VR-SysML+Traceability: Immersive Requirements Traceability and Test Traceability with SysML to Support Verification and Validation in Virtual Reality

Roy Oberhauser[0000-0002-7606-8226]
Computer Science Dept.
Aalen University
Aalen, Germany
e-mail: roy.oberhauser@hs-aalen.de

Abstract - **As systems grow in complexity, the interdisciplinary nature of systems engineering makes the visualization and comprehension of the underlying system models challenging for the various stakeholders. This, in turn, can affect validation and realization correctness. Furthermore, stakeholder collaboration is often hindered due to the lack of a common medium to access and convey these models, which are often partitioned across multiple 2D diagrams. This paper contributes VR-SysML, a solution concept for visualizing and interacting with Systems Modeling Language (SysML) models in Virtual Reality (VR). Our prototype realization shows its feasibility, and our evaluation results based on a case study shows its support for the various SysML diagram types in VR, cross-diagram element recognition via our Backplane Followers concept, and depicting further related (SysML and non-SysML) models side-by-side in VR.**

Keywords - Systems Modeling Language (SysML); virtual reality; systems modeling; systems engineering; requirements traceability; test traceability; system testing; verification and validation.

I. Introduction

This paper extends the immersive Systems Modeling Language (SysML) model visualization and interaction capabilities in VR-SysML [1]. Towards supporting immersive software (SW) verification and validation (V&V), it contributes semi-automated requirements traceability and test tracing capabilities visualized in VR.

Systems engineering (SysE) is an interdisciplinary collaborative engineering field dealing with the design, integration, and management of complex system solutions over their lifecycle. The field faces a continuous challenge of growing system complexity, an increasing share of functionality shifted to software, system resource constraints, while coping with compressed development timeframes and project budget and resource constraints. Furthermore, the interdisciplinary nature of SysE means that diverse stakeholder types and groups with their specialty competencies and concerns are involved and who may not be readily acquainted with the model types and modeling languages involved. Any models may be digitally isolated or practically inaccessible to all stakeholder types, "hidden" within "cryptic" modeling tools that certain modeling specialists may understand. Due to the interdisciplinary nature of SysE, the inaccessibility and lack of model comprehension can hamper collaboration and affect overall system validity and correctness with regard to requirements. Visualized requirements traceability can help support validity checking. Furthermore, visualizing test traceability can help with the analysis of the testing effort and support V&V.

While SysE can involve various models including physical, mechanical, electrical, thermodynamic, and electronic, the focus of this paper is on the Systems Modeling Language [2]. SysML is a dialect of the Unified Modeling Language (UML®) and defined as a UML 2 Profile. Views and their associated diagrams can help reduce cognitive overload, yet their divided nature also risks overlooking a relation or element and comprehending the overall model. Ideally, a model should be whole and complete to the appropriate degree for the reality it is depicting and simplifying. Yet the modeling languages and associated tooling typically assumes a 2D display and portrays portions of models sliced onto 2D diagrams. Although 3D models can be portrayed on 2D displays, they lack an immersion quality.

VR is a mediated visual environment which is created and then experienced as telepresence by the perceiver. VR provides an unlimited immersive space for visualizing and analyzing a growing and complex set of system models and their interrelationships simultaneously in a 3D spatial structure viewable from different perspectives. Lacking a proper 3D system modeling notation, in the interim we propose retaining the well-known SysML notation and interconnecting 2D SysML diagrams in VR, which can suffice for depicting the relations between elements across diagrams and assist with navigating and validating complex models. As system models grow in complexity and reflect the deeper integration and portrayal of their system reality and environment, an immersive digital environment provides an additional visualization capability to comprehend the "big picture" model for structurally and hierarchically complex system models via interconnected diagrams and associated digital elements.

As to our prior work in visualizing architecture in VR, VR-UML [3] provides VR-based visualization and interaction with UML models. VR-EA [4] visualizes Enterprise Architecture (EA) ArchiMate models in VR. Extending VR-SysML [1], this paper contributes VR-SysML+Traceability, a VR-based solution concept for visualizing and interacting with SysML while adding additional SysML automated requirements traceability and test tracing capabilities to support SW V&V. Our prototype realization shows its feasibility, and a case-based evaluation provides insights into its capabilities.

The remainder of this paper is structured as follows: Section 2 discusses related work. In Section 3, the solution concept is described. Section 4 provides details about the realization. The evaluation is described in Section 5 and is followed by a conclusion.

II. Related Work

As to visualization approaches with SysML, Nigischer and Gerhard [5] proposed a lightweight 3D visualization for SysML models in Product Data Management. They describe an approach and concept, but no prototype is shown. Barosan et al. [6] describes a 3D SysML digital-twin-in-loop virtual simulation environment of a distribution center for truck driving test scenarios integrating IBM Rhapsody with Unity3D; VR and immersion are not considered. Mahboob et al. [7] describe a model-based approach to generate VR object collision simulation scenes from SysML behavior models.

Besides our own VR-UML [3], VR features are not yet commonplace in UML tools: Ozkaya [8] analyzed 58 different UML tools without any mention of VR, and Ozkaya and Erata [9] surveyed 109 practitioners to determine their UML preferences without any mention of VR. Non-VR 3D-based UML visualization includes X3D-UML [10], VisAr3D [11], and the case study by Krolovitsch and Nilsson [12].

Work related to requirements traceability visualization includes Li & Maalej [13], which found traceability matrices and graphs preferrable for management tasks. Matrices were preferred for an overview, while graphs were preferred for navigating linked artifacts. They noted that users were not always capable of choosing the most suitable traceability visualization. Abad et al. [14] performed a systematic literature review on requirements engineering visualization. Madaki & Zainon [15] performed a review on tools and techniques for visualizing SW requirement traceability. None of the above literature mentioned immersive or VR techniques; our literature search did not find similar work.

With regard to test traceability, we found no VR work directly addressing this topic. VR-related work regarding software analysis includes VR City [16], which applies a 3D city metaphor. While it briefly mentions that its work might be used for test, it shows no actual results in this regard and in this regard only a trace mode visualization is depicted.

In contrast, VR-SysML+Traceability provides an immersive visualization and experience with SysML models, providing automatic layout of views as stacked 3D hyperplanes, visualizing the reality of inter-view relations and recurrence of elements, and enabling interactive modeling in VR. Furthermore, it provides traceability of requirements and test status to immersively support V&V. Hypermodeling support enables SysML, UML, and other relevant models to be simultaneously visualized in the same virtual space, supporting cross-model analysis across various diagram types and stakeholder concerns.

III. Solution Concept

Our solution concept is based on VR. In support of our view that an immersive VR experience can be beneficial for model analysis, Müller et al. [17] compared VR vs. 2D for a software analysis task, finding that VR does not significantly decrease comprehension and analysis time nor significantly improve correctness (although fewer errors were made). While interaction time was somewhat less efficient than the common daily 2D interactions one is used to and has been trained in for years, it is important to note that VR improved the user experience, was more motivating, less demanding, more inventive/innovative, and more clearly structured.

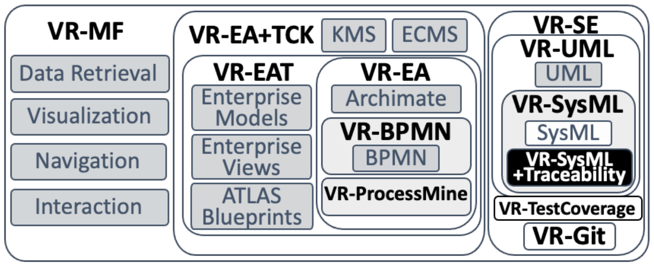


Figure 1. Conceptual map of our various VR solution concepts.

SysML is a general-purpose architecture modeling language for systems and systems-of-systems, supporting their specification, analysis, design, verification, and validation. Out of UML 2's diagrams, it reuses seven (modifying four of these) while adding two additional ones. Thus, for VR-SysML (Figure 1) we chose to extend our VR-UML [3] solution concept, which is based on our generalized VR Modeling Framework (VR-MF) (detailed in [4]). VR-MF provides a VR-based domain-independent hypermodeling framework addressing four aspects requiring special attention when modeling in VR: visualization, navigation, interaction, and data retrieval. Our other VR architectural modeling solutions include VR-BPMN [18], VR-ProcessMine [19], VR-EA [4], and VR-EAT [20], which integrates the EA tool Atlas to provide dynamically-generated EA diagrams in VR. VR-EA+TCK [21] adds additional capabilities, integrating enterprise Tool, Content, and Knowledge such as a Knowledge Management Systems (KMS) and/or Enterprise Content Management Systems (ECMS). While SysML is popular for embedded and model-based systems, it is also applicable to domains such as EA. In the Software Engineering (SE) area, which we group under VR-SE, whereby VR-TestCoverage [22] and VR-Git [23] address test coverage and code repository aspects in VR.

A. Visualization in VR

Our concept attempts to leverage the best of 2D and VR: to support diagram comprehension, we chose not to diverge significantly from the SysML notation. Yet placing 2D SysML images like flat screens in front of users would provide little added value in the 3D VR space. A plane is used to intuitively represent a diagram. Stacked hyperplanes are used to support viewing multiple diagrams at once, while permitting a user to readily have an overview of the number and types of diagrams. Furthermore, hyperplanes serve a grouping function and allow us to utilize the concept of a common transparent or invisible backplane to indicate common elements across diagrams via multi-colored inter-diagram followers. Versus side-by-side, stacked diagrams are a scalable approach for larger projects since the distance to the VR camera is shorter. Multiple stacks can be used to group diagrams or delineate heterogeneous models. Diagrams of interest can still be viewed side-by-side by moving them from the stack via an anchor sphere affordance on a diagram corner, which is also used to hide or collapse diagrams to reduce visual clutter. To distinguish SysML elements types, 2D icon images can be placed on generic (e.g., block) model elements, in order to reduce the effort of modeling each SysML element type as a separate 3D form for VR.

B. Navigation in VR

One navigation challenge arising from the immersion VR offers is supporting intuitive spatial navigation while reducing potential VR sickness symptoms. Thus, we incorporate two navigation modes in our solution concept: the default uses gliding controls for fly-through VR, while teleporting instantly places the camera at a selected position. Although potentially disconcerting, it may reduce the likelihood of VR sickness induced by fly-through for those prone to it.

C. Interaction in VR

As VR interaction has not yet become standardized, in our concept user-element interaction is supported primarily through VR controllers and a *Virtual Tablet*. The VR-Tablet provides detailed element information with context-specific Create, Retrieve, Update, Delete (CRUD) capabilities including a *virtual keyboard* for text entry via laser pointer key selection. The aforementioned corner *anchor sphere* affordance supports moving / hiding / displaying diagrams. Inter-diagram element *followers* can be displayed, hidden, or selected (emphasized).

D. Traceability

A modeling tool such as Sparx Systems Enterprise Architect can be used to provide requirement and test traceability information via SysML. Our solution then extracts traceability-related information from relevant SysML diagrams (Requirements and/or Use Case diagrams), including elements such as Requirement (stereotype «requirement») and Test Case, and relations such as «satisfy», «verify», and «deriveReqt», etc.

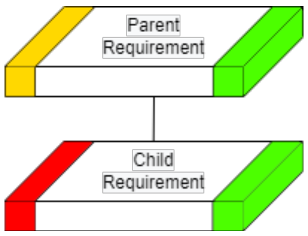


Figure 2. Depicting (sub/super)-requirement dependencies, with degree of implementation on the left and test implementation on the right edge.

For tracing requirements with their dependent code implementation status, annotations are placed in the code to indicate with requirement IDs are addressed. These are extracted by a tool that parses all (test) code files and generates a report. The result is then visualized on the relevant requirement element edges in the diagram in VR (e.g., red means that requirement ID was not found in the (test) code, green if at least one reference was found, and for parent requirements yellow if partially addressed based on some child element(s) missing a reference (see Figure 2). Also, a total degree of implementation considering the elements on that diagram level is also provided on the side of a diagram.

To trace test results to their requirements, the test results are extracted from a test tool report, e.g., in the JUnit XML format used by pytest and JUnit. The test result is then visualized on the relevant requirement diagram elements in VR (e.g., red if no test was found, yellow if not all passed, and green if passed). Also, a total degree of test implementation considering the elements on that diagram level is also provided on the diagram side.

Connectors can be followed to trace these to the actual artifacts, the content of which can be shown in the VR-Tablet.

Note that while the traceability model utilizes information from SysML in addition to other sources, we chose to visualize it in VR independent of SysML conformant constraints, opting for a more intuitive visual depiction of traceability for the stakeholder. Placing a VR-SysML model next to a traceability model is intended and supported. That way, the VR-SysML expresses the exact model it does in a SysML tool, while the traceability model can include the additional automatically extracted implementation and test features without encumbering the SysML model.

For our traceability visualization, we thus chose to layer the information ordered by degree of abstraction as shown in Figure 3. The top layers are SysML model-related: Use cases being the highest abstraction and thus on top, requirements being more concrete and a level below, and test cases being used to verify requirements and thus below requirements but shifted to the side to indicate they relate to testing. The lower layers consist of file trees that visualize the test source code files implementing test cases, and the implementation layer consisting of source code files that implement the requirements.

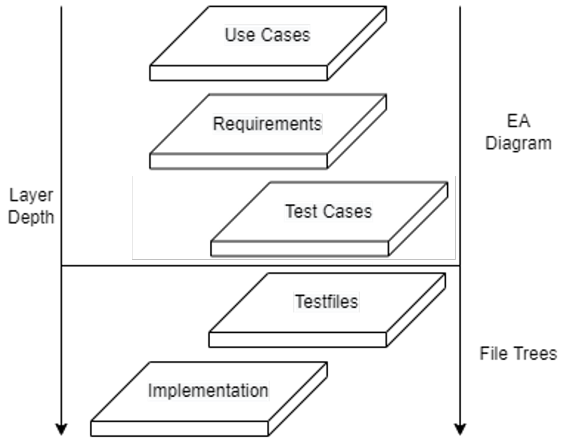


Figure 3. VR-SysML visual layering for traceability.

In general, a backplane is used with colored trace lines to show all the traces, thus indicating the total available traces and the degree of traceability. This avoids a spider web-like tracing of lines across all elements. However, when an element of interest is selected, then direct trace lines specific to that element are also depicted (analogous to a spotlight).

IV. Realization

The realization of the solution focused on two aspects: 1) realizing a correct portrayal of the various SysML diagrams and providing a way to trace the same element to its occurrence in other diagrams (which we name VR-SysML), and 2) the extraction and visualization of requirements traceability information from SysML diagrams and implementation files and test code files.

A. VR-SysML Realization

The logical architecture for our VR-SysML prototype realization is shown in Figure 4.

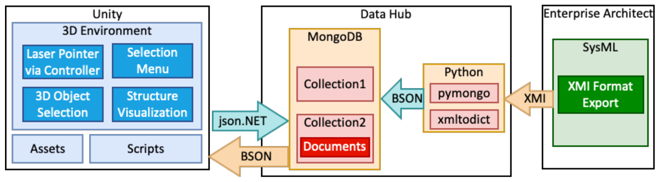


Figure 4. VR-SysML logical architecture.

SysML models are imported in XMI format to our Data Hub that is implemented in Python. Xmitodict is used to convert the XMI to a key-value dictionary and the built-in JSON package is used for JSON conversion. Pymongo is used to store the JSON (as BSON) in the NoSQL document database MongoDB. The scripts in the Unity environment utilize json.NET. SysML XMI files produced from SparxSystems Enterprise Architect were used. Our prototype currently does not consider the Allocation Table (relationship matrices).

B. Requirements Traceability Realization

The logical architecture for our traceability realization is shown in Figure 5.

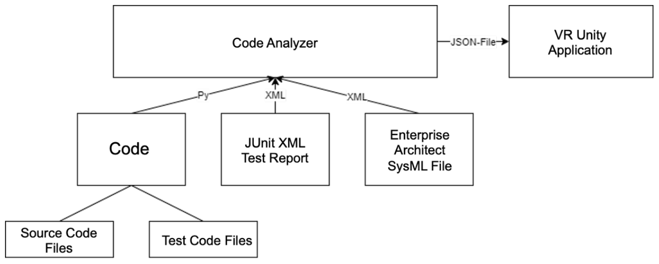


Figure 5. VR-SysML logical architecture for traceability realization.

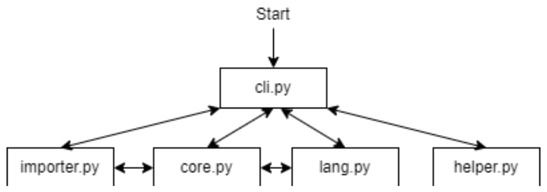


Figure 6. Python modules contained in the Code Analyzer tool.

The Code Analyzer tool has a Command Line Interface (CLI) and is implemented in Python. Based on input parameters, it scans the given files and extracts information such as requirements IDs from code, test reports in JUnit XML, and SysML Enterprise Architect XMI files, producing a JSON file as output that is then imported by the VR application running on the Unity platform. Its modular realization is shown in Figure 6.

Within SysML models, a diagram element with the property "id" serves as the reference for identifying and differentiating requirements as shown in Figure 7 and for test cases as shown Figure 8. The Python library xml.dom.minidom is used to extract the ID, element types, element position and size, relations between elements, and properties, comments, or annotations.

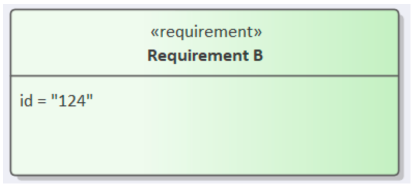


Figure 7. Requirement "id" as property in SysML.

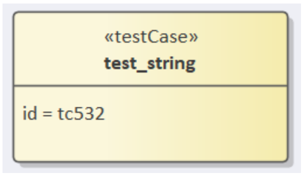


Figure 8. Test case "id" as property in SysML.

```
def test_method(): # REQID: 123, 312 TESTID: tc1232 TESTTARGET:
    ↪ network.py
    pass
```

Figure 9. Traceability annotation example: associating a test method in test code to a requirement and test target (the implementation).

```
"""Test File"""
# REQID: 123

def test_method():
    pass
```

Figure 10. Traceability annotation example: associating all methods in a test source file to one (or more) requirement(s).

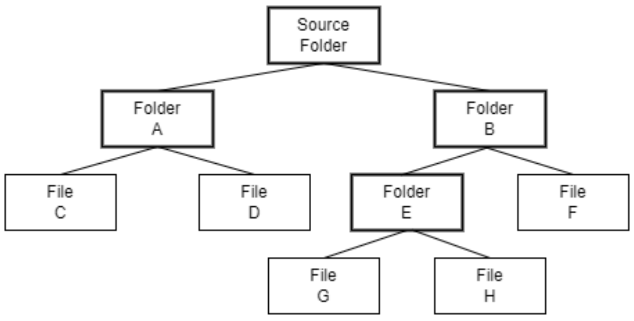


Figure 11. Example file tree with folders.

Since almost all programming languages support comments, the annotations are provided as comments and can thus be utilized in any programming language. For this, lang.py must be extended for each additional language. Within source code, the keyword "REQID" indicates the unique identifier (ID) of a requirement, and can be associated with a (test) method, as shown in Figure 9. When a (test) method satisfies multiple requirements, multiple IDs can be separated via commas. The keyword "TESTID" is used to associate a SysML diagram. Finally, "TESTTARGET" is only used within test files to indicate the test target. Multiple references are possible separated by commas. Some requirements are overarching and it would be arduous to associate each method separately. Thus, for the case when all methods in a file address one (or more) requirement(s), "REQID" can be placed at the top of the file (separate from method declarations or definitions), thus implicitly indicating it is associated with all methods in that file, as exemplified in Figure 10.

To build balanced file trees to portray the test source and implementation source, with each tree consisting of folders and files (see Figure 11), an implementation of Reingold-Tilford algorithm [24] was adapted.

V. Evaluation

We base the evaluation of our solution concept on design science method and principles [25].

A. *VR-SysML Evaluation*

A case study is used with an emphasis on SysML diagram type support, how these are visualized in VR, and additional capabilities in VR. A sample SysML project with all 9 SysML diagram types is used to compare the visualization in Enterprise Architect to that in VR-SysML, grouped as requirement, behavior, or structure diagram types.

As shown in Figure 12, the various diagrams of the SysML model are mapped to stacked hyperplanes that provide an anchor affordance (black sphere) with which to expand, collapse, or move a diagram. Planes and elements have a shallow 3D depth with labeled edges to support recognition from different viewing angles. The colors of the planes can be configured to help with differentiation or grouping. Furthermore, our backplane concept creates followers that allow one to quickly find the same element across different diagrams in the same model, to readily see in which diagrams that element participates, or to determine that the element is only shown on one diagram (it not having a follower). The colored followers can be selected (made bold) and the other followers can be hidden if desired to reduce visual clutter for larger models.

1) SysML Requirement Diagram. SysML extends UML with an additional diagram type, the Requirement diagram. It can be used to specify functional and non-functional requirements for the model. An example viewed in EA is shown Figure 13 and in VR in Figure 14. In VR, elements are labeled on edges to support reading from different angles. The VR Tablet can provide more details or interaction capabilities for a selected element, and while support for modeling capabilities is shown on the interface, these are currently placeholders and have not yet been fully implemented in the prototype (create, modify, delete, export).

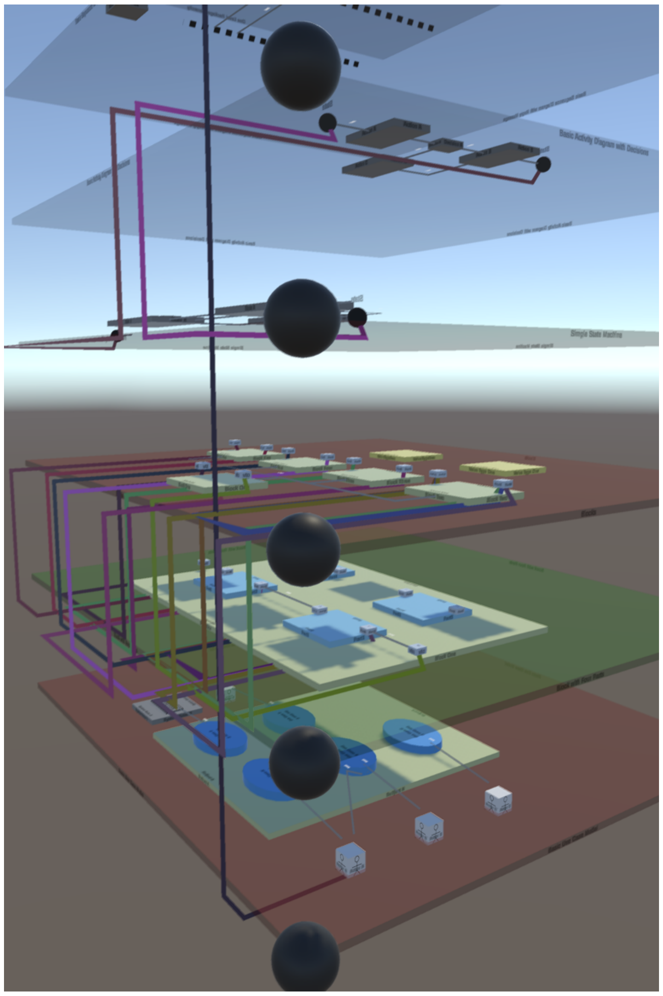


Figure 12. VR-SysML backplane with inter-diagram followers.

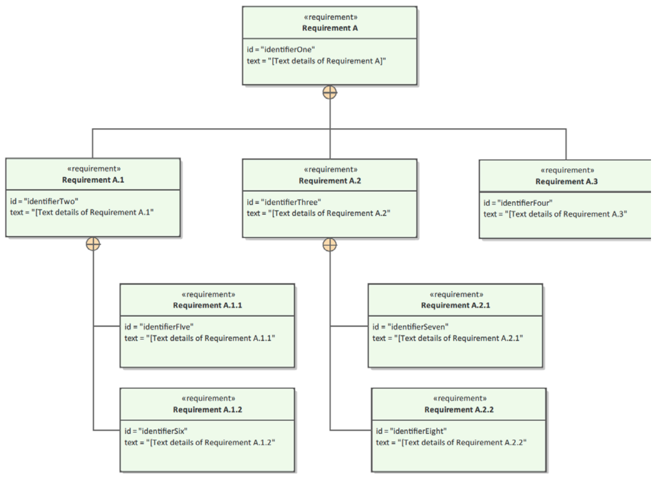


Figure 13. Requirement Diagram in EA.

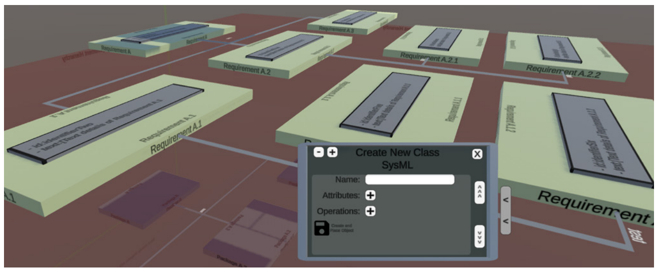


Figure 14. Requirement Diagram in VR.

2) SysML Use Case Diagram. As a behavior diagram, SysML includes the Use Case Diagram from UML as shown from EA in Figure 15 and VR in Figure 16. In order to more readily recognize and differentiate the diagram type, an oval shape was used for the use cases. However, the actors utilize our generic cube concept with notation symbols placed on the various sides. This provides a flexible mechanism for quickly supporting various notation element types and tailoring or extending model element types using any icons or images.

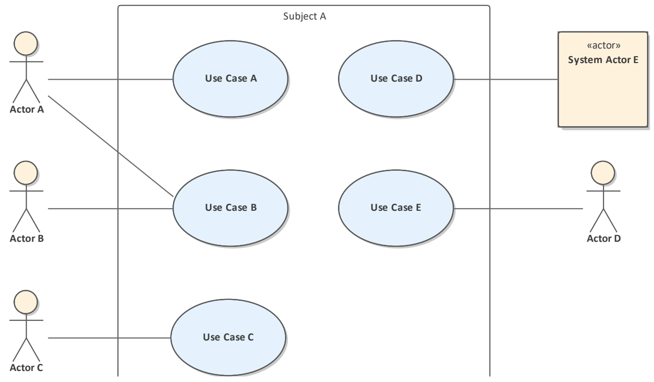


Figure 15. Use Case Diagram in EA.

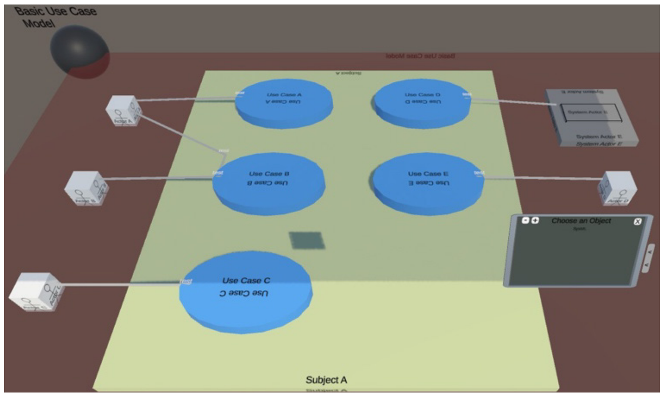


Figure 16. Use Case Diagram in VR.

3) SysML Activity Diagram. Another dynamic behavior diagram type that can be used to specify dynamic system behaviors, such as control flow and object flows, is the Activity diagram in SysML from EA in Figure 17 and VR in Figure 18. It is slightly modified from that in UML, adding additional semantics for Continuous Flow and Probability.

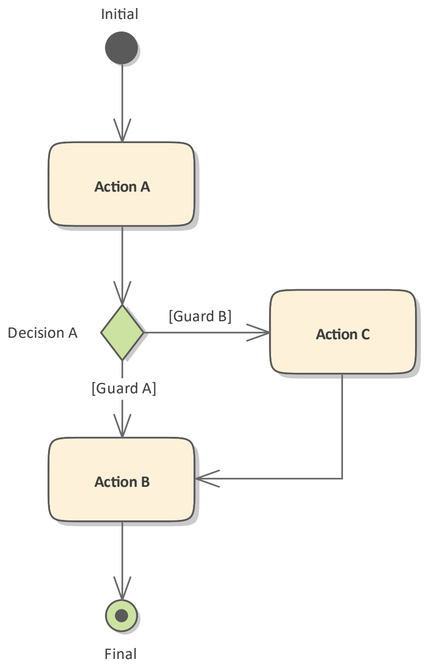


Figure 17. Activity Diagram in EA.

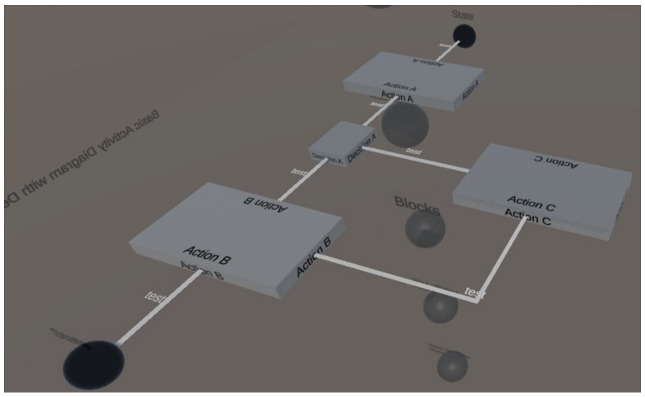


Figure 18. Activity Diagram in VR.

4) SysML Sequence Diagram. Sequence diagrams (unmodified from UML) provide a further dynamic behavior diagram, showing interactions via message sequences, from EA in Figure 19 and VR in Figure 20.

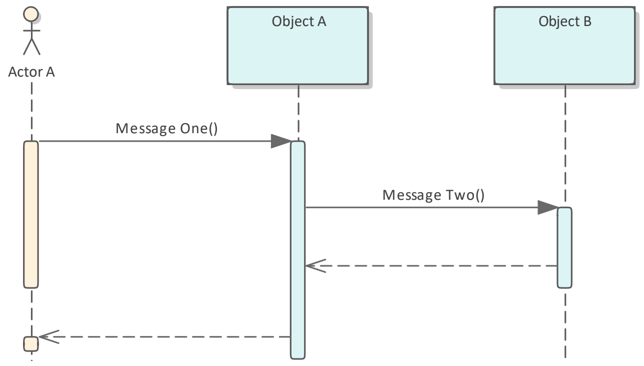


Figure 19. Sequence Diagram in EA.

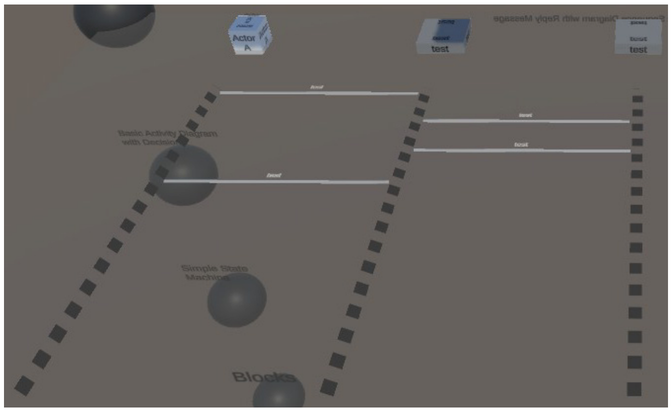


Figure 20. Sequence Diagram in VR.

5) *SysML State Machine Diagram.* State machine diagrams (unmodified from UML) are a dynamic behavior diagram showing states transitions that occur in response to events, from EA in Figure 21 and VR in Figure 22.

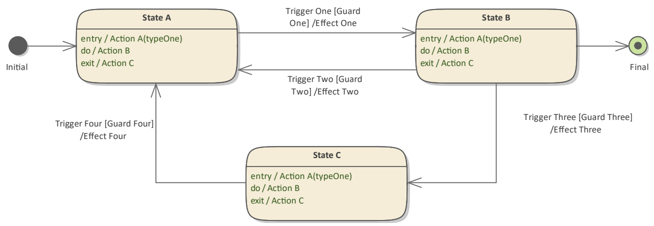

State A
entry / Action A(typeOne)
do / Action B
exit / Action C
State B
entry / Action A(typeOne)
do / Action B
exit / Action C
State C
entry / Action A(typeOne)
do / Action B
exit / Action C
Initial
Final
Trigger One [Guard One] /Effect One
Trigger Two [Guard Two] /Effect Two
Trigger Three [Guard Three] /Effect Three
Trigger Four [Guard Four] /Effect Four

Figure 21. State Machine Diagram in EA.

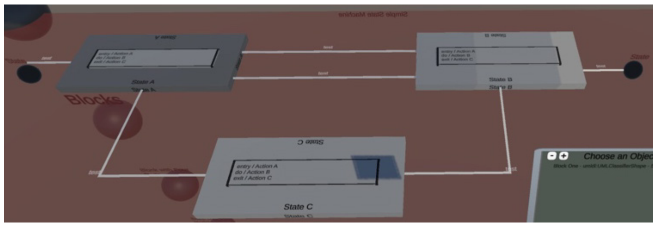


Figure 22. State Machine Diagram in VR.

6) *SysML Block Definition Diagram (BDD).* A BDD is a static structural diagram, analogous to the UML Class diagram type with certain modifications, and shows system components, their contents (as properties, behaviors, constraints), interfaces, and relationships. See Figure 23 for an example from EA and Figure 24 for VR. It can be used for describing the system structure as a hierarchy of relations between systems and subsystems typically consisting of "black-box" blocks. As a possible specialization, it can be useful to explicitly model constraints separately, referred to as Constraint Block diagrams (see Figure 25 for an EA example and Figure 26 for VR), which can be referenced by Parametric diagrams.

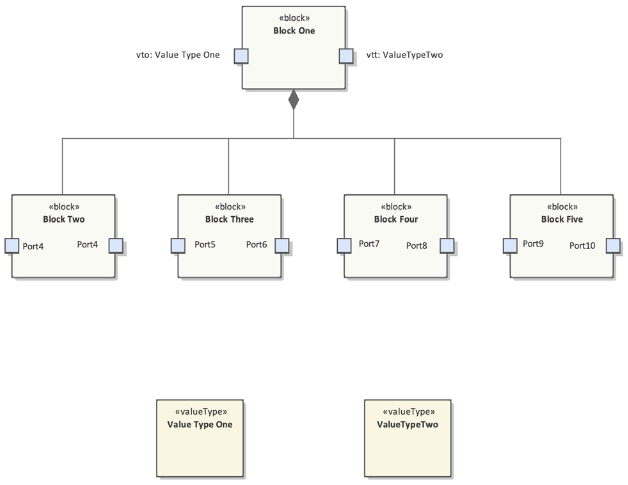


Figure 23. Block Definition Diagram (BDD) in EA.

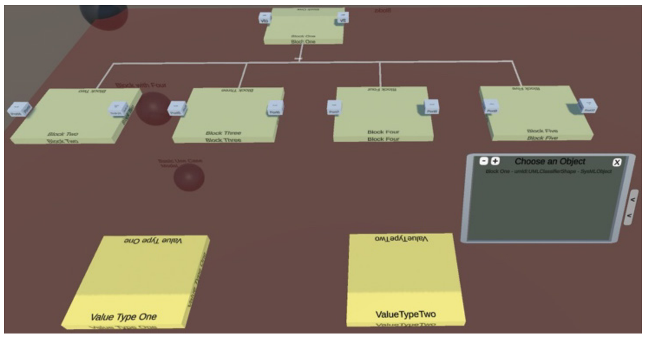


Figure 24. Block Definition Diagram (BDD) in VR

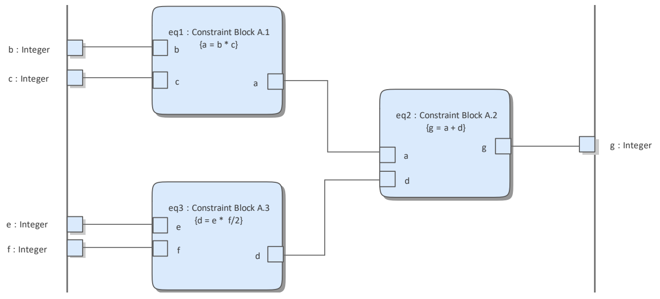


Figure 25. Constraint Block Diagram in EA.

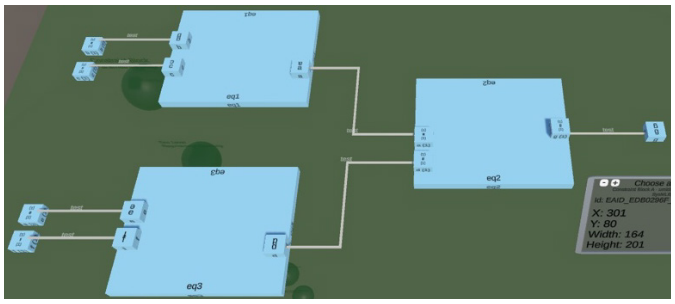


Figure 26. Constraint Block Diagram in VR.

7) SysML Internal Block Diagram (IBD). An IBD is a static structural diagram that depicts the internal (encapsulated) composition (structural contents) of a Block in a BDD, i.e., a "white-box" view. This includes properties, parts, interfaces, connectors, and ports, and can be used to depict the flow of inputs and outputs between them. See Figure 27 for an example in EA and Figure 28 for VR.

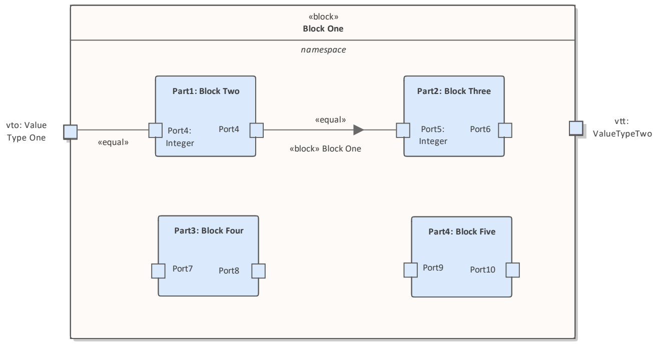


Figure 27. IBD in EA.

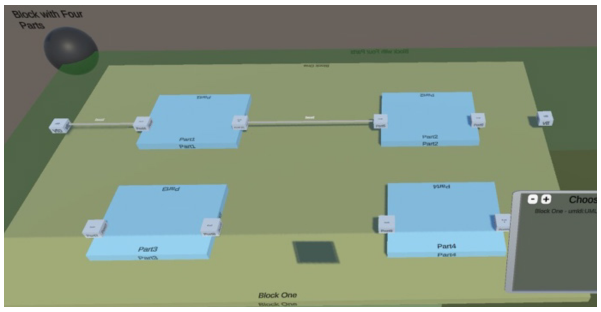


Figure 28. IBD in VR.

8) SysML Parametric Diagram. A static structural diagram type, Parametric diagrams (see Figure 29 for EA and Figure 30 for VR) are a specialization of IBD to model equations with parameters and can be used to enforce mathematical rules or constraints defined via Constraint Blocks.

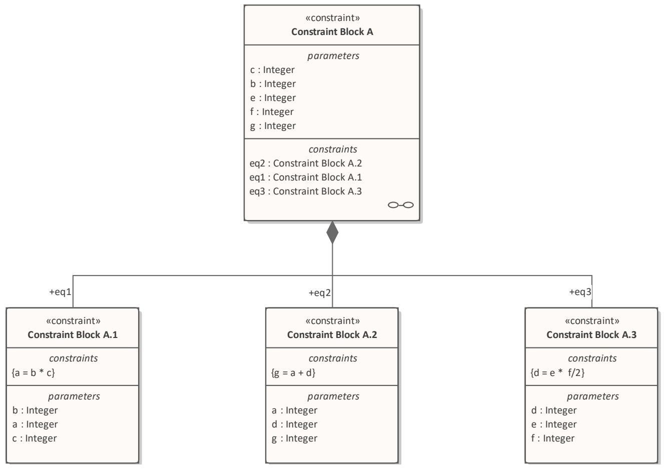


Figure 29. Constraint Parametric Diagram in EA.

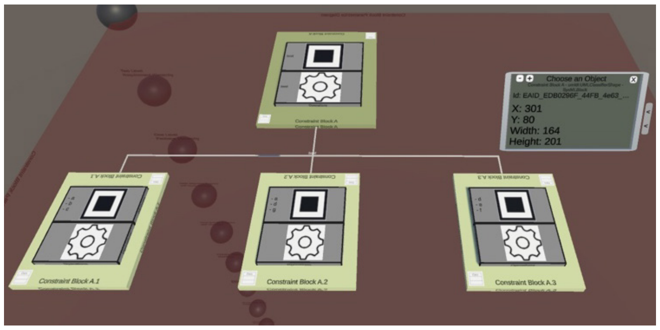


Figure 30. Constraint Parametric Diagram in VR.

9) SysML Package Diagram. A SysML Package diagram (see Figure 31 for EA and Figure 32 for VR) is further static structural diagram based on the equivalent UML type (with minor modifications). Packages provide a general-purpose mechanism for grouping model elements and diagrams, and the diagram can be used to show their contents and the relationship between them.

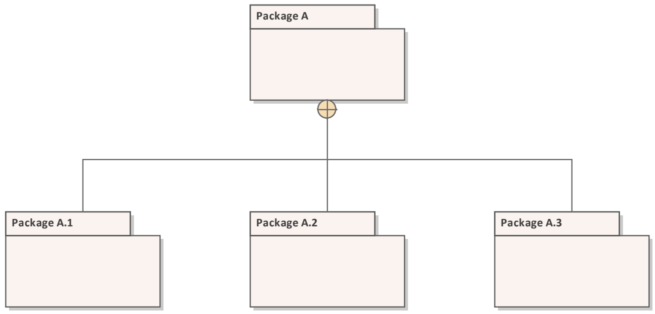


Figure 31. Package Diagram in EA.

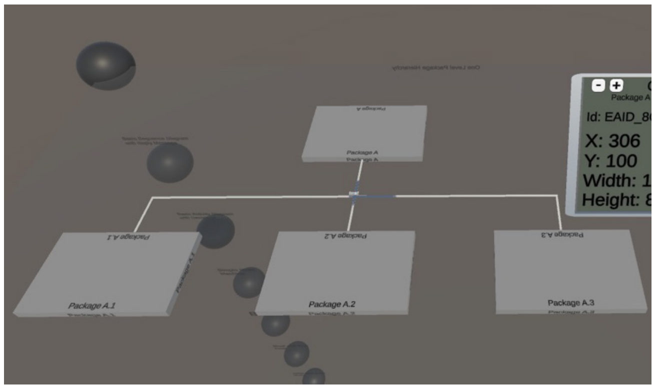


Figure 32. Package Diagram in VR.

10) Multi- and Heterogeneous Model Depiction in VR. VR's unlimited virtual space provides the potential to view, compare, and analyze multiple SysML (left and center models in Figure 33) or heterogeneous models side-by-side, exemplified with an ArchiMate enterprise architecture model on the right in Figure 33. For SysE, this immersive approach also has the potential to support interdisciplinary collaboration between specialization experts for complex systems.

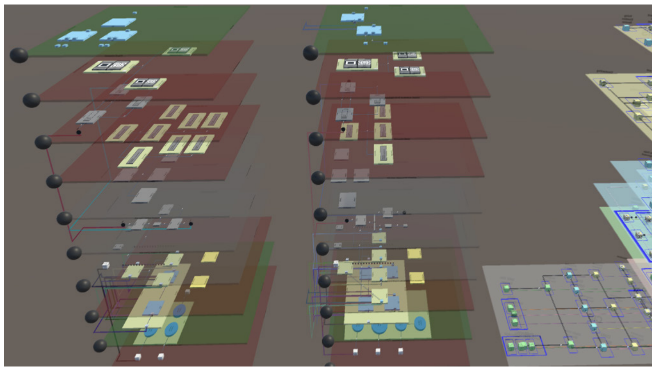


Figure 33. Multiple and heterogeneous side-by-side models in VR.

B. *Traceability Scenario: Requirements and Tests*

To evaluate the traceability scenario, an example project is used, consisting of a various source and test files with a SysML requirements diagram in EA (see Figure 34). As can be seen in the diagram, use cases related to requirements via satisfy, and further child requirements via derived relations, and test cases having a verify relation to the requirements.

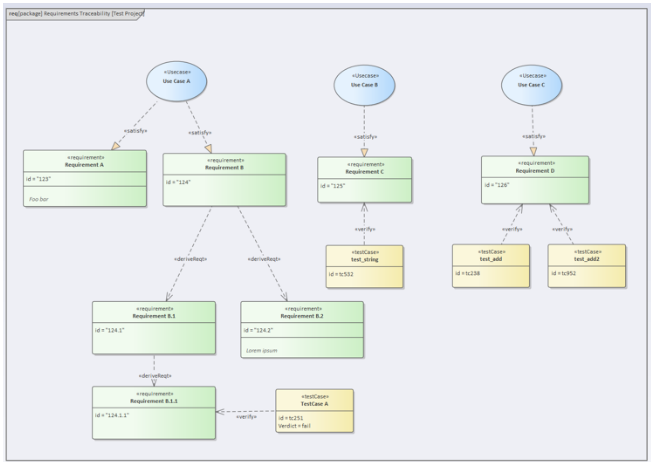
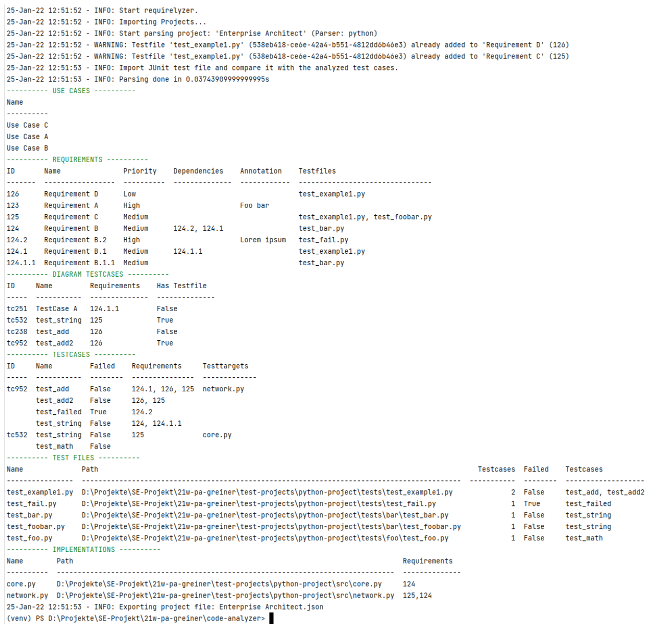


Figure 34. Example requirements diagram in EA.

Output of the code analysis tool is shown in Figure 35. A separate JSON file (not shown) with this information is generated for VR to process the information for visualization. The output shows the scanned use cases, derived requirements, test cases, test files, and implementation source files, and their relations.

The visualization of the traceability layers (Figure 36) and relations backplane (Figure 37) are shown in VR. Use cases (biege) are the most abstract at the top (Figure 38), with satisfy relations following requirements. The number of requirements layers (dark grey) (Figure 39) are variable, depending on the depth of derivation hierarchy, in this case two additional layers. The test cases layer (Figure 40) is shown in (blue) and offset to the right of the stack. File trees for the implementation and the tests (shifted to the right) are depicted on the lowest purple planes as seen in Figure 41.


```
25-Jan-22 12:51:52 - INFO: Start requirelyzer.
25-Jan-22 12:51:52 - INFO: Importing Projects...
25-Jan-22 12:51:52 - INFO: Start parsing project: 'Enterprise Architect' (Parser: python)
25-Jan-22 12:51:52 - WARNING: Testfile 'test_example1.py' (538eb418-ce6e-42a4-b551-4812dd6b46e3) already added to 'Requirement D' (126)
25-Jan-22 12:51:52 - WARNING: Testfile 'test_example1.py' (538eb418-ce6e-42a4-b551-4812dd6b46e3) already added to 'Requirement C' (125)
25-Jan-22 12:51:53 - INFO: Import JUnit test file and compare it with the analyzed test cases.
25-Jan-22 12:51:53 - INFO: Parsing done in 0.03743909999999995s
---------- USE CASES ----------
Name
----------
Use Case C
Use Case A
Use Case B
---------- REQUIREMENTS ----------
ID       Name              Priority    Dependencies    Annotation    Testfiles
-------  ----------------  ----------  --------------  ------------  --------------------------------
126      Requirement D     Low                                       test_example1.py
123      Requirement A     High                        Foo bar
125      Requirement C     Medium                                    test_example1.py, test_foobar.py
124      Requirement B     Medium      124.2, 124.1                  test_bar.py
124.2    Requirement B.2   High                        Lorem ipsum   test_fail.py
124.1    Requirement B.1   Medium      124.1.1                       test_example1.py
124.1.1  Requirement B.1.1 Medium                                    test_bar.py
---------- DIAGRAM TESTCASES ----------
ID     Name         Requirements    Has Testfile
-----  -----------  --------------  --------------
tc251  TestCase A   124.1.1         False
tc532  test_string  125             True
tc238  test_add     126             False
tc952  test_add2    126             True
---------- TESTCASES ----------
ID     Name         Failed    Requirements    Testtargets
-----  -----------  --------  --------------  -------------
tc952  test_add     False     124.1, 126, 125  network.py
       test_add2    False     126, 125
       test_failed  True      124.2
       test_string  False     124, 124.1.1
tc532  test_string  False     125             core.py
       test_math    False
---------- TEST FILES ----------
Name              Path                                                                                  Testcases  Failed    Testcases
----------------  ------------------------------------------------------------------------------------  ---------  --------  -------------------
test_example1.py  D:\Projekte\SE-Projekt\21w-pa-greiner\test-projects\python-project\tests\test_example1.py        2  False     test_add, test_add2
test_fail.py      D:\Projekte\SE-Projekt\21w-pa-greiner\test-projects\python-project\tests\test_fail.py            1  True      test_failed
test_bar.py       D:\Projekte\SE-Projekt\21w-pa-greiner\test-projects\python-project\tests\bar\test_bar.py         1  False     test_string
test_foobar.py    D:\Projekte\SE-Projekt\21w-pa-greiner\test-projects\python-project\tests\bar\test_foobar.py      1  False     test_string
test_foo.py       D:\Projekte\SE-Projekt\21w-pa-greiner\test-projects\python-project\tests\foo\test_foo.py         1  False     test_math
---------- IMPLEMENTATIONS ----------
Name        Path                                                                          Requirements
----------  ----------------------------------------------------------------------------  --------------
core.py     D:\Projekte\SE-Projekt\21w-pa-greiner\test-projects\python-project\src\core.py     124
network.py  D:\Projekte\SE-Projekt\21w-pa-greiner\test-projects\python-project\src\network.py  125,124
25-Jan-22 12:51:53 - INFO: Exporting project file: Enterprise Architect.json
(venv) PS D:\Projekte\SE-Projekt\21w-pa-greiner\code-analyzer> 
```
Figure 35. Code analyzer output.

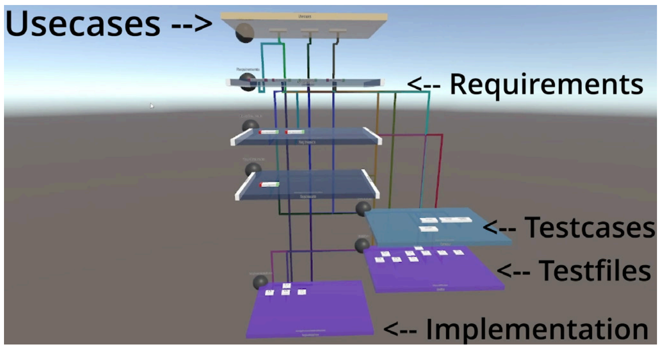


Figure 36. Annotated traceability layers and relations backplane in VR (without element selected).

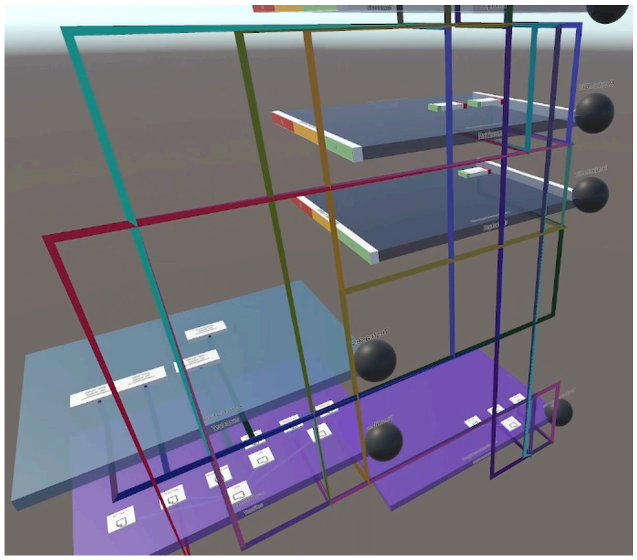


Figure 37. Traceability relations on backplane.

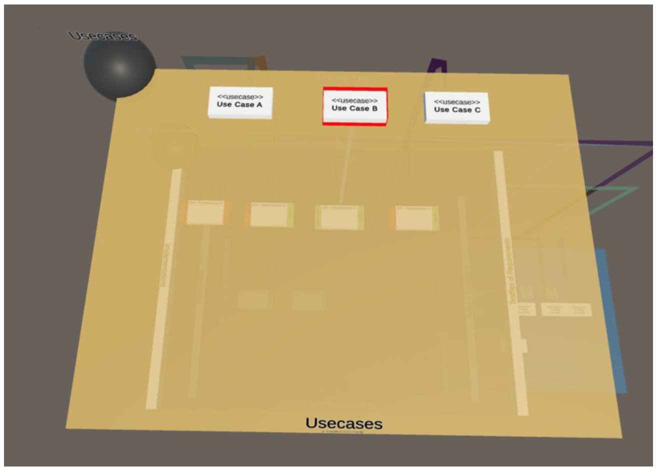


Figure 38. Use case layer.

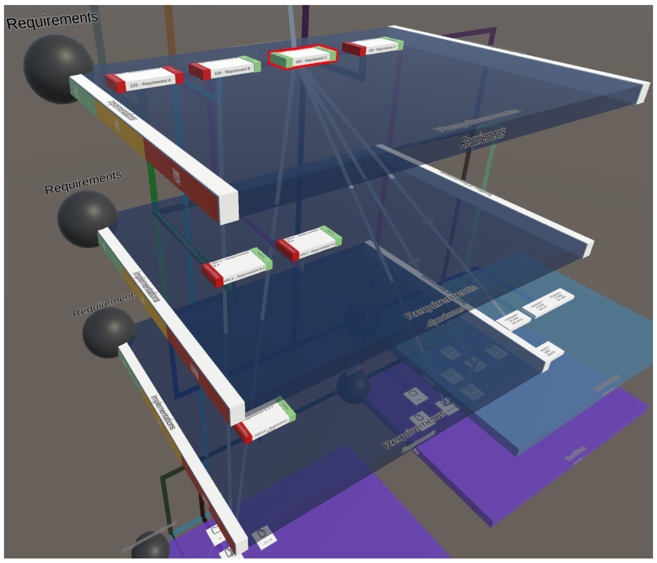


Figure 39. Requirements layers (one element selected showing direct spotlight relations).

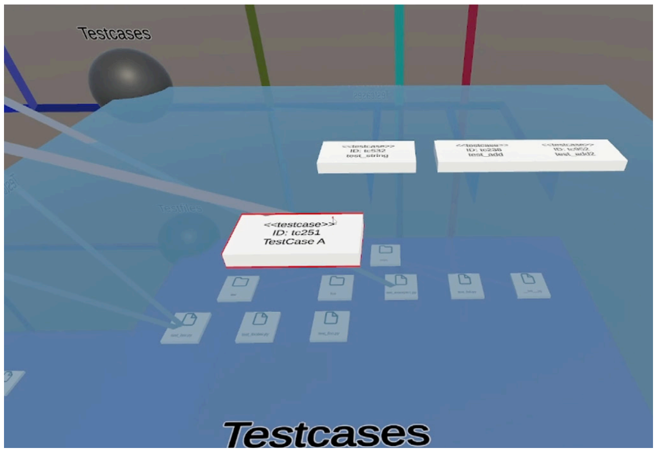


Figure 40. Test cases layer (one element selected).

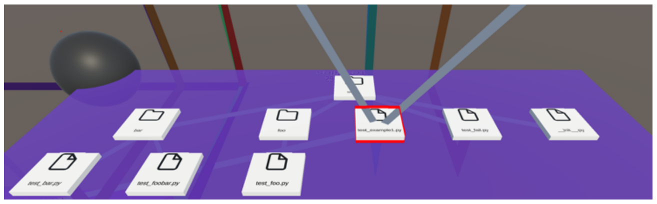


Figure 41. Test files layer showing tree (faintly in grey) of folders and files (with one element selected).

Requirements elements are colored on the edges (see Figure 42): the left side for implementation status and the right side for test case implementation status. The status is colored in three segments: red indicates the percentage of a (product or test) implementation missing, green the percentage completed, and yellow partial fulfillment. Thus, from the perspective of one side of all layers one can get a quick impression of the overall requirements implementation status or from the right side the test implementation status on a per-layer basis.

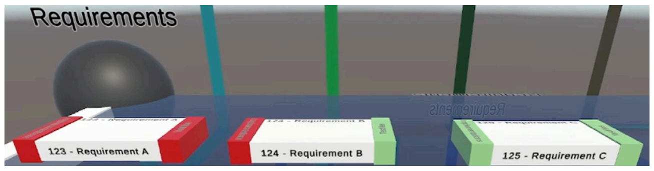


Figure 42. Use case layer.

The requirements layer shows a total fulfillment degree on the diagram edges: left for implementations and right for test cases. For example, in Figure 43 on that requirements layer three requirements are unimplemented while one is, so on the left edge 25% is shown for green and 75% in red, while in Figure 44 since three out of four requirements have tests, on the right edge red shows 25% and green shows 75%. This might be the case if the test team is ready with tests before the implementation has made progress, e.g., as with Acceptance Test-Driven Development (ATDD).

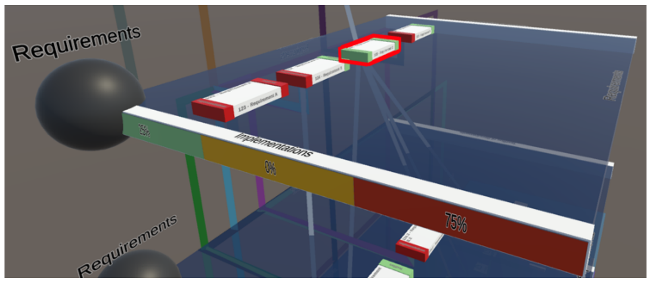


Figure 43. Requirements layer left side (implementation fulfillment).

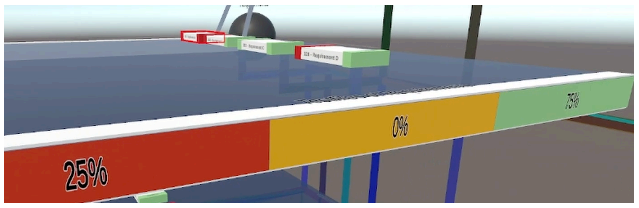


Figure 44. Requirements layer right side (test fulfillment).

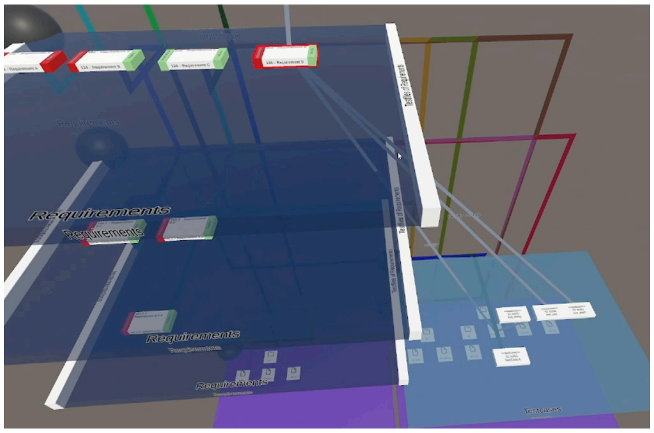
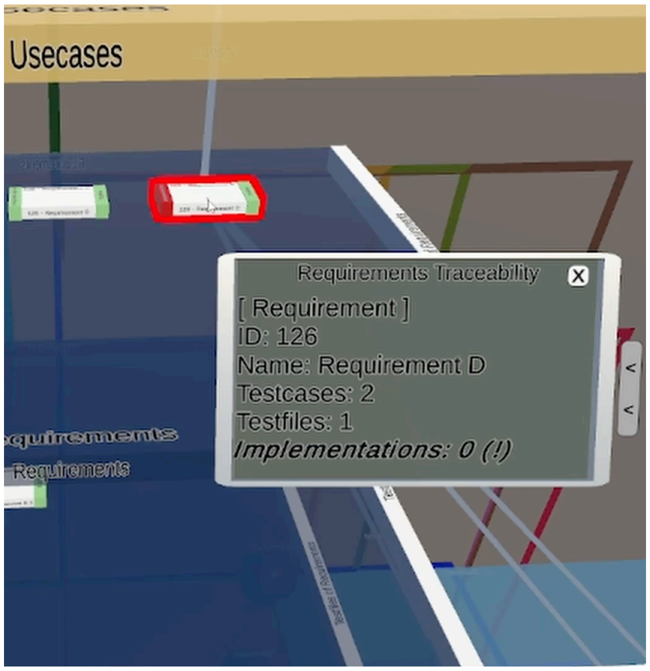


Figure 45. Selected requirement highlights the verifying test cases.

Figure 45 shows the traceability between a requirement to its associated verification test cases.

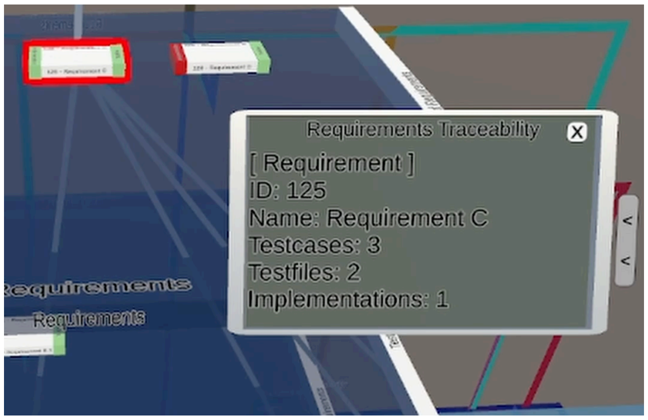
As to the VR-Tablet, additional detailed information about an object is shown. In Figure 46 for the selected requirement ID 126 the name (Requirement D) is shown, the number of test files (1) that exist, the test cases covered (2), and that no implementation was found. Also note no downward trace to the left towards the implementation is shown, only to the right towards tests. In contrast, Figure 47 shows test files and implementations exist for ID 125 (with downward traces in both directions), which is why both its edges are green, while Figure 48 shows a requirement with neither tests nor implementation.



Requirements Traceability

[Requirement]
ID: 126
Name: Requirement D
Testcases: 2
Testfiles: 1
Implementations: 0 (!)

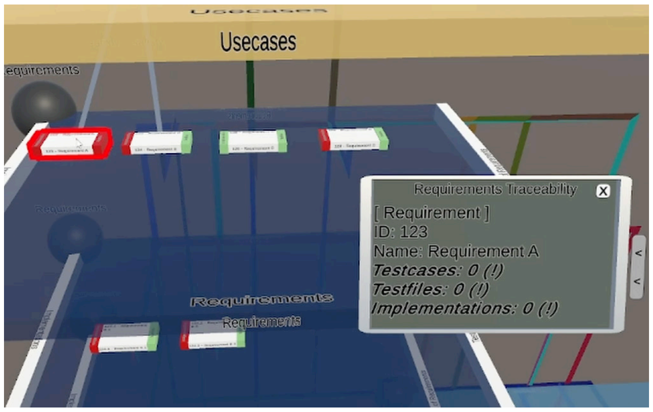
Figure 46. VR-Tablet shows detailed information about selected object.



Requirements Traceability

[Requirement]
ID: 125
Name: Requirement C
Testcases: 3
Testfiles: 2
Implementations: 1

Figure 47. VR-Tablet shows detailed information about selected object.

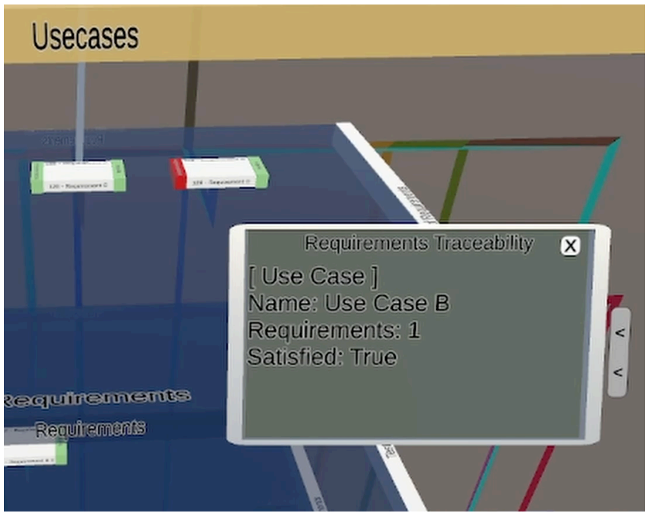


Requirements Traceability

[Requirement]
ID: 123
Name: Requirement A
Testcases: 0 (!)
Testfiles: 0 (!)
Implementations: 0 (!)

Figure 48. VR-Tablet shows detailed information about selected object.

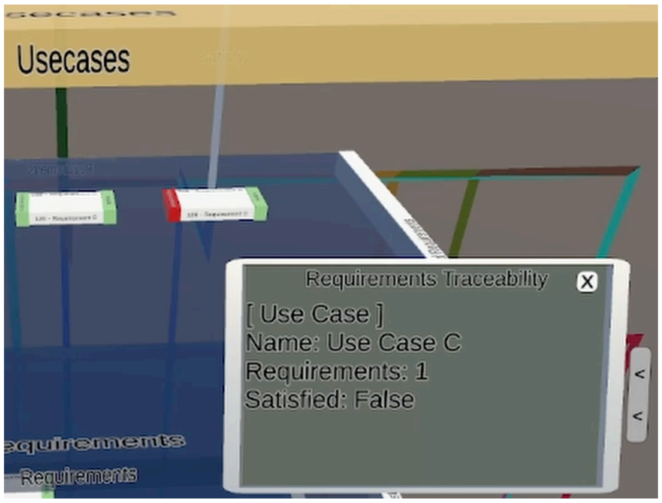
To determine use case satisfaction, the VR-Tablet indicates if a selected use case is satisfied (Figure 49) or not (Figure 50).



Requirements Traceability

[Use Case]
Name: Use Case B
Requirements: 1
Satisfied: True

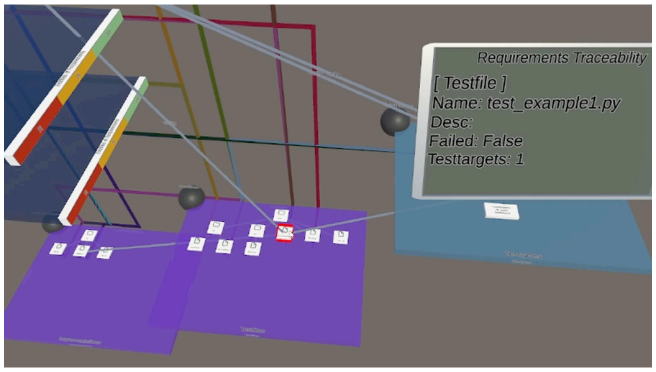
Figure 49. VR-Tablet shows selected use case satisfied since requirement implemented.



Requirements Traceability
[Use Case]
Name: Use Case C
Requirements: 1
Satisfied: False

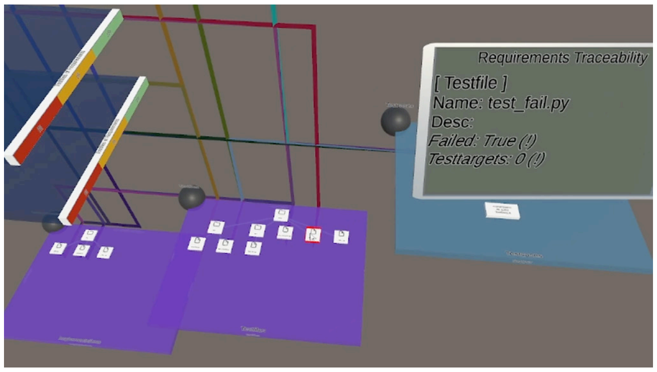
Figure 50. VR-Tablet shows selected use case unsatisfied since requirement not fulfilled.

When viewing test files, the VR-Tablet indicates the number of test targets that exists (the target can be found by following the trace), and if the test passed (Figure 51) or failed (Figure 52).



Requirements Traceability
[Testfile]
Name: test_example1.py
Desc:
Failed: False
Testtargets: 1

Figure 51. VR-Tablet shows associated test implementation name as test_example1.py with a single test target.



Requirements Traceability
[Testfile]
Name: test_fail.py
Desc:
Failed: True (!)
Testtargets: 0 (!)

Figure 52. VR-Tablet showing test case status Failed as "True" and this test case having no associated test target information.

C. *Discussion*

The case-based evaluation of VR-SyML showed its ability to depict the various SysML diagram types in VR. Furthermore, the backplane supports the ability to quickly find the same elements across the various diagram types. As systems grow in complexity, these permits one to quickly find an element of interest in its various contexts. VR-SysML enables an immersive experience in the model, with the unlimited VR space supporting for larger models, multiple diagrams, prior or legacy model versions for comparison, and even heterogenous models displayed simultaneously.

Our traceability evaluation showed the code analyzer was able to automatically scan source code, test files and SysML diagram files and automatically ascertain traceability relevant information. The evaluation showed the ability of VR to provide an intuitive way to simply portray (via layers, the backplane, spotlights, and the VR-Tablet) the relevant artifacts and trace the relations between requirements, tests, and implementation to support V&V activities. This allows independent stakeholders who may not be system experts to evaluate the implementation and testing fulfillment without being inundated with perhaps irrelevant details in typical developer documentation.

VI. Conclusion

VR-SysML+Traceability contributes a VR-based solution concept for visualizing and interacting with SysML while adding automated requirements traceability and test tracing capabilities to support SW V&V. An immersive SysML model experience is provided for visually depicting and navigating SysML diagrams of models in VR. The solution concept was described, a VR prototype realized, while an evaluation using case studies showed its capabilities. Using VR hyperplanes, SysML diagrams are enhanced with 3D depth, color, and automatically-generated inter-diagram element followers based on our back-plane concept. Interaction is supported via a virtual tablet and keyboard. The unlimited space in VR facilitates the depiction and visual navigation of large models, while relations within and between elements, diagrams, and models can be analyzed. Furthermore, additional related (SysML or non-SysML) models can be visualized and analyzed simultaneously in VR, benefiting complex systems-of-systems architectures or collaboration. The sensory immersion of VR can support task focus during model comprehension and increase modeling enjoyment, while limiting the visual distractions that typical 2D display surroundings incur. The semi-automated traceability capability enhances the V&V opportunities by creating a simple and intuitive way to navigate and readily determine the degree of implementation and test progress and requirement fulfillment, even for quality assurance personnel who are not deeply acquainted with the project.

Future work includes support for creating models directly in VR, integrating further SysML tooling and simulation capabilities, supporting tighter and more comprehensive model verification and validation within the SysML diagrams, and conducting a comprehensive empirical study to evaluate usability with various stakeholder groups in collaborative situations.

Acknowledgment

The author would like to thank Shadrach Arulrajah, Marie Bähre, and Christian Greiner for their assistance with the design, implementation, figures, and evaluation.

References

[1] R. Oberhauser, "VR-SysML: SysML Model Visualization and Immersion in Virtual Reality," International Conference of Modern Systems Engineering Solutions (MODERN SYSTEMS 2022), IARIA, 2022, pp. 59-64.

[2] OMG, "OMG Systems Modeling Language Version 1.6", Object Management Group, 2019.

[3] R. Oberhauser, R., "VR-UML: The unified modeling language in virtual reality – an immersive modeling experience," International Symposium on Business Modeling and Software Design (BMSD 2021), Springer, Cham, 2021, pp. 40-58.

[4] R. Oberhauser and C. Pogolski, "VR-EA: Virtual Reality Visualization of Enterprise Architecture Models with ArchiMate and BPMN," In: Shishkov, B. (ed.) BMSD 2019. LNBIP, vol. 356, Springer, Cham, 2019, pp. 170–187.

[5] C. Nigischer and D. Gerhard, "Lightweight visualization of SysML models in PDM systems," DS 87-3 Proc.eedings of the 21st International Conf. on Engineering Design (ICED 17) Vol 3: Product, Services and Systems Design, 2017, pp. 61-70.

[6] I. Barosan, A. Basmenj, S. Chouhan, D. Manrique, "Development of a Virtual Simulation Environment and a Digital Twin of an Autonomous Driving Truck for a Distribution Center," Software Architecture (ECSA 2020). CCIS, vol 1269, Springer, Cham, 2020, pp. 542–557.

[7] A. Mahboob, S. Husung, C. Weber, A. Liebal, and H. Krömker, "SYSML behaviour models for description of Virtual Reality environments for early evaluation of a product," In DS 92: Proc. 15th Int'l Design Conf. (DESIGN), 2018, pp. 2903-2912.

[8] M. Ozkaya, "Are the UML modelling tools powerful enough for practitioners? A literature review," IET Software, vol. 13, 2019, pp. 338-354. https://doi.org/10.1049/iet-sen.2018.5409

[9] M. Ozkaya and F. Erata, "A survey on the practical use of UML for different software architecture viewpoints," Information and Software Technology, Vol. 121, 106275, 2020.

[10] P. McIntosh, "X3D-UML: user-centered design, implementation and evaluation of 3D UML using X3D," Ph.D. dissertation, RMIT University, 2009.

[11] A. Krolovitsch and L. Nilsson, "3D Visualization for Model Comprehension: A Case Study Conducted at Ericsson AB," University of Gothenburg, Sweden, 2009.

[12] C.S.C. Rodrigues, C.M. Werner, and L. Landau, "VisAr3D: an innovative 3D visualization of UML models," In 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), IEEE, 2016, pp. 451-460.

[13] Y. Li and W. Maalej, "Which Traceability Visualization Is Suitable in This Context? A Comparative Study," In: Regnell, B., Damian, D. (eds) Requirements Engineering: Foundation for Software Quality (REFSQ 2012), LNCS, vol 7195. Springer, Berlin, Heidelberg, 2012. https://doi.org/10.1007/978-3-642-28714-5_17

[14] Z. S. H. Abad, M. Noaeen and G. Ruhe, "Requirements Engineering Visualization: A Systematic Literature Review," 2016 IEEE 24th International Requirements Engineering Conference (RE), Beijing, China, 2016, pp. 6-15, doi: 10.1109/RE.2016.61

[15] A.A. Madaki and W.M.N.W. Zainon, "A Review on Tools and Techniques for Visualizing Software Requirement Traceability," In: Mahyuddin, N.M., Mat Noor, N.R., Mat Sakim, H.A. (eds) Proceedings of the 11th International Conference on Robotics, Vision, Signal Processing and Power Applications, Lecture Notes in Electrical Engineering, vol 829, Springer, Singapore, 2022. https://doi.org/10.1007/978-981-16-8129-5_7.

[16] J. Vincur, P. Navrat, and I. Polasek, "VR City: Software analysis in virtual reality environment," In 2017 IEEE international conference on software quality, reliability and security companion (QRS-C), IEEE, 2017, pp. 509-516.

[17] R. Müller, P. Kovacs, J. Schilbach, and D. Zeckzer, "How to master challenges in experimental evaluation of 2D versus 3D software visualizations," In: 2014 IEEE VIS International Workshop on 3Dvis (3Dvis), IEEE, 2014, pp. 33-36.

[18] R. Oberhauser, C. Pogolski, and A. Matic, "VR-BPMN: Visualizing BPMN models in Virtual Reality," In: Shishkov, B. (ed.) BMSD 2018. LNBIP, vol. 319, Springer, Cham, 2018, pp. 83–97. https://doi.org/10.1007/978-3-319-94214-8_6

[19] R. Oberhauser, "VR-ProcessMine: Immersive Process Mining Visualization and Analysis in Virtual Reality," International Conference on Information, Process, and Knowledge Management (eKNOW 2022), IARIA, 2022, pp. 29-36.

[20] R. Oberhauser, P. Sousa, and F. Michel, "VR-EAT: Visualization of Enterprise Architecture Tool Diagrams in Virtual Reality," In: Shishkov B. (eds) Business Modeling and Software Design. BMSD 2020. LNBIP, vol 391, Springer, Cham, 2020, pp. 221-239. doi: 10.1007/978-3-030-52306-0_14

[21] R. Oberhauser, M. Baehre, and P. Sousa, "VR-EA+TCK: Visualizing Enterprise Architecture, Content, and Knowledge in Virtual Reality," In: Shishkov, B. (eds) Business Modeling and Software Design. BMSD 2022. LNBIP, vol 453, pp. 122-140. Springer, Cham. doi:10.1007/978-3-031-11510-3_8

[22] R. Oberhauser, "VR-TestCoverage: Test Coverage Visualization and Immersion in Virtual Reality," The Fourteenth International Conference on Advances in System Testing and Validation Lifecycle (VALID 2022), IARIA, 2022, pp. 1-6.

[23] R. Oberhauser, "VR-Git: Git Repository Visualization and Immersion in Virtual Reality," The Seventeenth International Conference on Software Engineering Advances (ICSEA 2022), IARIA, 2022. To be published.

[24] E.M. Reingold and J.S. Tilford, "Tidier drawings of trees," IEEE Transactions on software Engineering, (2), 1981, pp.223-228.

[25] A.R. Hevner, S.T. March, J. Park, and S. Ram, "Design science in information systems research," MIS Quarterly, 28(1), 2004, pp. 75-105.