Interactive Visualization of Software Components with Virtual Reality Headsets

(preprint)

Andreas Schreiber
Intelligent and Distributed Systems
German Aerospace Center (DLR)
Köln, Germany
Email: andreas.schreiber@dlr.de

Marlene Brüggemann
Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences
Berlin, Germany
Email: marlenebr@outlook.de

***Abstract*—In large software projects, it can be hard to understand the actual architecture of the implemented software systems if current design documents are outdated or do not exist at all. For systems based on the OSGi component specification, which are used to build quite large applications with back-end and front-end services, tools for visualizing the actual architecture can help to understand the system. We provide an interactive tool that visualizes OSGi-based systems with their components, packages, services, and dependencies in 3D using Virtual Reality headsets. It uses the metaphor of modular electronic systems. The tool allows software engineers, project managers, or clients to explore the architecture and get a first impression about component sizes and their dependencies.**

I. Introduction

Visualization of the actual structure of a software system is important to get some insight into it. Many approaches exist for various aspects of the software and its architecture, both in 2D and 3D [1]. Software visualization in 3D is able to provide more details compared to 2D visualization, but navigation through the visualization and interaction with visual objects can be harder. Using Virtual Reality (VR) can lead to an improvement because it naturally extends the human environment.

In recent years, the availability of high-quality VR headsets as consumer devices with easy-to-use SDKs allows to explore VR approaches for software visualization without overwhelming effort. Also with cheap VR headsets, it would be possible to provide many if not all developers, project manager, or customers with VR headsets on their workplace. These headsets allows them to use VR visualization as a standard tools as part of their development or test environment.

To explore the potential of VR headsets for visualization of component-based software architecture, we developed a tool for visualizing OSGi-based applications. Based on previous visualizations and a tool chain for 2D [2], we present the following contributions:

- A visual concept based on modular electronic systems for OSGi components and dependencies (Section II).
- Description of the interaction patterns for navigation and selection (Section III).
- Details and visual impressions of a tool implementation for VR headsets (Section IV).

II. Visual Concept

To represent modules of OSGi-based applications, we used the basic visual metaphor of modular electrical component systems. Single components of those systems are replaceable, they can be connected to each other, and have often control elements (e.g., sliders or switches). In practice, electrical control panels or satellite on-board systems are designed using such a modular layout. For our software systems, we chose to represent software modules as simple as possible to not distract users.

A. Representation of Bundles and Packages

Each OSGi *bundle* is represented as a cubical object. Also each *package* is represented as a cubical object, which has a different color and size as the bundle object. It should be possible to recognize that packages are parts of bundles, but packages should be placed inside of bundles. Instead, packages are stacked over each other and placed on top of bundles (Figure 1). The intention using this representation, is to illustrate which packages belong to a bundle. The height of this stack allows to visually recognize the size of a bundle compared to neighboring bundles.

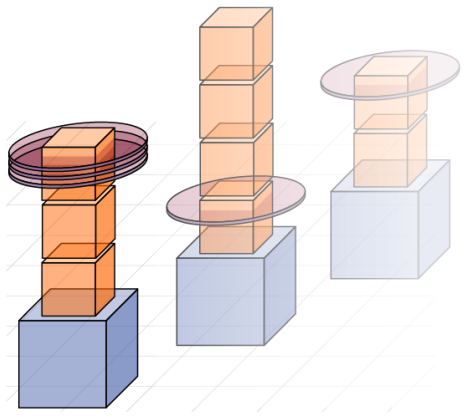

Fig. 1. Visual representation of *bundles* (blue), its *packages* (orange), and *services* that are provided by a package (red).

B. Representation of Services

OSGi *services*, which are provided by a package, are represented as disc objects. They are placed in a package with a larger size to make them recognizable. If a package provides one service only, it is placed at the upper edge of the package object. For more than one service, the discs are "stacked" down under each other with a small gap to make them distinguishable (Figure 1).

C. Representation of Classes

Each *class* of a package is represented as a cylinder-shaped object. The size of these cylinders are all the same, independent of class sizes. The class cylinders are displayed only of a selected package. They are placed spirally around the stack of packages, starting from the selected package (Figure 2). The first cylinder is placed within the selected package and each following cylinder is placed spirally with slightly increased radius and angle. A maximum radius ensures, that classes of packages from different bundles cannot touch each other.

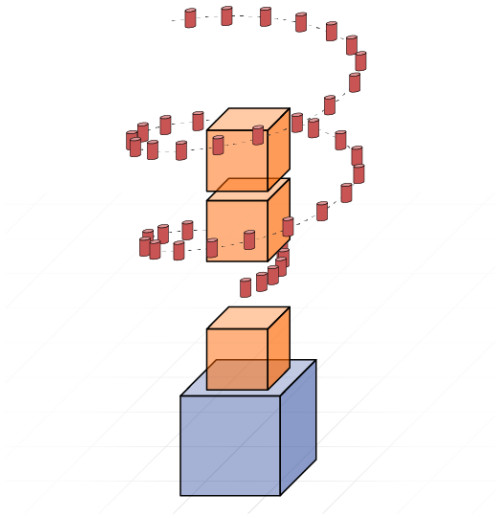

Fig. 2. Representation and positioning of *classes* (red).

The name of each class is displayed a half-transparent text next to the cylinder objects. With selection of a class the name will be drawn without transparency.

D. Dependencies between Bundles and Packages

The dependencies between packages and bundles are visualized using a node-link-representation. Dependencies are *imports* and *exports* of packages, which are represented by lines with distinct colors.

If the visualization of package objects is activated, the dependencies are displayed as follows (Figure 3):

- **Imports**: A bundle is connected to all packages that is imports.
- **Exports**: Each package that is exported by a bundle is connected to all bundles, which import them.

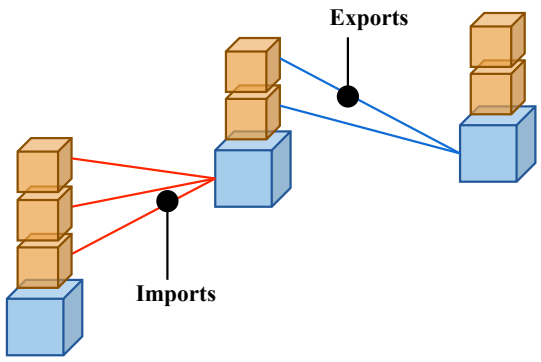

Fig. 3. Dependencies (imports and exports) between bundles and packages.

E. Arrangement of Bundles

The base scene should show OSGi bundles, which are placed at different locations within the three-dimensional space. The most basic placement is with random positions on x-axis and z-axis. Since for larger software systems with many bundles this becomes confusing, we provide other layout strategies based on clustering. Clustering of bundles is realized by grouping them and placing them with certain layouts.

For *grouping* of bundles, we decided to use the symbolic name of each bundle. We assume that each symbolic name uses the reverse domain convention, which is recommended by the OSGi core specification [3]. An optional settings parameter determines to which depth of the reversed domain name the grouping will occur. For example, it's possible to just group according to the top-level domain ("de," "com," "org," etc.)

For *placement* of bundles, we implemented two different layouts. An *arrangement in a quadratic field*, which places bundles quite near to each other, and an *arrangement in staggered rows*, which gives a more spacious layout.

1) Arrangement in quadratic field: All clustered groups of bundles are placed within a quadratic field in the x-z-plane. The quadratic field is divided into cuboids (Figure 4). The length of a cuboid along the x-axis is fixed and equal to the length of the quadratic field. The length of a cuboid along the z-axis is divided; it is proportional to the size of the respecting bundle cluster, which allows to oversee the proportions of cluster sizes.

Within a cluster cuboid, bundles are sorted along the z-axis according to the number of packages they contain. Bundles with fewer packages are placed more to the front and bundles with more packages are placed to the end.

2) Arrangement in staggered rows: Each clustered group of bundles is placed along the x-axis. The distance between the different groups as well as the distance between each bundle along the z-axis is equidistant, which leads to rows of bundles. Each of these rows is places along a curve within the x-z-plane (Figure 5), which allows that from a standard front view as much as possible of the bundles are visible. We use the following curve for placement in the x-z-plane:

$$x = z^2/160 \tag{1}$$

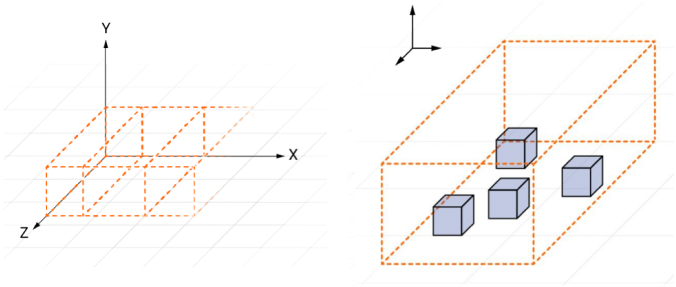

Fig. 4. Clustered groups (left) and bundles (right) arranged in a quadratic field.

Within each row, bundles a placed equidistant along the x-axis. The position along the y-axis is also defined by a curve. This leads to a better visibility if a user views the scene from a standard front position. We use the following curve for placement in the y-z-planes:

$$y = -z^2/360 \tag{2}$$

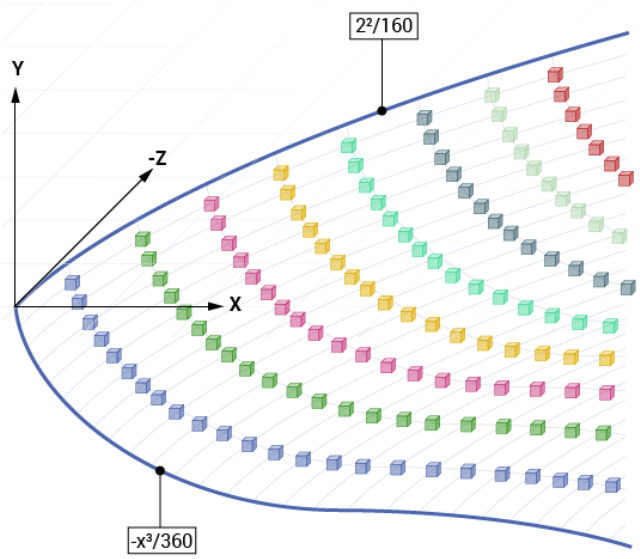

Fig. 5. Bundles and cluster arranged in staggered rows.

In both placement layouts, we choose to use distinct colors for each clustered group of bundles.

III. Navigation and Interaction

The basic actions are triggered using a button of the VR headset. Within the visualization, the user has the possibility to select certain elements using a hairline cross or a small circle, which we use as a *pointer*. Pointing to an element for a certain time selects that element.

Our approach does not use gestures or handheld devices for interaction since we wanted to support VR headset without any sensors for gesture detection and without handheld controllers. Navigation ("flying") through the scene is supported by selecting a "Move" mode (see Section III-A2) and then steering the movement by motion of the head.

A. GUIs for Navigation and Interaction

In the visualization, several GUIs are displayed, depending on the current status or selected element.

1) Main GUI: The settings of this GUI change the representation of bundles as well as their placements. It contains:

- Switching displaying of packages on or off.
- Switch between the two arrangement layouts (Section II-E) for bundles.
- Selecting a "top-view" to see the scene from the top.

2) Navigation GUI: The navigation GUI (or "Move GUI"; see Figure 6) allows users to control movement within the scene or to rotate the whole scene.

- The button "Move in scene" activates movement mode where the viewer is able to move within the scene. The direction of movement changes with the viewing direction, which allows to fly through the scene by movement of the head.
- The button "Rotate objects" allows to rotate the whole scene. A three-dimensional rotation cross allows to select the axis on which the scene rotates.

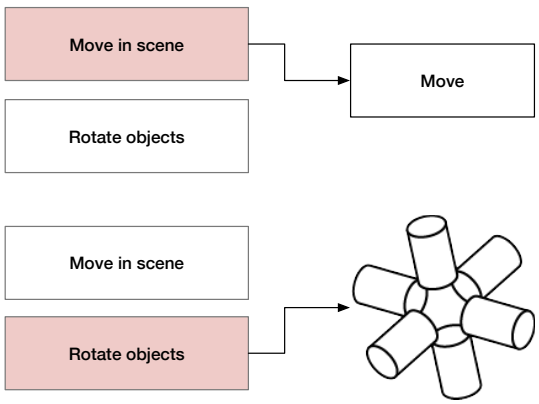

Fig. 6. Navigation GUI.

3) Bundle GUI: The bundle GUI allow users to interact with single bundles (Figure 9). This GUI is activated after selecting a bundle. It contains:

- Switch display of packages on or off.
- Activate or deactivate display of imports and exports from/to other bundles.

4) Settings GUI: The settings that can be activated with this GUI does not affect the architecture visualization itself. It contains:

- Activation or deactivation of the VR mode.
- Move view to a standard default position.

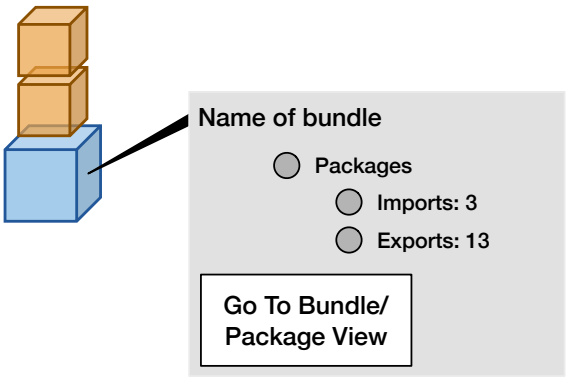

Fig. 7. Bundle GUI.

IV. Tool Implementation

The implementation of the described visual concepts is possible with several technologies, such as OpenGL or more high level frameworks. We decided to implement our tool based on the 3D game engine Unity,[1] because effort for development is quite low. Using Unity, we are able to generate the tool for a number of target platform (i.e., headsets), such as Oculus Rift, Gear VR, Playstation VR, Microsoft HoloLens, HTC Vive, and Google Cardboard & Daydream. We tested our implementation on Oculus Rift and Google Cardboard. Especially, support for cheap VR solutions, such as Cardboard, is very good with an SDK provided by Google [4]. This allows to deploy the tool via the Google Play Store.[2]

A. Data Source

The data that is visualized is extracted from the source code tree with an existing tool [2]. It reads information about OSGi components from the following sources:

- *General program structure* is retrieved from the directory hierarchy, since OSGi has conventions about the expected directory structure.
- *Information about OSGi bundles*, its imported bundles, the packages it provides (exports), and its metadata is read from the Manifest file.
- *Information about services* are read from XML files with the directory "OSGI-INF."

B. Visual Impressions

All of the visual concepts from Section II are implemented in the tool. In the following, some example screen shots from the regular 3D mode give an impression of the visualization and interaction controls. The screen shots shows the OSGi components of the Open Source framework RCE [5].

[1] http://unity3d.com/

[2] Dependency Diver — A Virtual Reality app for Google Cardboard for software modules and dependencies. https://play.google.com/store/apps/details?id=de.dlr.sc.DependencyDiver

1) Visualization of bundles, packages, and services: The general scene that is visible after starting the tool (Figure 8) shows bundles (large cubical objects), packages (small cubical objects) stacked over each other on top of bundles, and services (discs around packages). A white circle represents the pointer, which helps to select bundles or packages.

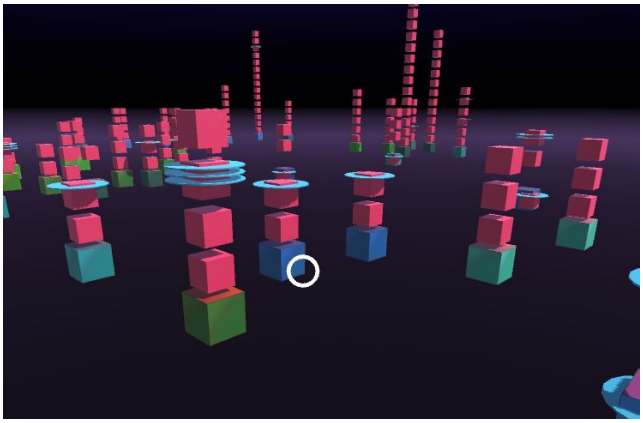

Fig. 8. General scene that shows bundles (large cubical objects), packages (small cubical objects), and services (discs around packages).

2) Visualization of Bundle GUI: After selecting a bundle, the bundle GUI appears, where the user see the bundle name as well as the number of imports and exports (Figure 9). Using this GUI, the user can select to display dependencies between bundles and packages.

Fig. 9. Bundle GUI that shows name of the bundle, number of import, and number of exports.

3) Visualization of Imports: For a bundle, all imports can be displayed. They are represented by thin lines from the bundle to each of the imported packages (Figure 10).

4) Visualization of Exports: For a bundle, all exported packages to other bundles are displayed. A thin lines connects the exported packages to the bundle that imports this package (Figure 11).

5) Rotation: To rotate the whole scene, the user can activate a rotation cross consisting of cylinders that represents coordinate axes (Figure 12). It allows to rotate around all three axes, both clockwise and counterclockwise, by pointing on one of the six cylinders.

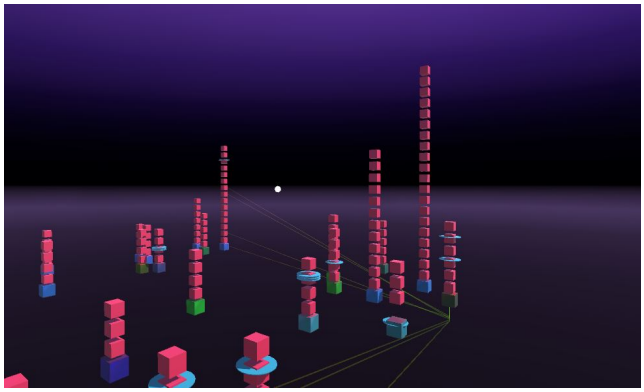

Fig. 10. Imported packages of a bundle.

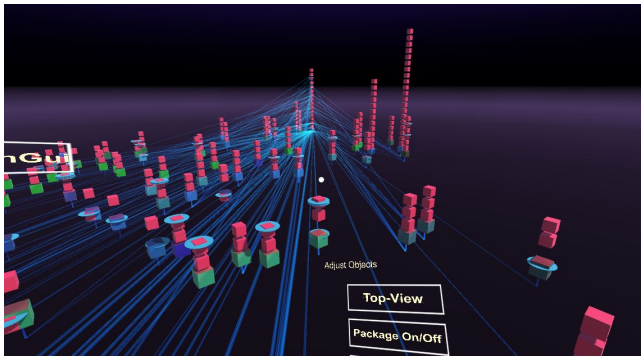

Fig. 11. Export dependencies of a bundle.

V. Related Work

The survey by Ghanman and Carpendale [6] gives an overall overview about visualizing software architectures. Müller and Zeckzer [7] conducted a literature analysis, which outlines the state-of-the-art, shows trends, and tries to identify research gaps especially for 3D software visualization.

Many 3D visualization tools and approaches for software visualization use the *city metaphor* [8], such as DAHLIA for database usage [9] or EXLORVIZ [10] for software landscapes and applications. Especially, EXLORVIZ visualizes software cities in VR [11]. Software cities provide a compact visualization of the hierarchical information of object-oriented software applications while our approach is intended to visualize the software in the component and service level, which is not necessarily hierarchical.

VI. Conclusions and Future Work

We developed and presented a tool to visualize the architecture of OSGi-based applications with VR headsets. The basic visual metaphor is taken from the design of modular electric components. Our visualization displays a scene where OSGi bundles are represented as blocks and their containing packages are stacked up on that blocks. Dependencies (import and exports) are represents as lines. The visualization was tested with Oculus Rift headsets and Google Cardboard on an Android phone. The tool allow users to "fly" through the

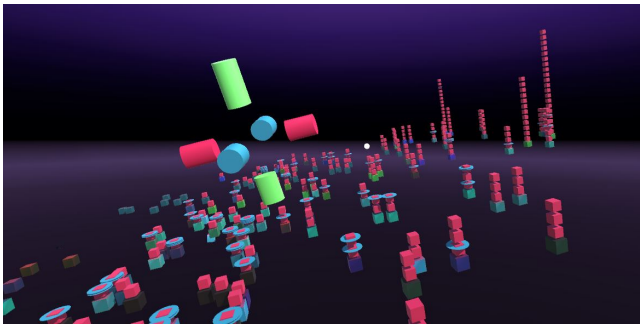

Fig. 12. Visual rotation cross.

bundles and package stacks which gives an impression about bundles, their sizes, and their clustering. For bundles and packages, the user can interactively explore further information, such as dependencies. The tool is intended to give a first impression about the components of the software but the visual concept can have deeper levels of information, if available.

Future work will foremost focus on conducting a user study for a systematic evaluation of the visualization. For the user study, we plan to use an HTC Vive with eye tracking. Based on the user study result we will derive directions for further improvements. Furthermore, we will evaluate whether the visual concept might be suitable for Augmented Reality (AR) headsets, such as Microsoft's *HoloLens*.

References

[1] S. Diehl, *Past, Present, and Future of and in Software Visualization*. Cham: Springer International Publishing, 2015, pp. 3–11. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25117-2_1

[2] D. Seider, A. Schreiber, T. Marquardt, and M. Brüggemann, "Visualizing modules and dependencies of OSGi-based applications," in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, Oct 2016, pp. 96–100.

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

[4] Google, "Cardboard sdk for unity." [Online]. Available: https://developers.google.com/cardboard/unity/

[5] 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

[6] 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

[7] R. Müller and D. Zeckzer, "Past, present, and future of 3d software visualization - a systematic literature analysis," in *Proceedings of the 6th International Conference on Information Visualization Theory and Applications (VISIGRAPP 2015)*, 2015, pp. 63–74.

[8] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, June 2007, pp. 92–99.

[9] L. Meurice and A. Cleve, "Dahlia 2.0: A visual analyzer of database usage in dynamic and heterogeneous systems," in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, Oct 2016, pp. 76–80.

[10] F. Fittkau, A. Krause, and W. Hasselbring, "Software landscape and application visualization for system comprehension with explorviz," *Information and Software Technology*, vol. 87, pp. 259 – 277, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584916301185

[11] ——, "Exploring software cities in virtual reality," in *2015 Third IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2015, pp. 130–134.