Visualizing Modules and Dependencies of OSGi-based Applications

Doreen Seider, Tobias Marquardt, Marlene Brüggemann, and Andreas Schreiber
German Aerospace Center (DLR), Simulation and Software Technology
Linder Hoehe, 51147 Cologne, Germany
{doreen.seider,tobias.marquardt,andreas.schreiber}@dlr.de

***Abstract*—The architecture of software it not tangible; but in different situations it is preferable to have it tangible. For example, while reviewing it against the intended design, introducing the software to others, or starting to develop on a new part. Basic aspects of a software architecture are the modules the software is constructed of and the dependencies between them. To comprehend these aspects is important especially for software using a technology such as OSGi, which key concept is modularization.**

In this paper, we describe interactive visualization tools that we developed to comprehend OSGi-based applications with their modules and dependencies. We focus on concepts to treat large number of modules and dependencies: navigation, filtering, and selection. We applied our solution for OSGi-based applications with hundreds of modules containing multiple submodules each. With the resulting visualizations, we can explore the modularization of the software architecture.

I. Introduction

Every software has an architecture. The architecture is a result of the software implementation. The implementation can be driven by an architecture design which describes the intended software architecture. Such a design does not guarantee that the actual architecture always follows the intended design. Whereas for small software, it might be derived from the implementation, of large software the actual architecture is invisible [1].

Comprehending the actual architecture is important in many situations in the day-to-day life of software development (see Section IV-C for examples). For that, the actual software architecture needs to be visible. Software architecture per se is an intangible concept [2]. To make it visible metaphors are used that visualize certain aspects such as the cartographic metaphor [3].

We present tools to visualize aspects of the architecture of OSGi-based applications. We focus on a basic aspect of software architectures, the modules a software is constructed of, and the dependencies between the modules. Section II introduces OSGi-based applications and the characteristics we considered. We describe the extraction of information regarding modules and their dependencies from OSGi-based applications in Section III. In Section IV, we present an interactive browser-based visualization and a virtual-reality-based tool. Section V concludes the paper and gives ideas for future work.

II. OSGi-based Applications

OSGi (Open Services Gateway Initiative) [4] is a specification that defines a dynamic module system for Java. Multiple implementations exist such as Apache Felix [5], Eclipse Equinox [6], or Knopflerfish [7].

A module is a unit of functionality. In Java, different types of modules exist:

- Class: unit of state and behavior
- Package: set of classes

OSGi adds two additional types of modules:

- Bundle: set of packages
- Service: class instances that are shared between bundles

The concept of OSGi Declarative Services (OSGi DS) [8] adds another module type:

- Service component: class in a bundle that provides or consumes services

An OSGi-based application consists of a runtime and a set of bundles. A bundle can provide services that can be used by other bundles [4]. When using OSGi Declarative Services, a bundle can declare service components as the units in the bundle that provide and use services (Fig. 1).

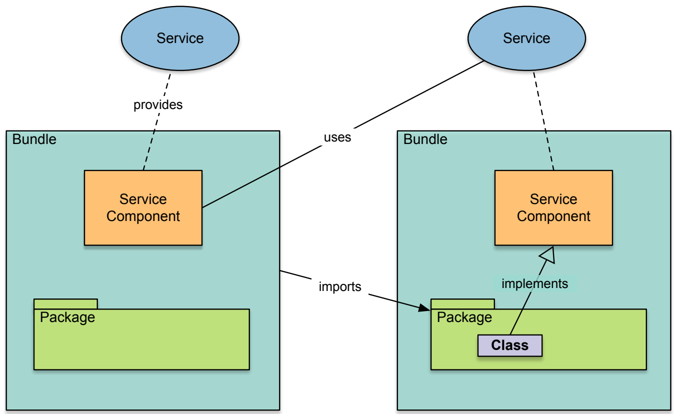


Fig. 1. OSGi packages and services.

Bundles are designed to hide their implementation from each other. A bundle defines which packages are allowed to be used by other bundles; it declares those packages as exported. A bundle also defines which packages it requires; it declares them as imported. Usually, it does not care from which bundle the package is imported. At startup, the OSGi runtime resolves the package dependencies and wires the bundles. The same applies for services. A service component gets required services injected if another service component is providing a matching one.

The following dependencies in an OSGi-based application can be derived:

- Package level: bundle ↔ package ↔ class
- Service level: bundle ↔ service component ↔ service

OSGi-based applications are dynamic regarding their modules: bundles can be installed and uninstalled, services can be registered and deregistered at any time.

III. Extracting Information about Modules and Dependencies

Information about the modules and dependencies of OSGi-based applications are distributed among different files. To visualize modules and dependencies, the information need to be extracted and aggregated, which requires information about:

1) Imported and exported packages per bundle (purpose: dependencies between bundles based on imported packages)
2) Service components (per bundle) and their services provided and required (purpose: dependencies between bundles based on service usage)
3) Foreign class usage per class (purpose: dependencies between packages and classes)
4) Number of packages per bundle, number of classes per package, LOC of classes (module size)

Extracting information and visualizing them are decoupled from each other. First, the code base of the OSGi-based application is analyzed, which results in a generated model that contains the required information. The model is stored in a file and passed to the visualization (Fig. 2). The model is the interface between analysis and visualization. To guarantee a reliable data interchange, the information in the model is structured according to a metamodel that we defined.

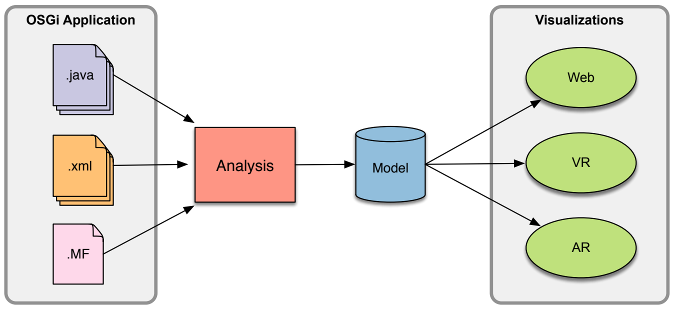


Fig. 2. Decoupled analysis (information extraction) and visualization.

IV. Visualizing Modules and Dependencies

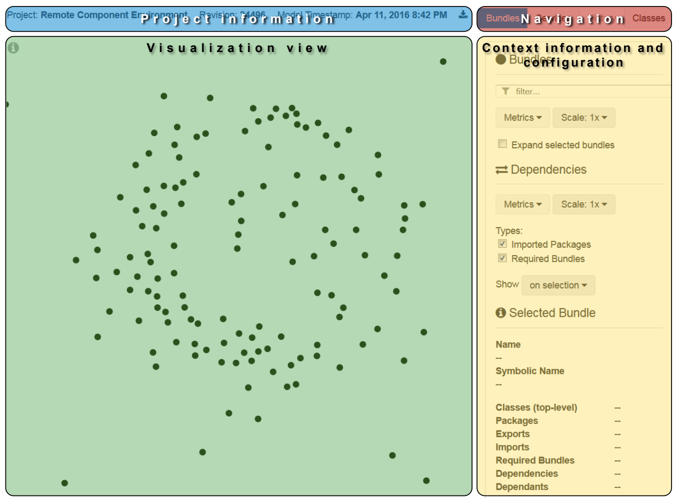
Software visualization can be classified by the aspects it considers: structure, evolution, and behavior [9]. In this section, we present structural visualizations of modules and dependencies. It focuses on the modules of OSGi-based applications on different abstraction levels and the dependencies between them. It intends to improve the comprehension of an OSGi-based application for different kind of persons: developers, project managers, or third-party persons (see Section I).

The amount of information to visualize increases with the number of bundles and services. Visualizing modules (bundles, packages, classes, services) and all of their dependencies at once including metrics (Section II) does not scale for applications with hundreds of bundles and services. To reduce the perceived complexity we applied techniques of interaction [10], [11]: navigation, filtering, selection, reconfiguration, encoding, links, and abstraction/refinement.

A. Browser-based Visualization

The data visualization of the browser-based tool follows a multi-view concept. Multiple views exist which focus on certain information each. The views are linked between each other. When navigating from one view to another selections are considered and filtering is applied. The graphical user interface is divided into four parts (Fig. 3):

1) *Project information*: static information about the OSGi-based application under consideration (e.g., name and time the model was generated)
2) *Navigation*: button-based bar for selecting the type of module that should be visualized
3) *Visualization view*: data visualization, depending on the selected module type
4) *Context information and configuration*: information about selected elements and configuration of the view (e.g., applying metrics)



Project: Remote Component Environment Revision: 24496 Model Timestamp: Apr 11, 2016 8:42 PM

[ ] Expand selected bundles

Types:
[✓] Imported Packages
[✓] Required Bundles

Show on selection

Name
--
Symbolic Name
--

Classes (top-level)	--
Packages	--
Exports	--
Imports	--
Required Bundles	--
Dependencies	--
Dependants	--

Fig. 3. Graphical user interface of the browser-based visualization

At start, the visualization view shows all bundles of the application, each as a filled circle. The dependencies are hidden and no metrics are applied. It is a neutral entry point for further exploration.

The configuration part allows to reconfigure the visualization. For the bundle visualization, the dependencies can be

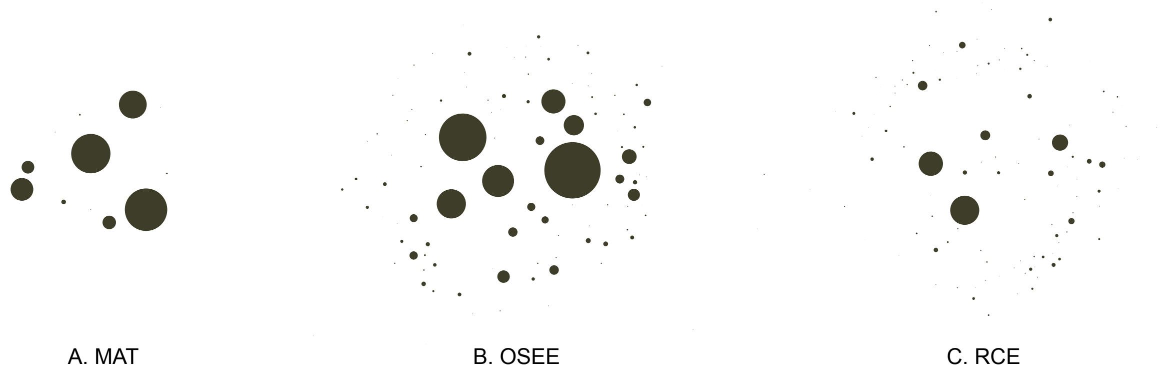


Fig. 4. Bundles visualized as force-directed graph with hidden edges; lines of code aggregated over the classes of a bundle is mapped to the size of the circles. Applications from left to right: Memory Analyzer (MAT), Open Systems Engineering Environment (OSEE), Remote Component Environment (RCE)

shown as edges between the circles to get a graph. Metrics can be applied so that the circles and edges are of different size and color. The configuration part also provides a filter option to search for modules.

With the navigation bar, the other views can be reached which visualize services, packages, and classes as well as the dependencies between them. They are intended to visualize snippets of the OSGi-based application on a lower abstraction level. The snippet is defined by the selection of the bundles in the first view.

The browser-based visualization is implemented as a single HTML file which loads JavaScript code for the data visualization; an installation on a Web server is not necessary. The data visualization is implemented with the JavaScript library *Data-Driven Documents* (D3) [12]. It generates interactive data visualization using the Web standards HTML5, CSS, and SVG. The concept of a single page application (SPA) does not require a reloading of the site at any time so that is feels similar to a desktop application.

B. Virtual-Reality-based Visualization

For very large applications and higher dimensional data, we explore other types of visualizations such as *Virtual Reality* (VR). With VR techniques, one could utilize the three dimensional space to dive into the source code or architecture of a software (e.g., see the visualization of source code in 3D [18] or software cities in VR [19]). A cheap and convenient method to use VR are headsets such as Oculus Rift, Samsung Gear VR, or Google Cardboard. The availability of these devices could allow software engineers to use VR visualization on their desk. We implemented a VR version of the visualization of key bundles (Fig. 5) for Google Cardboard using the 3D game engine Unity [20]. The preliminary version allows to fly through the three dimensional space of bundles, represented as spheres (Fig. 9). The current interaction methods allows to select a bundle, which then become translucent. The sphere contains the packages of the bundle. While moving the head, users can point with a virtual cursor to any bundle or package, which then shows dependencies to other bundles.

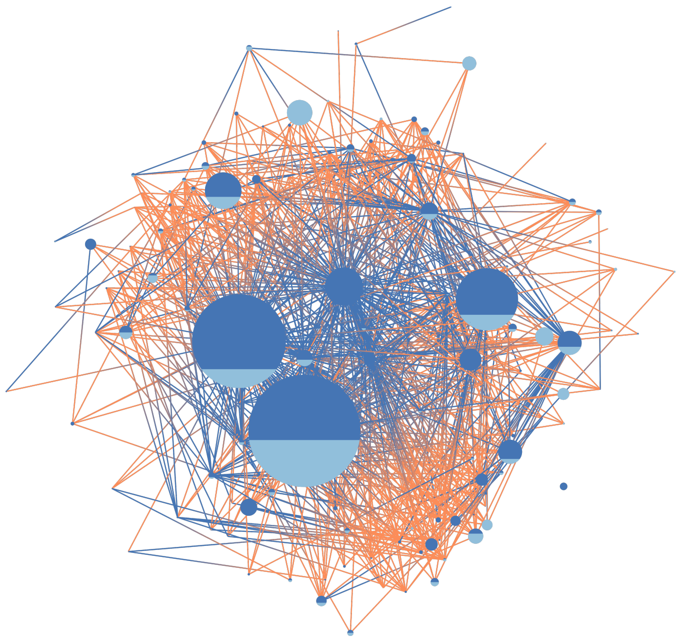


Fig. 5. Visualizing key bundles: largest ones in term of lines of code as well as notable number of private packages and incoming dependencies. Application: Remote Component Environment (RCE)

C. Use Cases for Visualization

To show the feasibility of our tools, we applied the visualization for three OSGi-based applications: *Memory Analyzer* (MAT) [13], *Open Systems Engineering Environment* (OSEE) [14], and *Remote Component Environment* (RCE) [15], [16]. MAT is of small size and has about 20 bundles. OSEE and RCE are of similar and larger size with over hundred of bundles and services.

1) *Getting an impression of the dimensions of the application.* For example, to give a project manager a clearer understanding of maintenance cost, it is helpful to give an impression of the size of the application. The dimension of an application can be described with

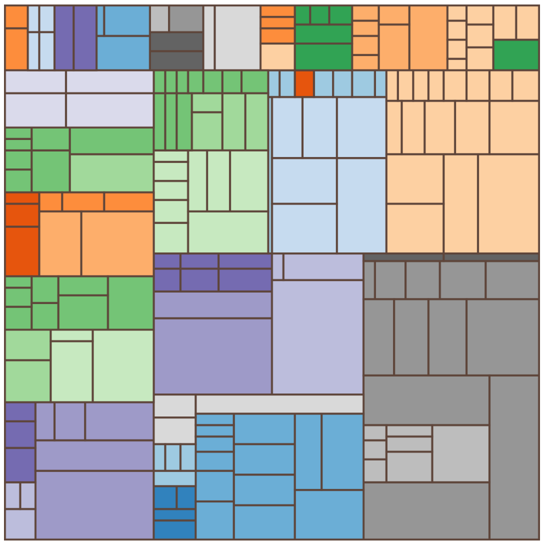


Fig. 6. Package structure of a bundle. Classes are assembled to packages by sharing the same color. The size of a class item is mapped to its lines of code. Application: Remote Component Environment (RCE)

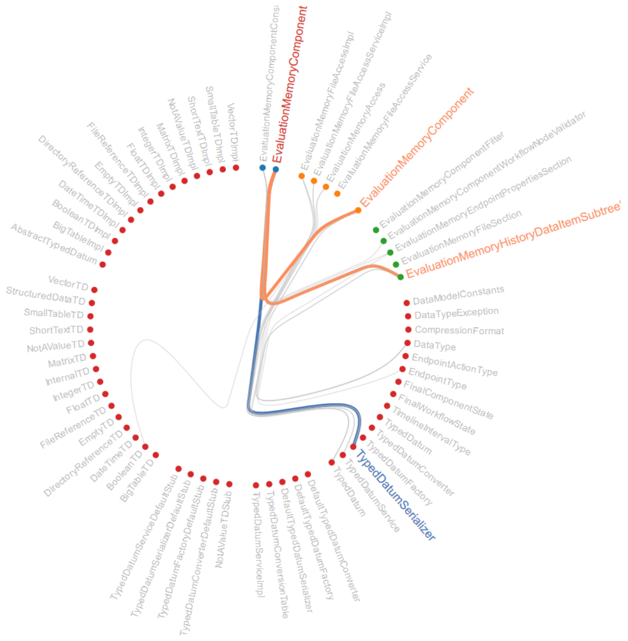


Fig. 7. Dependencies between two bundles broken down to classes. Only dependencies across bundles are shown; dependencies between and inside packages are hidden. Application: Remote Component Environment (RCE)

different parameters. For OSGi-based applications, the most important are: number of bundles and services as well as the size of the application and of the bundles (Fig. 4).

2) *Introducing a new member of the development team.* If a new developer joins the team it is helpful to give an overview of the software to develop. It involves the dimension of the project, key bundles, and basic structure that includes the semantic modularization beyond bundles (Fig. 5 and Fig. 9).

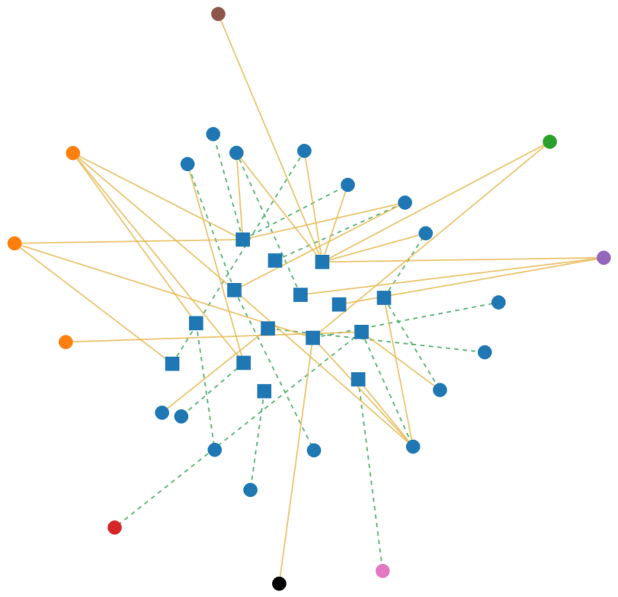


Fig. 8. Service components (rectangle) and services (circle) and the dependencies between them. Dotted edge if a service component uses the service and solid edge if a service component provides the service. Service components and services of one bundle share one color. Application: Remote Component Environment (RCE)

3) *Starting to work on a new module.* When starting to work on a new and unknown module, it is helpful to explore the overall structure in detail beforehand. For OSGi-based applications, it is of interest to know some characteristics of the affected bundles such as the size or the ratio between public and private packages. It is also good to know some insights of the bundles such as their package and class structure. To classify the bundles of the application, the number of dependencies are of interest including their direction and strength. Also, the services are noteworthy that are provided and used both within and in interaction with other bundles (Fig. 6, Fig. 7, and Fig. 8).

4) *Checking for abnormalities.* The actual architecture of a software does not always follow the intended design for different reasons. It must be continuously reviewed for abnormalities and adjusted if needed (Fig. 4, Fig. 6, Fig. 5, and Fig. 7).

V. Conclusion and Future Work

We presented tools to visualize the modularization of OSGi-based applications on different abstraction levels. We introduced an interactive, browser-based visualization and a virtual reality tool for VR headsets. Use cases demonstrated the usefulness of the visualization to comprehend the modularization of OSGi-based applications. The bundle graph with the option to show a bundle's package structure as a treemap and the circular net displaying dependencies on class level suit well. Compared to that, the benefit of the service graph is limited. It provides a good impression about the degree of service

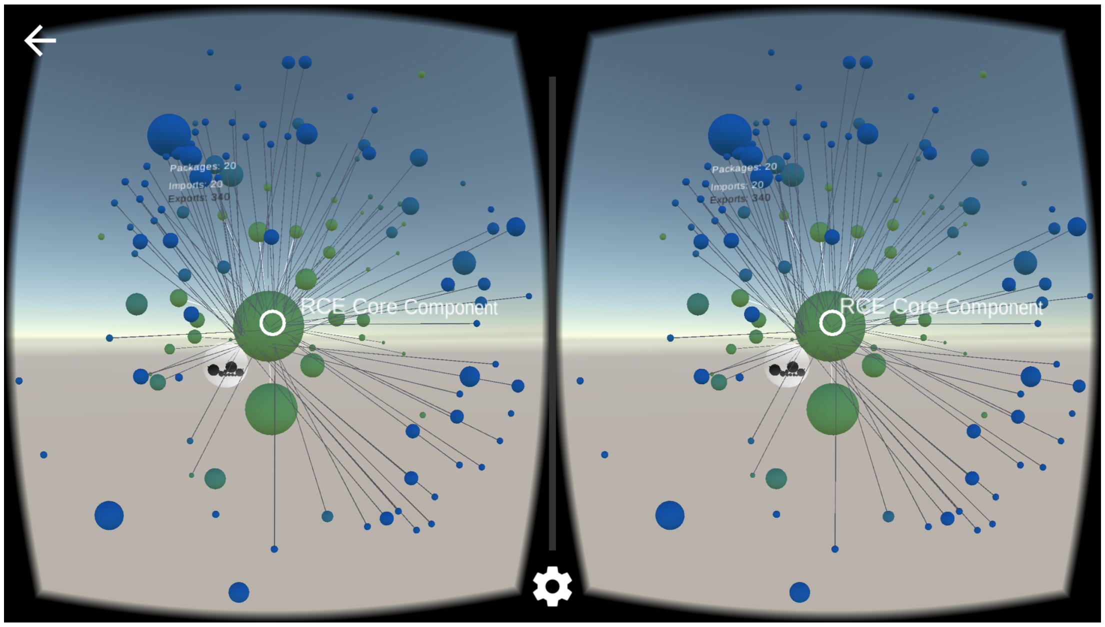


Fig. 9. VR version of the OSGi viewer. This preliminary version as available as a test version for Android and Google Cardboard in the Google Play Store: https://play.google.com/apps/testing/de.dlr.sc.OSGiViewer

usage, but lacks in supporting the comprehension of their dependencies. An alternative for the force-directed graph that was chosen should be found.

We already identified topics for future work. The metamodel could be extended with information on instruction level to add one more abstraction level in the visualization beyond classes. The treemap showing the package structure could be integrated with the dependencies. If a bundle is expanded, the edges that represents the dependencies would start and end at the classes in the treemap. Finding abnormalities by hand is not reliable. An application should be analyzed for abnormalities automatically and the ones found should be provided in the visualization. The challenge here is to define what abnormalities are and that a definition might differ from project to project.

References

[1] M. Petre and E. de Quincey, "A gentle overview of software visualisation," *Computer Society of India Communications*, pp. 6–11, Aug. 2006.

[2] Y. Ghanam and S. Carpendale, "A survey paper on software architecture visualization," University of Calgary, Tech. Rep., Jun. 2008. [Online]. Available: http://hdl.handle.net/1880/46648

[3] N. Hawes, S. Marshall, and C. Anslow, "Codesurveyor: Mapping large-scale software to aid in code comprehension," in *Software Visualization (VISSOFT), 2015 IEEE 3rd Working Conference on*, Sept 2015, pp. 96–105.

[4] A. L. Tavares and M. T. Valente, "A gentle introduction to osgi," *SIGSOFT Softw. Eng. Notes*, vol. 33, no. 5, pp. 8:1–8:5, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402521.1402526

[5] The Apache Software Foundation, "Apache felix," 2015. [Online]. Available: https://felix.apache.org

[6] The Eclipse Foundation, "Eclipse equinox," 2016. [Online]. Available: http://www.eclipse.org/equinox/

[7] The Knopflerfish Project, "Knopflerfish – open source osgi sdk and run-time container," 2015. [Online]. Available: http://www.knopflerfish.org/

[8] OSGi Alliance. Osgi service platform specification. [Online]. Available: https://www.osgi.org/developer/specifications/

[9] S. Diehl, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*, 1st ed. Springer-Verlag Berlin Heidelberg, 2007.

[10] M. O. Ward, G. Grinstein, and D. Keim, *Interactive Data Visualization: Foundations, Techniques, and Applications*. A K Peters/CRC Press, 2010.

[11] J. S. Yi, Y. ah Kang, J. Stasko, and J. Jacko, "Toward a deeper understanding of the role of interaction in information visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1224–1231, 2007. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TVCG.2007.70605

[12] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-driven documents," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011. [Online]. Available: http://vis.stanford.edu/papers/d3

[13] Eclipse, "Memory analyzer (mat)," 2015. [Online]. Available: https://www.eclipse.org/mat/

[14] "The open system engineering environment." [Online]. Available: http://www.eclipse.org/osee/

[15] D. Seider, P. M. Fischer, M. Litz, A. Schreiber, and A. Gerndt, "Open source software framework for applications in aeronautics and space," *2012 IEEE Aerospace Conference*, pp. 1–11, 2012. [Online]. Available: http://dx.doi.org/10.1109/AERO.2012.6187340

[16] German Aerospace Center (DLR), "Remote component environment (RCE)," http://rcenvironment.de/. [Online]. Available: http://rcenvironment.de/

[17] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TVCG.2006.147

[18] A. Marcus, L. Feng, and J. I. Maletic, "3d representations for software visualization," in *Proceedings of the 2003 ACM Symposium on Software Visualization*, ser. SoftVis '03. New York, NY, USA: ACM, 2003, pp. 27–ff. [Online]. Available: http://doi.acm.org/10.1145/774833.774837

[19] F. Fittkau, A. Krause, and W. Hasselbring, "Exploring software cities in virtual reality," in *2015 Third IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2015, pp. 130–134.

[20] Unity Technologies, "Unity," 2016. [Online]. Available: http://unity3d.com/