Live Visualization of Large Software Landscapes for Ensuring Architecture Conformance

Florian Fittkau
Software Engineering Group
Kiel University
24098 Kiel, Germany
ffi@informatik.uni-kiel.de

Phil Stelzer
Software Engineering Group
Kiel University
24098 Kiel, Germany
pst@informatik.uni-kiel.de

Wilhelm Hasselbring
Software Engineering Group
Kiel University
24098 Kiel, Germany
wha@informatik.uni-kiel.de

ABSTRACT

Large software landscapes are complex Systems-of-Systems. Systems are added to, modified in, or removed from the landscape at runtime. Architectural erosion typically occurs in such landscapes, resulting in increased maintenance and operation costs. Continuous monitoring can help to ensure the architecture conformance in such large landscapes. However, the emerging huge amounts of monitoring data have to be processed and presented in a scalable visualization.

In this paper, we present ExplorViz which aims for providing such a scalable live visualization of large software landscapes. We demonstrate how our visualization can be used for ensuring architecture conformance. Furthermore, we describe an applicability evaluation of ExplorViz concerning the prerequisite of scalability in our monitoring solution.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Large Software Landscapes, Software Visualization, Monitoring, Systems-of-Systems

1. INTRODUCTION

The increasing number of software systems in organizations and enterprises form large and complex software landscapes which are Systems-of-Systems (SoS). These landscapes evolve over decades [15] and, consequently, architecture erosion occurs. This architecture erosion causes high maintenance and operation costs, rendering architecture conformance checking important [3]. Architecture conformance enables faster change of functionality and adaptation to new challenges.

Challenges imposed by architecture conformance checking are, inter alia, multiple programming languages and platforms, and scalability concern due to the large data amount [3, 4]. Those challenges become even more apparent in large software landscape. An additional challenge in large software landscapes is that the architecture can be hard to analyze statically. Distributed applications dynamically communicate via networks such that architecture conformance checking can often only be conducted at runtime.

Furthermore, due to adaptation at runtime of the software landscape, continuous monitoring [16] and continuous architecture checking is important. In general, more support for evolution in SoS architectures is required [10]. For this purpose, we developed our ExplorViz[1] approach [7]. Due to its scalable live visualization, it can support in the task of manually ensuring the architecture conformance in large software landscapes during its runtime. In summary, our main contributions are:

- a presentation of our specific visualization for large software landscapes,
- an example of how ExplorViz can be used for manual SoS architecture conformance checking, and
- an applicability evaluation of our monitoring solution.

The remainder of this paper is organized as follows. Section 2 introduces our ExplorViz visualization. In the next section, a scenario for checking architecture conformance is demonstrated. The following Section 4 presents an applicability evaluation of our monitoring solution. Related work is discussed in Section 5. Finally, we draw the conclusions and illustrate future work in Section 6.

2. EXPLORVIZ

This section briefly introduces our web-based ExplorViz visualization. It is developed for large software landscapes and thus aims for maximal visual scalability. ExplorViz features two different perspectives: a landscape-level perspective and an application-level perspective. For a detailed discussion of the semantics and more use cases, we refer to [7].

An example of the landscape-level perspective of ExplorViz is illustrated in Figure 1. The displayed gray boxes represent software systems. For providing visual scalability, every system can be opened (showing internal details) or closed (hiding internal details). Opened systems contain their nodes represented in light green and nodes contain their applications (purple color). The dark green box around a node is another abstraction concept representing a node group. In, for instance, Cloud environments, some nodes have the same

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

ECSAW, August 25 - 29 2014, Vienna, Austria

http://dx.doi.org/10.1145/2642803.2642831

Figure 1: Landscape-level perspective modeling the Kiel Data Management Infrastructure for ocean science

configuration and can be grouped. Node groups can also be opened or closed interactively. The control flow is visualized by orange lines. The amount of communication activity is mapped to the thickness of the lines. In the bottom, the timeline of the activity (sum of method calls per time unit) in the software landscape is shown. It can be used to jump back and analyze situations of interest.

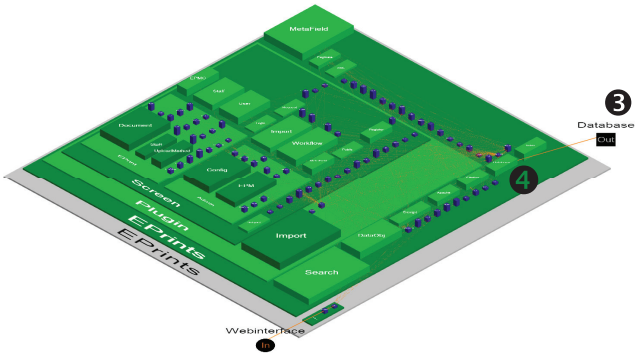

Figure 2: Application-level perspective visualizing the Perl-based application EPrints

ExplorViz enables the inspection of the control flow inside an application. Figure 2 shows our application-level perspective utilizing a 3D city metaphor [17]. The in and out ports (bottom and right side) represent the incoming and outgoing control flow to other applications. The flat green boxes represent opened component, i.e., showing its internal details. The higher green boxes are closed components which hide their internal details. The components can be opened or closed interactively. Classes are represented by purple boxes. The communication between classes or components is again displayed by orange lines.

We provide monitoring support for Java applications. However, to address the heterogeneous nature of large software landscapes, we provide an extensible adapter for reading external logs produced by software implemented in other programming languages. Thus, we can visualize logs from any systems which were generated by Kieker [14], for example.

3. SOS ARCHITECTURE CONFORMANCE

As a usage scenario, we model the GEOMAR's Kiel Data Management Infrastructure.[2] It represents a software landscape containing six systems and 30 applications. The infrastructure is used for planing, processing, and publishing data collected from scientific ocean measurements.

When a software architect intends to check whether the actual software landscape of the Kiel Data Management Infrastructure still conforms to his conceptual architecture, he opens the ExplorViz visualization of the landscape. In our scenario, he is only interested in the systems PubFlow, WDC-Mare, and OceanRep. After closing the other systems, his view looks like the one shown in Figure 1.

Now, he looks at the relevant systems and verifies that every application conforms to the conceptual communication paths. When detecting a mismatch, he jumps into the application and inspects the control flow pointing to the unwanted communication. In this scenario, the architect wonders why the EPrints (❶) application communicates to an external database (❷). In the application perspective (Figure 2), he follows the path from the outgoing communication port (❸) backwards and finds the `EPrints.Database` (❹) class, which he than inspects at the source code level with our integrated source code viewer. With this knowledge, he can take appropriate countermeasures to achieve architecture conformance.

[2] `https://portal.geomar.de/`

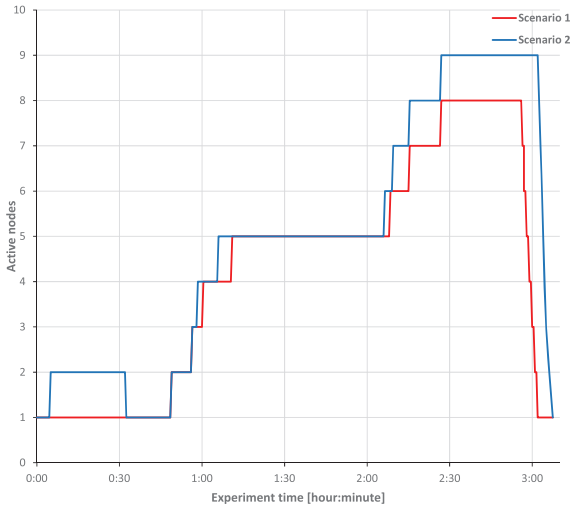

Figure 3: Node allocation of JPetStore instances in Scenario 1 and 2

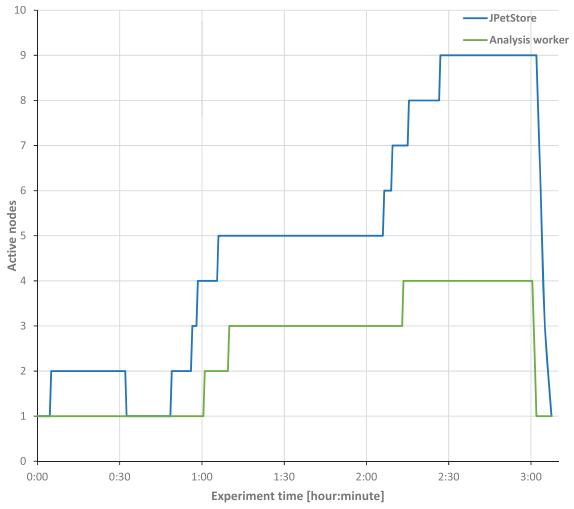

Figure 4: Node allocation of JPetStore instances and analysis worker instances in Scenario 2

4. EVALUATION OF APPLICABILITY

Particularly, in large software landscapes with often running hundreds applications, the scalability of the monitoring solution is crucial. Therefore, we have developed a flexible monitoring solution featuring cloud computing capabilities [6]. With the utilization of micro benchmarks, we have already shown that our monitoring solution has low overhead and is applicable for live analysis for one instance [6]. In this section, we provide an applicability evaluation for the scalability of our monitoring solution. Additional details can be found in [12]. We generate load on JPetStore instances. In Scenario 1 no monitoring is performed and in Scenario 2 the JPetStore instances are monitored with ExplorViz.

4.1 Setup

We utilize our private cloud containing 8 servers with 2x Intel Xeon E5-2650 (2.8GHz, 8 cores) and 128 GB of RAM each and use four types of nodes: workload generators, monitored applications, analysis workers, and load balancers. The workload generation is done with JMeter. The workload generators are configured with 16 VCPUs and 60 GB RAM each. The monitored JPetStore applications have 2 VCPUs and 4 GB RAM. Our analysis workers are configured with 2 VCPUs and 4 GB RAM and the load balancers with 4 VCPUs and 12 GB RAM.

For our study, the workload curve in our scenarios simulates a 24 hour period on an enterprise website [13]. The workload increases for a first peak at noon and goes on for its highest peak in the evening. For our scenarios, we scaled the workload down from 24 hours to a 3 hours duration. Our capacity management tool SLAstic [13] is configured to start new instances at an average CPU load above 50% and to terminate instances at an average CPU below 15%.

JMeter fetches IPs from our first load balancer and accesses the JPetStore instances with these IPs. Our monitoring solution uses analysis worker instances which are scaled according to the induced monitoring workload. The JPetStore instances fetch IPs from our second load balancer which determine the analysis worker where the monitoring data is transfered to.

4.2 Results and Discussion

The node allocation of JPetStore instances for Scenario 1 and 2 are shown in Figure 4. Due to the higher CPU utilization in Scenario 2 (monitoring enabled), the start of new instances in Scenario 2 takes place earlier and the termination is conducted later than in Scenario 1. Notably, the JPetStore node allocation only differs by a maximum of one instance. Therefore, our monitoring solution imposes only a low additional load (about 5% additional CPU load during the 5 instances period) on the monitored applications.

The node allocation of JPetStore and analysis worker instances for Scenario 2 is presented in Figure 3. The analysis workers are scaled from one instance to three instances for the first peek, and from three to four instances for the second peek. With the reduced workload, the amount is decreased to one afterwards. Therefore, the monitoring analysis scales with the workload on the monitored applications.

To summarize, we showed that our monitoring solution scales with the increasing and decreasing workload. Furthermore, it only induces only slight additional load on the monitored applications.

4.3 Threats to Validity

We conducted our scenarios on our private cloud. For external validity, it should also be evaluated in other environments and with other applications. Furthermore, the duration of our scenarios may be insufficient as the results may not be valid for longer running tests. To address these issues, more studies have to be conducted.

5. RELATED WORK

Work related to our ExplorViz approach comes from three areas: architecture conformance checking, software visualization, and monitoring solutions.

Passos et al. [11] categorize static analysis techniques for architecture conformance in dependency-structure matrices, source code query languages, and reflexion models. They recommend reflexion model [9] based tools for architecture

conformance checking during the development process (for example, Sotograph [1]). In contrast to our approach, most of those tools only provide static analysis capabilities. Furthermore, they are often only targeting one system and not a large software landscape architecture where a scalable visualization is required.

Application performance monitoring tools like AppDynamics, or ExtraHop visualize a landscape similar to our landscape-level perspective of ExplorViz. However, as far as we know, those visualizations are limited to visualizing nodes and applications. Hence, they do not provide suitable abstraction mechanisms for large software landscapes. In respect to our application-level perspective, many approaches for software visualization of single applications exist and due to space constraints we refer to [7] for a detailed comparison to other application visualization approaches.

Meng et al. [8] propose a Monitoring-as-a-Service solution for monitoring cloud infrastructures. To monitor the complex infrastructure of Cloud data centers, they developed a scalable and flexible monitoring topology consisting of different services. Compared to our approach, they also monitor the whole data center environment. Brunst and Nagel [2] present a parallel analysis infrastructure. In comparison to our approach, Brunst and Nagel do not use TCP connections to distribute produced monitoring workload to the analysis worker nodes. Instead a shared storage infrastructure is used and the monitoring data is saved in files.

6. CONCLUSIONS

In this paper, we presented our open source ExplorViz live visualization and its application for supporting in ensuring the architecture conformance during the runtime of large software landscape. The presented visualization is explicitly designed for large software landscapes and provides abstraction mechanisms for visual scalability. Another use case of ExplorViz in the context of large software landscapes (i.e., a control center concept) has been presented in [5].

In our future work, we will conduct a controlled experiment for our ExplorViz visualization targeting comprehension tasks of large software landscapes and its visualization scalability. Furthermore, capabilities for modeling the conceptual software landscape architecture in our tool will enable automatic continuous conformance checking.

7. REFERENCES

[1] W. Bischofberger, J. Kühl, and S. Löffler. Sotograph - a pragmatic approach to source code architecture conformance checking. In *Software Architecture*, volume 3047 of *LNCS*. Springer, 2004.

[2] H. Brunst and W. E. Nagel. Scalable performance analysis of parallel systems: Concepts and experiences. In *Proc. of the 10th Conf. on Parallel Computing: Software Technology, Algorithms, Architectures, and Applications*. Elsevier, 2003.

[3] F. Deissenboeck, L. Heinemann, B. Hummel, and E. Juergens. Flexible architecture conformance assessment with ConQAT. In *Proc. of the 32nd Int. Conf. on Software Engineering (ICSE 2010)*, volume 2. ACM/IEEE, May 2010.

[4] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE TSE*, 35, July 2009.

[5] F. Fittkau, A. van Hoorn, and W. Hasselbring. Towards a dependability control center for large software landscapes. In *Proc. of the 10th European Dependable Computing Conference (EDCC 2014)*. IEEE, May 2014.

[6] F. Fittkau, J. Waller, P. C. Brauer, and W. Hasselbring. Scalable and live trace processing with Kieker utilizing cloud computing. In *Proc. of the Symposium on Software Performance: Joint Kieker/Palladio Days 2013*, volume 1083. CEUR Workshop Proceedings, Nov. 2013.

[7] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: The ExplorViz approach. In *Proc. of the 1st Int. Working Conf. on Software Visualization (VISSOFT 2013)*, Sept. 2013.

[8] S. Meng, L. Liu, and V. Soundararajan. Tide: Achieving self-scaling in virtualized datacenter management middleware. In *Proc. of the 11th Int. Middleware Conf. (Middleware 2010)*. ACM, 2010.

[9] G. C. Murphy, D. Notkin, and K. Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In *Proc. of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT 1995)*. ACM, 1995.

[10] E. Y. Nakagawa, M. Gonçalves, M. Guessi, L. B. R. Oliveira, and F. Oquendo. The state of the art and future perspectives in systems of systems software architectures. In *Proc. of the 1st Int. Workshop on Software Engineering for Systems-of-Systems (SESoS 2013)*. ACM, 2013.

[11] L. Passos, R. Terra, M. Valente, R. Diniz, and N. Mendonca. Static architecture-conformance checking: An illustrative overview. *IEEE Software*, 27, Sept. 2010.

[12] P. Stelzer. Scalable and live trace processing in the cloud. Bachelor's thesis, Kiel University, Mar. 2014.

[13] A. van Hoorn, M. Rohr, I. A. Gul, and W. Hasselbring. An adaptation framework enabling resource-efficient operation of software systems. In *Proc. of the Warm Up Workshop (WUP 2009) for ICSE 2010*. ACM, Apr. 2009.

[14] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proc. of the 3rd Int. Conf. on Performance Engineering (ICPE 2012)*. ACM, Apr. 2012.

[15] M. Vierhauser, R. Rabiser, and P. Grünbacher. A case study on testing, commissioning, and operation of very-large-scale software systems. In *Proc. of the 36th Int. Conf. on Software Engineering (ICSE Companion 2014)*. ACM, 2014.

[16] M. Vierhauser, R. Rabiser, P. Grünbacher, C. Danner, and S. Wallner. Evolving systems of systems: Industrial challenges and research perspectives. In *Proc. of the 1st Int. Workshop on Software Engineering for Systems-of-Systems (SESoS 2013)*. ACM, 2013.

[17] R. Wettel and M. Lanza. Visualizing software systems as cities. In *Proc. of the 4th Int. Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007)*, June 2007.