VR-ISA: Immersively Visualizing Informed Software Architectures Using Viewpoints Based on Virtual Reality

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

Abstract - **Software is, in its essence, an inherently invisible digital construct, and thus its comprehension and its visualization remain a challenge. All software involves some underlying structure(s), and Software Architecture (SA) comprises the (intended) conceptual abstractions and structuring principles across this invisible construct. Agile development methods, DevOps, and continuous development results in a changing implementation and associated SA that is evolving and continually in flux. Any presumed SA understanding and (perhaps outdated or inconsistent) associated SA documentation may also diverge from the reality, while any shared SA concept across stakeholder minds may vary or differ, potentially resulting in a lack of conceptual integrity. In contrast, an Informed Software Architecture (ISA) is grounded in reality based on actual data and evidence, rather than being influenced by out-of-sync models, documentation, misconceptions, or assumptions. Yet the challenge remains of how best to visually convey ISA aspects, such as internal static software structures and behavioral and operational dynamics, to support evidence-based design, comprehension, and insights in an accessible way for a wider stakeholder spectrum. This paper contributes VR-ISA, a Virtual Reality (VR) solution concept to immersively support an ISA with the visualization of structural, behavioral, and operational aspects. To exemplify our solution concept, three VR-based viewpoints, framing different concerns for different stakeholder groups, are used to illustrate the potential of VR to support ISA: 1) components and connectors, for depicting dynamic distributed event and data streams, 2) modules and dependencies, for depicting static internal module composition and their dependencies, and 3) execution observability, for depicting operational execution, tracing, and observability aspects. Our realization shows its feasibility, while a case-based evaluation provides insights into its capabilities and potential.**

Keywords – informed software architecture; software architecture; virtual reality; event stream processing; data stream processing; event-driven architecture; static analysis; tracing; evidence-based design; observability; visualization.

I. INTRODUCTION

This paper extends our work on VR-EDStream+EDA [1] by extending it to include the visualization of what we refer to as Informed Software Architecture (ISA). To exemplify our solution concept, this paper elucidates descriptions of three VR-based viewpoints that frame operational, dynamic, and static concerns. Viewpoints in Software Architecture (SA) provide conventions for constructing, understanding, and using architectural views to frame certain stakeholder concerns [2]. Architectural views are informational parts of an architectural description that address one or more stakeholder concerns. Ideally, a SA is initially prescriptive, and with ongoing implementation transitions to being descriptive. For larger software (SW) development teams and projects, having SA documentation in sync with the implementation and operation is a challenge. In fact, an implementation may necessarily diverge from its prescriptive SA to address an issue, yet not have been communicated or incorporated in the SA documentation, thus resulting in inconsistencies.

It is said that "data is the new oil," with data playing a fundamental role in the digitalization and automation in various organizations, including enterprises, business, government, manufacturing, and IT (Information Technology). Yet to be valuable, this data is typically dependent on fundamental software building blocks (components such as modules or functions) and their interaction (connectors) to generate, transfer, transform, process, and store data. Moreover, events (a.k.a. records or messages) are a specific type of data consisting of a record of some occurrence. Modern SA is often networked and event-driven, utilizing microservices, Web APIs (Application Programming Interfaces), and/or reactive apps, frameworks, libraries, or services, etc. Microservice adoption in enterprises is growing, with IDC reporting 77% and GitLab reporting 71% of organizations (partially) using microservices [3][4]. Furthermore, in the enterprise, software has become pervasive with digitalization, and hence the number of different software components (apps, services, etc.) and their interdependence or coupling has grown. For instance, among enterprises it is said 57% utilize somewhere between 1000-5000 business applications [5]. Enterprise Service Buses (ESBs), Service Mesh, and the Side-Car pattern are further examples of how different apps and services can be coupled with each other without the apps necessarily being aware of any coupling. Thus, *operational coupling* is often obscure, and for any *dynamic behavior* of *components*, the associated *connectors* (such as events or data streams or event streams) are a concern for developer, IT stakeholders, and even end users (e.g., hidden, privacy, legal/geographic) and a challenge to readily discern and utilize for informing or improving a SA. We will refer to this stakeholder concern as *Concern:CompConn*.

To gain insights into the behavior and health of *deployed* software, a recent trend in software development is *observability*, with its three pillars of *logs*, *metrics*, and *traces* [6], whereby *operational* data is explicitly collected. Although observability is implicitly grounded in the reality of actual data, it is rarely directly used to explicitly inform SA. We will refer to this stakeholder concern as *Concern:Observability.*

Further stakeholders include developers and maintainers (with turnover impacts), faced with potentially nonexistent or incorrect SA documentation and differing confidence levels. From a *static development* perspective, the correlation between the potentially thousands of source codes files, their folder structure, and any actual (intended or unintended) *modules,* and (inter- and intra-)*dependencies* are a concern that can be difficult to readily and visually discern. We will refer to this stakeholder concern as *Concern:ModDep.*

One effect of the digitalization of information is an informed society. Hence, information should be explicitly incorporated to shape and influence future products and their structure, i.e., architecture. The concept of Informed Architecture (IA) has been proposed and explored in the context of construction [7] as well as in digital contexts [8][9].

In this paper, we posit that in the realm of SA, information should also be explicitly incorporated, continuously and readily flow, while being accessible to all stakeholders to address their various concerns. Any new information and insights from this ongoing information flow, analogous to a feedback loop, should result in informed adjustments and adaptations to the SA as applicable. An Informed Software Architecture (ISA) is grounded in the reality of current and ongoing data and facts to inform architectural decisions. Since any architecture is about addressing stakeholder concerns, this information flow should somehow be readily accessible to stakeholders, rather than exclusive to the architect alone. We therefore further posit that VR can offer visual accessibility to information for a wider stakeholder spectrum, while depicting and contextualizing SA-relevant information in new ways.

While various prior work involving SA may include the word "informed" as a verb or regarding decision with SA, we have not as yet found any prior or related work that specifically positions Informed Software Architecture (ISA) as a term. It is our assertion that ISA is essential for the future of SA, for SA to remain relevant, for improving decision making, for supporting comprehension (for developers, maintainers, operators), for ensuring conceptual integrity, for improving documentation, etc.

Additionally, the veracity of any SA-related information, such as models or documentation, is a relevant issue. Due to current Agile, DevOps, continuous development, with their rapid develop-release-deploy cycles and evolutionary architecture trends, any SA documentation can readily become out-of-sync with the reality. While evidence-based design has been touted [10], we believe it to have significant potential in the SA arena, even if it were employed less formally. In this paper, we take a more practical applied view to ISA, rather than employing rigid evidence-based scientific methods that, for instance, rely on hypotheses and proofs. Note that as data-driven SA can be readily confused to mean data-centric or data-oriented SA, this paper instead uses the term Informed SA (ISA), by which we mean a SA that is informed by data-based reality regarding its actual structure and behavior, rather than misconceptions that can readily arise based on assumptions not grounded in a data-grounded reality.

Moreover, there is a growing interest into the insight into the interactions between software and any related data and event processing that an ISA could convey by a wider spectrum of (grassroots or citizen) stakeholders, including domain experts, product owners, software developers, and IT administrators. For example, in DevOps, developer responsibilities are expanding to include operational aspects as well, including deployment, automation, performance management, user experience, and security, and increasingly responsible for the entire lifecycle of application development and operations [11]. And with Low-Code / No-Code (LCNC), an increasing set of additional stakeholders become involved in software development and may be interested in its structural and behavioral aspects, yet in an accessible and intuitive visual form to convey essential characteristics, without assuming Unified Modeling Language (UML) competency, nor necessitating the extraction of information across multiple diagrams to ascertain architecture concepts such as dependencies. To support a larger spectrum of stakeholders with ISA comprehension and insights, an intuitive form of generalized visualization for relevant aspects of an ISA is desirable. While Virtual Reality (VR) could offer a means to portray software structures, data, events, and observability data such as traces, and hence make such ISA aspects accessible to a wider set of stakeholders, VR solution concepts have not been sufficiently investigated.

In prior VR-related work, in the process area we developed VR-BPMN [12] to visualize Business Process Modeling Notation (BPMN) models, while VR-ProcessMine [13] addressed process mining. In the area of Enterprise Architecture (EA), VR-EA [14] contributed a VR solution for ArchiMate EA models, VR-EAT [15] presented a VR-based solution for integrating dynamically-generated EA tool diagrams in VR, while VR-EA+TCK [16] integrated enterprise content and knowledge management systems in VR. In the software architecture and software engineering area, VR-UML [17] supports UML, VR-SysML [18] supports the Systems Modeling Language (SysML), while VR-GitCity [19] supports Git repositories. VR-EDStream+EDA [1] generically supports immersive visualization and analysis data and event stream and Event-Driven Architecture (EDA).

This paper contributes VR-ISA, our VR-based solution concept for supporting ISAs immersively. Towards visualizing and analyzing both dynamic external, static internal, and operational internal information, we elucidate three VR viewpoints: 1) components and connectors for conveying dynamic distributed event and data streams, 2) modules and dependencies for conveying internal static SA structural aspects and metrics, and 3) execution observability for conveying operational aspects such as code traces. Our prototype realization shows its feasibility, and a case-based evaluation provides insights into its capabilities for addressing the aforementioned challenges.

The remainder of this paper is structured as follows: Section II discusses related work. In Section III, we describe our solution. Section IV provides details about the realization. The evaluation is described in Section V followed by a conclusion.

II. Related Work

Related work regarding event and data stream visualization includes the data visualization survey by Qin et al. [20], which only mention events streams with regard to SQL-like query support. A survey on immersive analytics by Fonnet and Prié [21] includes no citations related to streams, and only two related to events: IDEA [22], which depicts user activity logs in a 3D cylindrical scatterplot while tracking a mobile chair, and DebugAR [23], which uses Augmented Reality (AR) for debugging.

As to immersive toolkits, the DXR toolkit [24] offers support for building immersive visualizations, and does not mention events nor streams. IATK [25] is another immersive analytics toolkit, whereby events, messages, and streams are not mentioned nor addressed. Stream [26] uses head-mounted AR devices to support visual data analysis. Spatially-aware tablets are used for interaction and input. In contrast, our solution does not necessitate additional AR hardware or a real tablet, since a virtual VR tablet is provided. Furthermore, our solution does not require or utilize individual linked 2D scatter plots. This would potentially impede scalability depending on the connectedness and grouping of the nodes involved.

Reactive Vega [27] is a streaming dataflow architecture that supports declarative interactive visualization. Its architecture and parser are implemented in JavaScript, and intended to run in a web browser or with Node.js. Popular tools for visualizing event systems, such as Kafka and RabbitMQ, include the web applications Grafana and Kibana, or some tool implementation in combination with D3.js.

In the area of visualizing SA in VR, Zirkelbach et al. [28] integrate VR with ExplorViz for a web-based live trace analysis within a single application utilizing a 2D landscape and a 3D city metaphor; it does not directly visualize static dependencies nor external communication. IslandViz [29] visualizes OSGI-based software and its dependencies in VR using an island metaphor; it does not address dynamic aspects. BabiaXR [30] visualizes CodeCity in VR using a city metaphor; it does not explicitly show dependencies or dynamic aspects. Immersive Software Archaeology [31] utilizes solar system and city metaphors to visualize horizontal and vertical (abstraction) relations in VR; it does not address dynamic aspects.

In contrast to the above, VR-ISA provides a VR-based immersive generic (application and service independent, event platform independent, and programming language independent) visualization approach, elucidating three VR viewpoints for VR-based ISA support regarding: dynamic runtime behavioral aspects as components and connectors involving events and data streaming; static internal structural aspects, such as modules and dependencies; and operational execution aspects, such as code traces and observability.

III. Solution

VR is a mediated simulated visual environment in which the perceiver experiences telepresence. VR provides an unlimited space for visualizing a growing and complex set of models and processes and their interrelationships simultaneously in a spatial structure. As the importance, scale, inter-dependence, and coupling of software, data, and events for IT infrastructure grows, and reasoning about their interactions, an immersive environment can provide an additional visualization capability to comprehend and analyze an ISA, from both the structurally complex and interconnected static relations and the dynamic behavioral interactions between digital elements such as data, events, and traces.

Support for our approach for using VR for ISA type tasks can be gleaned from work done in related areas. For instance, regarding possible benefits of an immersive VR experience vs. 2D for performing an analysis task, Müller et al. [32] investigated a software analysis task that used a Famix metamodel of Apache Tomcat source code dependencies in a force-directed graph. They found that VR does not significantly decrease comprehension and analysis time nor significantly improve correctness (although fewer errors were made). While interaction time was less efficient, VR improved the UX (user experience), being more motivating, less demanding, more inventive/innovative, and more clearly structured. The empirical study by Narasimha et al. [33] for a collaborative information architecture design task, determined that the usability of VR was significantly higher and felt more productive and enjoyable, while the quantitative and qualitative data support that VR did not perform worse than in-person or video screen-sharing. Furthermore, the empirical study by McGuffin et al. [34] found that path tracing was less error-prone in 3D vs. 2D, that VR vs. physicalized showed no difference in error rates, and users preferred VR.

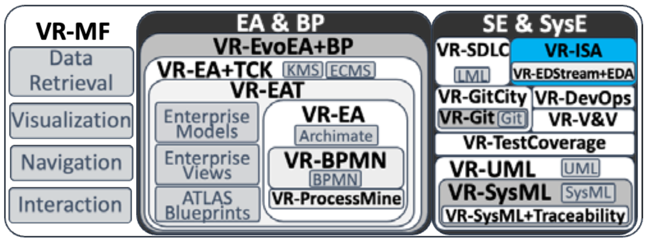


Figure 1. The VR-ISA solution concept (blue) in relation to our prior VR solution concepts.

To provide a context and background for our solution concept for SA, we position our V-ISA solution (marked in blue) in relation to our other prior VR concepts in Figure 1. VR-ISA (shown in blue) utilizes our generalized VR Modeling Framework (VR-MF), described in [14], which provides a VR-based domain-independent hypermodeling framework, which addresses four primary aspects that require special attention when modeling in VR: visualization, navigation, interaction, and data retrieval.

Our VR-based solutions specific to the SE and Systems Engineering (SysE) areas include: VR-SDLC [35], which supports immersive VR visualization of the Software Development LifeCycle (SDLC) and uses the Lifecycle Modeling Language (LML); VR-EDStream+EDA [1] is extended by this paper and addresses VR-based EDA and event and data stream visualization; VR-DevOps [36] supports VR-based visualization of DevOps pipelines; VrR-V&V (Verification and Validation) [37], for visualizing aspects related to quality assurance; VR-Git [38] and VR-

GitCity [19] supporting different visualization modes for Git repositories in VR; VR-TestCoverage [39] for visualizing in VR which tests cover what test target artefacts; VR-UML [17] supports UML; VR-SysML [18] supports SysML; and VR-SysML+Traceability [40] adds traceability.

In the Enterprise Architecture (EA) and Business Process (BP) space (under EA & BP in Figure 1), we developed VR-EA [14] to support mapping EA models to VR, including both ArchiMate as well as BPMN via VR-BPMN [12]; VR-EAT [15] adds enterprise repository integration (Atlas and IT blueprint integration); VR-EA+TCK [16] extends these capabilities by integrating further enterprise knowledge, information, and content repositories such as a Knowledge Management Systems (KMS) and Enterprise Content Management Systems (ECMS); VR-EvoEA+BP [41] adds EA evolution and Business Process animation, while VR-ProcessMine [13] supports process mining in VR.

A. Visualization in VR

Rather than attempting a one-size-fits-all view, our solution concept utilizes different forms of visualization for the different types of information and associated context. We refer to the well-known 4+1 View Model [42] as a way of portraying key views for SA, namely: logical, process, physical, and development, with scenarios as an overarching view. Note that the original article states the views are not fully independent. Our prior VR-UML and VR-SysML work can portray such 4+1 views in VR for UML or SysML diagrams via our hypermodeling capability, when those diagram types exist and are desired by the stakeholders. However, this would typically be the case when a model-first forward-engineering approach was used, or tool-generated diagrams from code artifacts when a reverse-engineering approach was used. However, in this paper we are focused on a data-first ISA approach that is independent of specialized notations (such as UML - to make it accessible to various stakeholders), while extracting data related to both operational and logical aspects of the SA from artifacts, to give us a true data-driven depiction of reality. This can also be viewed as a form of SA extraction, recovery, or archeology. Furthermore, we focus on areas where VR can provide some visualization advantages, due to its large unlimited space. Thus, for instance, we focus on support traces in VR, which can quickly become quite complex, yet we do not highlight metric or log file support (further observability pillars), which could readily be viewed with existing two-dimensional (2D) web-based mechanisms. Such 2D data could still be accessed within VR using our VR-Tablet concept that includes a web browser.

Architectural viewpoints are generic and provide conventions for constructing and using a view, whereas views are specific to a certain system architecture. Just as there are various 2D diagram types in UML that each can be used in for different views depending on the stakeholder concerts, in VR many visualization concepts for each view are feasible for an ISA. Thus, the scope of our solution concept and realization prototype will focus on illustrating VR support for three viewpoints (summarized in Table I), each of which is associated with one or more *4+1 view type(s)*:

- The distributed **Components and Connectors VR Viewpoint** (VR:VP:CompConn) for *process views* or *logical views*, typically involving runtime components and connectors, addressing *Concern:CompConn*. It addresses stakeholder concerns regarding dynamic(distributed or remote) communication and interaction, particularly event- and/or data- stream processing, workflow or pipeline processing, or network topology by depicting streams of events and/or data between producers and consumers (e.g., between microservices, data services, or an event bus).
- The **Modules and Dependencies VR Viewpoint** (VR:VP:ModDep) for *development views* or *logical views*, addressing *Concern:ModDep*. This addresses stakeholder concerns regarding the internal static structural organization of the software codebase and packages or functional decomposition by depicting element grouping / clustering and intra-dependencies.
- The **Execution Observability VR Viewpoint** (VR:VP:ExOb) for *process views* or *physical views*, addressing *Concern:Observability*. This involves stakeholder concerns regarding (internal software) operational (i.e., runtime) deployment insights into (distributed) code tracing, metrics, and event logs involving the operational deployment of processes, threads, and time-synced spans (logical units of work), which can be used to support debugging, root cause analysis, performance analysis, etc. The viewpoint lends support towards insights into operational aspects.

TABLE I. VIEWPOINT DEFINITIONS

Viewpoint (VP) name	***Components and Connectors***	***Modules and Dependencies***	***Execution Observability***
VP ID	VR:VP:CompConn	VR:VP:ModDep	VR:VP:ExOb
Viewpoint type	Dynamic operational	Static structural	Dynamic Deployment Execution
Posssible 4+1 View(s)	Process and/or Logical	Development and/or Logical	Process and/or Physical
Primary Stakeholders	Developers, Maintainers	Developers, Maintainers	Developers, Maintainers
Example Secondary Stakeholders	Testers, IT Admin, Auditors, Microservice or Data Consumers / Providers, etc.	Testers, Auditors, Quality Assurance, etc.	Testers, DevOps, Quality Assurance
Concerns	*Concern:CompConn* Monitoring remote (event, data) communication and processing workflows, producers & consumer topology & interaction	*Concern:ModDep* Code organization, modularization, dependencies	*Concern: Observability* Deployed processes, threads, operations, workflow, root cause analysis, optimization
Modeling technique	3D nexus sphere surface layered with colored interconnected balls (sources, sinks) animating time-based event/data capsules between producers and consumers	3D glass boxes representing (sub-)modules of colored linked balls (code element dependencies)	Hierarchically-stacked colored 3D blocks representing traces of time-based spans (associated with processes, threads, ops)

1) VR Viewpoint: Components and Connectors (VR:VP:CompConn)

This VR viewpoint provides a generic operational portrayal of streams of events or data (records or packets) as (distributed or remote) communication or interaction at (external) interfaces between producers (sources) and consumers (sinks). For this, a Directed Acyclic Graph (DAG) of nodes (sinks or sources) is utilized as shown in Figure 2. Note that events (messages) might be grouped and stored in topics, which are accessible to multiple producers and / or consumers.

In VR:VP:CompConn, this DAG is visualized as a nexus of elements (nodes) as 3D balls laid on the surface of a 3D sphere, while 3D empty pipes are used for the edges (interaction), and 3D capsules in the pipe portrays events or data records, which are dynamically animated within the pipe.

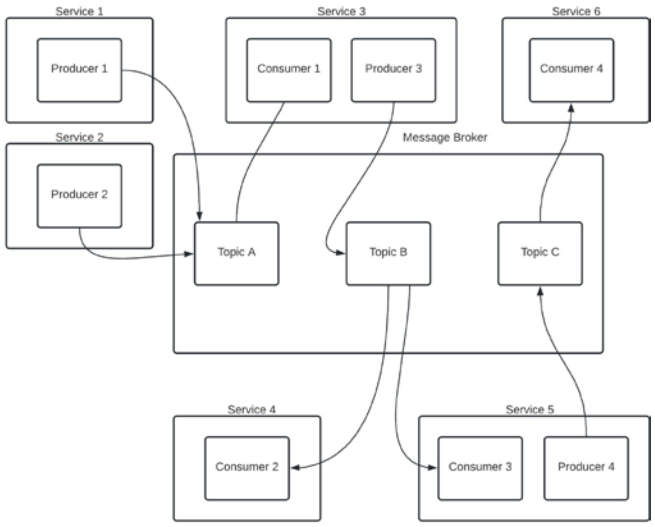


Figure 2. Example EDA couplings between services.

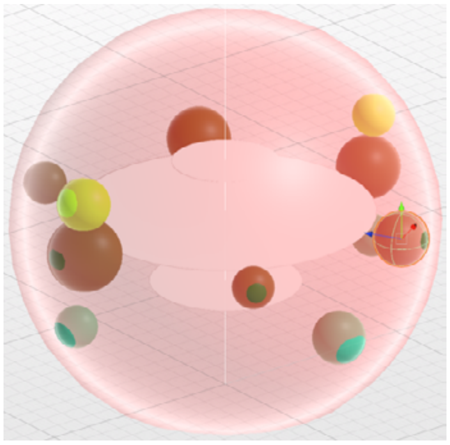


Figure 3. Nexus node placement on spherical edge aligned to planar circles.

For the layout of the DAG in VR, in the immersive space of VR navigation efficiency can affect analysis efficiency. Thus, we chose to initially place objects in relative proximity to each other to mitigate such delays. While a force-directed graph rebalances the distance of object automatically, it takes time to reach a steady state and can be distracting. Inspired by 2D chord diagrams used in visual data analytics, we considered how to use the third dimension to reduce clutter, reduce connector collisions, and retain order and legibility while supporting scalability. Using a *nexus*, nodes are initially placed on the outer edge of an imaginary sphere, while node groups follow along a planar circle on the sphere's edge as shown in Figure 3. Nodes can be optionally grouped in the configuration, in which case the largest sized group (based on number of nodes) is placed near the equator and serves as the basis for the sphere circumference, while smaller groups are placed accordingly closer to the poles. This grouping thus creates an implicit layering effect. Nodes in the same group have the same color, and the size of a node (sphere) is dependent on the number of connectors (streams).

To depict a stream, transmission, or processing of events or data in VR, a semi-transparent tube is used with nodes portrayed as spheres on both ends, and an animated capsule indicating the direction of source and sink, shown in Figure 4.

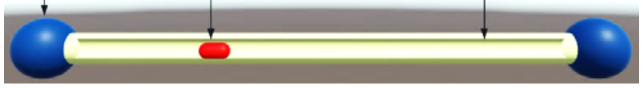


Figure 4. Event stream portrayal in VR: nodes as spheres (left arrow), semitransparent tube as stream (right arrow), and animated capsule as event (middle arrow).

2) VR Viewpoint: Modules and Dependencies (VR:VP:ModDep)

This VR Viewpoint addresses the structural aspects of software regarding modularization by visualizing the (de)composition of modules and internal structural dependencies.

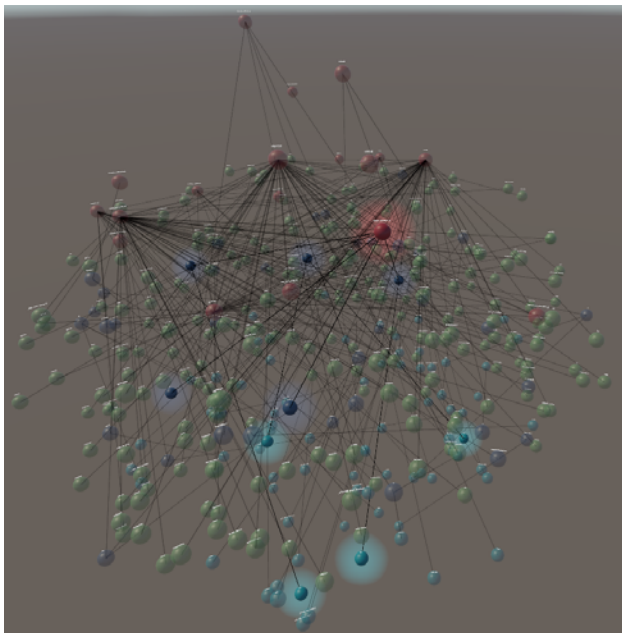


Figure 5. Nexus-based alternative view of modules and dependencies for a small project (Python Aspects sample project, 41 classes, 2 KLOC).

Initially, as a generic approach to viewing modules and dependencies, a graph-based approach as a 3D nexus was considered, as shown in Figure 5. However, modules and dependencies typically relate to some internal structural order, often known as a *development view*, and potentially related to the logical architecture. Developers may follow some intrinsic or predetermined structural and modularization order in allocating files to folders or directories, allocating classes to files, and the methods (associated with classes) or functions (independent of classes) to certain files. Although the software instruction stream invoked as a binary (or script) and does not actually concern itself about various original code pieces, source code locations, and how neatly they were modularized or what architecture was intended, developers do.

Structures can also be seen as a form of communication between minds that affects comprehensibility. Since the focus is on *informed* SA, we minimize the assumptions about modularity and associations (or interchangeably referred to as dependencies), and rather base it on the actual data available. Thus, in this case no diagrams or other documents about intents and principles are consulted, but rather the facts as extracted (reverse engineered) by static analysis tools. The concept of (sub)containment and encapsulation becomes relevant as a possible way to deal with granularity, details, and complexity. While the aforementioned modularization terms can be understood differently in various contexts, by *module* we mean some grouping or clustering at whatever granularity is provided by some (static analysis) extraction tool as input. Depending on the programming language, developers might make the "modules" explicit (such as declaring an element as belonging to a package, module, or component) or it may be discovered by a tool based on, for instance, file granularity and directory paths.

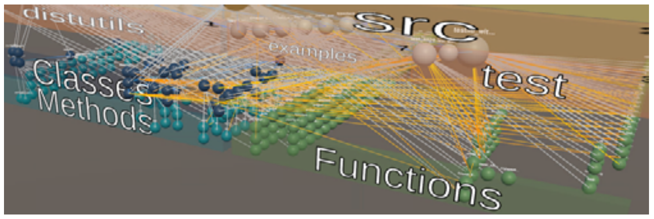


Figure 6. Logical view depicting modular containment and element dependencies (Python Aspects sample project, 41 classes and 2 KLOC).

In VR:VP:ModDep, a DAG is also utilized, whereby elements are visualized as nodes (3D spheres) colored by type (functions=green, methods=light blue, classes=dark blue, files=white) and grouped by type and module, as shown in Figure 6. The sphere size indicates number of associations relative to other nodes (larger spheres having more). The project's *hierarchical directory structure* is used as an organizing schema of *layers*, depicted via colored labeled boxes from highest to lowest (colored from yellow to darker orange as the hierarchy becomes deeper). File nodes are then positioned at both the vertically and horizontally appropriate box level. These colored layers act as both a legend (provide directory names) and provide a placement grouping and ordering. As a metric, the number of elements contained a directory is indicated in the upper right corner of a directory rectangle. We chose not to use encasing transparent colored 3D boxes as layers, as the coloring would interfere with visual differentiation of other elements and types, since alone their geo-placement in space already provides the intended information. Coloring of only box edges of a layer was also considered but rejected, since it only added additional visual clutter when viewing dependencies, which consist of lines also. On the top right of the layers, overall project metrics are provided for quick quantitative assessment or confirmation of the scope of what is being visually depicted.

As the actual software binary execution is uninterested in the original file location, we group classes, methods, and functions within their type. However, to indicate affiliation (relation to its residing source location), non-communication affiliations known as *connections* (white lines) are used: a method to the class it belongs to, or a class or function to the source file in which it resides. Since arrow shapes would add additional visual clutter, directed graph edges between nodes (lines) indicate their direction by color, with the source darker and the target lighter. point to the direction of element dependency with aqua color end representing the "to" or target and dark blue end the source of the line. Red lines are used to indicate bidirectional (circular) dependencies, since these are usually not desirable. *Dependencies* (in classes, methods, and files) are visualized as blue lines (dark blue as the source to aqua as the target). *Calls* are shown (also darker to lighter), the caller in orange and the callee target in yellow.

To reduce the amount of crisscrossing or collisions with dependencies, proximity is utilized in the placement of elements. Within its layer and type, an element is placed closer to the location of dependencies in another layer. For example, if a function is associated with a file in a directory that is towards the left, that function element is placed on the left side of the functions, and vice-versa.

3) VR Viewpoint: Execution Observability (VR:VP:ExOb)

This VR viewpoint focuses on visualizing dynamic behavioral execution trace information (typically application internal) regarding (internal) operational runtime deployment and execution behavior. This information is used to better understand how the software is functioning, e.g., to confirm its health or in support of optimization or debugging. In contrast to VR:VP:CompConn, it is more concerned with internal software information, and not necessarily directly related to intended, external communication.

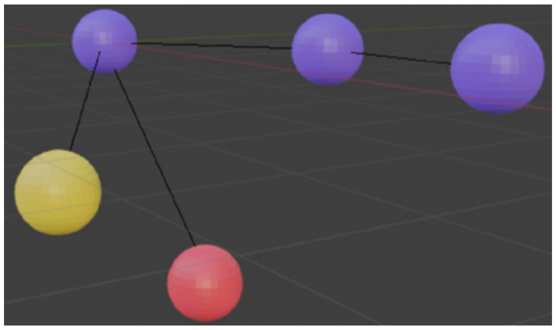


Figure 7. Process view with tree graph for trace span parent-child relations.

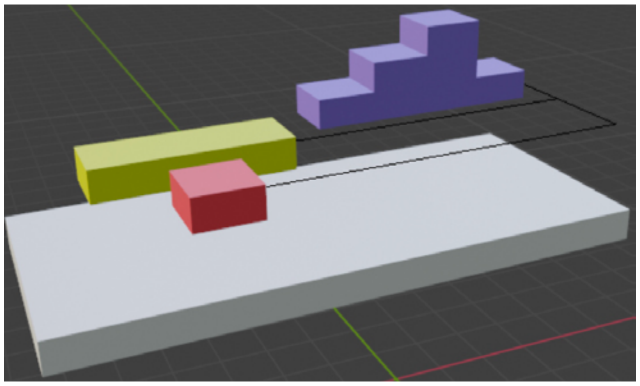


Figure 8. Process view of trace spans aligned to time axis.

Also based on a DAG, this VR viewpoint depicts stacked spans (of processes, threads, operations) relative to their deployment on some physical node (CPU). Spans represent logical units of work that can be nested, with each having an operation name, start time, and duration. A trace is some data or execution path, and can be viewed as a DAG of spans. In order to describe the relationship of parent to child spans, we utilize a 3D DAG of relations as shown in Figure 7. The trace information can also be viewed aligned in relation to time, as shown in Figure 8. Here, the grey area serves as the timeline base, above which spans (e.g., of different threads) can be located. Color is used to differentiate processes. E.g., the red and yellow spans occur at the same time but in different threads; the blue spans are executed within the same thread and at a different timepoint from the other threads. The lowest of the blue spans also acts as the root or parent span of the child spans above it. This span also produces the red and yellow spans, which is indicated via the black connecting lines.

B. Interaction in VR

Elements can be freely moved via drag-and-drop to support analysis. Where appropriate, an affordance as a ball in the corner of an object can be used to drag or to collapse / expand an element. Since interaction with VR elements has not yet become standardized, in our VR concept, user-element interaction is handled primarily via the VR controllers in combination with a virtual tablet. Our VR-Tablet concept provides detailed context-specific element information, and can provide a virtual keyboard for text entry fields (via laser pointer key selection), as seen in Figure 9.

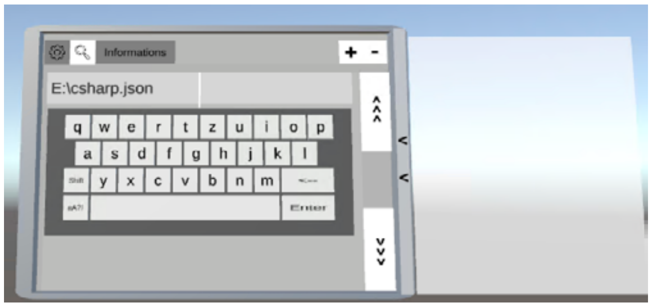


Figure 9. VR-Tablet showing a virtual keyboard and possible search query results on optional extended plane on right.

C. Navigation in VR

The immersion afforded by VR entails addressing how to navigate the space while reducing the likelihood of potential VR sickness symptoms. Thus, two navigation modes are included in the solution: the default uses gliding controls, enabling users to fly through the VR space and view objects from any angle they wish. Alternatively, teleporting permits a user to select an element (via a VR controller or by selecting an item of interest on our VR-Tablet) and be instantly placed there (i.e., by instantly moving the camera to that position); while this can be disconcerting, it may reduce the susceptibility to VR sickness for those prone to it that can occur when moving through a virtual space.

IV. Realization

As a realization of our solution concept, our prototype is inspired by the hexagonal architecture pattern (a.k.a. ports and adapters). It is partitioned into a common Data Hub, which supports various Extract-Transform-Load (ETL) adapters for various input formats from the associated tools and offers (REST) APIs and attached data storage appropriate for the data type. The VR frontend is implemented with Unity, accessing the Data Hub to retrieve data.

A. VR Viewpoint: Components and Connectors (VR:VP:CompConn)

For this VR viewpoint, our prototype realization provides a tool-independent network-based mechanism for monitoring and collecting data or events (connectors) from endpoints (components). To support collecting JSON events or data records generically - independent of a specific tool, a Web API-based microservice was implemented in Python using the FastAPI web framework. In addition to our REST interface, Telegraf (part of InfluxData platform) offers an open-source server-based agent written in Go for collecting and sending metrics and events from databases, systems, and sensors to InfluxDB. Either interface can be flexibly used to extract or collect events, applying an interceptor, proxy, or decorator pattern as appropriate.

Integration with two different event systems was performed. Apache Kafka is an open-source distributed event streaming platform. Kafka Connect supports data integration between databases, key-value stores, search indexes, and file systems. The connectors receive and transmit data to and from topics as a source or sink, and various extensible implementations are available (e.g., a Source Connector that streams database updates to a topic, collects server metrics to a topic, forwards topic records to Elasticsearch, etc.).

As to storage in the Data Hub, the InfluxDB was used as a database due to: 1) its time series support and 2) since its storage requirements were deemed significantly smaller for large time series datasets than the alternatives, a benefit when scaling the solution. Metainformation collected via REST or Telegraf and retained in the database with each record are as follows: source, target, timestamp, payload. Thus, the payload can be data, an event, a message, etc. If no target exists, then any null or fake named node can be used (equivalent to a null device in Unix).

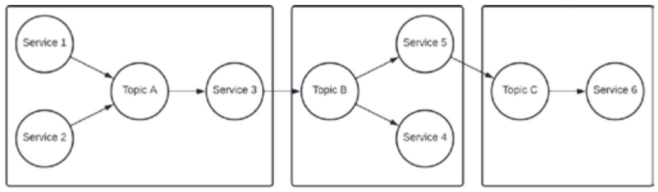


Figure 10. Abstracted node grouping EDA example.

Configuration information in JSON can be stored and loaded using the VR-Tablet, enabling stakeholders to tailor the grouping, placement, and coloring of nodes and streams based on their concern or interest. An example cross-service EDA is shown in Figure 10. Nodes in a group are assigned the same color. In the VR-Tablet, the relevant event flow time period can be selected and event flow steps and speed can be dynamically controlled.

B. *VR Viewpoint: Modules and Dependencies (VR:VP:ModDep)*

To support the realization of this viewpoint for ISA, static code analysis tools can provide information on modules, dependencies, and metrics. However, each tool usually supports only certain programming languages. Furthermore, there is a lack available (adopted) standards for data access or export from such tools, so any data extraction, when even supported, is tool-specific. To minimize tool dependencies, we use an adapter and JSON transformation approach to integrate extracted data into our data hub.

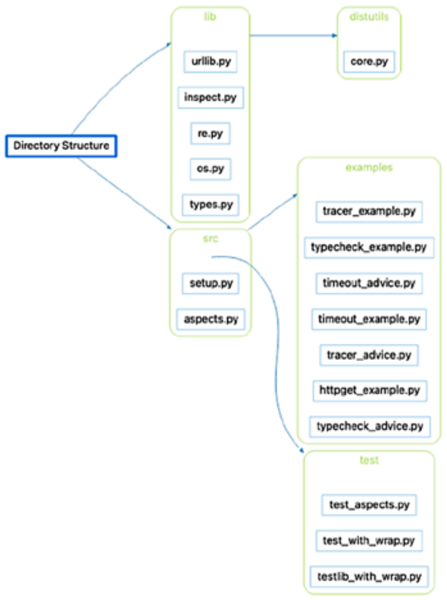


Figure 11. Example Dependency Graph Diagram in Understand for small project (Python Aspects).

To exemplify our solution concept, our prototype realization integrates the Understand tool by Scientific Toolworks, Inc. It offers static analysis support for multiple languages including C/C++, C#, Java, JavaScript, Python, etc., and offers APIs and various visualizations (UML, dependency graphs, control-flow graphs, call tree graphs, butterfly graphs). An example dependency graph is shown in Figure 11. Among its graph variants, it offers an Architecture Dependency graph with focus on dependencies, and a Graph Architecture view that depicts the structure of the architecture, with clustering granularity that can be varied across function, class, file, or architecture level.

In support of the VR:VP:ModDep viewpoint, the solution was realized as follows. Understand is run in a separate Docker container to utilize the Python environment required with Understand and avoid certain runtime issues using its APIs for information extraction. The data retrieved from the Understand APIs was transformed into our JSON format, a sample of which is shown in Figure 12.

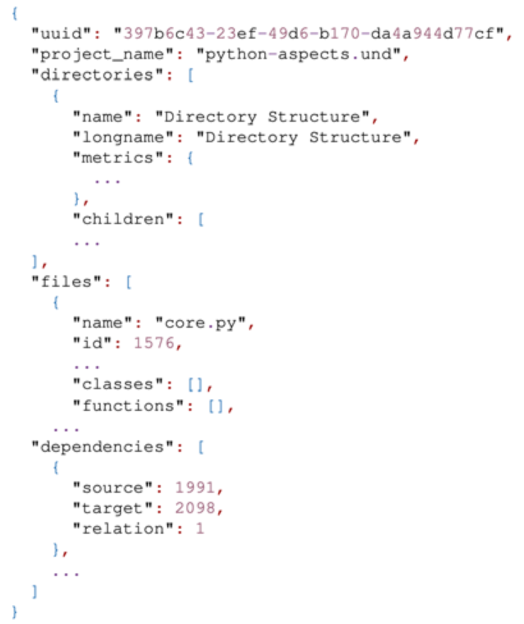

```
{
  "uuid": "397b6c43-23ef-49d6-b170-da4a944d77cf",
  "project_name": "python-aspects.und",
  "directories": [
    {
      "name": "Directory Structure",
      "longname": "Directory Structure",
      "metrics": {
        ...
      },
      "children": [
      ...
  ],
  "files": [
    {
      "name": "core.py",
      "id": 1576,
      ...
      "classes": [],
      "functions": [],
    ...
  "dependencies": [
    {
      "source": 1991,
      "target": 2098,
      "relation": 1
    },
    ...
  ]
}
```

Figure 12. Snippet of structural information from Understand as JSON.

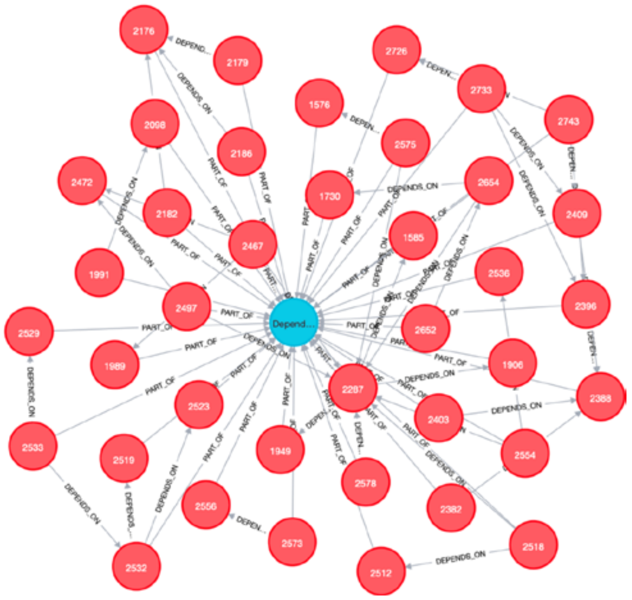


Figure 13. Dependencies stored in our Data Hub in a Neo4j database.

Data related to directories, files, and metrics is stored in a Docker-based MongoDB database, whereas graph-related data such as dependencies is stored directly in a Docker-based Neo4j database, a sample of which is shown in Figure 13. Separating data across two database types was done initially to ensure full flexibility for storing unstructured JSON data from various tool types, and which thus might include various other data such as metrics, etc., yet enabling us to leverage the Neo4j graph database capabilities for graphs such as dependencies. Note that the use of two database types is not necessarily required, but related to assumptions made at the beginning of the realization; consolidation to a single database type such as Neo4j could be considered.

C. *VR Viewpoint: Execution Observability (VR:VP:ExOb)*

For a prototype realization of this VR viewpoint, the distributed tracing platform Jaeger was chosen to collect trace information from various clients. Jaeger offers a timeline visualization (see Figure 14), a tree diagram that depicts span relationships (Figure 15 top), and the raw trace data in JSON (Figure 15 bottom). For implementing tracing in client code, the OpenTelemetry and OpenTracing libraries were used, and clients can use either the library APIs directly or available annotations, as exemplified in Figure 16. Jaeger agents are network daemons that listen for spans, which are batched and sent to collectors. Jaeger collectors can persist these or pass them on to Kafka.

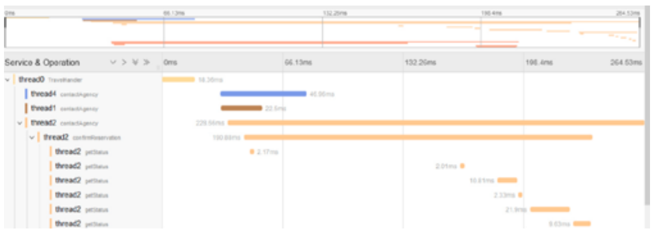
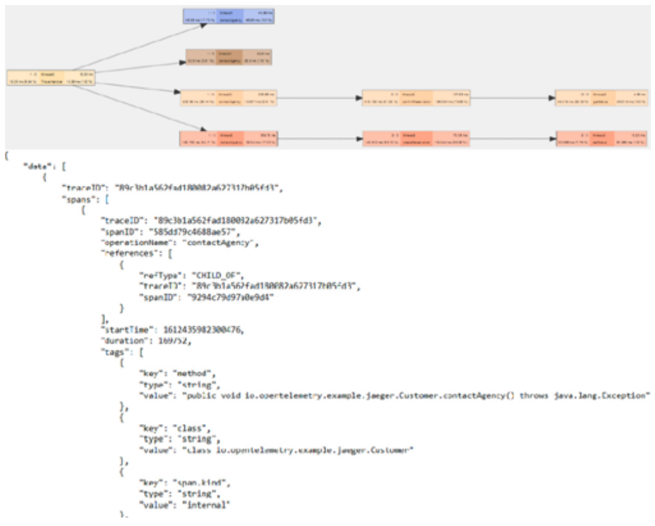
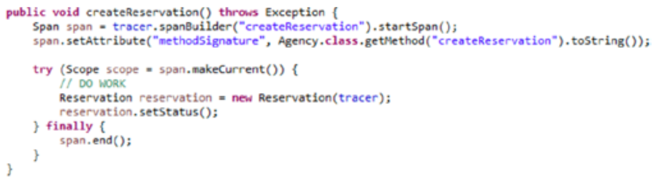


Figure 14. Screenshot of spans in Jaeger's trace timeline visualization.



```
{
    "data": [
        {
            "traceID": "89c3b1a562fad180082a627317b05fd3",
            "spans": [
                {
                    "traceID": "89c3b1a562fad180032a627317b05fd3",
                    "spanID": "585dd79c4688ae57",
                    "operationName": "contactAgency",
                    "references": [
                        {
                            "refType": "CHILD_OF",
                            "traceID": "89c3b1a562fad180082a627317b05fd3",
                            "spanID": "9294c79d97a0e9d4"
                        }
                    ],
                    "startTime": 1612435982300476,
                    "duration": 169752,
                    "tags": [
                        {
                            "key": "method",
                            "type": "string",
                            "value": "public void io.opentelemetry.example.jaeger.Customer.contactAgency() throws java.lang.Exception"
                        },
                        {
                            "key": "class",
                            "type": "string",
                            "value": "class io.opentelemetry.example.jaeger.Customer"
                        },
                        {
                            "key": "span.kind",
                            "type": "string",
                            "value": "internal"
                        },
```

Figure 15. Screenshot of spans in Jaeger's trace tree diagram and as JSON.



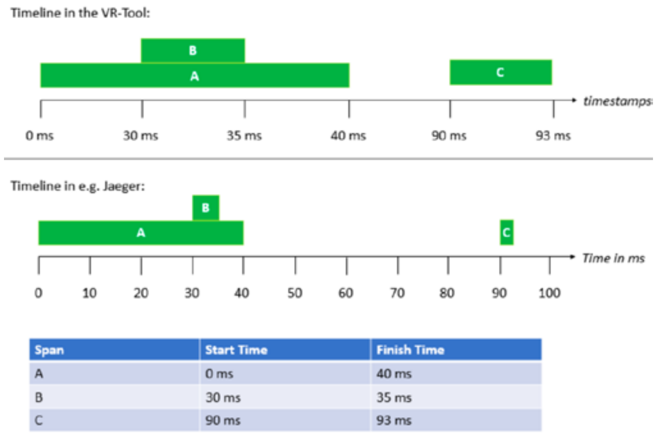
```
public void createReservation() throws Exception {
    Span span = tracer.spanBuilder("createReservation").startSpan();
    span.setAttribute("methodSignature", Agency.class.getMethod("createReservation").toString());

    try (Scope scope = span.makeCurrent()) {
        // DO WORK
        Reservation reservation = new Reservation(tracer);
        reservation.setStatus();
    } finally {
        span.end();
    }
}
```

Figure 16. Example client code snippet of OpenTelemetry span definition.

Trace results are exported from Jaeger and stored in our Data Hub. Processes are differentiated by color, thus spans in the same process share that color. Telemetry trace data is placed directly on each side of an individual span.

The timeline visualization in Jaeger depicts which spans were active when. For VR, instead of using a constant scale for the time axis, an event-sequencing with fixed-size units (blocks) of varying timescales is used, marking off the beginning or end of a span, as in Figure 17. Benefits include: 1) reduced virtual space needed for navigation while offering a better overview, and 2) concurrency, parallelism, nesting, and synchronization of active spans is highlighted and more comprehensible, rather than relative durations and possibly overlooking significant events.



Span	Start Time	Finish Time
A	0 ms	40 ms
B	30 ms	35 ms
C	90 ms	93 ms

Figure 17. Example VR:VP:ExOb trace variable timescale axis depiction (above) vs. a fixed timescale axis (below).

V. Evaluation

For the evaluation of the solution concept, we utilize the design science method and principles [43], in particular, a viable artifact, problem relevance, and design evaluation (utility, quality, efficacy). As our solution concept is focused on VR visual support for an ISA, a case study based on scenarios applicable to each viewpoint is used. An informed SA depends on the digital reality of the information provided by tooling. Hence, in contrast to explicit (UML) models that can be inconsistent with reality, this evaluation highlights our generic approaches for visualizing the data provided by the tooling in the various viewpoints. Note that in our prior work with VR-UML [17] and VR-SysML [18], we have shown our hypermodeling capability in VR, whereby such prescriptive, intended, or explicit models and associated diagrams can be portrayed in 3D in VR alongside the VR-ISA viewpoints we describe in this paper.

A. VR Viewpoint: Components and Connectors (VR:VP:CompConn)

For this VR viewpoint, which informs an ISA regarding components and connectors, our scenarios focus on generically depicting components and connectors, integration support for popular broker and streaming platforms, and VR interaction and tailoring support.

For the test applications, Confluent ksqlDB was used as a database supporting SQL queries for stream processing applications based on Kafka Streams. For generating event data for the evaluation, the Confluent Quickstart Demo using ksqlDB in combination with Kafka Connect was used with two connectors to the topics pageviews und users. A second configuration based on Confluent Kafka consisted of one producer and three consumers in Python. To ensure the solution was not Kafka dependent, a third configuration using only RabbitMQ with our microservice was also tested.

1) Event System or Streaming Platform Integratability

To test the integratability of the generic approach, a second popular publish/subscribe message broker event system, RabbitMQ, was also utilized in addition to Kafka in the evaluation. For more details and a comparison of these distributed event systems, we refer to Dobbelaere and Esmaili [44].

2) Single Large Group Connected to One Node

As a scalability scenario, a single group of 100 nodes all connected to a single node is shown in Figure 18. Note that although difficult to depict as a figure due to the limited space, in VR, due to its unlimited space, there are no actual technical limitations in visualizing, navigating, and comprehending very large models.

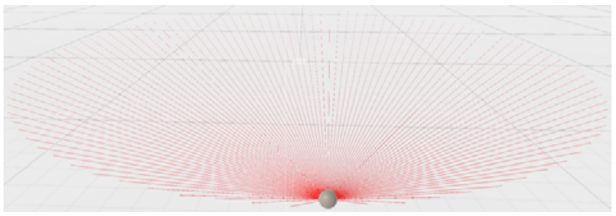


Figure 18. Scalability test: a group of 100 nodes connected to one node.

3) Unbalanced Groups Randomly Interconnected

This scenario consisted of three unbalanced groups: one group with 20 randomly intra-connected nodes, and two inter-connected groups consisting of a single node each, as portrayed in Figure 19. Note each group has a different node color, and more connected nodes are larger, and smaller groups are near the poles of the sphere, with the largest group at the equator.

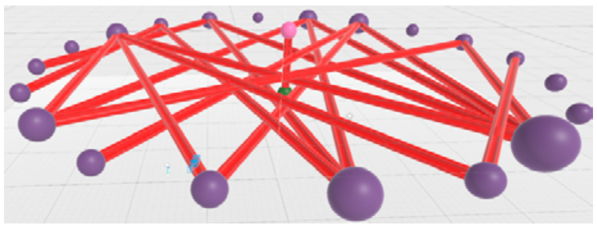


Figure 19. Three groups: one with 20 randomly intra-connected nodes and two inter-connected groups consisting of a single node each.

4) Multiple Balanced Highly Interconnected Groups

In this scenario, three balanced groups of 20 nodes each are randomly inter- and intra- connected with other nodes, as shown in Figure 20.

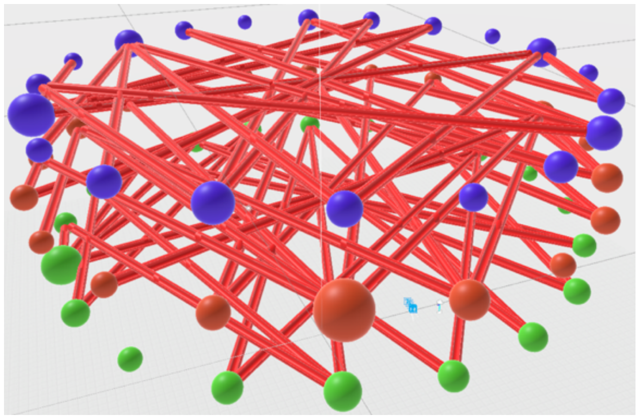


Figure 20. Three groups of 20 nodes each with random coupling.

5) Multiple Unbalanced Groups Irregularly Interconnected

To test many unbalanced groups with different degrees of connectedness, this scenario had five groups, one group with 20 nodes and the rest consisting of 5-10 nodes with random unbalanced coupling. The result is shown in Figure 21.

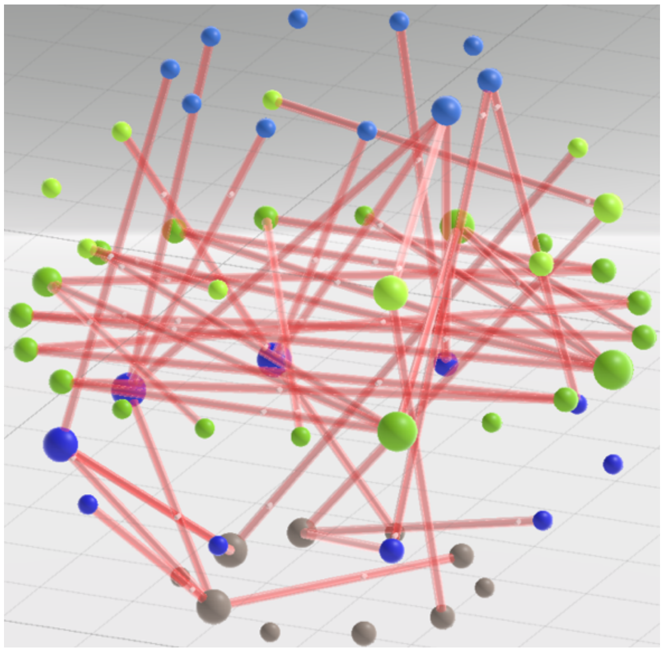


Figure 21. Five groups (with 20 and 5-10 nodes) and random coupling.

6) Interaction Support via VR-Tablet

VR interaction in this viewpoint is supported using our VR-Tablet via the following display modes:

- Animated Timeline for controlling dynamic stored or real-time playback (Figure 22 left),
- Querying the event or data store (Figure 22 right),
- Color customization (Figure 23),

- Object details for a selected node (Figure 24)
- Event or data record details (i.e., capsule, Figure 25),
- Settings for storing and fetching configurations.

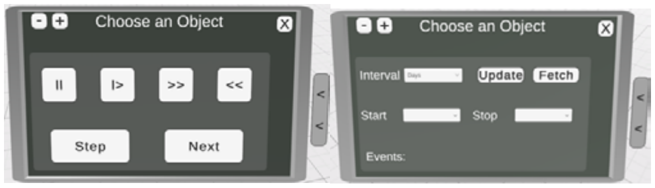


Figure 22. Dynamic animation interface (left) and Query interface (right).

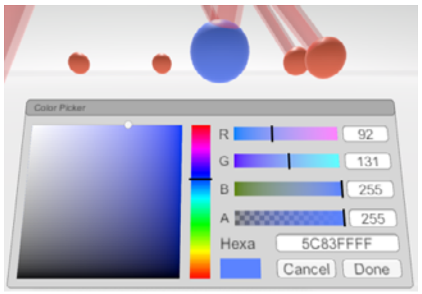
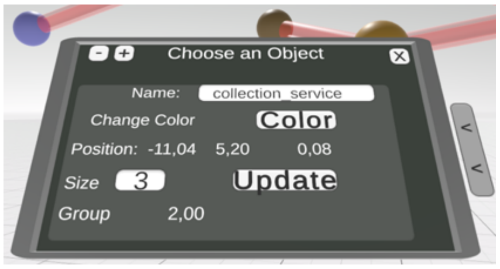
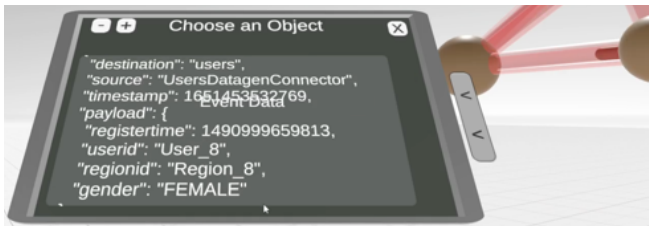


Figure 23. Object color customization.



Name: collection_service
Change Color
Position: -11,04 5,20 0,08
Size 3
Group 2,00

Figure 24. Node detail interface after selecting a node.



```
"destination": "users",
"source": "UsersDatagenConnector",
"timestamp": 1651453532769,
"payload": {
 "registertime": 1490999659813,
 "userid": "User_8",
 "regionid": "Region_8",
 "gender": "FEMALE"
```

Figure 25. Example event details after selecting the red capsule.

7) *VR:VP:CompConn Discussion*

The above scenarios used our prototype realization to demonstrate the feasibility of our generic solution concept for supporting this VR viewpoint. It can be used to simplify the understanding of inter-software communication and interactions regarding events and data streams for stakeholders, using generic components (endpoints such as microservices or stream processing steps) and generic connectors (in particular, event, message, or data flow). It by immersively depicting sources and sinks as nodes in a spatially compact (3D spherical) layout, while animating any time-based interaction between them.

In focusing only on the essential flows and communication streams for data and events, while hiding all else, it is readily scalable. By immersively visualizing and animating these key aspects, various (grassroot) stakeholders can access, experience, and comprehend the digital reality of the flow of event or data streams. The default configuration provides a starting point for any analysis, and users can tailor the views by moving and recoloring nodes, and can query datasets and timespans of interest.

B. *VR Viewpoint: Modules and Dependencies (VR:VP:ModDep)*

For this VR viewpoint, which informs an ISA regarding modules and dependencies, the scenarios focus on module and dependency depiction and VR interaction support. Programming language independence is demonstrated via two example projects provided with the Understand tool (Sokoban Pro in C# and python-aspects in Python). Understand APIs were used to extract project information to our Data Hub.

1) *Module Visualization*

For module and element visualization, labeled node types are differentiated by color: functions (green), methods (light blue), classes (dark blue), and files (white). Nodes are then grouped by type, with directories above that contain file nodes (behind the directory structure layers), function nodes (green, right bottom), and bottom left classes (dark blue) with their methods (light blue). This is exemplified for a small sample project (Python Aspects, 18 files, 2 KLOC) in Figure 26. Nodes with the most connections or dependencies are largest, and likely more significant to the architecture. The modular decomposition of files by subdirectories is depicted, whereby *examples* and *test* are subdirectories of *src*, and *lib* is parallel to *src* and contains the *distutils* subdirectory. The number of elements as a metric is shown in the legend numerically, and can be readily viewed and discerned visually relative to other elements from the side, as shown in

Rather than hiding various aspects of the reality, views in this VP initially depict all elements, to allow the stakeholder to see the relative number and location of elements. These details are often hidden and dispersed across text-based Command-Line Interface (CLI) file systems, while 2D analysis tools often must simplify and reduce the sheer number due to their limited 2D space. In contrast, our approach leverages the unlimited space of VR for a comprehensive depiction that is nevertheless ordered and can be readily filtered and explored. For instance, dependencies and connections, when not of interest, can be hidden to reduce visual clutter, as seen for a large project in Figure 51.

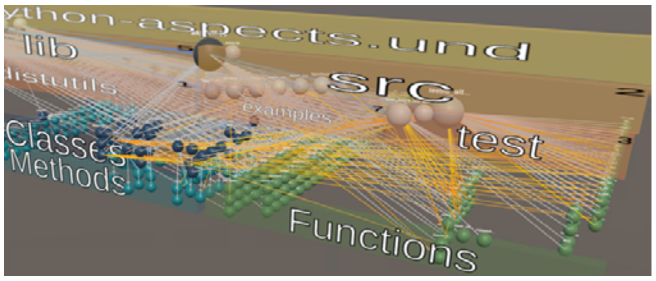


Figure 26. VR:VP:ModDep view visualizing modular containment and element connectors (affiliation), callers, and dependencies (containment) (Python Aspects sample project, 41 classes, 329 functions, 2 KLOC).

To indicate affiliation (relation to a location), *connections* (white lines) are used: a method to the class it belongs to, or a class or function to the source file in which it resides.

To evaluate the scalability of our solution concept and prototype, a large sample project (GitAhead C++, 444 files, 496 classes, 9747 functions, 252 KLOC) was used. As to modularization, directories with file containment, the number of files in each directory (number of spheres) can be readily discerned, and the metric is depicted in the upper right corner of each directory as shown in Figure 27. Overall project metrics shown to the right of the legend. A perspective from above without dependencies is shown in Figure 28. A full front perspective without dependencies is shown in Figure 48. A top view is shown in Figure 49. A full side perspective without dependencies is shown in Figure 50.

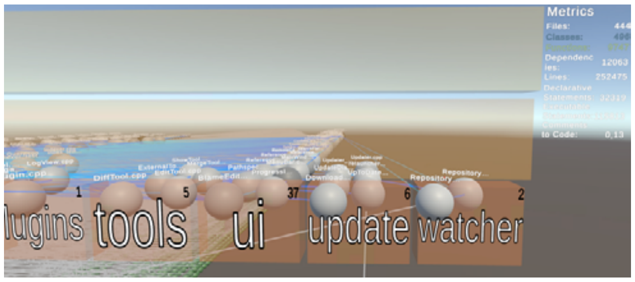


Figure 27. VR:VP:ModDep view visualizing files containment and depicting file grouping and relative number by directory (large GitAhead C++ sample project), with smaller directories shallow and larger directories, such as “ui” (37 files), deeper.

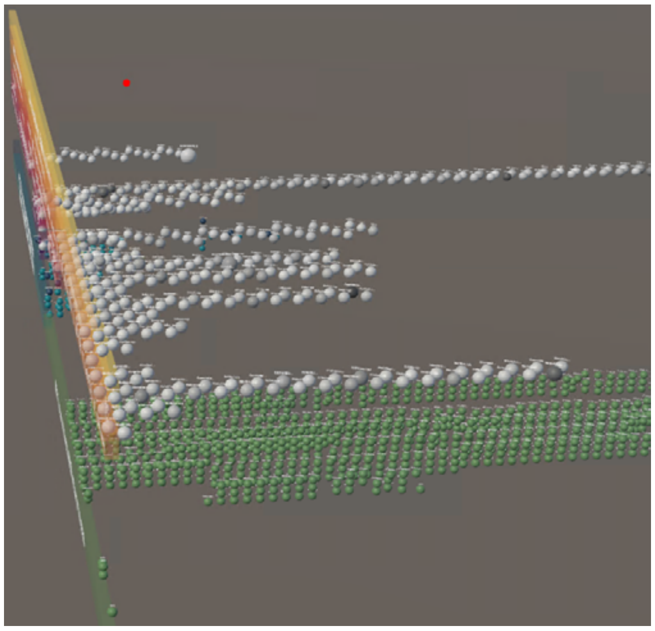


Figure 28. VR:VP:ModDep view visualizing nodes grouped by directory and type without dependencies (large GitAhead C++ sample project).

2) *Dependency Visualization*

Dependencies (in classes, methods, and files) are visualized as blue lines. For directed dependencies, we found that arrow shapes created unnecessary visual clutter; so instead, color transitions are used to indicate direction (darker to lighter), from dark blue as the source to aqua as the target. Bidirectional or circular dependencies are colored red. Calls are shown (also darker to lighter), the caller in orange and the callee target in yellow.

The Python Aspects sample project was used, containing 18 files, 41 classes, 329 functions, and 2 KLOC. A side perspective is shown in Figure 29. A rear perspective is shown in Figure 30. A top perspective shows the spacing between graph edges and nodes, as shown in Figure 31.

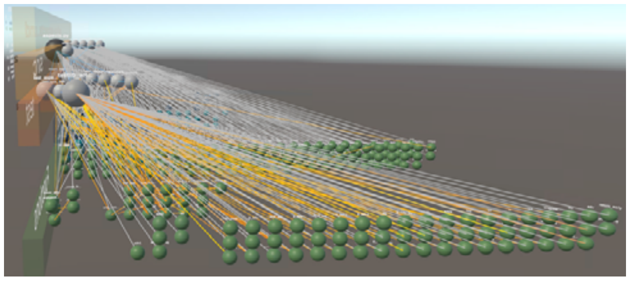


Figure 29. VR:VP:ModDep side perspective with calls (orange) and connectors (affiliations) (in white) (Python Aspects sample, 41 classes, 329 functions, 2 KLOC).

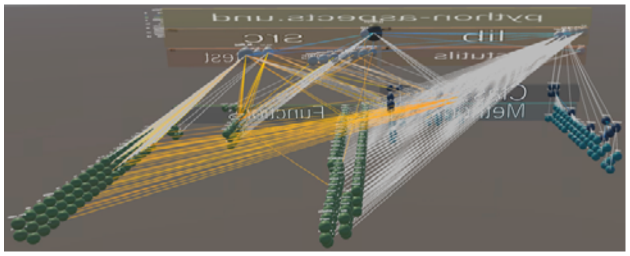


Figure 30. VR:VP:ModDep rear perspective showing element grouping placement to minimize connector/dependency crisscrossing and collisions (Python Aspects sample, 41 classes, 329 functions, 2 KLOC).

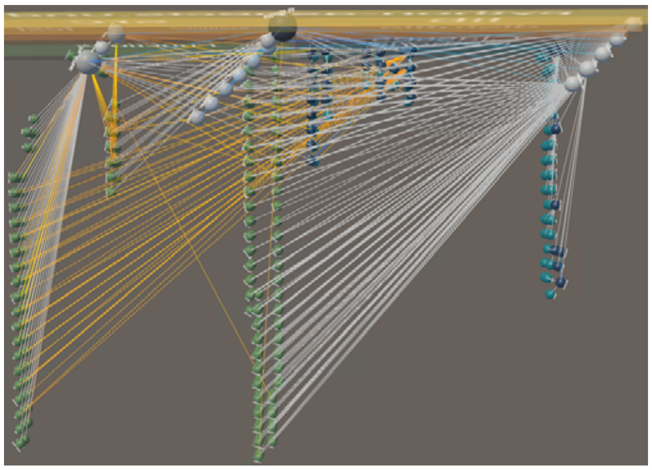


Figure 31. VR:VP:ModDep top perspective showing calls (orange) and affiliations as connectors (white) (Python Aspects sample, 41 classes, 329 functions, 2 KLOC).

To support analysis and investigation, once an element of interest is selected, it and first-degree neighbors are left colored, while other unrelated elements are ghosted, as shown in Figure 32.

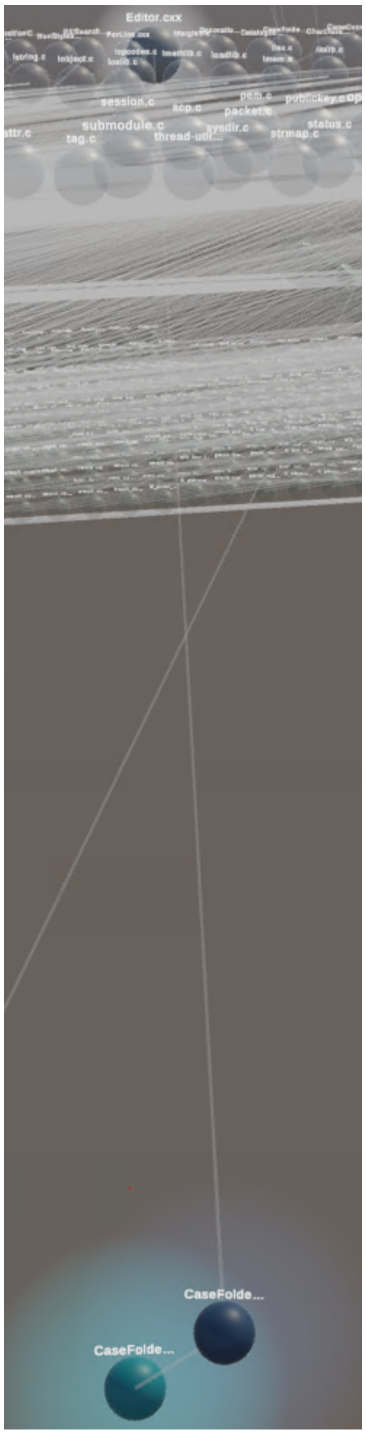


Figure 32. Analysis support for connections between a selected method element (CaseFolderASCII constructor, bottom, aqua glow), its containing class (CaseFolderASCII, dark blue), and its affiliated file (Editor.cxx, top).

With regard to the scalability of our solution concept and prototype for dependencies, the large sample project (GitAhead C++, 444 files, 496 classes, 9747 functions, 252 KLOC) was used. A front perspective with all dependencies depicted is shown in Figure 51. A side view with all dependencies depicted is shown in Figure 52.

3) *Interaction Support via VR-Tablet*

Interactive support for VR:VP:ModDep is provided by the VR-Tablet. It offers a search capability for an element of interest as shown in Figure 33. Selecting a resulting node on the result pane on the right will highlight that node. A filtering option shows or hides desired elements as shown in Figure 34.

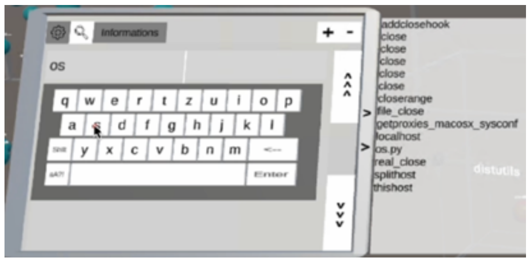
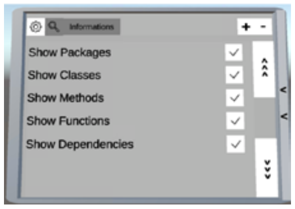

Informations
os
addclosehook
close
close
close
close
close
closerange
file_close
getproxies_macosx_sysconf
localhost
os.py
real_close
splithost
thishost

Figure 33. VR-Tablet showing virtual keyboard-based search and results.


Informations

Show Packages	✓
Show Classes	✓
Show Methods	✓
Show Functions	✓
Show Dependencies	✓

Figure 34. VR-Tablet showing filtering option settings.

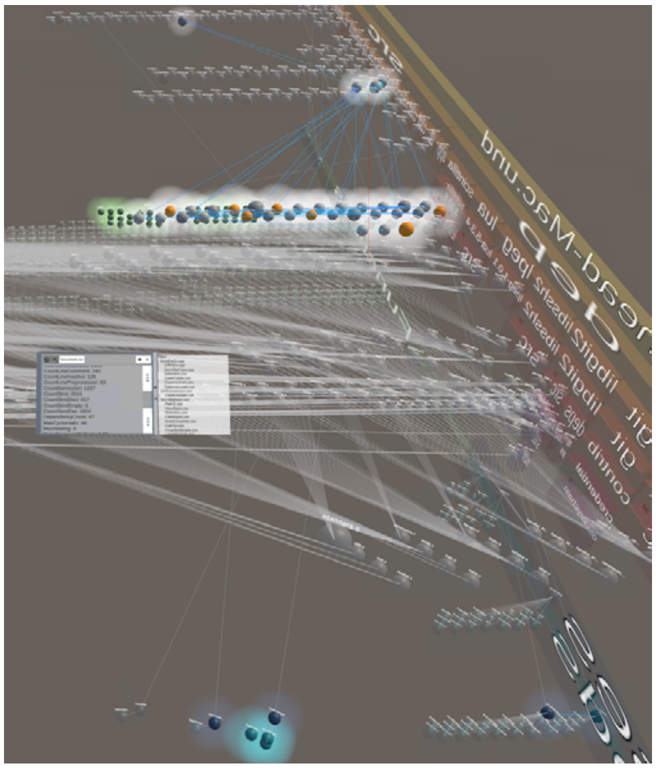
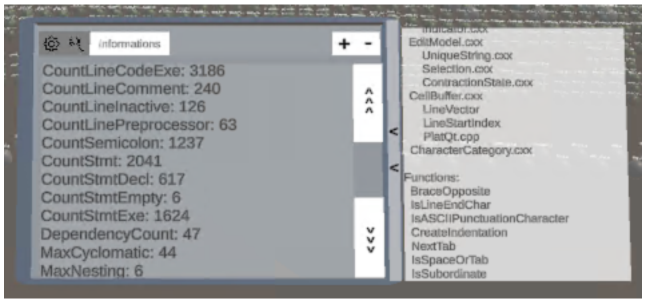


Figure 35. For a selected file node Document.cxx, the VR-Tablet (center left) displays metrics, aqua glow (bottom) highlights its classes and methods, green glow (upper left) highlights dependent functions, white glow highlights dependent files (dependencies as blue lines), and orange nodes indicating files containing called functions.

Selecting an element provides detailed contextual information on the selected element, ghosting irrelevant elements leaving its overall direct context and dependencies visible, while the VR-Tablet shows various metrics on the left, and named listed context in the right pane. Furthermore, colored glows indicate the directly associated elements. In Figure 35, a file node Document.cxx is selected, whereby the VR-Tablet displays various metrics, while aqua glow (bottom) highlights its class and method connections, green glow highlights its dependent functions (upper left), and white glow shows dependent files (dependencies as blue lines). The nodes colored orange indicate files containing functions called by the selected node. Element-relevant metrics and an extended context pane is shown in Figure 36. Teleporting functionality can rapidly navigate to a related element of interest when an element is selected in the context pane.



CountLineCodeExe: 3186
CountLineComment: 240
CountLineInactive: 126
CountLinePreprocessor: 63
CountSemicolon: 1237
CountStmt: 2041
CountStmtDecl: 617
CountStmtEmpty: 6
CountStmtExe: 1624
DependencyCount: 47
MaxCyclomatic: 44
MaxNesting: 6

EditModel.cxx
UniqueString.cxx
Selection.cxx
ContractionState.cxx
CellBuffer.cxx
LineVector
LineStartIndex
PlatQt.cpp
CharacterCategory.cxx

Functions:
BraceOpposite
IsLineEndChar
IsASCIIPunctuationCharacter
CreateIndentation
NextTab
IsSpaceOrTab
IsSubordinate

Figure 36. VR-Tablet in VR:VP:ModDep showing metrics for a selected node (left pane) and its context (right pane).

4) *VR:VP:ModDep Discussion*

The scenarios show that this VR viewpoint can provide insights to the internal structural aspects of software regarding modularization, internal structural dependencies, and internal static analysis metrics, and does so across programming languages. While the depicted images may seem difficult to discern within the limitations of such a paper, the immersion of VR permits the user to explore the various aspects. Since nothing is hidden until a node is selected, the user is aware of the nature and scope of what they are dealing with visually, not just numerically. By filtering and ghosting, specific elements of interest can be explored, without losing contextual insights.

C. *VR Viewpoint: Execution Observability (VR:VP:ExOb)*

For this VR viewpoint, which informs an ISA regarding operational and observability aspects such as execution traces, the scenarios focus on trace and span depiction and VR interaction support.

1) *Tree Graph and Timeline Visualization*

Our trace span tree graph in VR is shown in Figure 37. Our trace stacked span timeline visualization in VR, which uses a variable scale, offers two draggable cross-span timepoint plates (green for start, purple for finish) to compare active spans across two different timepoints, as shown in Figure 38. This supports concurrent trace span analysis for distributed or parallel computing, threading or concurrency issues. If no concurrency is used, then the analysis is simplified.

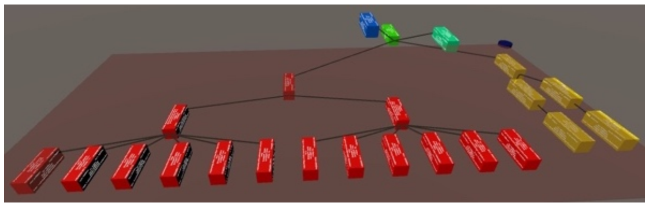


Figure 37. Tree graph in VR.

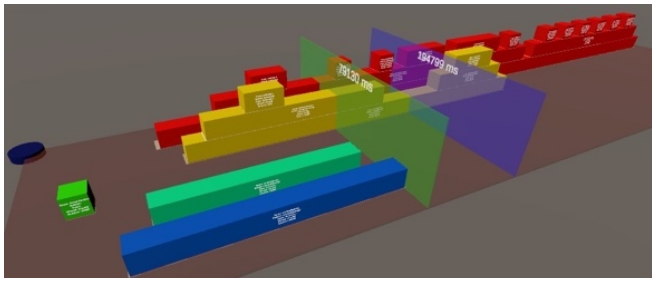


Figure 38. Timeline diagram

2) *Contextual Trace and Model Information*

Initially the view provides contextual support via an overview of the available information, with a model if available placed in the center, showing VR diagrams (such as our VR-UML or our VR-EA ArchiMate), which help provide context for the tracing information, as shown in Figure 39. Here, in the center, a stack of various VR-UML diagrams is shown with the bottom being a class diagram, while the trace information is placed as a tree graph on the left side, and a time axis representation is seen on the right.

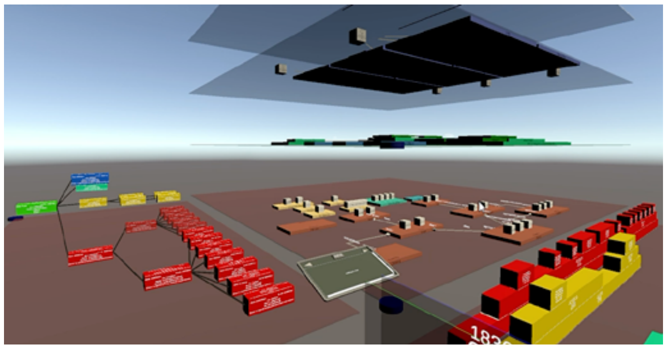


Figure 39. VR:VP:ExOb overview showing stacked VR-UML diagrams (center) with related tracing information on the sides.

3) *Span Information Depiction*

The following information is projected onto the sides of span blocks to readily provide relevant data and reduce the frequency of VR-Tablet interaction: Start timestamp, Finish timestamp, Duration, Name, (if available) name of the Method in which the span was created, (if available) name of the Class in which the span was created; this is shown in Figure 40. Further information such as Process name, Thread name, or Service name in which the span is located, is also shown on the blocks in the timeline diagram in Figure 41. This and additional detailed information can also be retrieved in the VR-Tablet by selecting a specific span.

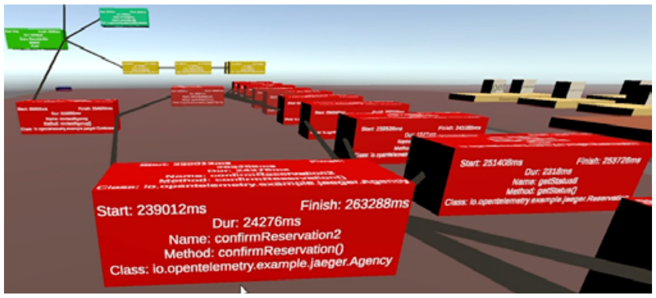


Figure 40. Trace span information shown on blocks in VR tree diagram.

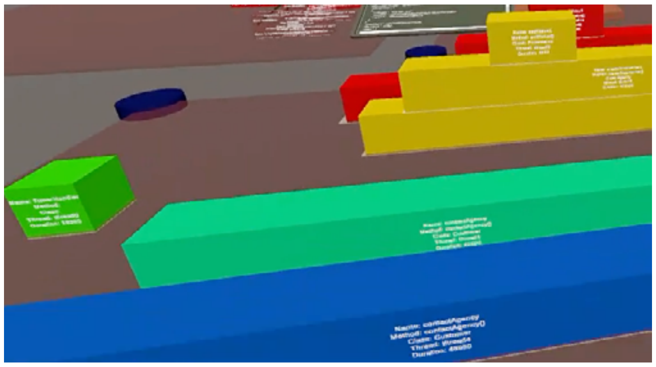


Figure 41. Span information shown on blocks in VR timeline diagram.

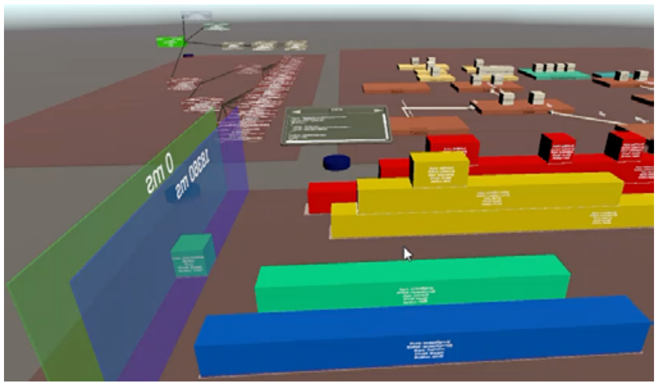


Figure 42. Active span synchronization from timeline to tree graph.

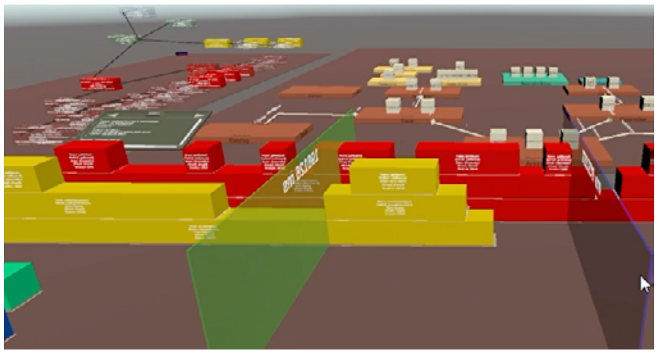


Figure 43. Synchronization from timeline to tree graph showing those spans that were active during that duration (the other spans are ghosted).

4) *Span Tree Graph and Timeline Synchronization*

Selecting a single span will ghost other inactive spans at its timepoint. The timeline and tree graph visualizations are synchronized such that moving or activating a timepoint plane will cause other non-active spans, even in the other diagram, to be ghosted. The start and end cross-plane timepoints are positioned such that only one green span was active, with the tree graph in the back top left showing a single green span and all other non-active spans ghosted, as shown in Figure 42. Between the start and end timepoint planes, all spans that were active at any time during that duration remain colored, and the rest are ghosted, as shown in Figure 43. Here, a parent yellow had two child spans at some point, and a red span had a child span, which in turn had 3 child spans during that duration. Hence, these all are colored at the different levels in the tree graph and the others are ghosted.

5) *Bidirectional Model and Trace Synchronization*

In support of trace context, when selecting a span, if it has an associated class or method, then a green line is drawn to indicate where that class or method is in the (VR-UML or VR-EA ArchiMate model), as shown in Figure 44 and Figure 45. This selection could also be reversed from method to spans.

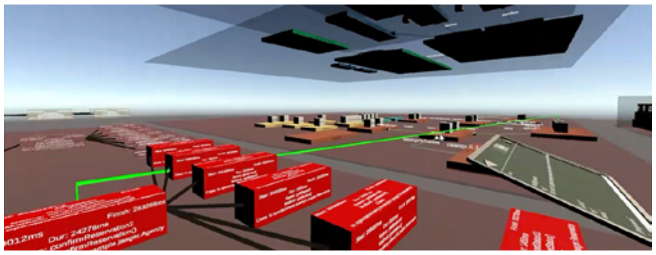


Figure 44. Selecting a span ghosts other spans and highlights (in green) the location of the corresponding method/class in the VR-UML class diagram.

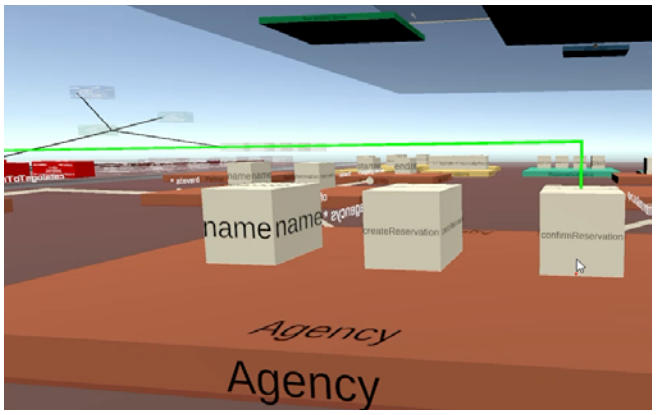


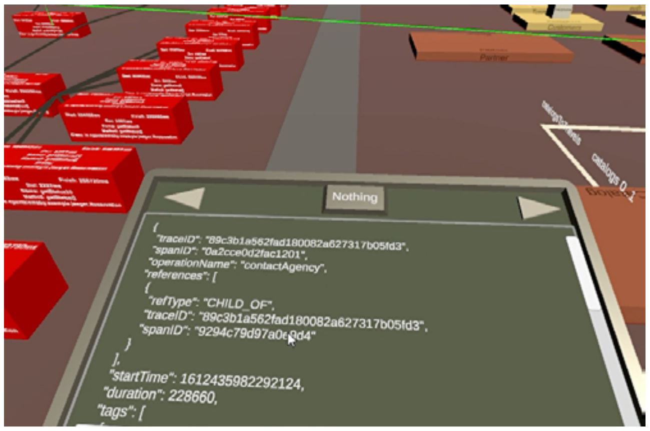
Figure 45. VR-UML class method span connector (in green).

6) *Interaction Support via VR-Tablet*

To support VR:VP:ExOb, our VR-Tablet provides the JSON raw data for a selected trace object, as shown in Figure 46. Deployment-related information is also provided, as shown in Figure 47. This information could also be used to browse and search for applicable spans.

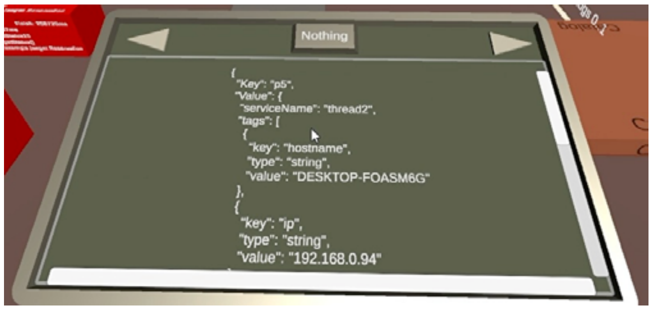
7) *VR:VP:ExOb Discussion*

The scenarios with our prototype implementation show that this viewpoint can support an ISA in VR with analysis of trace execution and spans, including their relation to VR-UML and VR-EA ArchiMate models. The prototype shows its feasibility, and could be readily extended to include other observability data such as relevant logs and metrics via the VR-Tablet.



```
{
 "traceID": "89c3b1a562fad180082a627317b05fd3",
 "spanID": "0a2cce0d2fac1201",
 "operationName": "contactAgency",
 "references": [
  {
   "refType": "CHILD_OF",
   "traceID": "89c3b1a562fad180082a627317b05fd3",
   "spanID": "9294c79d97a0e9d4"
  }
 ],
 "startTime": 1612435982292124,
 "duration": 228660,
 "tags": [
```

Figure 46. VR-Tablet showing trace information.



```
{
 "Key": "p5",
 "Value": {
  "serviceName": "thread2",
  "tags": [
   {
    "key": "hostname",
    "type": "string",
    "value": "DESKTOP-FOASM6G"
   },
   {
    "key": "ip",
    "type": "string",
    "value": "192.168.0.94"
```

Figure 47. VR-Tablet showing deployment information (hostname, IP).

VI. Conclusion

This paper contributes VR-ISA, a Virtual Reality (VR) solution concept that supports ISA to improve the quality of software architectures by immersively integrating information and supporting its visualization and accessibility to a spectrum of stakeholders. To demonstrate our VR-based ISA solution concept, three VR-centric viewpoints were elucidated: 1) dynamic distributed event and data streams, 2) static internal module composition and dependencies, and 3) operational execution tracing and observability. Our prototype realization showed its feasibility, and a case-based evaluation provided insights into its capabilities.

The invisibility of software remains an essential challenge for its development, and thus integrating fact-based information can help support better architectural decisions, support comprehensibility, and maintain conceptual integreity. Virtual reality offers a way to visualize a digital reality such as software, and to do so immersively. An informed software architecture can help to improve the quality of software architectures, and VR-ISA integrates ISA intuitively and immersively. By utilizing VR-based viewpoints, stakeholder concerns can be addressed to help make ISA accessible to a wide spectrum of stakeholders and support the adoption of ISA in industry. Additional VR-based viewpoints are readily feasible to support various additional views and concerns.

Future work includes defining and realizing additional VR viewpoints, holistic integration with our other VR-based SE and EA solutions, and a comprehensive empirical study in the industry. Additionally, simultaneous viewing of VR-ISA with our VR-UML and VR-SysML solutions could be used to highlight differences between the intended (prescriptive) SA and the actual informed SA (descriptive), which could be analyzed and lead to either corrections to the SA model or to the implementation to address potential architectural drift.

Acknowledgment

The author thanks Markus Harder, Maurizio Jenak, and Patrick Kneifel for their assistance with the design, implementation, evaluation, and diagrams.

References

[1] R. Oberhauser, "VR-EDStream+EDA: Immersively Visualizing and Animating Event and Data Streams and Event-Driven Architectures in Virtual Reality," the Fifteenth International Conference on Information, Process, and Knowledge Management (eKNOW 2023), IARIA, 2023, pp. 71-76.

[2] ISO/IEC/IEEE, "ISO/IEC/IEEE 42010:2022(E) - International Standard for Software, systems and enterprise--Architecture description," IEEE/ISO/IEC, 2022, doi: 10.1109/IEEESTD.2022.9938446.

[3] M. Loukides and S. Swoyer, "Microservices Adoption in 2020," O'Reilly Media, Inc., 2020. [Online]. Available from: https://www.oreilly.com/radar/microservices-adoption-in-2020/ 2024.11.28

[4] GitLab, "A Maturing DevSecOps Landscape," 2021. [Online]. Available from: https://about.gitlab.com/images/developer-survey/gitlab-devsecops-2021-survey-results.pdf 2024.11.28

[5] Business Wire, "New Ponemon Study Reveals Application Security Risk At All Time High: 1 in 2 Enterprises Need Better Protection," 2015. [Online]. Available from: https://www.businesswire.com/news/home/20151210006098/en/New-Ponemon-Study-Reveals-Application-Security-Risk-At-All-Time-High-1-in-2-Enterprises-Need-Better-Protection 2024.11.28

[6] C. Sridharan, "Chapter 4. The Three Pillars of Observability". Distributed systems observability: a guide to building robust systems, O'Reilly, 2018, ISBN 978-1-4920-3342-4.

[7] T. McQuillan, "Informed Architecture: Three Tensions," In: Writings in Architectural Education. EAAE PRIZE 2003-2005. EAAE, 2005, p.49-63.

[8] M. Hemmerling, "Informed Architecture," In: Hemmerling, M., Cocchiarella, L. (eds) Informed Architecture. Springer, Cham, 2018, pp. 3-10. https://doi.org/10.1007/978-3-319-53135-9_1.

[9] A. Figliola and A. Battisti, "Exploring Informed Architectures," In: Post-industrial Robotics. Springer, Singapore, 2021, pp. 1-45. https://doi.org/10.1007/978-981-15-5278-6_1

[10] G. H. Chong, R. Brandt, and W. M. Martin, *Design informed: Driving innovation with evidence-based design*. John Wiley & Sons, 2010.

[11] M. Shirer, “IDC Survey Illustrates the Growing Importance of Developers to the Modern Enterprise,” IDC, 2021. [Online]. Available from: https://www.idc.com/getdoc.jsp?containerId=prUS48058021 2024.11.28

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

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

[14] R. Oberhauser and C. Pogolski, "VR-EA: Virtual Reality Visualization of Enterprise Architecture Models with ArchiMate and BPMN," In: Shishkov, B. (ed.) Business Modeling and Software Design (BMSD 2019), LNBIP, vol. 356, Springer, Cham, 2019, pp. 170–187, doi.org/10.1007/978-3-030-24854-3_11.

[15] 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.org/10.1007/978-3-030-52306-0_14.

[16] 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, Springer, Cham, 2022, pp. 122-140, doi.org/10.1007/978-3-031-11510-3_8.

[17] R. Oberhauser, “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, doi.org/10.1007/978-3-030-79976-2_3

[18] 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.

[19] R. Oberhauser, “VR-GitCity: Immersively Visualizing Git Repository Evolution Using a City Metaphor in Virtual Reality,” International Journal on Advances in Software, 16 (3 & 4), 2023, pp. 141-150.

[20] X. Qin, Y. Luo, N. Tang, and G. Li, "Making data visualization more efficient and effective: a survey," The VLDB Journal, 29, pp.93-117, 2020.

[21] A. Fonnet and Y. Prié, "Survey of Immersive Analytics," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 3, pp. 2101-2122, 1 March 2021, doi: 10.1109/TVCG.2019.2929033.

[22] A. Fonnet, F. Melki, Y. Prié, F. Picarougne, and G. Cliquet, “Immersive Data Exploration and Analysis,” Student Interaction Design Research Conference, Helsinki, Finland, hal-01798681, 2018, https://hal.science/hal-01798681.

[23] P. Reipschläger et al., “DebugAR: Mixed dimensional displays for immersive debugging of distributed systems,” In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1-6.

[24] R. Sicat et al., "DXR: A Toolkit for Building Immersive Data Visualizations," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 715-725, Jan. 2019, doi: 10.1109/TVCG.2018.2865152.

[25] M. Cordeil et al., "IATK: An Immersive Analytics Toolkit," *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, Osaka, Japan, 2019, pp. 200-209, doi: 10.1109/VR.2019.8797978.

[26] S. Hubenschmid, J. Zagermann, S. Butscher, and H. Reiterer, “Stream: Exploring the combination of spatially-aware tablets with augmented reality head-mounted displays for immersive analytics,” *Proc. 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1-14.

[27] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer, "Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization," In: *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 659-668, 2016, doi: 10.1109/TVCG.2015.2467091.

[28] C. Zirkelbach, A. Krause, and W. Hasselbring, “Hands-on: experiencing software architecture in virtual reality,” Technical Report, Kiel University, 2019.

[29] A. Schreiber, L. Nafeie, A. Baranowski, P. Seipel, and M. Misiak, "Visualization of Software Architectures in Virtual Reality and Augmented Reality," *2019 IEEE Aerospace Conference*, Big Sky, MT, USA, 2019, pp. 1-12, doi: 10.1109/AERO.2019.8742198.

[30] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J. M. González-Barahona, and M. Lanza, "CodeCity: On-Screen or in Virtual Reality?," *2021 Working Conference on Software Visualization (VISSOFT)*, Luxembourg, 2021, pp. 12-22, doi: 10.1109/VISSOFT52517.2021.00011.

[31] A. Hoff, L. Gerling, and C. Seidl, "Utilizing Software Architecture Recovery to Explore Large-Scale Software Systems in Virtual Reality," *2022 Working Conference on Software Visualization (VISSOFT)*, Limassol, Cyprus, 2022, pp. 119-130, doi: 10.1109/VISSOFT55257.2022.00020.

[32] R. Müller, P. Kovacs, J. Schilbach, and D. Zeckzer, "How to master challenges in experimental evaluation of 2D versus 3D software visualizations," *2014 IEEE VIS International Workshop on 3DVis (3DVis)*, Paris, France, 2014, pp. 33-36, doi: 10.1109/3DVis.2014.7160097.

[33] S. Narasimha, E. Dixon, J. W. Bertrand, and K.C. Madathil, “An empirical study to investigate the efficacy of collaborative immersive virtual reality systems for designing information architecture of software systems.” Applied ergonomics, 80, 175-186, 2019.

[34] M. J. McGuffin, R. Servera, and M. Forest, "Path Tracing in 2D, 3D, and Physicalized Networks," in IEEE Transactions on Visualization and Computer Graphics, 2023, doi: 10.1109/TVCG.2023.323898

[35] R. Oberhauser, "VR-SDLC: A Context-Enhanced Life Cycle Visualization of Software-or-Systems Development in Virtual Reality," In: Business Modeling and Software Design (BMSD 2024), LNBIP, vol 523, Springer, Cham, 2024, pp. 112-129, https://doi.org/10.1007/978-3-031-64073-5_8.

[36] R. Oberhauser, "VR-DevOps: Visualizing and Interacting with DevOps Pipelines in Virtual Reality," In: Proceedings of the Nineteenth International Conference on Software Engineering Advances (ICSEA 2024), IARIA, 2024, pp. 43-48. ISBN: 978-1-68558-194-7.

[37] R. Oberhauser, “VR-V&V: Immersive Verification and Validation Support for Traceability Exemplified with ReqIF, ArchiMate, and Test Coverage,” Int’l Journal on Advances in Systems and Measurements, 16 (3 & 4), 2023, pp. 103-115.

[38] R. Oberhauser, “VR-Git: Git Repository Visualization and Immersion in Virtual Reality,” The Seventeenth International Conference on Software Engineering Advances (ICSEA 2022), IARIA, 2022, pp. 9-14.

[39] R. Oberhauser, “VR-TestCoverage: Test Coverage Visualization and Immersion in Virtual Reality,” In: Proc. Fourteenth Int’l Conf. on Advances in System Testing and Validation Lifecycle (VALID 2022), IARIA, 2022, pp. 1-6.

[40] R. Oberhauser, “VR-SysML+Traceability: Immersive Requirements Traceability and Test Traceability with SysML to Support Verification and Validation in Virtual Reality,”

International Journal on Advances in Software, 16(1 & 2), pp. 23-35, IARIA, 2023.

[41] R. Oberhauser, M. Baehre, and P. Sousa, "VR-EvoEA+BP: Using Virtual Reality to Visualize Enterprise Context Dynamics Related to Enterprise Evolution and Business Processes," In: Shishkov, B. (eds) Business Modeling and Software Design (BMSD 2023), LNBIP, vol 483, Springer, Cham, 2023, pp. 110-128. https://doi.org/10.1007/978-3-031-36757-1_7

[42] P. B. Kruchten, "The 4+1 View Model of architecture," in IEEE Software, vol. 12, no. 6, pp. 42-50, Nov. 1995, doi: 10.1109/52.469759

[43] 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.

[44] P. Dobbelaere and K. S. Esmaili, "Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations," In: Proc. 11th ACM int'l conference on distributed and event-based systems, 2017, pp. 227-238.

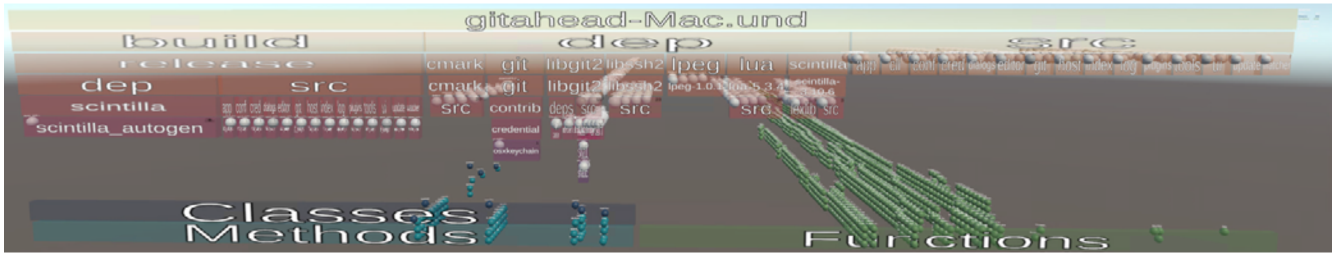


Figure 48. VR:VP:ModDep for large project (GitAhead C++ sample, 444 files, 9747 functions) depicting layered folder structure as containers for files.

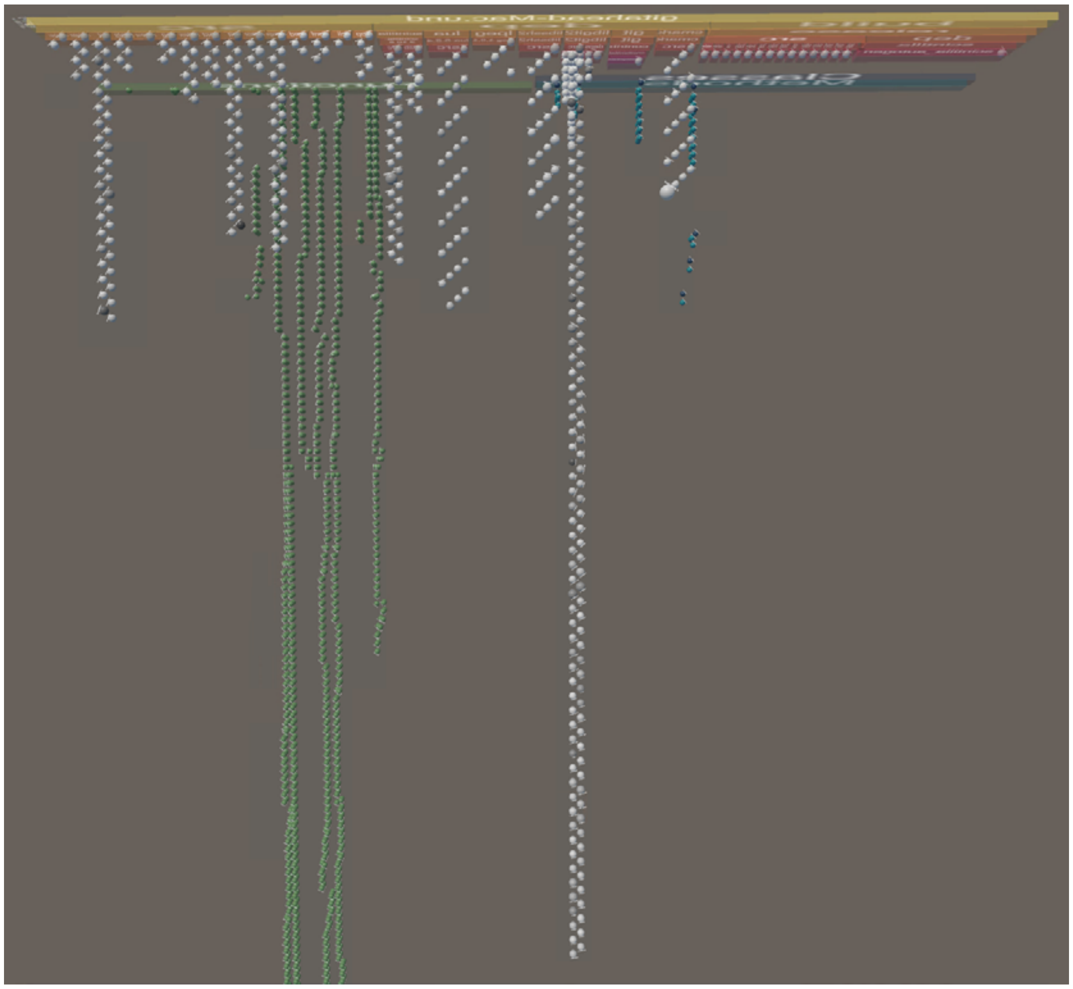


Figure 49. VR:VP:ModDep for large project (GitAhead C++ sample, 444 files, 9747 functions) from the top depicting a complete project perspective.

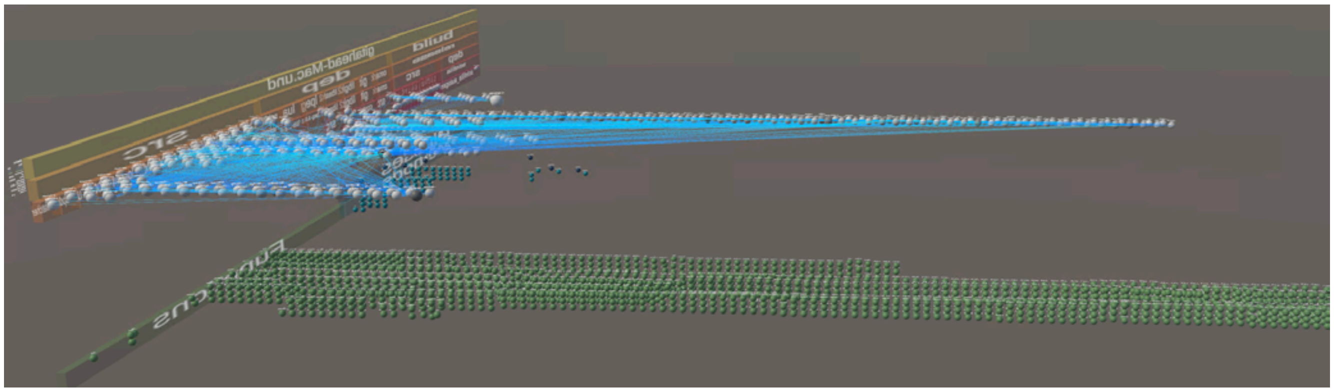


Figure 50. VR:VP:ModDep side perspective for large project (GitAhead C++, 444 files, 9747 functions) depicting directed dependencies (blue lines).

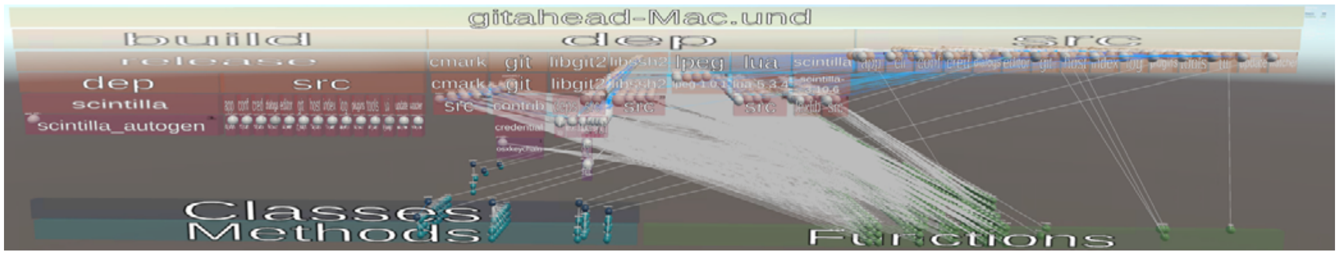


Figure 51. VR:VP:ModDep front perspective for large project (GitAhead C++ sample) depicting all dependencies (blue lines) and connections (white lines).

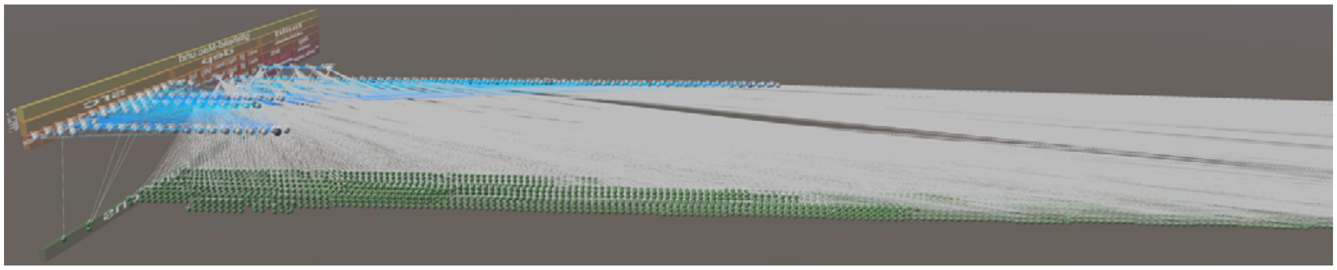


Figure 52. VR:VP:ModDep side perspective for large project (GitAhead C++ sample), depicting all dependencies (blue lines) and connections (white lines).