Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach

Florian Fittkau, Jan Waller, Christian Wulf, and Wilhelm Hasselbring
Software Engineering Group, Department of Computer Science, Kiel University, Kiel, Germany
Email: (ffi, jwa, chw, wha) @informatik.uni-kiel.de

***Abstract*—The increasing code complexity in modern enterprise software systems exceeds the capabilities of most software engineers to understand the system's behavior by just looking at its program code. Large software landscapes, e.g., applications in a cloud infrastructure, further increase this complexity.**

A solution to these problems is visualizing the applications of the software landscape to ease program comprehension and to understand the respective communication. An established visualization concept is the 3D city metaphor. It utilizes the familiarity with navigating a city to improve program comprehension. Dynamic analysis, e.g., monitoring, can provide the required program traces of the communication.

In this paper, we present our live visualization approach of monitoring traces for large software landscapes. It combines a landscape and a system level perspective. The landscape level perspective provides an overview of the software landscape utilizing the viewer's familiarity with UML. The system level perspective provides a visualization utilizing the city metaphor for each software system.

I. Introduction

In many enterprises the number of systems is constantly increasing, forming large software landscapes. The knowledge of the communication, internal behavior, and utilization of these software landscapes often gets lost. This circumstance renders tools supporting the system comprehension important.

For visualizing the communication between multiple software systems in a software landscape, UML deployment diagrams are in wide use. For visualizing a single software system, several approaches [e. g., 1, 2] utilize a city metaphor since UML class diagrams provide poor scalability.

We propose a combination of both presentations to support the comprehension process of large software landscapes. Our landscape level perspective provides knowledge about the interaction of different applications and nodes in the software landscape utilizing the familiarity with UML deployment and activity diagrams. This leads to a fast overview. Our system level perspective is concerned with a single software system. It utilizes the 3D city metaphor to support the comprehension of the internal communication on the system level.

Software visualizations are either performed live or offline. Offline approaches usually require manual steps in gathering the required traces for the visualization. Live trace visualization inherently shows the collected data over a limited time span [3]. Therefore, it requires an automation of the required trace gathering which also becomes important in the comprehension process of large software landscapes consisting of hundreds of applications.

Our approach is based on incrementally revealing relations rather than showing all of them at once. Groups of objects are substituted by a single object until the user requests to reveal its content [4]. Thus, our approach provides higher scalability than showing all relationships at once and follows the Visual Information-Seeking Mantra of Shneiderman [5].

We propose our ExplorViz (http://www.explorviz.net) approach for live trace visualization as a solution to support system and program comprehension in large software landscapes. Possible scenarios include the discovery of the real communication between components and the amount of usage for each component on both the landscape and system level. In summary, our main contributions are:

- An interactive approach for the live, explorable visualization of monitoring traces
- A combination of the software landscape perspective with the system level perspective
- A landscape level live visualization based on a mix of UML deployment and activity diagram elements
- A system level live visualization utilizing the city metaphor for each software system

The rest of the paper is organized as follows. Section II describes our ExplorViz approach. An overview of our visualization is provided in Section III. Finally, related work (Section IV) and our conclusions (Section V) are discussed.

II. ExplorViz Approach

Our live trace visualization approach includes five activities (A1 to A5) which are detailed in the following. Figure 1 provides an overview of the activities.

A1 – Monitoring: The existing applications in the software landscape are monitored with, e. g., Kieker [6]. Besides monitoring operation calls in each application, remote calls are also monitored. This provides information on the communication between applications. The result of this activity is a stream of monitoring logs for the executed operations.

A2 – Preprocessing: Most servers cannot process huge amounts of incoming monitoring records typical for large software landscapes. Therefore, we use many worker nodes to preprocess the monitoring logs utilizing, for example, cloud computing. For this purpose, the logs are consolidated into traces. Due to the large amount of data, we employ trace reduction techniques [7], e. g., equivalence classes, on each worker node. The result of this activity are preprocessed traces.

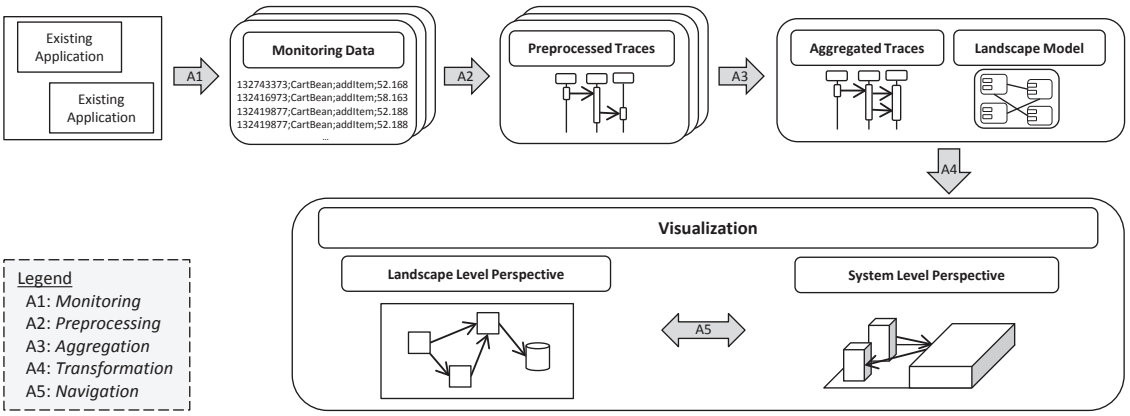

Figure 1. Activities in our ExplorViz approach for live trace visualization of large software landscapes

A3 – Aggregation: To enable a global view of the software landscape, the distributed, preprocessed traces are collected and aggregated on a single node (using the trace reduction techniques of A2). In addition, a model representation of the software landscape is created and updated. This model keeps track of the entities (e. g., nodes, applications, and components) that were discovered during runtime. This activity provides the aggregated traces and the landscape model.

A4 – Transformation: This activity consists of a transformation from the aggregated traces and the landscape model into a visualization model. The resulting visualization model only includes information relevant for the requested view.

A5 – Navigation: Our live trace visualization includes two perspectives – one for the landscape level and one for the system level. We decided to provide two different perspectives and thus metaphors because we want the user to clearly differentiate between landscape and system level. In the following section, we describe the two perspective and their navigation in more detail.

III. Live Trace Visualization

In this section, our live trace visualization approach for large software landscapes, named ExplorViz, is presented. It consists of two perspectives, namely a landscape and a system level perspective. The former perspective uses a 2D visualization employing a mix of UML deployment and activity diagram elements. The latter perspective consists of a 3D visualization. Our ExplorViz approach will be implemented as a web application utilizing WebGL.

In a large software landscape, information must be presented in a limited time span. Hence the presentation of the information must be quickly comprehensible and only reveal required information. Therefore, the major concept of ExplorViz is based on revealing additional details, e.g., the communication on deeper system levels, on demand. This provides the *scalability* of our approach.

For some analyses, particular traces or situations are of interest. During a live trace visualization the software engineer can only analyze a situation for a short duration before the visualization updates itself. ExplorViz supports the analysis of specific traces through a *time shift mode*. The user can pause the visualization to switch into an offline mode where she is able to forward or rewind the traces to a particular timestamp. A switch back to live visualization is always possible resulting in a combination of live and offline trace visualization.

Our approach is not relying on upfront static information since it would be necessary to conduct a static analysis for every application in the software landscape. Therefore, our approach needs consistent layouts [8], i.e., minimal layout changes during runtime for preserving the viewer's mental model. An additional advantage of this approach is not displaying unused code.

A visualization should provide macro, relationship, and micro views [9]. In the following, we introduce these three views for the landscape and system level perspective.

A. Landscape Level Perspective

For the layout, we utilize a layout by Klauske et al. [10] conceptualized for data-flow diagrams, i.e., communication direction is represented by layout direction. We choose this layout because the visual noise is reduced by encoding the communication direction. To support a stable layout, our landscape model introduced in A3 stores discovered entities of the software landscape.

A sketch of the *macro view* on the landscape level is displayed in Figure 2. It visualizes the application nodes and the communication between each application. Furthermore, it automatically combines similar node configurations into a single entity to support quick comprehension of the existing applications and their respective communication. The communication is visualized by edges from one application to another. The thickness of the edges represents the number of requests in the current time window. In the *relationship view*, the user can request additional details from the combined entities to further explore the relationships between instances of applications. This action results in showing the previously combined entities as single entities revealing the communication of each entity. The *micro view* on the landscape level is provided by linking to the macro view of the system level perspective.

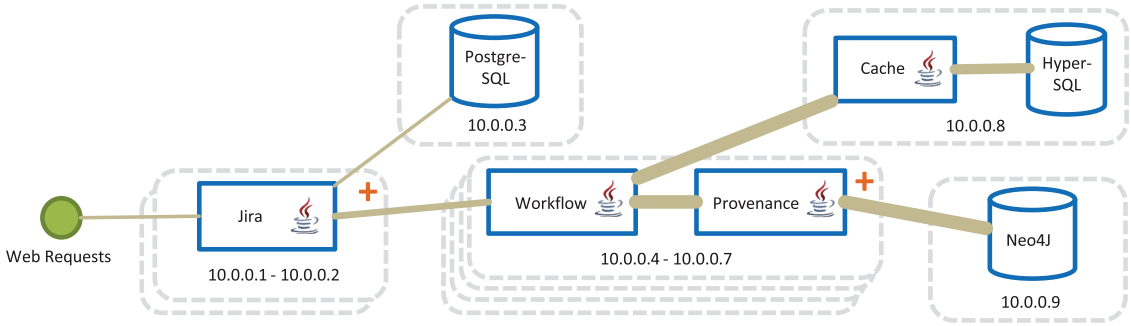

Figure 2. Macro view on landscape level showing the communication between applications in the PubFlow (http://pubflow.de) software landscape

B. System Level Perspective

The system level perspective is based on the *3D city metaphor* [1, 2]. In the following, we use the term entity as generic term for a component, a subcomponent, or a class.

Districts: Components or subcomponents (e. g., packages in Java) form the districts in our city metaphor. Each component is visualized as a rectangular layer with a fixed height. Several components are stacked upon one another to display their subcomponent hierarchies.

Buildings: Buildings represent entities, i.e., components, subcomponents, or classes. In our city metaphor, buildings become districts when they are opened (see Figure 3). The maximal count of current instances in each entity maps to the height of the corresponding building. If the entity is a class, the current instance count of the class forms the height of the building. Furthermore, the width of a building is determined by the number of classes inside the represented entity. If the entity is a class, the width is a constant minimal value.

Streets: The streets visualizing the communication are represented by pipes between entities. Different to other city metaphors, components and subcomponents can be part of the communication in addition to classes. Similar to the landscape level perspective, the thickness of the streets represents the current call count between the entities.

The *macro view* on the system level provides information on the communication of components. Figure 3a sketches the four components of jPetStore and their communication. The layout is a first prototype to show our concept of revealing details on demand in the city metaphor. In the future, we will evaluate other layouts, e.g., by Caserta et al. [2]. Again, the *relationship view* enables the user to reveal the subentities and their communication. An example is displayed in Figure 3b. It shows the opened service component. The *micro view* is provided by the link to the source code, if it is available, to the visualization and vice versa.

IV. Related Work

In this section, we discuss related approaches to our live trace visualization. Due to space constraints, we only list closely related work focusing on displaying program traces. For a more general overview on several 3D visualizations of software systems, we refer to [11].

A. 2D Visualization of Program Traces

Web Services Navigator [12] provides 2D graph visualizations of the communication of web services. Contrary, we cluster similar node configurations in our landscape perspective. Jive and Jove [3] visualize Java applications during their execution. They use a 2D visualization to achieve live trace visualization. Contrary to these tools, our approach utilizes the familiarity of UML and the city metaphor to visualize program traces for multiple software systems in a software landscape. ExtraViz developed by Cornelissen et al. [13] visualizes single program traces in two synchronized views, namely a circular bundle view and a massive sequence view. The former view utilizes hierarchical edge bundles to display the interaction of the program trace. Trümper et al. [14] visualize traces in a sequence visualization with sub ranges for details. In contrast to both approaches, ExplorViz focuses on the visualization of software landscapes and uses an exploration-based approach for displaying program traces.

B. 3D Visualization of Program Traces

Balzer and Deussen [4] provide a visualization of relations in 3D using a software landscape metaphor based on hemispheres. They define the concept of a Hierarchical Net which substitutes a group of entities with a single entity to display relations. Bohnet and Döllner [15] combine the static structure and dynamic properties of a software system in a single 3D view. The programs traces are visualized live during the runtime of the software system. TraceCrawler [16] visualizes prerecorded program traces for one feature based upon a 3D graph metaphor. Caserta et al. [2] utilize the hierarchical edge bundling technique for visualizing relations in the city metaphor of a single software system. In contrast to the former approaches, ExplorViz substitutes groups of objects by single objects for exploring relations and focuses on live visualization of multiple software systems in a large software landscape. EvoSpaces [1] represents the underlying system utilizing a 3D city metaphor. An important feature is the day view for static analysis and the night view for dynamic analysis. We also use two perspectives to visualize different properties. However, we visualize landscape and system level issues.

(a) Macro view visualizing four components of jPetStore

(b) Relationship view with opened service component

Figure 3. Mockup of system level perspective on the example of jPetStore for demonstrating the exploration concept

V. Conclusions

Our ExplorViz approach promises an effective means to ease comprehension of large and complex software landscape. In the paper, we presented a web-based visualization approach which supports in the comprehension process of large software landscapes by visualizing program traces during execution of the applications. It combines a 2D visualization similar to UML deployment and activity diagrams on the landscape level perspective and a city metaphor-based 3D visualization perspective for each application. Furthermore, the relations are explorable providing scalability and information on demand. Our open research questions are:

- Which stable layout is suitable for our 3D visualization, especially with regards to explorable relations?
- Does the direction of the communication have to be directly perceivable in the visualization or can it also be provided on demand?
- Which clustering approaches are suited to provide a synthetic hierarchy and thus scalability in our approach, when no hierarchy is present in the software system?
- Which baseline to choose, when evaluating the assistance in program comprehension by our ExplorViz approach with a controlled experiment?

References

[1] S. Alam and P. Dugerdil, "Evospaces: 3D visualization of software architecture," in *Proc. of 19th Int. Conf. on Soft. Eng. and Know. Eng.* IEEE, 2007, pp. 500–505.

[2] P. Caserta, O. Zendra, and D. Bodenes, "3D hierarchical edge bundles to visualize relations in a software city metaphor," in *Proc. of 6th IEEE Workshop on Visualizing Soft. for Understanding and Analysis*, 2011, pp. 1–8.

[3] S. Reiss and A. Tarvo, "What is my program doing? Program dynamics in programmer's terms," in *Runtime Verification*. Springer, 2012, pp. 245–259.

[4] M. Balzer and O. Deussen, "Hierarchy based 3D visualization of large software structures," *Visualization Conference, IEEE*, p. 4p, 2004.

[5] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proc. of the Symposium on Visual Languages*, 1996, pp. 336–343.

[6] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in *Proc. of the 3rd ACM/SPEC Int. Conf. on Perf. Eng.*, 2012, pp. 247–248.

[7] B. Cornelissen, L. Moonen, and A. Zaidman, "An assessment methodology for trace reduction techniques," in *Proc. of 24th Conf. on Soft. Main.*, 2008, pp. 107–116.

[8] A. Kuhn, P. Loretan, and O. Nierstrasz, "Consistent layout for thematic software maps," in *Proc. of the 15th Working Conf. on Reverse Eng.*, 2008, pp. 209–218.

[9] M. Lima, *Visual Complexity: Mapping Patterns of Information*. Princeton Architectural Press, 2011.

[10] L. K. Klauske, C. D. Schulze, M. Spönemann, and R. von Hanxleden, "Improved layout for data flow diagrams with port constraints," in *Proc. of 7th Conf. on Theory and App. of Diag.*, vol. 7352. Springer, 2012, pp. 65–79.

[11] A. R. Teyseyre and M. R. Campo, "An overview of 3D software visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 87–105, 2009.

[12] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J. F. Morar, "Web services navigator: Visualizing the execution of web services," *IBM Systems Journal*, vol. 44, no. 4, pp. 821–845, 2005.

[13] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. van Wijk, and A. van Deursen, "Understanding execution traces using massive sequence and circular bundle views," in *Proc. of the 15th IEEE Int. Conf. on Program Comprehension*. IEEE, 2007, pp. 49–58.

[14] J. Trümper, A. Telea, and J. Döllner, "Viewfusion: Correlating structure and activity views for execution traces," in *Proc. of 10th Theory and Practice of Comp. Graph. Conf.* Euro. Asso. for Comp. Graph., 2012, pp. 45–52.

[15] J. Bohnet and J. Döllner, "Visual exploration of function call graphs for feature location in complex software systems," in *Proc. of Symp. Soft. Vis.*, 2006, pp. 95–104.

[16] O. Greevy, M. Lanza, and C. Wysseier, "Visualizing live software systems in 3D," in *Proc. of the 2006 ACM Symposium on Software Visualization*, 2006, pp. 47–56.