Visualization of Software Components and Dependency Graphs in Virtual Reality

Lisa Nafeie
German Aerospace Center (DLR)
Köln, Germany
lisa.nafeie@dlr.de

Andreas Schreiber
German Aerospace Center (DLR)
Köln, Germany
andreas.schreiber@dlr.de

ABSTRACT

We present the visualization of component-based software architectures in Virtual Reality (VR) to understand complex software systems. We describe how to get all relevant data for the visualization by data mining on the whole source tree and on source code level. The data is stored in a graph database for further analysis and visualization. The software visualization uses an island metaphor. Storing the data in a graph database allows to easily query for different aspects of the software architecture.

CCS CONCEPTS

• **Human-centered computing** → **Virtual reality**; **Graph drawings**; • **Information systems** → Data mining;

KEYWORDS

Software architecture, 3D visualization, OSGi, real-world metaphor, virtual reality, graph database

ACM Reference Format:
Lisa Nafeie and Andreas Schreiber. 2018. Visualization of Software Components and, Dependency Graphs in Virtual Reality. In *VRST 2018: 24th ACM Symposium on Virtual Reality Software and Technology (VRST '18), November 28-December 1, 2018, Tokyo, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3281505.3281602

1 INTRODUCTION

For visualizing software, many approaches exist [3] (e.g., *Unified Modeling Language* (UML) diagrams). Especially, techniques from graph visualization are suitable for visualizing software dependencies [1]. A common software visualization approach in 3D are *software cities* [8], which are also available as implementations for Virtual Reality (VR) or Augmented Reality (AR).

To get an high level overview of large component-based software systems classic software architecture visualizations such as UML diagrams might be too large or complex. Our software visualization uses the highest abstraction level, which deals with the entirety of the software architecture [2]. It conveys the underlying hierarchical component structure, the relationships between these components, and the visual representation of code quality metrics. The data is acquired from source code repositories and stored in a graph database. We describe the two essential parts of our architecture (Figure 1) as follows:

- Data mining of architecture information using Open Source software stack and graph database model (Section 2).
- Visualization of software components and dependencies in VR (Section 3).

2 DATA MINING

We analyze Java projects that are based on the OSGi framework. This framework modularize and manage software projects and their services. OSGi projects includes *bundles*. Each bundle is a `JAR` archive with a `MANIFEST.MF` file, which describes different informations such as *dependencies* and *services*.

We used the open source quality assurance tool jQAssistant as a basis for acquisition of all relevant data from the source code repository. jQAssistant relies on the graph database Neo4j. jQAssistant scans and analyzes software projects [5] as follows:

- **Scanning**: `jQAssistant` scans software artifacts either by using the Maven plug-in or the command line tool. We use command line tool to scan the source tree, which includes archives of all compiled software fragments. We focus on scanning projects that are based on the OSGi framework, where XML files in `OSGI-INF` directories are important for declarative service dependencies. Therefore we use the jQAssistant XML plug-in for all XML files at a specific directory (configuration file `scan.properties`) and activated scans for `pom.xml` files only.
- **Analyzing**: After scanning the project, jQAssistant analyzes the data with plug-ins, which are implemented using the Cypher graph query language. jQAssistant already provides different plug-ins, for example, for JSON, XML, or OSGi. We extended the OSGi plug-in and added additional rules

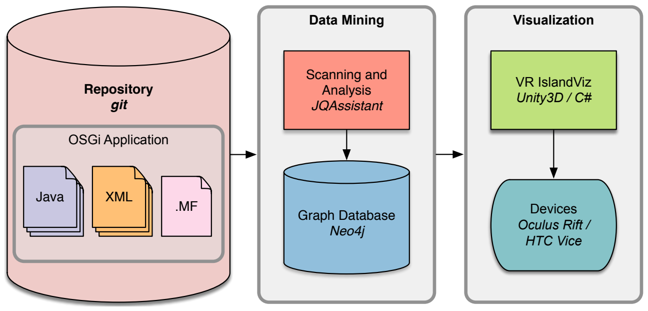


Figure 1: Architecture of the software stack from repository mining to virtual reality visualization.

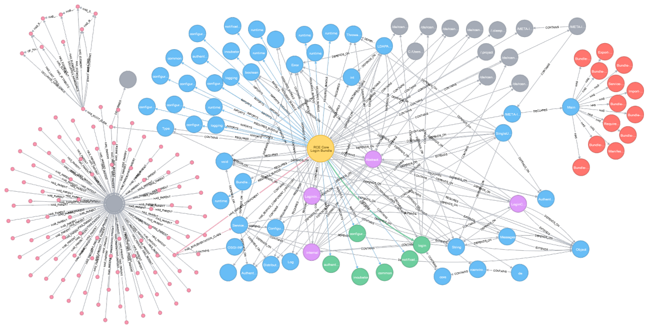


Figure 2: Visualization of the graph data for one single OSGi bundle ("`RCE Core Login Bundle`") and all dependent nodes.

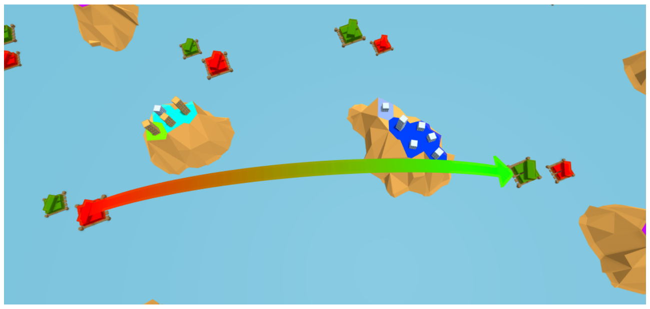


Figure 3: Visualization of software architecture as islands (components; bundles), regions (packages), and buildings (classes). Package dependencies are visualized via arced arrows.

for OSGi structures, which creates nodes and relationships for OSGi elements in the graph database, and for analyzing if the scanned projects violates certain software structures.

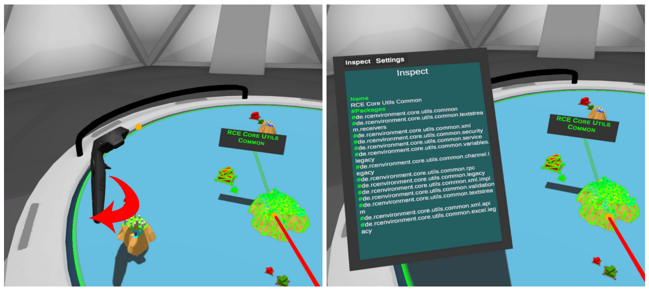
After scanning the repository and applying all rules the software architecture information with all relevant elements of OSGi is stored in the graph database (Figure 2).

3 VISUALIZATION OF SOFTWARE ARCHITECTURE IN VIRTUAL REALITY

We use the *island metaphor* [4] for visualization of software architectures. Islands on a virtual water level represents components, regions on the islands represents packages, buildings represents classes, and arced arrows represent dependencies (Figure 3). A fixed virtual table contains the virtual water level and a virtual tablet allows to present additional information on selected elements to the user (Figure 4).

The original implementation *IslandViz*[1] [6] of the VR visualization uses Unity3D and C# and reads the data from a JSON file [7]. We extended this implementation to use data directly from the Neo4j graph database. The following Cypher query is the most straight forward approach to collect all data that is then rendered in VR:

[1]https://github.com/DLR-SC/island-viz



Inspect
Name
RCE Core Utils Common
#Packages
#de.rcenvironment.core.utils.common
#de.rcenvironment.core.utils.common.textstream.receivers
#de.rcenvironment.core.utils.common.xml
#de.rcenvironment.core.utils.common.security
#de.rcenvironment.core.utils.common.service
#de.rcenvironment.core.utils.common.variables.legacy
#de.rcenvironment.core.utils.common.channel.legacy
#de.rcenvironment.core.utils.common.rpc
#de.rcenvironment.core.utils.common.legacy
#de.rcenvironment.core.utils.common.xml.impl
#de.rcenvironment.core.utils.common.validation
#de.rcenvironment.core.utils.common.textstream
#de.rcenvironment.core.utils.common.xml.api
#de.rcenvironment.core.utils.common.excel.legacy

Figure 4: Virtual table as environment for the virtual water level and virtual tablet for displaying information on selected elements.

```
MATCH
    (pf:PackageFragment)-[c:CONTAINS]->
        (cu:CompilationUnit),
    ()-[e:EXPORTS]->(),
    ()-[i:IMPORTS]->(),
    ()-[hs:HAS_SERVICE_COMPONENT]->(),
    ()-[hi:HAS_IMPLEMENTATION_CLASS]->(),
    ()-[ps:PROVIDES_SERVICE]->(),
    ()-[hfh:HAS_FRAGMENT_HOST]->(),
    ()-[hba:HAS_BUNDLE_ACTIVATOR]->(),
    ()-[rb:REQUIRES_BUNDLE]->()
RETURN pf, c, cu, e, i, hs, hi, ps, hfh, hba, rb
```

4 CONCLUSION AND FUTURE WORK

The presented software visualization makes the software architecture more tangible, which allows heterogeneous teams to talk about technical problems.

Future work include support for multiple programming languages and programming models, more devices (e.g., AR devices), and more interaction models to specify graph queries.

REFERENCES

[1] A. Bergel, S. Maass, S. Ducasse, and T. Girba. 2014. A Domain-Specific Language for Visualizing Software Dependencies as a Graph. In *2014 Second IEEE Working Conference on Software Visualization*. 45–49.
[2] Pierre Caserta and Olivier Zendra. 2011. Visualization of the Static Aspects of Software: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 17, 7 (July 2011), 913–933.
[3] Stephan Diehl. 2007. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software* (1 ed.). Springer-Verlag Berlin Heidelberg.
[4] Martin Misiak, Doreen Seider, Sascha Zur, Arnulph Fuhrmann, and Andreas Schreiber. 2018. Immersive Exploration of OSGi-based Software Systems in Virtual Reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*.
[5] Richard Müller, Dirk Mahler, Michael Hunger, Jens Nerche, and Markus Harrer. 2018. Towards an Open Source Stack to Create a Unified Data Source for Software Analysis and Visualization. In *The Sixth IEEE Working Conference on Software Visualization (VISSOFT 2018)*. IEEE, 107–111.
[6] Andreas Schreiber and Martin Misiak. 2018. Visualizing Software Architectures in Virtual Reality with an Island Metaphor. In *Virtual, Augmented and Mixed Reality: Interaction, Navigation, Visualization, Embodiment, and Simulation*, Jessie Y.C. Chen and Gino Fragomeni (Eds.). Springer International Publishing, Cham, 168–182.
[7] Doreen Seider, Andreas Schreiber, Tobias Marquardt, and Marlene Brüggemann. 2016. Visualizing Modules and Dependencies of OSGi-Based Applications. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. 96–100.
[8] Richard Wettel and Michele Lanza. 2007. Visualizing Software Systems as Cities. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. 92–99.