearly constant response times — assuming other parameters like the workload intensity to be constant — we conclude that a software aging problem exists and thus we have to trigger proactive countermeasures to ensure the QoS.

3) Planning Change: After the diagnosis, we want to act proactively to ensure that the failure will not occur. For this purpose, we envision a semi-automatic *planning perspective* for defining or changing an adaptation plan. Two possibilities for the opening of the planning perspective will exist. The first one is by manually creating a reconfiguration plan. The second one is an automatic suggestion by a plug-in on how to reconfigure the software landscape if needed. It first states the low QoS results and a short summary of how it is suggested to adapt the landscape. As a further indicator, the new QoS results and the resulting costs are displayed, e.g., by using model-based software performance prediction techniques [8]. The user has the ability to directly execute this plan or to manually refine the plan.

In our scenario, a dialog is shown reading *"The software landscape violates its requirements for response times. It is suggested to start a new node of type 'm1.small' with the application 'Neo4J' on it. After the change, the response time is improved and the operating costs increase by 5 Euro per hour"*. We choose to manually adapt the plan which opens the planning perspective. After opening the planning perspective, the reconfiguration plan that was computed by a plug-in is displayed (Figure 4).

In the planning perspective, the user has the possibility to manually refine the reconfiguration plan. The possibilities in the planning perspective are sketched in Figure 5. The user can, for instance, restart, terminate, or replicate applications. In our scenario, we have expert knowledge about the situation such that we know that the existing Neo4J application can simply be restarted to act against the software aging. Thus, we rely on our knowledge that the Neo4J application is not critical and can be restarted without violating the SLAs.

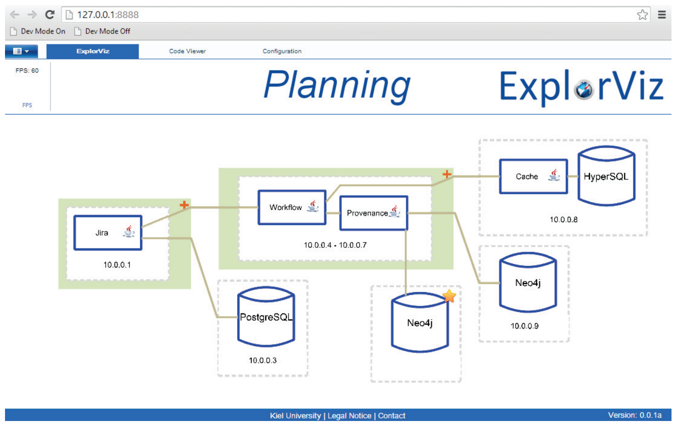
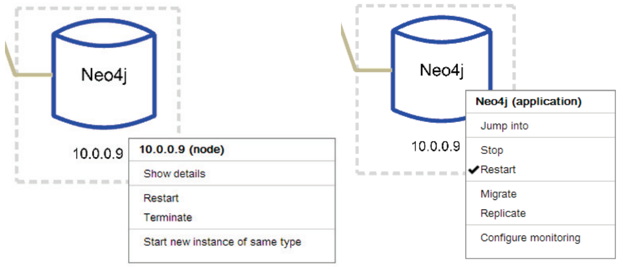


Fig. 4. Mockup of planning perspective



(a) Node context menu (b) Application context menu

Fig. 5. Mockup of reconfiguration options in planning perspective

4) Executing Change: The actual execution of the reconfiguration plan is triggered by pressing an *execute* button in the planning perspective. The tool used for this execution should provide a transactional way for this step. If the transaction fails, it should rollback the transaction and a message to the control center should be triggered. After triggering the execution, the *execution perspective* is opened. During the actual execution of the plan, the execution perspective shows what is planned and what has already been conducted. In our example, a restart of the Neo4J application is triggered and during the restart of the application, the perspective shows a blinking Neo4J application indicating that the application is currently starting.

B. *Plug-In Architecture*

The control center will provide extension points such that plug-ins can contribute to each described perspective. Furthermore, all plug-ins can read and update the model representation of the software landscape as provided by ExplorViz. The model representation of the software landscape holds entities, like nodes, applications, or generic meta-data for each entity.

Symptoms and diagnosis plug-ins can perform their analysis on the data from the landscape model repository. To visualize their results, they can enrich the visualization by, for instance, marking applications with warning or error signs. Plug-ins for planning or executing reconfigurations can provide their adaptation plan to ExplorViz for visualization and can enrich the planning perspective by, for instance, context menu entries.

In addition to the contribution to perspectives, plug-ins can contribute to the monitoring process since some information might be unavailable in the landscape model and has to be gathered from the running applications. In our envisioned control center, all applications in the software landscape will be monitored. This can be conducted with, for instance, Kieker [4]. From the monitoring data, we create and update the former mentioned model representation of the software landscape including structural and behavioral aspects. For the contribution to the monitoring process, the plug-ins can provide filter components which fit into the pipe-and-filter architecture of our monitoring data processing.

C. Prototype Plug-ins

In this section, we briefly outline three approaches that we will integrate into the control center by developing respective plug-ins. Note that each of these approaches is a result of our own previous work. However, in our future work, we will extend the set of plug-ins to integrate work by others.

1) ΘPAD: The ΘPAD approach — short for online performance anomaly detection — aims to detect anomalies in performance observations, e.g., unexpectedly high method response times. The general procedure performed by ΘPAD is that it (i) extracts a time series from incoming observations, (ii) forecasts expected future observations based on time series analysis, and (iii) computes an anomaly score by comparing forecasts and actual measurements. ΘPAD is a result of [9] and part of the Kieker monitoring framework [4]. With respect to the control center, an ΘPAD plug-in contributes to the symptoms perspective.

2) RanCorr: Based on component anomaly scores — e.g., provided by ΘPAD — RanCorr [3] localizes the root cause of problems by correlating the anomaly scores with architectural information in form of calling dependencies. RanCorr will contribute to the diagnosis perspective by providing information on problem root causes. The static visualization of RanCorr results described in [3] will then be reimplemented based on ExplorViz's interactive visualization capabilities.

3) SLAstic: SLAstic [10] is a MAPE-K framework for online capacity management of component-based software systems. Based on monitoring and predicting the current and future performance and efficiency measures, SLAstic plans and executes system adaptations, e.g., starting or stopping server nodes, or replication and migrating software components. SLAstic contributes to the planning and execution perspectives of the control center.

IV. Related Work

Work that is related to our control center approach comes from the areas of cloud management, APM, and MAPE-K tools. Amazon CloudWatch or the auto-scaling feature of Microsoft Azure provide management for capacity adaptation in the cloud. The user can specify rules that lead to starting or terminating cloud instances. In contrast to our envisioned control center, those tools provide no manual refinement of the reconfiguration plan and to the best of our knowledge no visualization of the execution of the plan. A further class are application performance monitoring (APM) [11] tools like AppDynamics, ExtraHop, or CA Wily Introscope. To the best of our knowledge, most APM tools only provide monitoring and reactive analysis of the software systems. Especially, they often do not provide capabilities to influence the system like, for instance, replicating over-utilized components. Many MAPE-K control loop tools [2], like Rainbow or TRAP, exist. A large part is focusing on automatically adapting the software systems to the environment and do not provide a semi-automatic approach for building trust in the system like our envisioned control center. Furthermore, most of them do not provide a visualization.

V. Conclusions

In this paper, we presented our envisioned semi-automatic control center for cloud dependability. The control center serves to provide a visual plug-in-based integration platform for dependability management approaches, including activities like problem detection, diagnosis, and resolution. It builds on our ExplorViz aproach for online trace visualization. We exemplified the usage by integrating approaches from our previous work for performance anomaly detection, diagnosis, planning, and adaptation. In our future work, we will provide an implementation for our control center concept and aim to develop plug-ins to integrate a number of reasonable dependability management approaches.

References

[1] A. Avritzer, A. Bondi, and E. J. Weyuker, "Ensuring system performance for cluster and single server systems," *J. Syst. Softw.*, vol. 80, no. 4, pp. 441–454, 2007.

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, 2009.

[3] N. S. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring, "Automatic failure diagnosis in distributed large-scale software systems based on timing behavior anomaly correlation," in *Proc. CSMR 2009*. IEEE, 2009, pp. 47–57.

[4] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in *Proc. ICPE 2012*. ACM, 2012, pp. 247–248.

[5] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring, "Live trace visualization for comprehending large software landscapes: The ExplorViz approach," in *Proc. VISSOFT 2013*, 2013.

[6] C. Knight and M. Munro, "Virtual but visible software," in *Proc. IV 2000*, 2000, pp. 198–205.

[7] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[8] V. Cortellessa, A. Di Marco, and P. Inverardi, *Model-based software performance analysis*. Springer, 2011.

[9] T. C. Bielefeld, "Online performance anomaly detection for large-scale software systems," Diploma Thesis, Kiel University, Germany, 2012.

[10] R. von Massow, A. van Hoorn, and W. Hasselbring, "Performance simulation of runtime reconfigurable component-based software architectures," in *Proc. ECSA 2011*. Springer, 2011, pp. 43–58.

[11] J. Kowall and W. Cappelli, "Gartner's magic quadrant for application performance monitoring," 2013.