

Towards Visualization of Evolution of Component-Based Software Architectures in VR

Elke Franziska Heidmann
Annika Meinecke
heidmann.elke@t-online.de
annika.meinecke@dlr.de
German Aerospace Center (DLR)
Intelligent and Distributed Systems
Cologne, Germany

Lynn von Kurnatowski
lynn.kurnatowski@dlr.de
German Aerospace Center (DLR)
Intelligent and Distributed Systems
Oberpfaffenhofen, Germany

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

ABSTRACT

To understand the development history of complex software architectures, software visualizations are very useful. They show dependencies and contexts in which design decisions were made, supporting programmers in understanding systems and helping them to recognize disadvantageous design decisions. IslandViz visualizes OSGi-based software architectures in Virtual Reality using an island metaphor; for now, the history of an architecture is not taken into account. We show how IslandViz can be extended to include changes on package and compilation unit level by changing the layout of islands. For this purpose we adapt the Enhanced Hexagon Tiling Algorithm and create growth corridors for each region of an island. As a result, island regions can change along with their respective packages without the possibility of being enclosed by other regions.

CCS CONCEPTS

• **Human-centered computing** → **Visualization theory, concepts and paradigms**; **Information visualization**; *Virtual reality*.

KEYWORDS

software visualization, software architecture, software evolution, history of software, virtual reality

ACM Reference Format:

Elke Franziska Heidmann, Annika Meinecke, Lynn von Kurnatowski, and Andreas Schreiber. 2020. Towards Visualization of Evolution of Component-Based Software Architectures in VR. In *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming (<Programming'20> Companion)*, March 23–26, 2020, Porto, Portugal. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3397537.3398473>

1 INTRODUCTION

The complexity of software architecture grows with its evolution. As a result, comprehending modules of a software with all their dependencies gets increasingly difficult during project development.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

<Programming'20> Companion, March 23–26, 2020, Porto, Portugal

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7507-8/20/03.

<https://doi.org/10.1145/3397537.3398473>

To support programmers in understanding software architecture—especially of large projects—the visualization of software has become a useful tool [9].

Different visualization techniques have been used for this purpose, in two dimensional visualizations as well as in three dimensional ones [1, 8]. Furthermore, 3D representations can be enhanced by using *Virtual Reality* (VR). In VR, navigational problems are reduced and additional information can be introduced using stereoscopic depth cues [7]. Especially in 3D representations of software architecture, the usage of real-world metaphors, such as the city metaphor or the solar system metaphor, is very popular and makes software architecture more accessible to the user [10].

Software visualizations focus mainly on a specific point in time to represent software architecture, not taking into account its history. However, rebuilding large software architectures step-by-step can give important insights. It helps in understanding all dependencies and gives context in which changes occurred. This makes it possible to recognize disadvantageous design decisions and to rectify them.

IslandViz [7, 12] visualizes the software architecture of a software system based on OSGi (Open Service Gateway Initiative) in VR, using the real-world metaphor of islands. Each OSGi bundle is represented by an island. A Java package inside a bundle is represented by a region on the island. Compilation units (i.e., classes) in one package are visualized as buildings in their respective region.

To include the history of a software architecture, it is necessary to consider how new, deleted, and changing bundles, packages and compilation units affect the visualization. For IslandViz we identified four main aspects that need to be considered. First, the database must be extended to include not only the current status of a project but its complete history as well [16]. In a second step, the visualization needs to draw changing island layouts due to changing packages, on which this paper focuses on. Third, the islands in IslandViz need to be able to drift according to their changing relationships. Lastly, the new extension needs to be made accessible for users and evaluated. In the remaining paper we present our contribution to this regard as follows:

- A brief overview on the current status of IslandViz with main focus on island layouts (Section 2).
- A concept on how to visualize changes in packages which affect the layout of an island (Section 3).
- A short description of the resulting visualization and interaction possibilities (Section 4).

2 CURRENT VERSION OF ISLANDVIZ

The IslandViz implementation [6] visualizes the architecture of an OSGi-based software project at a certain time. It uses an island metaphor representing the main hierarchy, organizing the project’s artifacts: The bundles of the project are displayed as islands forming an archipelago. The packages are regions on the islands. Compilation units are represented as buildings placed in the region of the package they belong to. The islands are located on the ocean’s surface relative to each other representing their package import/export relationships. OSGi services are displayed as a network of connections above the archipelago.

Our extension of IslandViz aims to visualize not only one status of software architecture but also its evolution. This means to extend the program to include the handling of appearing and disappearing islands, regions and buildings as bundles, packages or files are created or deleted.

2.1 Island Layouts in IslandViz

The form of islands in IslandViz is defined by the *Enhanced Hexagon Tiling Algorithm* (EHTA) [17]. The EHTA assigns to each region a number of cells in a hexagonal grid. The number of cells which are assigned to one region is proportional to the number of compilation units in each package.

Starting at a random cell in a grid, the EHTA distributes each new cell to a region as follows:

- The next cell assigned to a region is a free cell, that is neighboring at least one already assigned cell.
- The probability of a free cell being selected next depends on the number of already assigned neighbors.

The probability $P(n)$ that a cell with n assigned neighboring cells is selected is proportional to a score s_n of a cell.

$$P(n) \sim s_n = b^n \quad (1)$$

The compactness of the resulting selection is defined by b .

Has one region reached the necessary number of cells, the EHTA starts building the next region bordering the first one, until every region has enough assigned cells. Backtracking ensures that no region is placed inside a too small space enclosed by cells of other regions. Different parameters b are used and packages are placed according to the number of their compilation units in decreasing order. As a result, the largest regions are placed in the middle of an island.

While the EHTA uses a hexagonal grid, the grid of IslandViz is a Voronoi diagram based on random points to create a more realistic map [11]. To include the history of a software architecture in IslandViz, we use a hexagonal grid.

3 CONCEPT: EVOLVING ISLAND LAYOUT

The size of a region in IslandViz depends on the number of buildings on it, which are representations of compilation units inside a package. Since this number changes during the life cycle of a software, the number of buildings in one region also changes. As a result, the size of the regions must adapt as well. Our layout concept ensures that each region has enough space to grow while not changing the general layout overly much. This supports the user’s

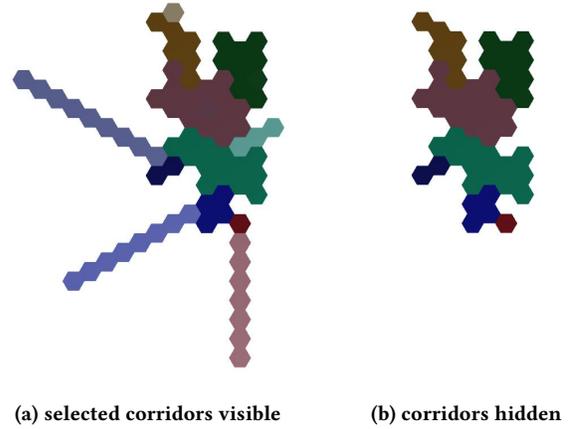


Figure 1: Island built using growth corridors.

ability to retain a mental map of the island layout throughout the development of a software architecture.

Our solution extends the in 2 described EHTA by adding conditions and parameters for the random selection of new cells to ensure that each region has coastal access. Our basic idea is to define absolute growth corridors for each newly created region (Figure 1). Cells lying on a growth corridor can only be assigned to its respective region. For each newly created region we define a growth corridor which does not overlap with any other region or their growth corridors. As a result, a new region cannot be placed in an already enclosed space, and a region can never be confined by its neighbors.

3.1 Absolute Growth Corridors

For each new region we create a growth corridor from a randomly selected starting cell. If a growth corridor overlaps with another region or growth corridor, another starting cell must be selected. The Probability $P_{start-opt}(n)$ that a border cell is checked for being a possible starting cell is:

$$P_{start-opt}(n) = b^6 - b^n \quad (2)$$

with n being the number of assigned neighbors and b being the same as in equation 1. Thus, regions are more likely to start at cells which are less surrounded by other regions.

3.2 Refinement

While our approach solves the problem of enclosed regions, island layouts can be unrealistic when regions start exactly between two growth corridors, forcing the region to only grow in its own corridor of a one cell width (Figure 2). Our solution is, to include the distance to the nearest growth corridors in our calculation. Our new formula is:

$$P_{start-opt}(n, step_r, step_l) = \begin{cases} 0.1 & \text{if } step_r = 0 \vee step_l = 0 \\ step_r * step_l * (b^6 - b^n) & \text{else} \end{cases} \quad (3)$$

$step_r$ and $step_l$ are the number of cells to the nearest growth corridors to the right and to the left side of a potential starting cell. Is a

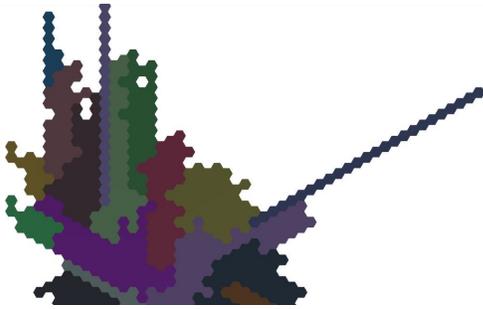


Figure 2: Example for regions with a width of only one cell

cell located next to an existing corridor ($step_r = 0 \vee step_l = 0$), the probability will have a low constant value. This ensures that a cell can be selected as starting cell if only free cells are located next to a growth corridor.

4 VISUALIZATION AND USER INTERACTION

In our visualization, we include time in form of the height of cells as an indicator of its age, resulting in islands that grow with the evolution of a software architecture. Since newer and thus lower compilation units are at the border of a region, and older ones are placed higher and more towards the center, over time islands will develop a cone shape. In our metaphor, islands can therefore be seen as volcano islands (Figure 3).

In Addition, newly created buildings are highlighted by green discs beneath the buildings. Deleted packages are represented as regions without buildings named 'Deleted Package' (Figure 3b). Regions of deleted packages cannot be included in other regions to support the user in retaining a mental map of IslandViz.

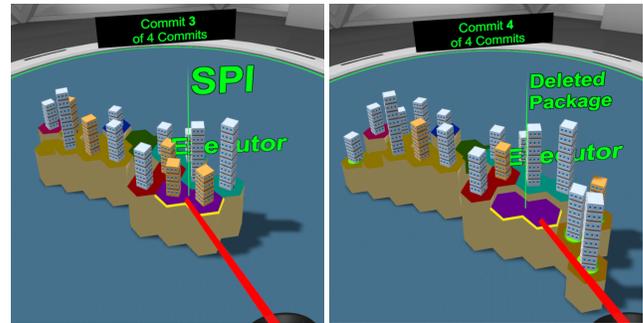
The user can navigate through software history either by going one single step forward or backward in software history, to analyze changes that occurred at a specific point in time. To get an impression on how the software architecture evolved over time, the user can also start a animation playing every commit sequentially.

We developed our visualization using Unity3D (v2019.3). The targeted HMD is HTC Vive.

5 RELATED WORK

The visualization of software architecture has become popular in recent years. However, most approaches concentrate on a specific status, not considering the history of a software. Nonetheless, there are different approaches on how to visualize the evolution of software. In “Evolution Matrix” [5], each class is represented by a box in a matrix. The width and height of each box is defined by two arbitrary metrics. The rows of the matrix contain one class of the software system, the columns show the different points in time. Looking only at the rows of the matrix displays the evolution of classes over time, while focusing on the columns gives an overview of the system at a specific point in time. A very similar approach to “Evolution Matrix” uses evolution spectographs [4] in which fields of a matrix are color-coded to represent different metrics.

A visualization of software evolution in a three dimensional space was presented in [3]. The software architecture was visualized



(a) Commit 3 of 4

(b) Commit 4 of 4

Figure 3: Visualization of changes between commits. Deleted packages are shown as regions without buildings. New cells are highlighted by green discs beneath buildings. Existing cells grow in height over time.

at specific points in time as two dimensional graphs. Aligning these results in a third dimension that introduces time in the visualization. In contrast, “Evolution Storyboard” [2] allows to compare two states of a software architecture by showing two dimensional graph representations of them in panels next to each other.

The visualization “EvoStreets” [13] uses a city metaphor that also considers the history of a software architecture. Package hierarchy is visualized as software streets with packages branching off orthogonal, representing their superordinate package. Classes are represented as buildings at the side of their respective software street. If a new class is created, the software street is extended at its end. To include software evolution a topology is added to their software map. Height is used to visualize a time component, with older structures being placed on a higher elevation level. Thus the city grows from top to bottom over time.

In [14], Voronoi Treemaps are used to monitor the evolution of software quality attributes. The authors employ scaled Hilbert curves to place Voronoi sites in the plane, thus gaining a more stable visualization. Regarding treemaps, [15] propose a methodology and associated quality metrics to measure the quality of dynamic treemaps for the specific use case and context of software evolution visualization.

6 CONCLUSION AND FUTURE WORK

We presented our approach to extend IslandViz towards showing the evolution of a software architecture by adapting island layouts according to changes in packages and compilation units. To ensure that new regions of an island have enough space to grow, we extended the enhanced hexagon tiling algorithm (EHTA). Our approach creates a growth corridor for each new region. The growth corridor cannot overlap with any other region or growth corridor, thus ensuring that the new region is not placed in an already enclosed space. An additional advantage of the use of the EHTA for our solution is that the borders remain somewhat irregular and support the island metaphor. To further ensure more realistic island layouts, we refined our extension by adding additional constraints,

prohibiting growth corridors of a new region being placed directly between two existing corridors.

While this paper focuses on the island layouts, the evolution of a software architecture also changes the relationships between OSGi bundles. As a result the positions of islands must change accordingly. Our extension of IslandViz will include islands which will be able to drift according to their changing relationships.

Our visualization is specifically designed with the goal to support users in their understanding of complex software architectures. Therefore, we want to enhance our interface and interaction techniques to make our extension of IslandViz more accessible to users. To this regard, we also plan to conduct user studies to evaluate our extension.

REFERENCES

- [1] L. Bedu, O. Tinh, and F. Petrillo. 2019. A Tertiary Systematic Literature Review on Software Visualization. In *2019 Working Conference on Software Visualization (VISOFT)*. 33–44. <https://doi.org/10.1109/VISOFT.2019.00013>
- [2] D. Beyer and A. E. Hassan. 2006. Animated Visualization of Software History using Evolution Storyboards. In *2006 13th Working Conference on Reverse Engineering*. 199–210. <https://doi.org/10.1109/WCRE.2006.14>
- [3] H. Gall, M. Jazayeri, and C. Riva. 1999. Visualizing software release histories: the use of color and third dimension. In *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99)*. 'Software Maintenance for Business Change' (Cat. No.99CB36360), 99–108. <https://doi.org/10.1109/ICSM.1999.792584>
- [4] A. E. Hassan, Jingwei Wu, and R. C. Holt. 2005. Visualizing historical data using spectrographs. In *11th IEEE International Software Metrics Symposium (METRICS'05)*. 10 pp.–31. <https://doi.org/10.1109/METRICS.2005.54>
- [5] Michele Lanza. 2001. The Evolution Matrix: Recovering Software Evolution Using Software Visualization Techniques. In *Proceedings of the 4th International Workshop on Principles of Software Evolution (Vienna, Austria) (IWPSE '01)*. ACM, New York, NY, USA, 37–42. <https://doi.org/10.1145/602461.602467>
- [6] Martin Misiak, Rawi85, and Andreas Schreiber. 2018. DLR-SC/island-viz: IslandViz 1.0. <https://doi.org/10.5281/zenodo.1464633>
- [7] M. Misiak, D. Seider, S. Zur, A. Fuhrmann, and A. Schreiber. 2018. Immersive Exploration of OSGi-Based Software Systems in Virtual Reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 1–2. <https://doi.org/10.1109/VR.2018.8446057>
- [8] Richard Müller and Dirk Zeckzer. 2015. Past, Present, and Future of 3D Software Visualization: A Systematic Literature Analysis. *IVAPP 2015 - 6th International Conference on Information Visualization Theory and Applications; VISIGRAPP, Proceedings*. <https://doi.org/10.5220/0005325700630074>
- [9] Blaine Price, Ronald Baecker, and Ian Small. 1993. A Principled Taxonomy of Software Visualization. *J. Vis. Lang. Comput.* 4 (09 1993), 211–266. <https://doi.org/10.1006/jvlc.1993.1015>
- [10] Meike Schaller and Andreas Schreiber. 2019. Visualization of Component-Based Software Architectures: A Comparative Evaluation of the Usability in Virtual Reality and 2D. In *Human Interface and the Management of Information. Visual Information and Knowledge Management*, Sakae Yamamoto and Hirohiko Mori (Eds.). Springer International Publishing, Cham, 208–222.
- [11] Andreas Schreiber and Martin Misiak. 2018. Visualizing Software Architectures in Virtual Reality with an Island Metaphor. In *VAMR 2018: Virtual, Augmented and Mixed Reality: Interaction, Navigation, Visualization, Embodiment, and Simulation*, Jessie Y. C. Chen and Gino Fragomeni (Eds.). *Lecture Notes in Computer Science* 10909, 168–182.
- [12] Andreas Schreiber, Lisa Nafeie, Artur Baranowski, Peter Seipel, and Martin Misiak. 2019. Visualization of Software Architectures in Virtual Reality and Augmented Reality. In *2019 IEEE Aerospace Conference*. 1–12. <https://doi.org/10.1109/AERO.2019.8742198>
- [13] Frank Steinbrückner and Claus Lewerentz. 2010. Representing Development History in Software Cities. In *Proceedings of the 5th International Symposium on Software Visualization (Salt Lake City, Utah, USA) (SOFTVIS '10)*. ACM, New York, NY, USA, 193–202. <https://doi.org/10.1145/1879211.1879239>
- [14] Rinse van Hees and Jurriaan Hage. 2015. Stable Voronoi-based visualizations for software quality monitoring. In *2015 IEEE 3rd Working Conference on Software Visualization (VISOFT)*. IEEE. <https://doi.org/10.1109/visoft.2015.7332410>
- [15] Eduardo Faccin Vernier, Alexandru C. Telea, and Joao Comba. 2018. Quantitative Comparison of Dynamic Treemaps for Software Evolution Visualization. In *2018 IEEE Working Conference on Software Visualization (VISOFT)*. IEEE. <https://doi.org/10.1109/visoft.2018.00018>
- [16] Lynn von Kurnatowski. 2018. *Visualisierung der Evolution von Softwarearchitektur*. Master's thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg. <https://elib.dlr.de/121897/>
- [17] Muye Yang and Robert P. Biuk-Aghai. 2015. Enhanced Hexagon-Tiling Algorithm for Map-Like Information Visualisation. In *Proceedings of the 8th International Symposium on Visual Information Communication and Interaction (Tokyo, AA, Japan) (VINCI '15)*. Association for Computing Machinery, New York, NY, USA, 137–142. <https://doi.org/10.1145/2801040.2801056>