EXPLORVIZ: VISUAL RUNTIME BEHAVIOR ANALYSIS OF ENTERPRISE APPLICATION LANDSCAPES

Complete Research

Fittkau, Florian, Kiel University, Kiel, Germany, ffi@informatik.uni-kiel.de
Roth, Sascha, Technische Universität München, Garching, Germany, roth@tum.de
Hasselbring, Wilhelm, Kiel University, Kiel, Germany, wha@informatik.uni-kiel.de

Abstract

Enterprise application landscapes are complex and hard to manage systems. Enterprise models abstract from this complexity and seek to capture relevant information to cope with business requirements, i.e., increase flexibility while reducing costs of IT. Recent automated approaches focus on utilizing existing information sources. We identified two issues in current approaches. First, in line with these approaches we observe that enterprise models often get outdated and, thus, are an unreliable basis for decision making. Second, these approaches lack detail and after larger transformations commonly the entire application landscape exhibits an altered load profile.
In this paper, we present how ExplorViz can be utilized to ensure consistency between an enterprise model and the actual information systems. We exemplify our approach via modeling the application landscape of the Kiel Data Management Infrastructure at the Helmholtz Centre for Ocean Research Kiel (GEOMAR). In our scenario, we depict an application landscape and exemplary drill-down to our EPrints productive installation to the source-code level. We explain the importance of underlying concepts and features, which help to optimize the responsiveness of an entire application landscape based on runtime information.

Keywords: Software Visualization, Enterprise Application Landscape, Dynamic Analysis, Application Monitoring.

1 Introduction

Enterprise IT infrastructures form intrinsically complex enterprise application landscapes. As a reaction to an increased market pressure, enterprises change their business models, products, and processes that implement the value chain at an increasingly rapid pace — enterprises transform cf. (Ross, Weill, and Robertson, 2006). This affects not only organizational structures but also respective information systems. In particular during turbulent times such enterprises transform at a large-scale, e.g., due to mergers and acquisitions. At the same time, enterprise models utilized to describe and manage an enterprise in a holistic manner are quickly outdated and differ considerably from the real-world. Hence, they are no more a reliable sources for decision making and planning. Even if these models are not outdated, severe runtime problems often arise after transformation projects due to an altered load profile of the entire application landscape. We conclude to the following research question: *What is an appropriate means for enterprise application landscape monitoring/planning with respect to performance characteristics?*

Enterprise Architecture (EA) researchers propose to derive plans for application landscape transitions based on information sources that commonly embrace only static data (Farwick et al., 2013; Fischer, Aier, and Winter, 2007) or probabilistic methods (Lagerström et al., 2009). While EA management does have an understanding about the relationships among applications (Winter and Fischer, 2007), they do not provide information about the internal details of an application, e.g., classes, components, and their communication. In addition, the provided data quality often becomes challenging (Roth et al., 2013b), due to, e.g., missing information, unstructured data, and wrong or outdated information.

As a consequence, recent approaches seek to gather information in a semi-automated manner from existing information sources, e.g., (Buschle et al., 2012; Farwick et al., 2013; Hauder, Matthes, and Roth, 2012; Roth et al., 2013b). These approaches typically focus on information about relationships and aggregated data which is typical for EA management (Fischer, Aier, and Winter, 2007). Hence, information does provide only little insights in application details. This especially is the case for runtime information, system load, and response times (of components). At the same time, IT operations commonly does not monitor load behavior of transitive relationships between systems. They only monitor single applications such that interconnected structures of the system of systems is not reflected by their models ignoring the complexity of an enterprise application landscape.

In line with most EA researchers and practitioners, we claim that during large IT transformations, interrelationships between applications are an essential means to derive a transition from the current to an envisioned state of the EA (Buschle et al., 2012). While current EA practices use a black-box perspective on an application landscape, we argue that application runtime behavior must be identified when deriving planned states that implement the adaptation of IT to new business demands. In contrast to related work, we argue that besides a holistic perspective also details matter when planning migrations. The identification of these details is cumbersome and the derivation of respective enterprise models usually is time-consuming (Hauder, Matthes, and Roth, 2012) for IT managers and system engineers.

Information on user traces, load profiles, and response times across the entire application landscape can help to analyze and resolve load issues in an operating enterprise application landscape. We envision monitoring of the application landscape to cope with these shortcomings. In this paper, we present how ExplorViz can be used to acquire data from monitoring different information systems that make up the enterprise application landscape and thus to keep the model of it in line with the actual applications and their communication.

Application landscapes of large-scale enterprises may embrace many hundreds or even thousands of applications (Roth and Matthes, 2014). Visualizations are a common means to analyze an application landscape (Roth, Zec, and Matthes, 2014). They help to identify patterns. Based on these ideas, ExplorViz provides a highly-specialized visualization to analyze runtime behavior and load profiles based on traces. In line with Matthes, 2008, ExplorViz provides a live visualization of large enterprise application landscapes that features different abstraction levels. We choose Design Science (Peffers et al., 2007, 2012) as research methodology and apply its two main steps Build and Evaluate.

To summarize, ExplorViz

- provides mechanisms to acquire up-to-date (live) information that is consistent with the actual enterprise application landscape and information systems,
- introspects applications and information systems (Fittkau et al., 2013a) to a fine-grained level, and
- facilitates to improve the resource capacity (Hoorn et al., 2009) in enterprise application landscapes by means of a control center (Fittkau, Hoorn, and Hasselbring, 2014).

The remainder of the paper is organized as follows. We outline a trend towards support for semi-automated enterprise application landscape analysis and revisit related work in Section 2. Section 3 describes ExplorViz and the concept for live visualization of large enterprise application landscapes. Afterwards, we demonstrate its applicability in Section 4. In Section 5, we reveal implementation details of ExplorViz and conclude the paper in Section 6 outlining limitations and future prospects.

2 Toward Automated Enterprise Application Landscape Analysis

From a mere feature perspective, contributions published by other researchers are threefold, i.e., enterprise modeling, information gathering, and visualizing enterprise application landscapes or single applications. In this section, we provide results of a comprehensive literature study on the different fields and relate the research communities with contributions made in the aforementioned areas.

The research community around Matthes presents results of a survey on Enterprise Architecture (EA) tools in (Matthes et al., 2008; Roth, Zec, and Matthes, 2014). Thereby, they underpin the importance of visual means for the analysis of enterprise models. This community further investigates issues arising when gathering information for an enterprise model (Hauder, Matthes, and Roth, 2012). They propose to extract data from existing information sources within an enterprise. Additionally, they provide an evolutionary approach that requires manual user intervention to extend and maintain an enterprise model (Roth, Hauder, and Matthes, 2013). Moreover, this community contributes means for ad-hoc analyses of enterprise models (Hauder et al., 2012; Roth and Matthes, 2014; Roth et al., 2013a; Schaub, Matthes, and Roth, 2012). In contrast to their approach, we advocate that not only relationships between applications but also their details matter in the course of a root cause analysis in complex systems of systems.

The research community around Leymann is also concerned about gathering, visualizing, and analyzing data that describes the infrastructure, cf. (Binz et al., 2013). They present the visual concept of Enterprise Topology Graph which — as the name suggests — is made up of nodes and edges, cf. (Binz et al., 2012). While this community has shifted their architecture to plug-in-based monitoring components, they also considered to read log files (Agrawal, Gunopulos, and Leymann, 1998). However, they do not provide details on the actual root cause in the event of a potential bottleneck in the application landscape.

Many other researchers provide insights in enterprise modeling. For instance, a multi-perspective view on enterprise models is provided by Frank (2011). Winter and Fischer (2007) advocate that an enterprise model captures aggregated information and stakeholders are not particularly interested in element details. Fischer, Aier, and Winter (2007) detail the process to gather information from different stakeholders. However, they do not detail how to get the information in an efficient manner. The effort strongly varies with respect to the level of detail. We refer the interested reader to (Frank et al., 2014) for a comprehensive overview of aspects relevant model in the context of business information systems. Breu et al. (2011) present a concept coined ‘living models’. Living models aim to bring together a coherent management, design, and operations of IT. While practice shows clear dependencies among each other, artifacts created and maintained by the efforts of these disciplines are viewed separately. We regard this concept ambitious and agree that — at least to a certain extend — approaches should intent to propagate changes on enterprise models to the real-world, i.e., influence reality.

With respect to software visualization, e.g., Wettel (2010) applies and evaluates the city metaphor as a three dimensional visualization. This solution focuses on a single application and aims to foster an understanding during the maintenance of the application rather then illuminating potential bottlenecks

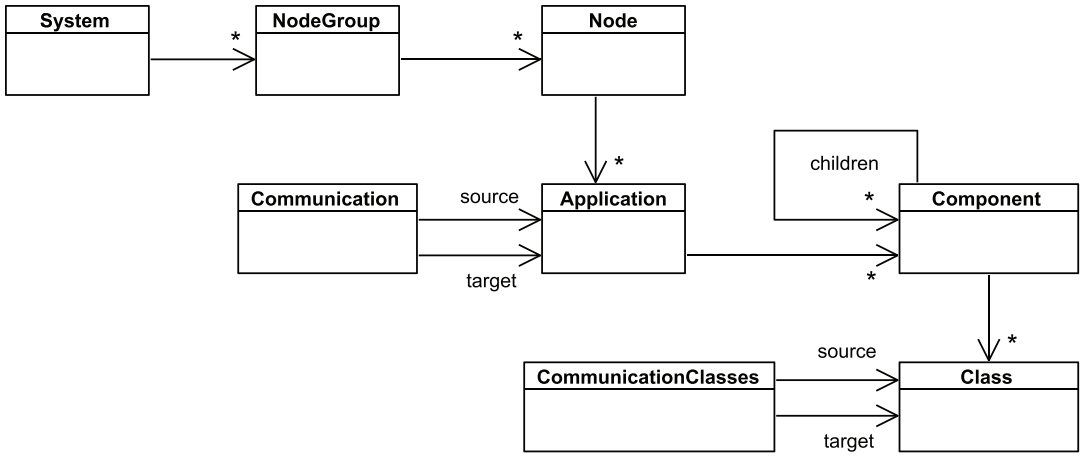


Figure 1. Overview of our enterprise application landscape meta-model

within the application landscape that may only arise through (re-)combination of applications, i.e., their identification requires runtime information.

Another class of software visualization are Application Performance Management (APM) tools. They provide monitoring and management of performance of an application landscape. Examples for APM tools include AppDynamics, dynaTrace, or CA Wily Introscope. However, to the best of our knowledge, most APM tools do not provide different abstraction levels at the landscape visualization. Furthermore, if the APM tool also allows detailed analysis of one application, the used visualization often is a tree-based viewer, which can hinder the analysis of traces with thousands of events.

3 ExplorViz Approach

ExplorViz is designed for monitoring and providing an overview of large enterprise application landscapes (Fittkau et al., 2013a). It provides different abstractions on each perspective and features two of them, namely the landscape and the application level perspective. The former visualizes the application landscape utilizing 2D elements. The latter utilizes the 3D city metaphor (Knight and Munro, 2000) to visualize one application running in the landscape. Therefore, ExplorViz enables to view into each application while still providing the landscape overview. This exposes details of the application and its architecture, and also lets the user jump to the concrete underlying source code, if available.

At first, we describe our enterprise application landscape meta-model to introduce our terminology. Afterwards, visual concepts of both perspectives, i.e., landscape and application perspective, are detailed. The example for the landscape level perspective is modeled on the basis of the Kiel Data Management Infrastructure for ocean science. The application perspective is generated from monitoring data which resulted from the work of Wechselberg (2013).

3.1 Enterprise Application Landscape Meta-Model

Figure 1 provides an overview of our enterprise application landscape meta-model. A landscape consists of different `Systems` and contains the `Communication` between the `Applications`. A `System`, e.g., PubFlow (Brauer and Hasselbring, 2012) or OceanRep from Figure 2, includes different `NodeGroups`. `NodeGroups` are formed of equal `Nodes` who are running the same application configuration. One `Node` contains identification attributes, e.g., its IP address and hostname, its current resource utilization, like CPU, free, and used RAM measured by the JVM, and a list of its applications.

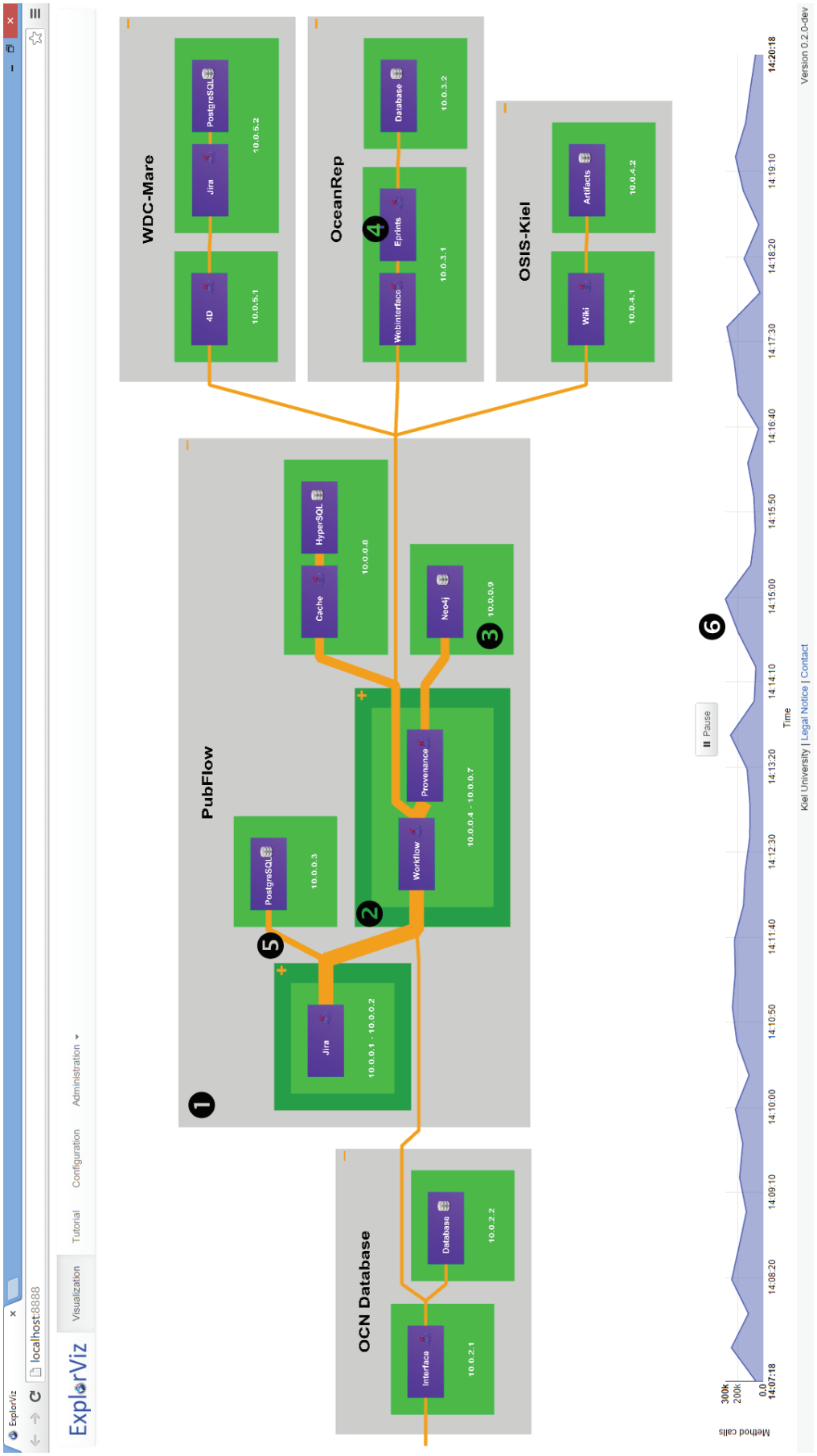


Figure 2. Landscape perspective modeling the Kiel Data Management Infrastructure for ocean science

An `Application` holds a list of its `Components`. These may contain a list of child components and a list of directly belonging `Classes`. Communication between `Classes` is modeled and a `Class` provides its current instance count allocated within the running application.

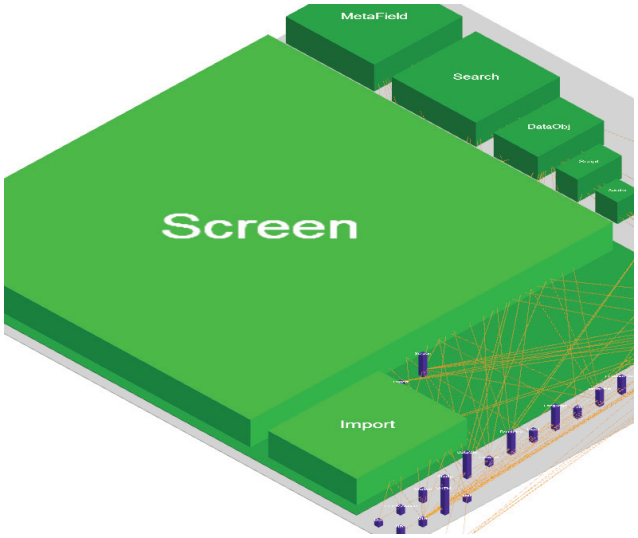
3.2 Landscape Perspective

Figure 2 shows the modeled infrastructure of the Kiel Data Management for ocean science on the landscape level. The large boxes with, e.g., PubFlow (❶), represent the information systems present in the application landscape. They can also be minimized such that only the information system and its communication are visible, without their interior. Thus, providing abstraction on the level of information systems and only visualizing systems currently in focus.
The landscape level perspective utilizes a flow-based layout. Therefore, the information systems are arranged following their communication direction, i.e., the source is on the left side and the target is on the right side.
The smaller boxes in one information system represent the contained node groups (❷) or nodes (❸). Node groups are labeled with a textual representation of their contained nodes, for example, '10.0.0.1 – 10.0.0.7'. We introduced node groups because in cloud computing, for instance, nodes are scaled for performance reasons, but typically keep their application configuration. For providing an overview, these nodes are grouped. However, they can be extended with the *plus* symbol near the node group.
A node can contain different applications (❹). The communication between applications is visualized by lines. In accordance to their call count, the line thickness changes, i.e., higher amount of communication leads to thicker communication lines (❺). Since we are interested in the runtime behavior and do not monitor data transfers, we only visualize the control flows. The user can navigate to the application level perspective by choosing one application.
ExplorViz features a time-shift feature to analyze specific situations (see ❻ of Figure 2). To provide a clue, when large amounts of calls are processed, the call count of the entire landscape is shown on the y-axis. A configurable time window is shown on the x-axis.
Furthermore, we feature a code viewer as opening dialog where the source code of an application can be analyzed. This code viewer is linked to different entities in our visualization, i.e., an application, its components, and its classes. From the logged method signatures, we can correlate components or classes with the corresponding source code, if available on the server.

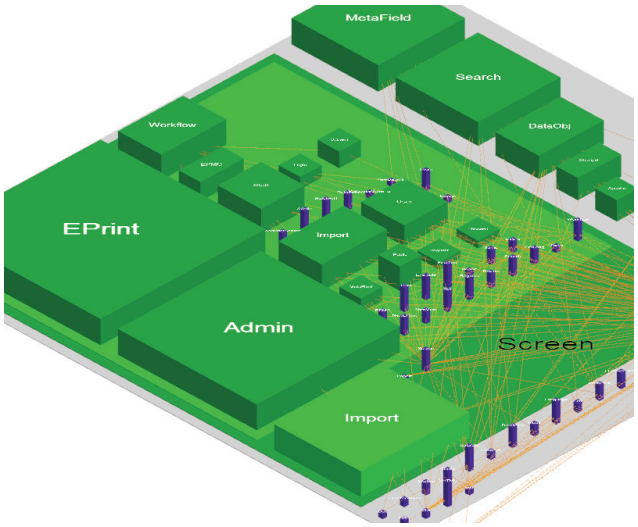
3.3 Application Perspective

In the application level perspective, we utilize the city metaphor to visualize its components and classes, and their interaction derived from the dynamic monitoring data without up-front static information. Our notion of the city metaphor and its semantics are exemplified in the following.
In Figure 3a, the structure of our EPrints[1] installation (see ❹ of Figure 2) is visualized as an example. The smallest boxes, for instance, on the bottom, represent classes. The height of each class maps to its currently active instance count in the application and the width of a class is defined as one unit.
The larger boxes represent the components of application. Notably, to be programming language independent as far as possible, we only specify components to be an organizing unit, for example, packages can be used for Java. However, also folders can be components. The height of a component is the maximum height of its contained classes or components, i.e., the highest instance count. The width property maps roughly to the amount of classes contained in it. The interaction amount of the components or classes is visualized by the thickness of the connecting pipes. The layout is inspired by the layout of CodeCity Wettel, 2010.
Contrary to other approaches utilizing the city metaphor, e.g., (Caserta, Zendra, and Bodenes, 2011; Steinbrückner, 2010; Wettel, 2010), we do not visualize the whole application in class level detail at once.

[1] `http://www.eprints.org/`



(a) EPrints with closed component `Screen`



(b) EPrints with opened component `Screen`

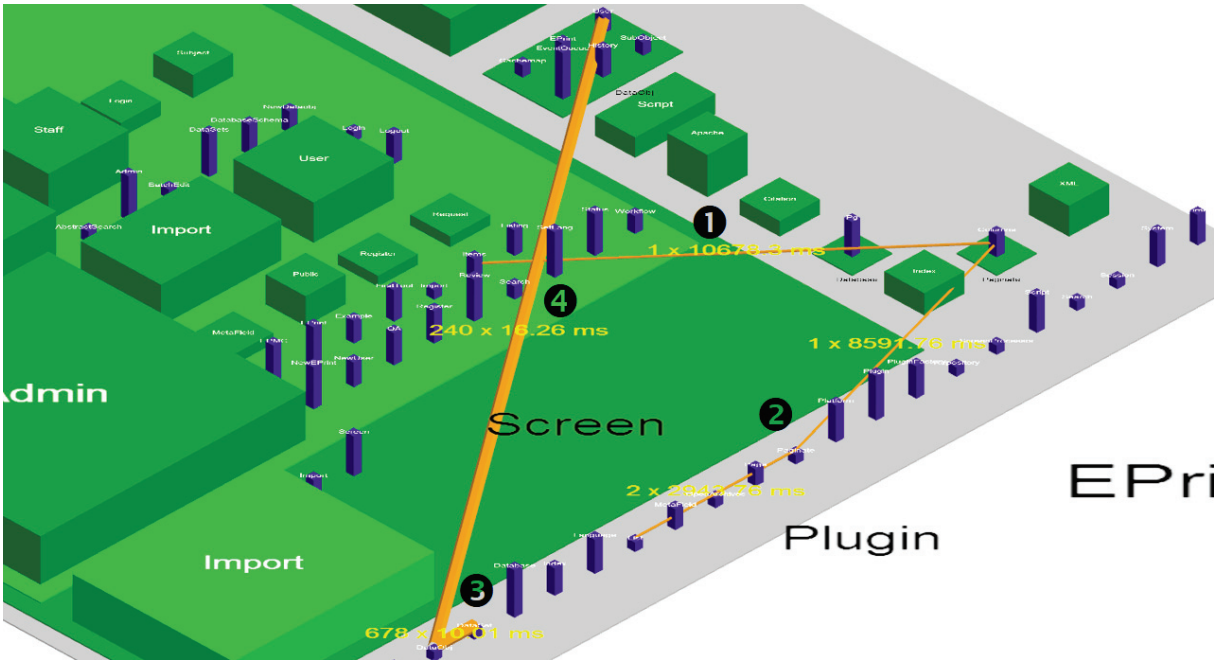
Figure 3. Application level perspective visualizing the Perl-based application EPrints

We follow a top down approach, where only top-level components and their relationships are shown, i.e., hiding internal details of those components. Our navigation concept bases on focusing the entities and interactions that are currently of interest. This concept is visualized by the transition from Figure 3a to Figure 3b. After analyzing the interaction between the top level components, we might want to get more information by opening the `Screen` component to find the root cause of high interaction.

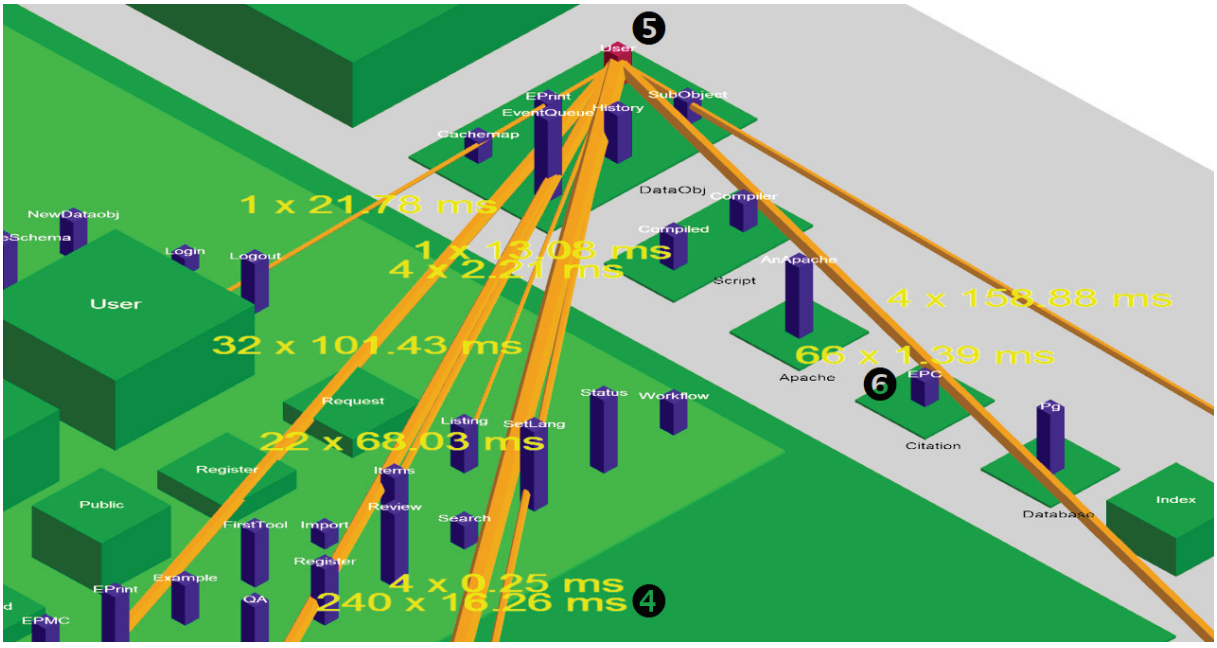
Notably, our approach of the application perspective assumes a monitored application written in a language that provides a class concept and an organization structure of those, e.g., Java packages.

4 Applying ExplorViz

To illustrate the utility of ExplorViz, we apply it in a manual runtime behavior analysis of our application landscape (cf. Figure 2). Our OceanRep installation (❹) of the Perl-based literature repository software EPrints (http://www.eprints.org, last accessed 2014-11-26) required over 10 seconds to respond in the web portal, which resulted in low user satisfaction (Wechselberg, 2013).



(a) Visualizing the highest 1% of the product of average response time and the call count



(b) Visualizing the communications of the `DataObj.User` class

Figure 4. Visualizing method call count and average response times in EPrints

We utilize the gathered monitoring data from Wechselberg (2013), who manually searched for bad performing calls in log files, and proceed with a performance analysis of EPrints employing ExplorViz. The entire application landscape has already been shown in Figure 2. Thus, we directly zoom into the details of EPrints (cf. ❹ in Figure 2). In our scenario, we are currently experiencing performance problems. Hence, we analyze the method calls between applications and within the EPrints application visually.
Figure 4 illustrates the upper 1% of invocation time, i.e., duration of a method call multiplied with the average response time. The highest time of 10,678 ms occurs between `Plugin.Screen.Items` and `Paginate.Columns` (❶) which results from its incoming method calls (❷). After analyzing the method calls to the `List` class of EPrints, where the high response time originates, we did not find any potential for optimizations. Therefore, we looked at the class interaction between `DataSet` and `DataObj` (❸).
Those classes are typically used when returning results from the database and consume 6,780 ms. This cannot be attributed to a concrete use case directly. Thus, we investigated the communication between `DataObj` and `DataObj.User` (❹) and analyzed the methods of the `DataObj.User` class (❺) to find promising optimization options. The source code revealed extensive method calls like 'has_privilege' to DataSet and meta field calls to the `Repository` class (❻). Looking at the call amount and their average response time, we concluded that the database queries should be reduced. A common means to reduce this kind of problem is the introduction of a cache which confirms the conclusions of Wechselberg (2013).

5 ExplorViz Implementation

In this section, we detail our implementation, how we achieve to monitor and analyze the large amount of gathered information, which can be millions of method call events per second in a large enterprise application landscape.
Our implementation of ExplorViz consists of three steps. The first step is monitoring the applications and their distributed communication (Section 5.1). Afterwards, the monitoring records, representing one method call each, are aggregated to traces and compressed in an elastic manner (Section 5.2). Then, our web application visualizes this created model representation of the enterprise application landscape (Section 5.3). To setup ExplorViz, one needs to instrument each application in the landscape with our monitoring component and start the server component (WAR file) in an application server.
On our website[2] we provide three out of the box components, i.e., monitoring, analysis worker, and the web application. To keep the overhead as low as possible, we benchmarked and engineered towards high throughput and minimal impact on the monitored application (Fittkau et al., 2013b; Waller, Fittkau, and Hasselbring, 2014). The techniques and concepts of each component are detailed in the following sections.

5.1 Monitoring

The first step is monitoring all information systems. Basically, we distinguish between application level monitoring and the monitoring of remote procedure calls. The later is often very specific to the utilized technology.

5.1.1 Application Level Monitoring

For monitoring each application in the application landscape, we developed a specialized, high-throughput monitoring component. Via an adapter, we still support other inputs such as Kieker (Hoorn, Waller, and Hasselbring, 2012) monitoring records, which provides the ability of monitoring other programming languages than Java, for example, C, C++, or Perl.
One concept is the spatial separation of the generation of the monitoring records and its processing. Hence, we aim for the collection of the minimal data and directly transfer those data in an as small as possible format onto another server by using a TCP binary writer.

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

For Java-based programs, we utilize AspectJ to inject monitoring probes into the applications running in an enterprise application landscape. Those probes generate monitoring records consisting of information about, for instance, the trace identifier, order index, and logging timestamp. To monitor a Java-based application, one only needs to include the provided monitoring component as an agent at application start.

Extensive micro-benchmarks revealed a low monitoring overhead of 6 μs in contrast to 59 μs of our comparison tool. For the measurement technique and details of the low overhead of our monitoring solution, we refer to (Waller, Fittkau, and Hasselbring, 2014).

5.1.2 Monitoring of Remote Procedure Calls

Since we aim for minimal impact on the monitored application and on high throughput, we utilize the concept of sending additional information with the remote procedure call, instead of, e.g., global trace identifier synchronization. For example, extending the HTTP header with information about the trace identifier from where the remote procedure call happened. The monitoring at the callee site provides the relation between the caller trace identifier and the own currently used trace identifier.

Notably, this technique also comes with some disadvantages. For instance, the addition of information to a remote procedure call is often technology dependent. Furthermore, the technology might not be designed for addition of information. To monitor such technology, often source code changes are required making the solution also technology version dependent.

5.2 Elastic Trace Analysis

A large enterprise application landscape can generate millions of monitoring records per second which have to be processed. Since we render a live visualization, this large amount has to be processed rapidly to guarantee user responsiveness and up-to-date information. Furthermore, typically the application landscape running, for instance, in a cloud environment, often steadily change due to adaptation to the workload. This often results in more generated monitoring records. Those challenges led to the design of a scalable trace analysis solution utilizing cloud computing.

Our analysis worker concept provides the possibility to scale with the workload. Different monitoring components can write to one analysis worker component, which then processed the received monitoring records. To consolidate the processed monitoring information, all analysis worker write their results to one analysis master, which again processes incoming monitoring information.

In this setup, one analysis master can become the bottleneck, if high workload is encountered in the system. Therefore, we will utilize different processing levels of analysis workers, i.e., chaining of analysis workers. Currently, we are working on guidelines when to use multiple levels and are evaluating this concept. For details, we refer to (Fittkau et al., 2013b).

5.3 Visualization

After creating or updating the model representing the current enterprise application landscape, the next step is visualizing the model. As frontend, we utilize web browsers, since live monitoring the landscape should be accessible on different clients without further software installation. Since our application level perspective is a 3D visualization, we chose WebGL as the rendering technology. In addition, WebGL provides the advantage that no additional plug-ins must be installed and mobile devices are also usable.

Since WebGL requires to write extensive JavaScript code, we utilize Google Webkit Tool (GWT). With GWT we are able to write our source code in Java. Afterwards, GWT generates the JavaScript code from the Java sources.

6 Conclusions

As a consequence of large-scale enterprise application landscapes transformations, the highly interwoven applications exhibit an altered load profile. This can lead to potential bottlenecks that are currently not anticipated by the EA management discipline when deriving planned states of an EA. A particular challenge addressed by other researchers is consistency between enterprise models and the real world. However, current approaches that seek to improve the situation do not consider details of applications, especially when it comes to source code and runtime information.

We presented an approach that utilizes application monitoring to improve consistency between the enterprise models and the real information systems. In contrast to related approaches, we not only visualize relationships among applications but also include details (source code) and incorporate runtime information. This way, optimizations can be driven by a visual analysis of load in an interactive visualization. Thus, it is a means for enterprise application landscape monitoring and planning.

Appropriate visual concepts have been introduced in this paper. We further exemplified our implementation based on an application landscape and narrowed the root cause of a real-life bottleneck in the productive EPrints installation to the respective source code. This example served to showcase the utility of our approach in a minimalistic example.

In some scenarios, the usability of the visualization is currently limited. For instance, in the absence of organizing units (components), all classes or functions are currently visualized at one level. Depending on the amount of concepts, the visualization would not scale and may require a clustering into hierarchical components. Furthermore, we can only visualize the systems which are instrumented and not instrumented system can only be handled as black boxes.

Future work lies in improving aspects of our visualization like the 3D layout algorithm. In addition, we are working on guidelines when to use multiple levels of analysis workers and currently are evaluating this concept. We are currently investigating clustering techniques and heuristics. We are also working on more advanced filters increasing the analysis such that users are able to visualize runtime information by specifying a query. Another direction seeks to develop plug-ins for ExplorViz. Those plug-ins allow to collaborate with other researchers and practitioners which potentially want to enrich the enterprise model and respective visualization with further details. Furthermore, we will asses the user experiences in a controlled experiment providing evidence for the improved effectiveness and efficiency of our approach. Visualization of performance anomalies (Marwede et al., 2009) and faults (Ploski et al., 2007) is also subject to future work.

Our prototypical implementation of ExplorViz and its source code, available under the Apache 2 license, can be downloaded at `http://www.explorviz.net`. It comes with three components, i.e., monitoring, analysis worker, and the web application (visualization).

References

Agrawal, R., D. Gunopulos, and F. Leymann (1998). *Mining Process Models From Workflow Logs*. Springer.

Binz, T., C. Fehling, F. Leymann, A. Nowak, and D. Schumm (2012). "Formalizing the Cloud through Enterprise Topology Graphs." In: *Proc. IEEE 5th International Conference on Cloud Computing (CLOUD)*. IEEE, pp. 742–749.

Binz, T., U. Breitenbucher, O. Kopp, and F. Leymann (2013). "Automated Discovery and Maintenance of Enterprise Topology Graphs." In: *Proc. IEEE 6th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, pp. 126–134.

Brauer, P. C. and W. Hasselbring (2012). "Capturing Provenance Information with a Workflow Monitoring Extension for the Kieker Framework." In: *Proceedings of the 3rd International Workshop on Semantic Web in Provenance Management*. Vol. 856.

Breu, R., B. Agreiter, M. Farwick, M. Felderer, M. Hafner, and F. Innerhofer-Oberperfler (2011). "Living Models - Ten Principles for Change-Driven Software Engineering." *International Journal of Software and Informatics* 5 (1-2), 267–290.

Buschle, M., M. Ekstedt, S. Grunow, M. Hauder, F. Matthes, and S. Roth (2012). "Automated Enterprise Architecture Documentation using an Enterprise Service Bus." In: *Proc. America's Conference on Information Systems (AMCIS)*.

Caserta, P., O. Zendra, and D. Bodenes (2011). "3D Hierarchical Edge Bundles to Visualize Relations in a Software City Metaphor." In: *Proc. 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE.

Farwick, M., R. Breu, M. Hauder, S. Roth, and F. Matthes (2013). "Enterprise Architecture Documentation: Empirical Analysis of Information Sources for Automation." In: *Proc. 46th Hawaii International Conference on System Sciences (HICSS)*.

Fischer, R., S. Aier, and R. Winter (2007). "A Federated Approach to Enterprise Architecture Model Maintenance." In: *Proc. 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA)*, pp. 9–22.

Fittkau, F., A. van Hoorn, and W. Hasselbring (2014). "Towards a Dependability Control Center for Large Software Landscapes." In: *Proc. 10th European Dependable Computing Conference (EDCC)*.

Fittkau, F., J. Waller, C. Wulf, and W. Hasselbring (2013a). "Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach." In: *Proc. 1st IEEE International Working Conference on Software Visualization (VISSOFT)*. IEEE.

Fittkau, F., J. Waller, P. C. Brauer, and W. Hasselbring (2013b). "Scalable and Live Trace Processing with Kieker Utilizing Cloud Computing." In: *Proc. Symposium on Software Performance: Joint Kieker/Palladio Days*. Vol. 1083. CEUR Workshop Proc., pp. 89–98.

Frank, U. (2011). *The MEMO Meta Modelling Language (MML) and Language Architecture (2nd Edition)*. Tech. rep. 24. ICB-Research Report.

Frank, U., S. Strecker, P. Fettke, J. Brocke, J. Becker, and E. Sinz (2014). "The Research Field "Modeling Business Information Systems"." *Bus. Inf. Syst. Eng.* 6 (1), 39–43.

Hauder, M., F. Matthes, and S. Roth (2012). "Challenges for Automated Enterprise Architecture Documentation." In: *Proc. Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation*. Springer, pp. 21–39.

Hauder, M., F. Matthes, S. Roth, and C. Schulz (2012). "Generating Dynamic Cross-Organizational Process Visualizations through Abstract View Model Pattern Matching." In: *Proc. Architecture Modeling for Future Internet enabled Enterprise (AMFInE)*.

Hoorn, A. van, J. Waller, and W. Hasselbring (2012). "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis." In: *Proc. 3rd ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, pp. 247–248.

Hoorn, A. van, M. Rohr, I. A. Gul, and W. Hasselbring (2009). "An Adaptation Framework Enabling Resource-efficient Operation of Software Systems." In: *Proc. of the Warm Up Workshop (WUP 2009) for ACM/IEEE ICSE*. Cape Town, South Africa: ACM, pp. 37–40.

Knight, C. and M. Munro (2000). "Virtual but Visible Software." In: *Proc. IEEE International Conference on Information Visualization 2000*. IEEE, pp. 198–205.

Lagerström, R., U. Franke, P. Johnson, and J. Ullberg (2009). "A Method for Creating Enterprise Architecture Meta-models – Applied to Systems Modifiability Analysis." *International Journal of Computer Science and Applications* 6 (5), 89–120.

Marwede, N., M. Rohr, A. van Hoorn, and W. Hasselbring (2009). "Automatic Failure Diagnosis Support in Distributed Large-Scale Software Systems Based on Timing Behavior Anomaly Correlation." *European Conference on Software Maintenance and Reengineering* 0, 47–58.

Matthes, F. (2008). "Softwarekartographie." *Informatik Spektrum* 31 (6), 527–536.

Matthes, F., S. Buckl, J. Leitel, and C. Schweda (2008). *Enterprise Architecture Management Tool Survey 2008*. Tech. rep. TUM.

Peffers, K., T. Tuunanen, M. Rothenberger, and S. Chatterjee (2007). "A Design Science Research Methodology for Information Systems Research." *Journal of Management Information Systems* 24 (3).

Peffers, K., M. Rothenberger, T. Tuunanen, and R. Vaezi (2012). "Design Science Research Evaluation." In: *Design Science Research in Information Systems*. Ed. by K. Peffers, M. Rothenberger, and B. Kuechler. Vol. 7286. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 398–410.

Ploski, J., M. Rohr, P. Schwenkenberg, and W. Hasselbring (2007). "Research Issues in Software Fault Categorization." *SIGSOFT Software Engineering Notes* 32 (6), 1–8.

Ross, J. W., P. Weill, and D. Robertson (2006). *Enterprise Architecture as Strategy: Creating a Foundation for Business Execution*. Harvard Business Press.

Roth, S., M. Hauder, and F. Matthes (2013). "Collaborative Evolution of Enterprise Architecture Models." In: *Proc. 8th International Workshop on Models at Runtime (Models@run.time)*.

Roth, S. and F. Matthes (2014). "Visualizing Differences of Enterprise Architecture Models." In: *Proc. International Workshop on Comparison and Versioning of Software Models (CVSM) at Software Engineering (SE)*. Kiel, Germany.

Roth, S., M. Zec, and F. Matthes (2014). *Enterprise Architecture Visualization Tool Survey 2014*. Tech. rep. Technische Universität München.

Roth, S., M. Hauder, M. Zec, A. Utz, and F. Matthes (2013a). "Empowering Business Users to Analyze Enterprise Architectures: Structural Model Matching to Configure Visualizations." In: *Proc. 7th Workshop on Trends in Enterprise Architecture Research (TEAR)*.

Roth, S., M. Hauder, M. Farwick, R. Breu, and F. Matthes (2013b). "Enterprise Architecture Documentation: Current Practices and Future Directions." In: *Proc. 11th International Conference on Wirtschaftsinformatik (WI)*.

Schaub, M., F. Matthes, and S. Roth (2012). "Towards a Conceptual Framework for Interactive Enterprise Architecture Management Visualizations." In: *Proc. Modellierung*. Bamberg, Germany.

Steinbrückner, F. (2010). "Coherent Software Cities." In: *Proc. IEEE International Conference on Software Maintenance (ICSM)*. IEEE.

Waller, J., F. Fittkau, and W. Hasselbring (2014). "Application Performance Monitoring: Trade-Off between Overhead Reduction and Maintainability." In: *Proceedings of the Symposium on Software Performance 2014*. University of Stuttgart, pp. 46–69.

Wechselberg, N. B. (2013). "Monitoring von Perl-basierten Webanwendungen mittels Kieker." Bachelor thesis. Kiel University.

Wettel, R. (2010). "Software Systems as Cities." PhD thesis. University of Lugano.

Winter, R. and R. Fischer (2007). "Essential Layers, Artifacts, and Dependencies of Enterprise Architecture." *Journal of Enterprise Architecture* 3 (2), 7–18.