PrismArch

Deliverable No D4.2

First version of the interfaces and interconnections allowing to interact with BIM and multi-simulation information in VR

Project Title:	PrismArch - Virtual reality aided design blending cross-disciplinary aspects of architecture in a multi-simulation environment
Contract No:	952002 - PrismArch
Instrument:	Innovation Action
Thematic Priority:	H2020 ICT-55-2020
Start of project:	1 November 2020
Due date of deliverable:	31 October 2021
Actual submission date:	15 November 2021
Version:	1.0
Main Authors:	Dimitrios Ververidis (CERTH), Dimitrios Gounaridis (CERTH), Christos Mouzakis (CERTH), Vittorio Bava (Mindesk), Evangelia Vangeli Margariti (CERTH), Maria Rousi (CERTH)

Project funded by the European Community under the H2020 Programme for Research and Innovation.

Deliverable title	First version of the interfaces and interconnections allowing to interact with BIM and multi-simulation information in VR
Deliverable number	D4.2
Deliverable version	Final
Contractual date of delivery	31 October 2021
Actual date of delivery	15 November 2021
Deliverable filename	PrismArch_D4.2_1.0.pdf
Type of deliverable	Other
Dissemination level	CO
Number of pages	43
Workpackage	WP4
Task(s)	T4.1, T4.2, T4.3, T4.4
Partner responsible	CERTH
Author(s)	Dimitrios Ververidis (CERTH), Dimitrios Gounaridis (CERTH), Christos Mouzakis (CERTH), Vittorio Bava (Mindesk), Evangelia Vangeli Margariti (CERTH), Maria Rousi (CERTH), Jeg Dudley (AKTII), Roberto De Loris (Mindesk)
Editor	Dimitrios Ververidis (CERTH), Evangelia Vangeli Margariti (CERTH), Spiros Nikolopoulos (CERTH)
Reviewer(s)	Roberto de Loris (Mindesk), Martin Broesamle (ETH),

Abstract	Software package and associated report containing the results from T4.1-T4.4, namely interconnection modules, BIM/CAE-Simulation interfaces and high-realistic graphics that will be integrated in WP5, the first prototype.
Keywords	Architecture Design Software, Virtual Reality, Interfaces

Copyright

Deliverable history

Version	Date	Reason	Revised by
0.1	14/9/2021	Table of Contents	Dimitrios Ververidis (CERTH)
0.2	10/10/2021	Initial content	Dimitrios Ververidis (CERTH)
0.3	15/10/2021	Semantics and interconnection content	Maria Rousi (CERTH)
0.4	20/10/2021	Explanations for developments	Evangelia Vangeli Margariti (CERTH)
0.7	8/11/2021	Final edits	Dimitrios Ververidis (CERTH)
0.8	10/11/2021	Ready for internal review	Dimitrios Ververidis (CERTH)
0.9	13/11/2021	Comments by internal reviewer	Martin Broesamle (ETH)
1.0	15/11/2021	Submission	Spiros Nikolopoulos (CERTH)

List of abbreviations and Acronyms

Abbreviation	Meaning
AEC	Architecture, Engineering and Construction
BIM	Building Information Modelling
CAD/CAM	Computer-Aided Design & Computer-Aided Manufacturing
GA	Grant Agreement
ICT	Information and communication technology
IP	Intellectual Property
IPR	Intellectual Property Rights
NDA	Non-Disclosure Agreement
PC	Project Coordinator
PHP	PHP: Hypertext Preprocessor
PM	Person-Month
PMB	Project Management Board
PTM	Project Technical Manager
R&I	Research and Innovation
SB	Project Supervisory Board
SBM	Supervisory Board Member
SME	Small and Medium-sized Enterprises
ToC	Table of Contents
UE4	Unreal Engine 4
UG	User Group
UML	Unified Modeling Language
VR	Virtual Reality
WP	Work Package

Executive Summary

The main goal of this deliverable is to present the updates in the Prismarch VR-aided design environment during WP4. We describe the progress that has been made in issues related to the VR interfaces, the interconnections across software, and the development of design tools. This deliverable builds on top of previous deliverables namely D4.1 that defined the overall structure of the interfaces as well as D5.1 that defines the overall architectural design of PrismArch.

The interfaces are developed in Unreal Engine that allow the three main disciplines, namely Architecture, MEP engineering, and Structural Engineering to communicate seamlessly with existing software such as Rhino, Revit and SAP2000. The main software selected for the developments is Unreal Engine that offers unique visualization capabilities in an efficient manner. Significant role is allocated to the Speckle system that allows asynchronous collaboration through a common format and a cloud database. Also, Mindesk plugins for Unreal, Rhino and Revit are also crucial to the implementation as it allows synchronous collaboration through real-time interconnection across software.

Table of Contents

1. INTRODUCTION

1.1 Position of Tasks WP4 in the Prismarch space

The developments in WP4 have been driven from user requirements (Deliverable D1.1) and overall System Architectural Design (Deliverable D5.1).

Overall, the collaboration of the AEC disciplines is foreseen in the context of Synchronous and Asynchronous collaboration as described in D5.1. Synchronous is the collaboration that happens in real-time, namely to transfer changes from Unreal to Rhino or Revit in real-time and vice versa. This is achieved with the Mindesk plugin using LiveLink technology with Rhino, but it is extended during PrismArch to be achieved also in Unreal. The asynchronous collaboration is achieved with an open-source software from an external collaborator, namely *Speckle*[1]. We have adopted its Unreal plugin and further developed it with more functionalities. It allows users to exchange 3D model data in a versioned manner based on operations like pushing or pulling commits into and from a central database in which each collaborator has access and can download the 3D models on a second step. This allows to remove the need of exchanging files but instead relying on a central database only that communicates 3D models in a certain JSON format.

In the following, namely Section 2, we describe the latest changes in the technical requirements. In Section 3, we provide the current developments. In Section 4, we describe the structure of the Gitlab repository where all the developments have been uploaded. Finally, in Section 5,we provide the future plans and prioritization of the developments.

2. UPDATED TECHNICAL REQUIREMENTS

The outcome of D5.1 was the system design which is also the blueprint where the developments are based to fulfill user requirements (D1.1). During the last months, the D1.2 has introduced new requirements and therefore the system design was updated. In Figure 1, we present the new overall system design where the blue colored boxes indicate the parts that have been updated or changed. These will be further analyzed in Section 3.

[1] https://speckle.systems

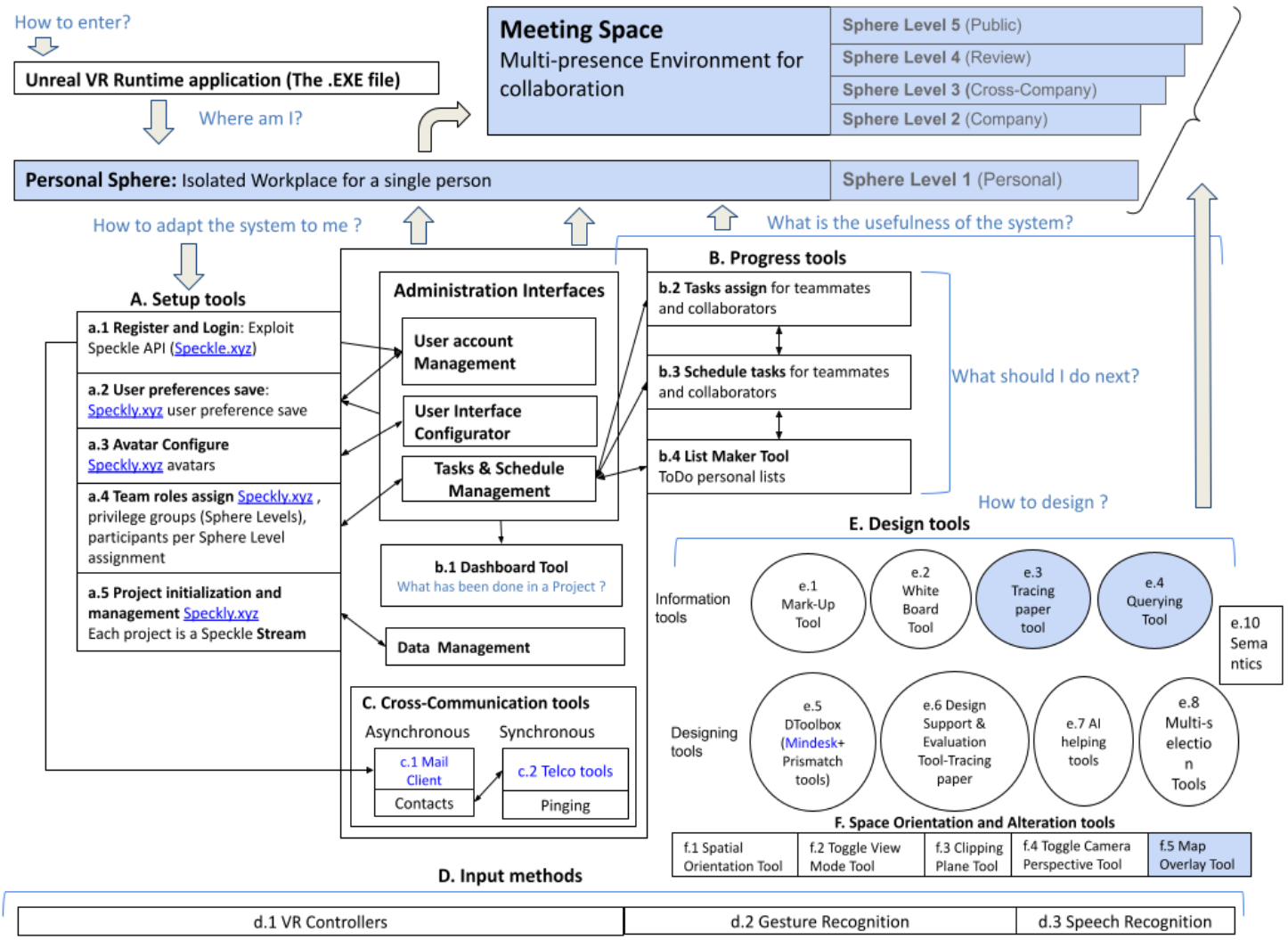


Figure 1: Changes in the technical requirements. Personal Sphere, Meeting Space are better defined. A Map overlay is added.

3. DEVELOPMENTS

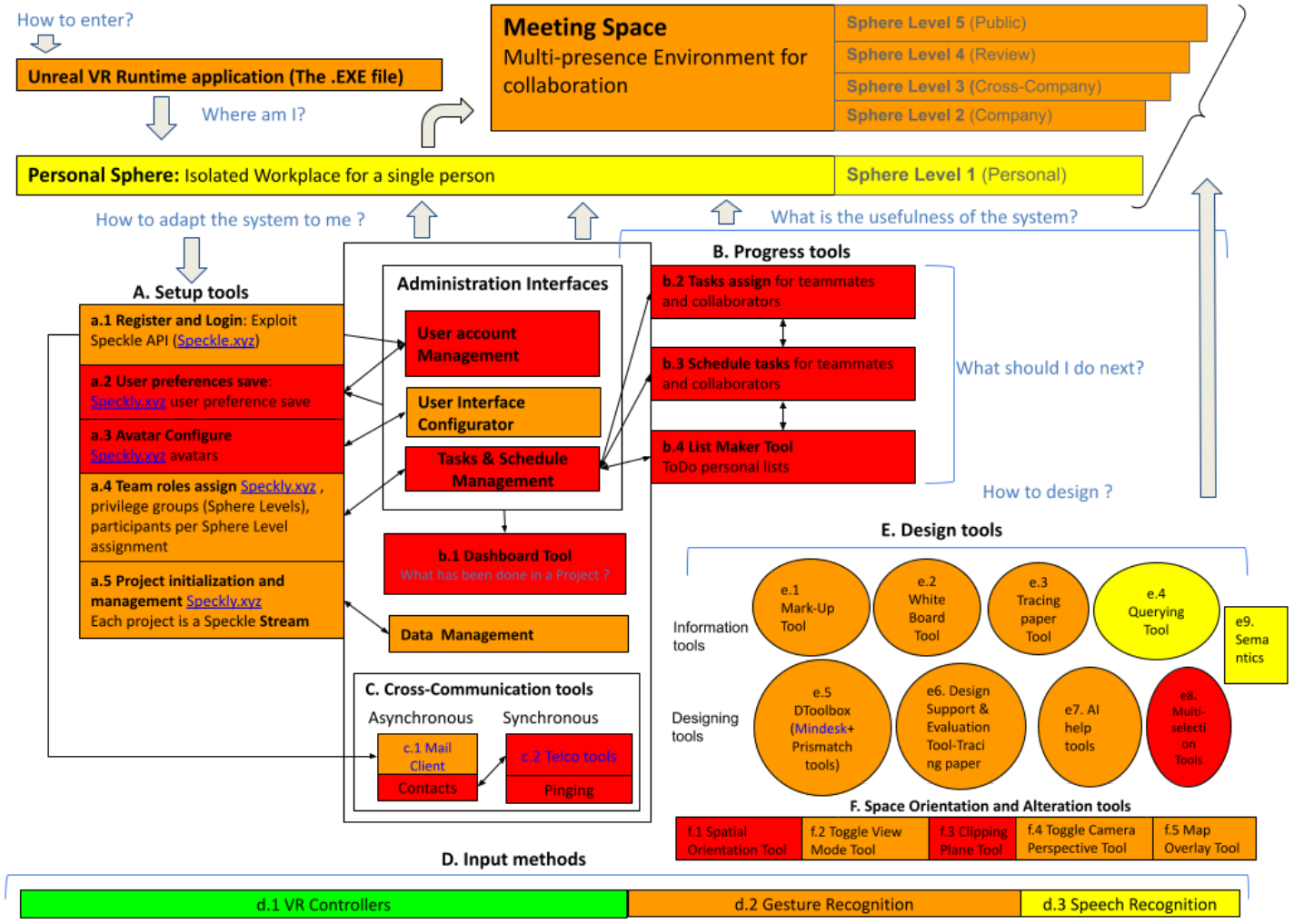
3.1 Interactive environment and toolset architecture progress diagram

D5.1 produced a technical blueprint as depicted in Figure 1 and briefly explained in the previous chapter. In this section, a colour code memo is formed, ranging from red to green colour to exhibit the progress of the individual categories and their tools. Let it be known that red describes an early stage of development while green describes the completed and the in-between colours range from medium to high level of progress (orange and yellow sequentially).

As regards third party technologies that were described in D5.1, we have so far extensively exploited the following third party plugins in order to speed up developments:

1) Cesium maps[2]: They were used to visualize the project in its actual place, and configure the environmental conditions (Sun light and shadows) in a realistic manner.
2) Speckle system[3]: It was used in order to interconnect the Unreal environment to a database that allows to store the design data in a format that is understood by software that are daily used by Architects and Engineers such as Revit and Rhino. This facilitates Asynchronous collaboration whereas the foreseen collaboration through Mindesk plugin is meant to be used Synchronous (real-time) collaboration.
3) Mozilla DeepSpeech[4]: It is software and data that are used to achieve voice recognition in VR and reduce the time needed for typing.

More details about the developments can be found in Section 3.2.



[2] https://cesium.com/
[3] https://speckle.xyz/
[4] https://deepspeech.readthedocs.io

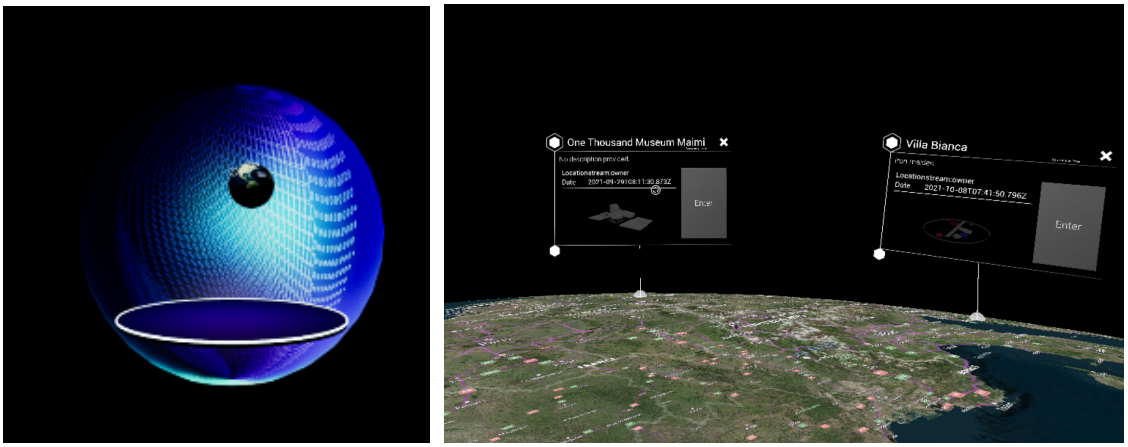
Figure 2: Overall progress of environment and toolset architecture in colour coded diagram.

3.2 Detailed progress of the Prismarch tools and methodology

The developments are organized as follows. We first outline the structure of the Personal Sphere (Section 3.2.1) and Meeting Space (Section 3.2.2), and then we overlay the features needed to enrich each of these spaces (Section 3.2.3).

3.2.1 Personal Sphere

The current developments during the implementation of the Personal Sphere is seen in Figure 3. In terms of spatial structure, it consists of an outer shell with a futuristic texture and a globe sphere (Figure 3a) where the avatars can navigate into projects (Figure 3b). The development was done in Unreal Engine 4.26 using Cesium Maps 1.7.0. The globe has a very high resolution where the users can navigate accurately at a spatial resolution of centimeters. This accuracy, from the inspection of the whole earth up to the centimeter, however, has the trade-off of increasing the world size by x50 otherwise numerical errors occur as scaling in Cesium maps can not be less than 0.00002. See relevant discussion between PrismArch and Cesium[5].



(a) (b)

Figure 3: First developments in Unreal Engine 4 of the personal project Sphere.

3.2.2 Meeting space

The initial version of the meeting space can be seen in Figure 4. It has been designed to be used as a space for collaborative activities such as design reviews, co-design etc. Furthermore, it has the capability to transfer designed models to their geographic location using Cesium maps and topography data, e.g. photogrammetry scanned new spaces, using the Cesium API (free for up to 5GB of data)[6]. The 3D model of the Meeting Space is designed by Zaha Hadid architects.

[5] https://community.cesium.com/t/decrease-cesium-world-terrain-size/15676
[6] https://cesium.com/

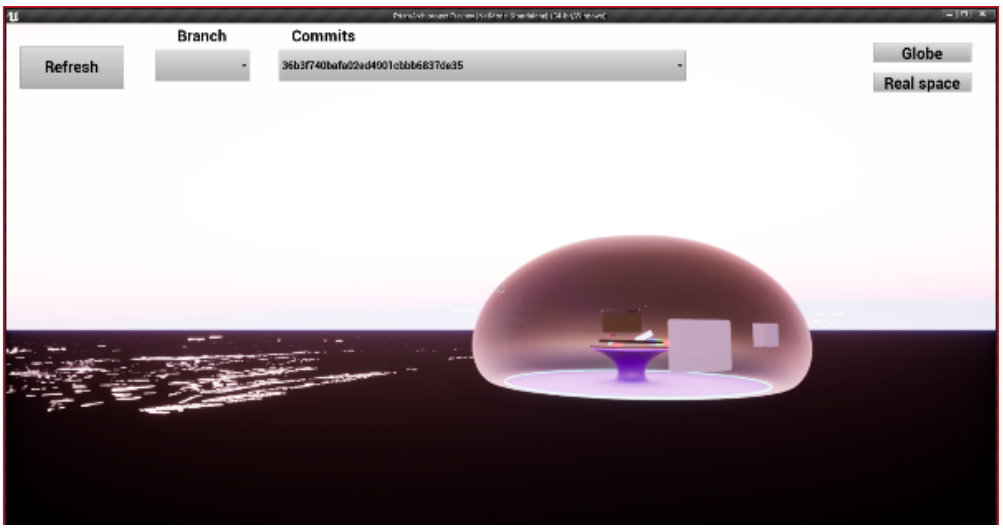


Figure 4: Meeting space is an isolated space for collaboration in VR but also allows to inspect 3D models in their actual geographic location.

3.2.3 Functionalities Developed

We have in general adopted the Speckle terminology, in turn allowing us to define a common data format across all necessary software. This terminology was explained in D4.1 and it is also repeated here in order for this document to be used standalone. In details:

a) A "**Stream**" is an architectural project

b) A "**Branch**" of a Stream is a thread where each discipline or a group of people on the same discipline can work.

c) The "**Main Branch"** is the Architectural design (otherwise not possible to work fluently)

d) A "**Commit**" is each insertion of data in the database.

There are obvious similarities with the Git system. Speckle can thus be considered as the Git of 3D designs. This should not be confused with Mindesk software which is working in real-time across two software and does not save the data but leave this task to each software independently.

A. SETUP TOOLS

a.1 Registration and Login: Towards achieving this functionality we have exploited Speckle[7] database that allows users to register and login from a web interface. Next, the users should copy and paste a token (Bearer) generated by the web interface into PrismArch Unreal Application. The Bearer token can be found in GraphQL interface of Speckle console (HTTP headers console[8]). In the future, a desktop application, namely Speckle Manager will be

[7] https://speckle.xyz/
[8] https://speckle.xyz/graphql

used by the users that allows OAUTH 2.0 authorization for 3rd party applications (give privileges). This will remove the copy-pasting of the Bearer token.

a.2 User preferences save: User preferences saving is a feature that will be developed in the second year of PrismArch. It is saving preferences locally in each PrismArch Unreal Application as Game Settings and then should be uploaded to Speckle Database in user data for synchronisation with other PrismArch instances used by the same user.

a.3 Avatar configure: Users can configure their appearance in the immersive environment's 3D space. It has been proposed by the use case partners to have a realistic 3D representation of the physical user as the avatar. However, it requires 3D scanning of end-users, and face and pose tracking and allocation to the scanned model. This is a very demanding functionality to be achieved in terms of development and outside the GA of PrismArch as no such task was foreseen. The options to be checked for an alternative approach that could be developed relatively quickly within the scope of PrismArch are

1. Meta Human creator by Unreal Early Access application[9].
2. MediaPipe Selfie-Segmentation solution by Google[10].

The first one regards the 3D scanning of the user, but the second is related to capturing the video stream of the user and removing the background. In this manner the user can be represented by its video as a texture onto a rectangular panel[11]. The second option will be checked in the second year of Prismarch as it allows for an increased level of user representation without the risk of falling in the uncanny valley[12]. We will base our efforts in a recently developed plugin for Unreal Engine, named as Mediapipe-ue4-plugin[13].

a.4 Team roles assignment: The Speckle user management is used that allows three types of users, namely Owners, Contributors, and Reviewers. Additionally, It has a switch for making a project Public so that anyone can view it. This mechanism partially fulfills end-user requirements as the Sphereing Levels as defined in D1.2 require an access level definition. Beyond what is provided by the Speckle-specific mechanism is the requirement to define the access-levels per asset rather than for the whole project. A project is named as Stream in Speckle terms. Additionally, the Sphere Levels also require some information to be only editable by a single author before making it visible to collaborators. Towards this end we foresee in the second year to increase Speckle DB definition schema with an additional field, namely that of the Sphere Level per asset.

[9] https://www.unrealengine.com/en-US/metahuman-creator
[10] https://google.github.io/mediapipe/solutions/selfie_segmentation
[11] https://www.youtube.com/watch?v=GWYrdczYFKU
[12] https://en.wikipedia.org/wiki/Uncanny_valley
[13] https://github.com/wongfei/ue4-mediapipe-plugin

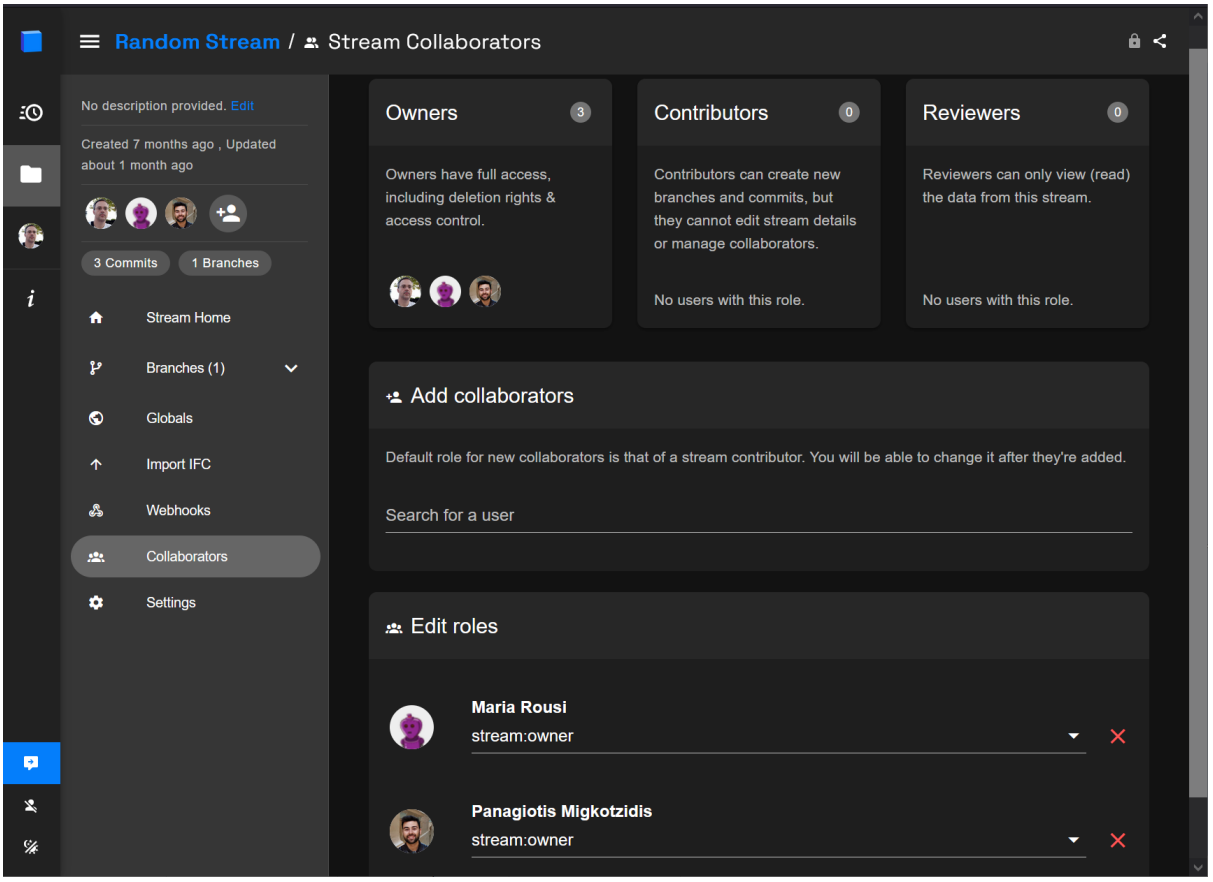


Figure 5: Team management exploiting the Speckle API.

a.5 Project initialization and management: This is where an Architectural Project, namely a "Stream" can be generated and managed. Towards this end, we have allocated resources into integrating the Speckle Unreal plugin into the PrismArch Unreal project. An example for the integration into the user interface is shown in Figure 6.

This allows to

a) to fetch architectural data from each Stream.
b) change branch where each branch represents a different discipline.
c) alternate across commits, where each commit represents a timestamp where a change was made and pushed to the Speckle database.

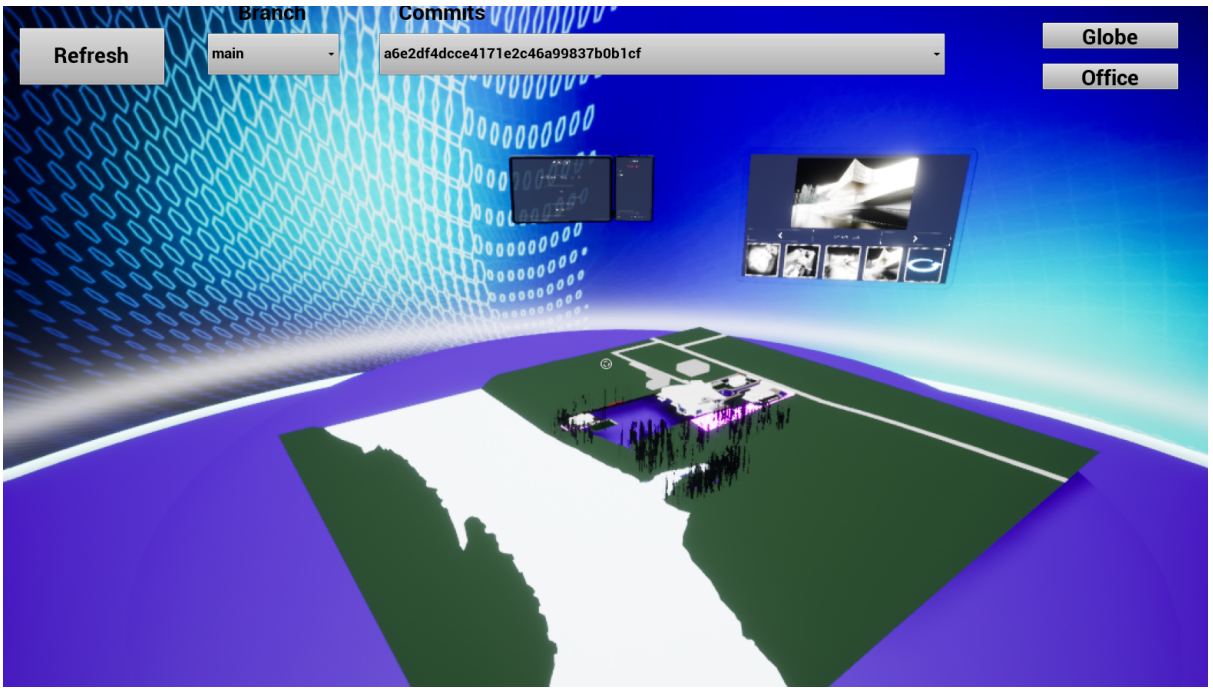


Figure 6: Fetching the latest commit of ZH Villa model from Speckle repository in Meeting Space according to a hash metatag from the drop-down menu located in the upper part of interface.

B. PROGRESS TOOLS

These tools are driving the users on how to contribute to the architectural project, which parts to be responsible for, which conflicts to resolve, and to provide deadlines for each task. Contextually it answers the question “What should I do next?”.

b.1 Dashboard tool: It is the briefing of the current project status. It informs the user for the project update, who is now in the system, and about the overall status of the project.

b.2 Tasks assigned: It is the interface and back-end system for assigning tasks to collaborators.

b.3 Schedule tasks: It is the time definition for tasks completion.

b.4 List maker tool: It is a list of notes of what a user has to do next in the system.

Although Asana was foreseen to be incorporated in the VR environment through a WebView, this is not feasible anymore since Asana has stopped serving its webpage for Unreal Engine 4 (UE4) web browser component. Furthermore, for other services such as Trello, the web browser of UE4 is serving the pages as textures over 3D objects which causes blurriness (Figure 7). Furthermore, the use of a web browser inside UE4 is also problematic as there is no interface for manipulating information such as storing cookies with user credentials.

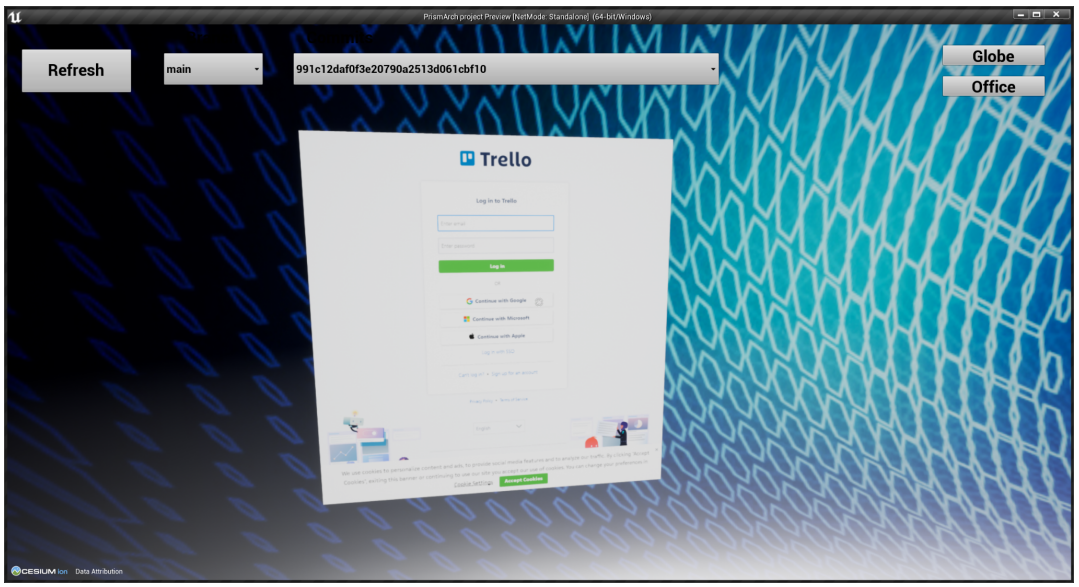


Figure 7: Web view is rejected due to blurriness.

An option to be examined in the second year is a UE4 plugin called “Bug Tracker”. It has an interface that can be used in-Editor-mode and in-Game-mode for UE4 (Figure 8) and directly report them in Trello using the REST API of it (Figure 9). Although it is meant to be used for reporting issues related to game development, it can be used in Architecture as it shares the same logic.

Figure 8: Bug tracker reporting an issue.

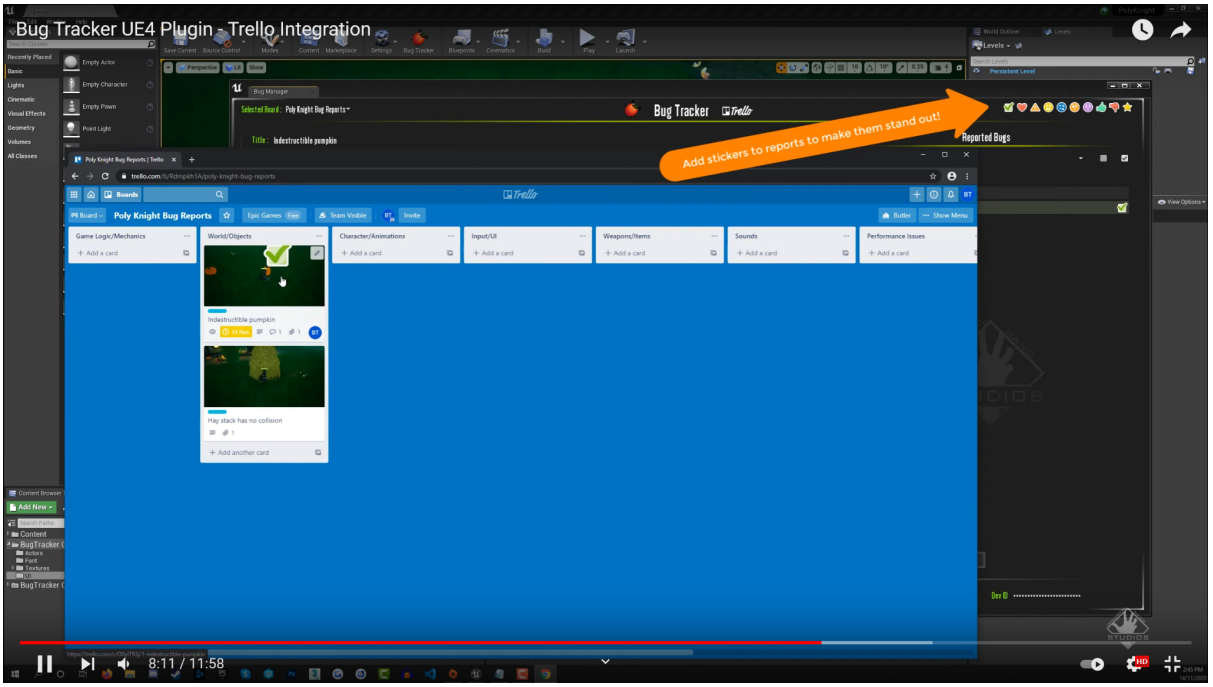


Figure 9: Bug tracker captures screenshots of the experience and ports them to Trello directly.

In general Bug Tracker has the following capabilities that can be used inside PrismArch for collaboration across teams. It is proprietary licensed and it costs about 54 euros[14]. It will not be distributed with PrismArch open source code, however, the interested organizations can purchase it from Epic Marketplace.

Bug Tracker Features:

- Browsing reports with a selection of filters.
- Bug markers – Spawn markers showing bug locations displayed as actors on game.
- "Jump to" button. One click to jump to the location associated with the report.
- Editing report details.
- Auto sends fixed bugs to patch notes.
- Assigning team members to reports.
- Add checklists to reports for team members to follow up on.
- In-Game bug reporting for players.
- Automated capturing of useful data like screenshots, level, camera position and more.
- Dynamically built UI fields.
- Customizable UI using UMG and Blutility.
- Viewing and editing report details in-Editor/Game or via the Trello website.

Trello API Features:

[14] https://www.unrealengine.com/marketplace/en-US/product/bug-tracker?lang=en

- User Tokens, Boards, Cards, Comment Cards, Lists, Labels, Stickers, Attachments (JPG), Checklists, Checkitems, Members, Members Notifications, Custom Fields (Limited)

C. CROSS-COMMUNICATION TOOLS

Two tools belong to this category.

c.1 Asynchronous communication tools: As an example of asynchronous communication we will focus on email communications, i.e. an e-mail client tool that can be connected to the mail addresses of the user to send and receive messages. It is like a standard mail client but adjusted for VR environments. We will base our developments on Easy-Email UE4 plugin[15] or nodemailer[16] using nodejs-ue4[17] where everyone can connect it with his/her mail server. It is foreseen to be developed in the second year. Easy-Email costs 21 euros whereas nodemailer is free. In the open-source version we will not distribute proprietary software but we will prompt developer that if they want this emailing functionality they should lease it from Epic Marketplace.

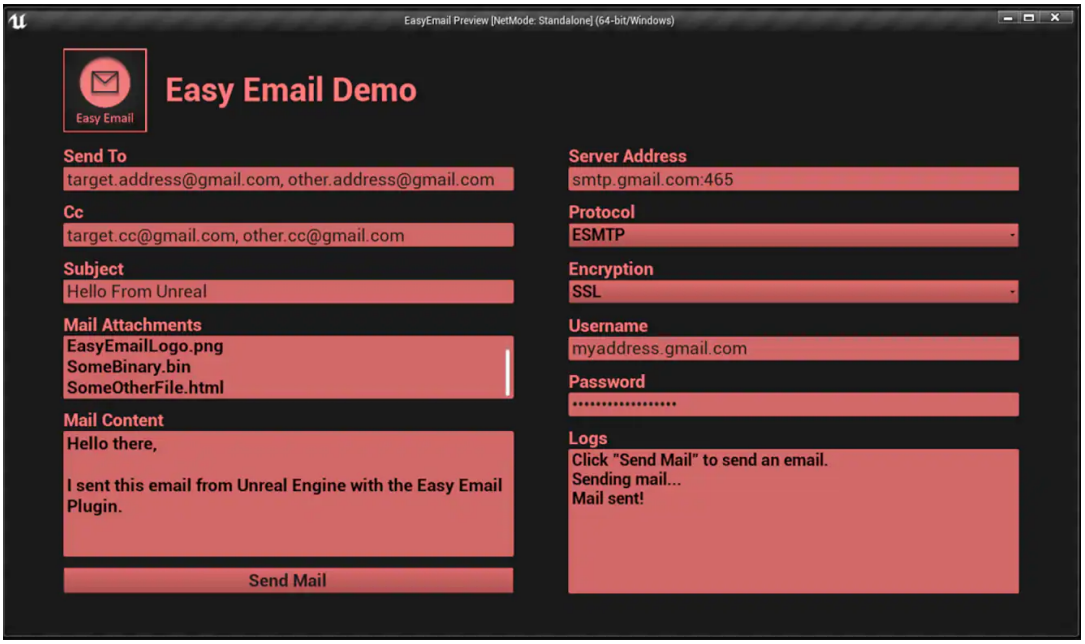


Figure 10: Easy email allows easily sending an email through the Unreal Engine environment.

c.2 Synchronous communication tools: For this purpose we envision to develop a teleconferencing tool to be used inside VR. It is actually a telecommunication tool for real time chatting especially useful for communicating to the users that are using onscreen

[15] https://unrealengine.com/marketplace/en-US/product/email-plugin
[16] https://www.npmjs.com/package/nodemailer
[17] https://github.com/getnamo/nodejs-ue4

software. Candidate tools are Vivox[18] or Agora[19]. Vivox is the state of the art used platform for communication in 3D games, but it is only for audio and text chatting. For the integration of Vivox, we will use the Vivox Core[20] and AVRF Vivox Core plugin[21]. Agora on the other hand offers also video chatting. For the integration of Agora we will use the Agora SDKs. Most preferable is Agora. It is a task foreseen for the second year of the project.

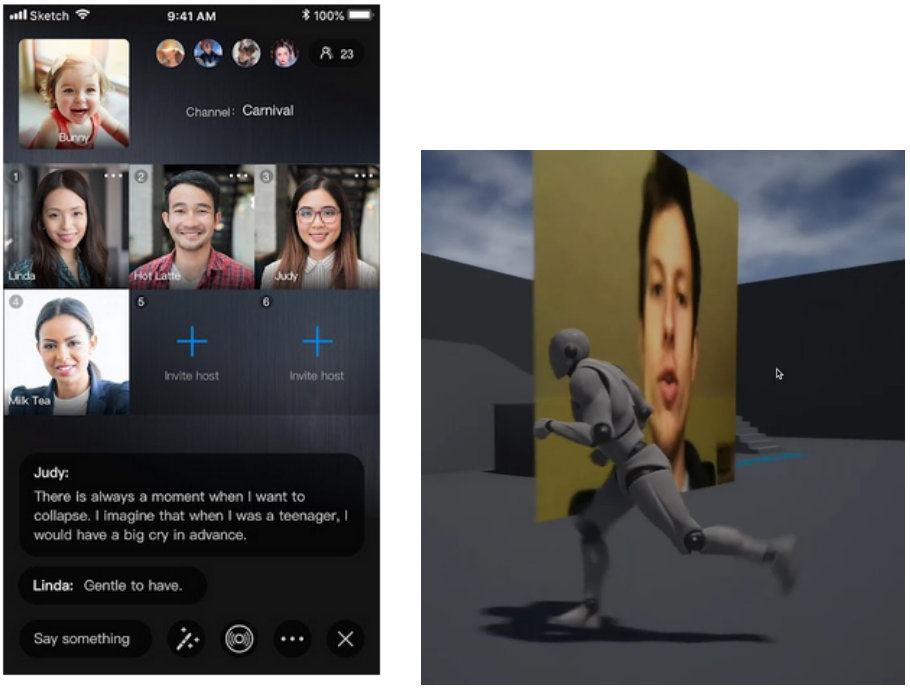


Figure 11: Agora has SDKs for Windows, Mobile, and Unreal video chatting.

D. INPUT METHODS

Input methods define the way how the user interacts with the system. There are three methods:

d.1 VR Controllers: They are the most reliable way that the user can interact with the system and have been tested with the tools prototypes described in the next category. HP Reverb G2 controllers and Oculus Quest 2 controllers are the two types of controllers that are currently used during the developments of PrismArch. The SDKs for Unreal of the respective companies are used for the integration of these devices. The developments on the first year were focused on how to add all the tools in a radial tree cascade menu manipulated by players hands (Figure 12).

[18] https://unity.com/products/vivox
[19] https://www.agora.io/en/products/voice-call/
[20] https://www.unrealengine.com/marketplace/en-US/product/vivoxcore
[21] https://dev.humancodeable.org/our-services-2/advanced-framework-utilities/

Figure 12: The radial menu allows to place theoretical infinite functionalities as it is developed to show functionalities in a cascaded form.

d.2 Gesture Recognition: Gesture recognition by Oculus Quest 2 headset is an alternative way of interacting with the system. Although it is not as robust as the VR controllers, it is mature enough to be used in the project and Unreal Engine already supports their input. The exploitation of gestures for designing 3D models and manipulating the information in VR is foreseen to be developed in the second year of the project.

d.3 Speech Recognition: We have incorporated AI methodologies for Speech Recognition. Namely, we have integrated the open source DeepSpeech libraries of the Common-Voice project[22]. The English AI model 0.9.3 was integrated as it is the best trained network among all the language sets available. In Figure 13, the details of each voice dataset are presented where a new AI model can be trained. The German and French languages are half the size of the English one. So, the system can be possibly extended to these languages.

[22] https://commonvoice.mozilla.org/en

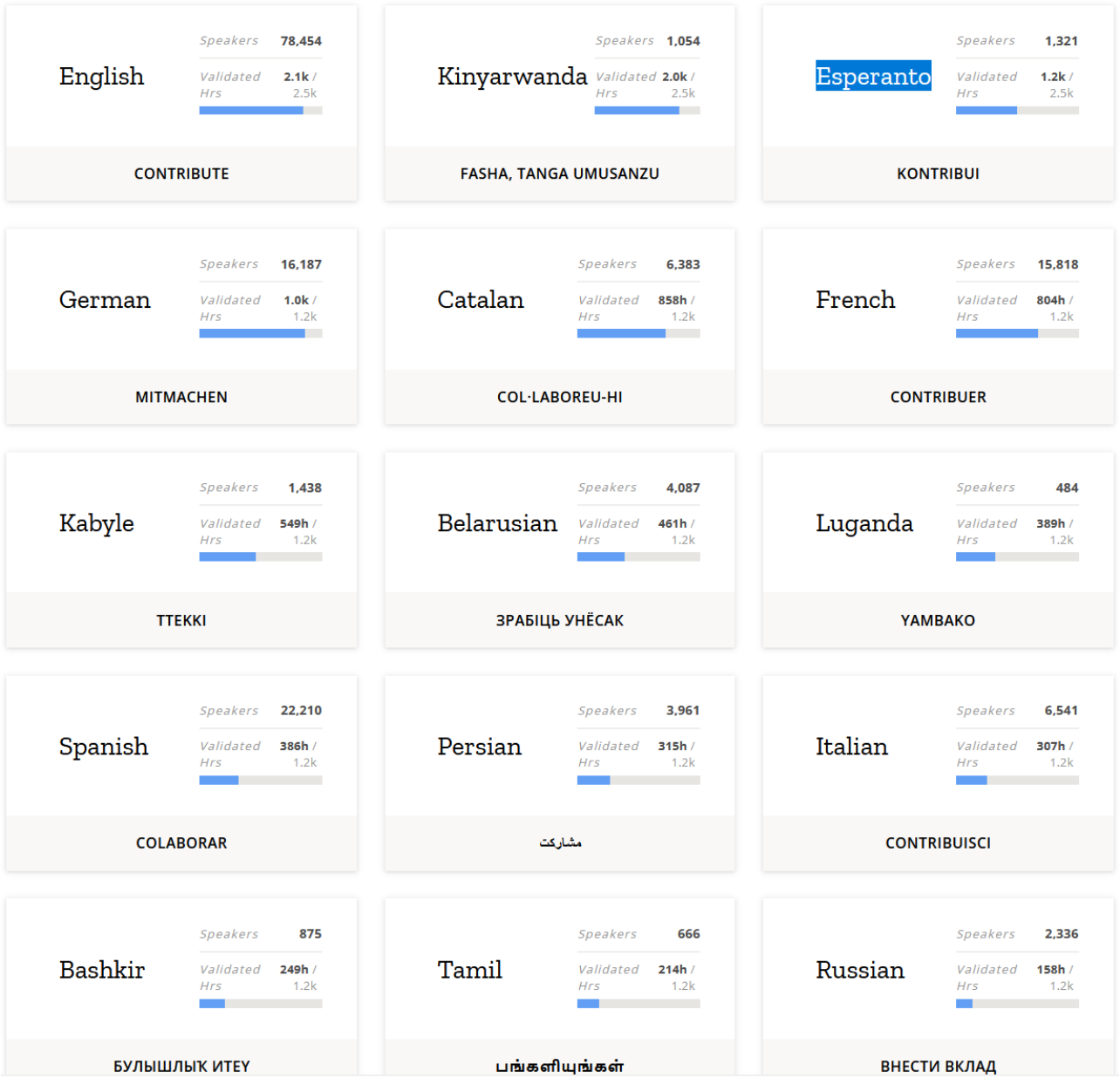


Figure 13: Open datasets for more AI models training for voice recognition.

In detail, the knowledge in English AI model is accumulated into two files[23], namely

1. Deepspeech-0.9.3-models.pbmm (200 MBs): That contains the mechanism for the phonetic transcription, i.e. the voice to letters mechanism; and
2. Deepspeech-0.9.3-models.scorer (800 MBs): That contains the Natural Language Processing mechanism, i.e. the meaningful transcription of letters into sentences.

The second AI model is not the optimum as regards Architectural Design environments as some technical words are missing. It is foreseen in the future to add such words, extracted from Rhino3D documentation.

23

For the integration of DeepSpeech into Unreal Engine, we have used the Javascript SDK of DeepSpeech and we did the connection through nodejs-ue4 plugin[24]. The final result is a standalone speech recognition mechanism that does not need web connection for recognition. The voice recognition interface has been firstly integrated into the Markup tool that allows to insert notes on each object.

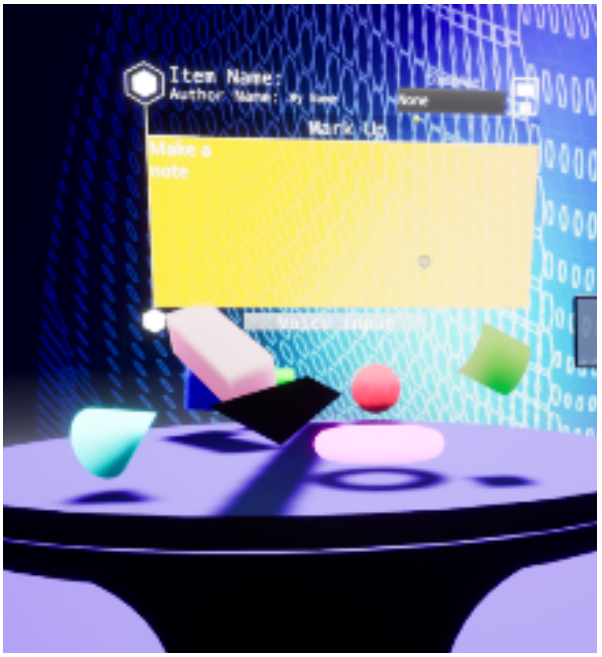


Figure 14: Adding notes on 3D objects through voice recognition interface.

E. DESIGN TOOLS

The next subcategory of Design tools is **Information Tools,** namely they are tools that are used for extrapolating or retrieving information about the building. Five tools can be found in this category as one more was added according to the updated user requirements.

e.1 Markup tool:

The MarkUp tool allows the user to create notes as annotations for 3D assets inside the VR environment and share them with collaborators. Various features that have already been developed/implemented:

- The user can select a 3D asset by pointing at it with the VR controller and select it to create a note.
- The MarkUps are always pointing towards the user.
- All 3D assets have the ability to be marked with a widget that is attached to them and can be activated and deactivated with the Unreal's select component.
- The tool supports Speech to text recognition and was tested by recording a title of a 3d asset and seeing it printed on the MarkUp note.
- The MarkUp widget has a spin box where the user can select and sign administration and partner collaboration group information (Sweco, Act II, Client, ZHVR). Each group

[24] https://github.com/getnamo/nodejs-ue4

has a unique colour coding system that assigns to the note's label as well when it is created.

In the future, the tool should support customization as regards colors and size, as well as log tracing so that the information on the user who created the note will be available in each markup item.

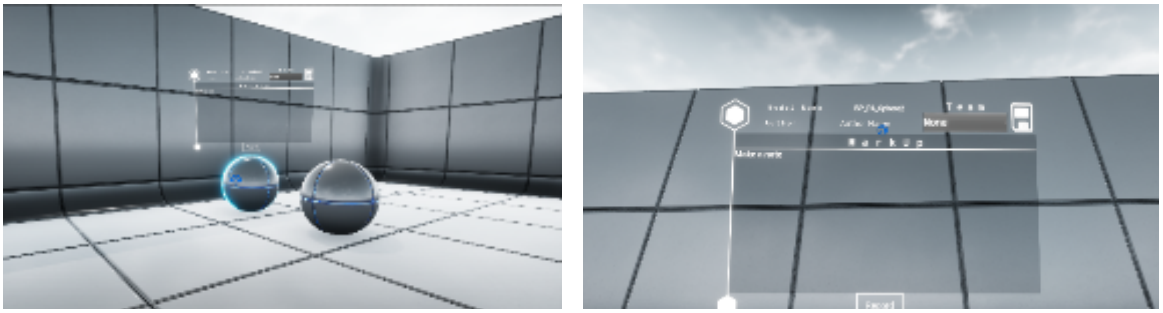


Figure 15: The Markup tool developed allows to insert notes for objects through voice to text.

e.2 Whiteboard - Tracing paper tool

The Whiteboard allows the user to fetch reference images inside VR space. A mockup was initially developed (Figure 16a) that led to the implementation presented in Figure 16b.

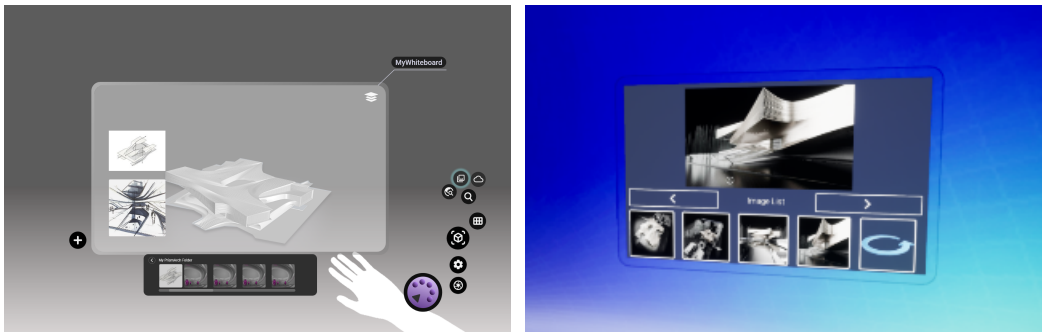


Figure 16: (a) First mockup of whiteboard; and (b) actual implementation.

The images are fetched from a specific folder in the path as jpg or png. If this path is shared through a file sharing mechanism such as Dropbox, OneDrive, Google, etc then the users can exchange and transfer images into VR.

Other features for future implementation are

- Adjust opacity,
- Transforming board size,
- Change position.

The Tracing Paper Tool allows one to draw on a transparent layer as it usually happens in architectural physical cases. A mockup was developed (Figure 17) that leads the developments (Figure 18).

Figure 17: Mockup of Tracing Paper tool. Drawing on a tracing paper drawing and over the actual model top view allows architects to keep notes and design changes.

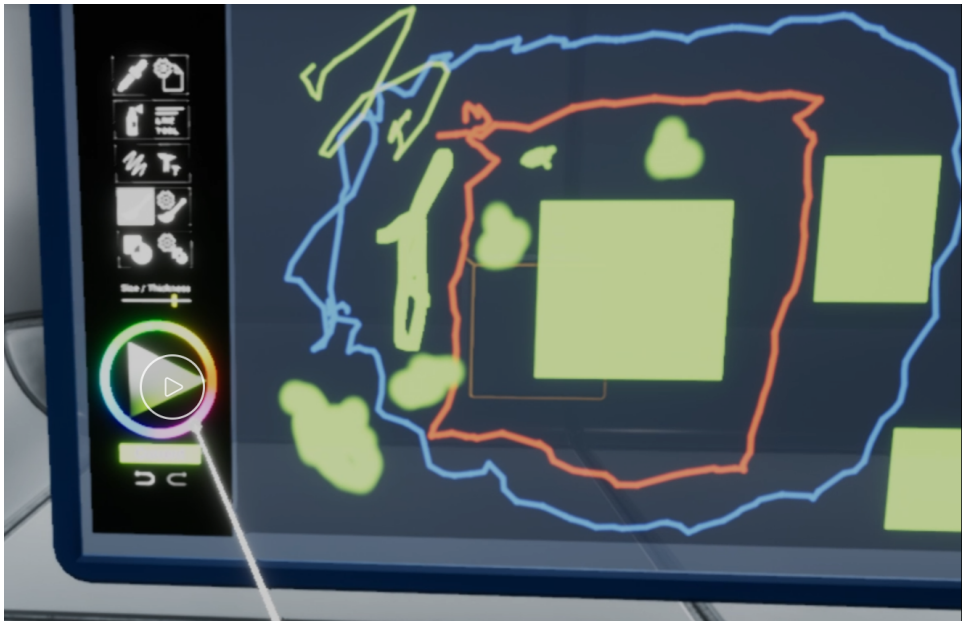


Figure 18: Developed Tracing Paper tool.

Features developed:

- Opacity change
- Drawing, marking, colouring
- Line width change
- Basic shapes drawing, e.g. lines, rectangles, etc.
- Text typing (voice recognition connected)
- Capture screen and save to exr file image
- Do and Undo buttons

Features that can be implemented in the future are tool resizing, logging user drawing history, and board layers.

e.7 Query Tool

The query tool allows users to browse Speckle JSON Objects information from the database in a 3D way using a 3D force directed graph that was developed during PrismArch based on a recently published work[25] (Figure 19). The parsing includes the dimensionalities of time, i.e. various commits (Figure 20), and the inspection of each model through its tags (Figure 21). Inside each node, a miniature of the related 3D model is presented. The same mechanism will be used for presenting AI generated information where the nodes will indicate AI alterations of the original model. The node bonds are elastic and they follow physics rules, e.g. when pulling one node the others react by following it, constrained by joint bonds.

In the future, searching filters will be implemented. The user will be able to selectively view specific meshes by applying filters to a search utility.

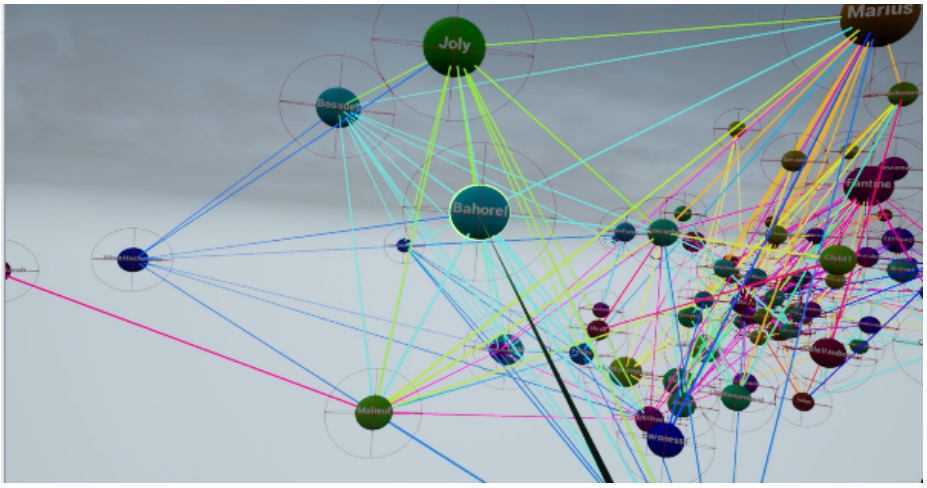


Figure 19: The 3D force directed graph for parsing information of an architectural project.

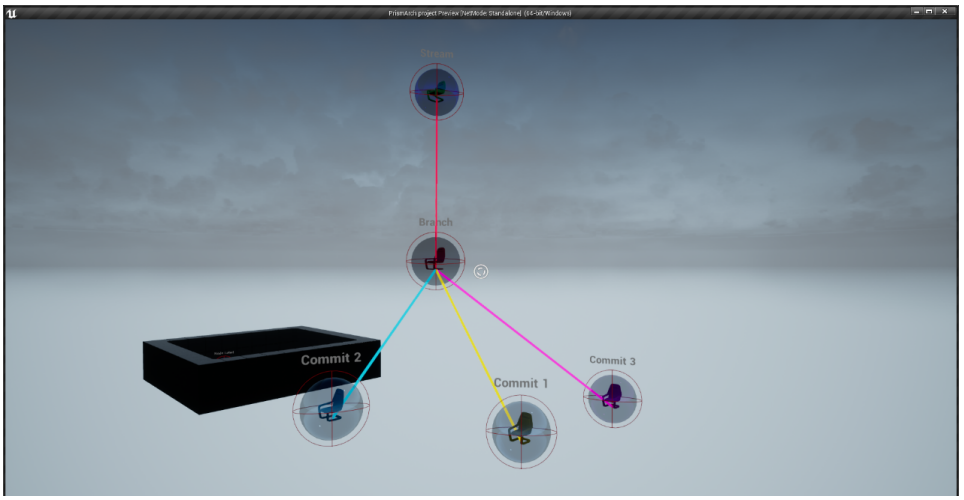


Figure 20: Browsing commits of a branch of the project.

[25] N. Capece, U. Erra and J. Grippa, "GraphVR: A Virtual Reality Tool for the Exploration of Graphs with HTC Vive System," *2018 22nd International Conference Information Visualisation (IV)*, 2018, pp. 448-453, doi: 10.1109/iV.2018.00084.

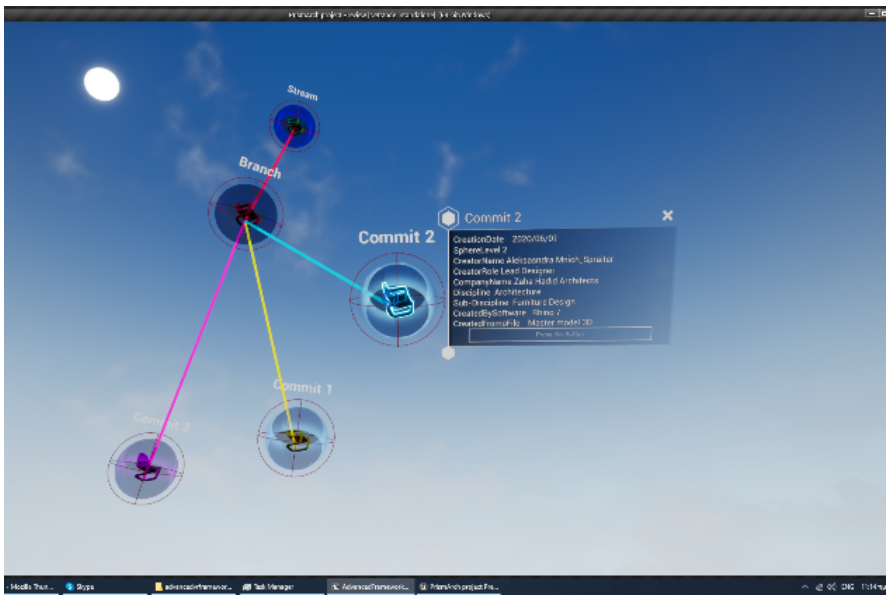


Figure 21: Manipulation of information inside VR with AVRF.

e.5 DToolbox: As part of the development work in PrismArch we have implemented design tools natively inside UE4. The design toolbox enables users to create 3D geometries inside the 3D space and in parallel transfer them synchronously inside Rhino and Revit. The Design Toolbox is empowered by the Mindesk Live Link and therefore based on industrial CAD or BIM geometric kernels. An example of a geometry creation is shown in Figure 22. In a first implementation, the DToolbox will include a basic set of geometric primitives including box, cone, cylinder, as well as a basic set of geometric transformation functions like move, rotate, and scale. Later on, more advanced primitives and functions will be integrated. The next batch of primitives will include planes, lines and certain NURBS surfaces; while the third batch of functions will include copy, deletion, multiple selection, layer filtering, Boolean combination and other NURBS-related functions offered by the source CAD software linked via Mindesk. This development concludes the bidirectional communication across VR and Rhino / Revit, since the direction from Rhino and Revit towards Unreal is already developed by Mindesk.

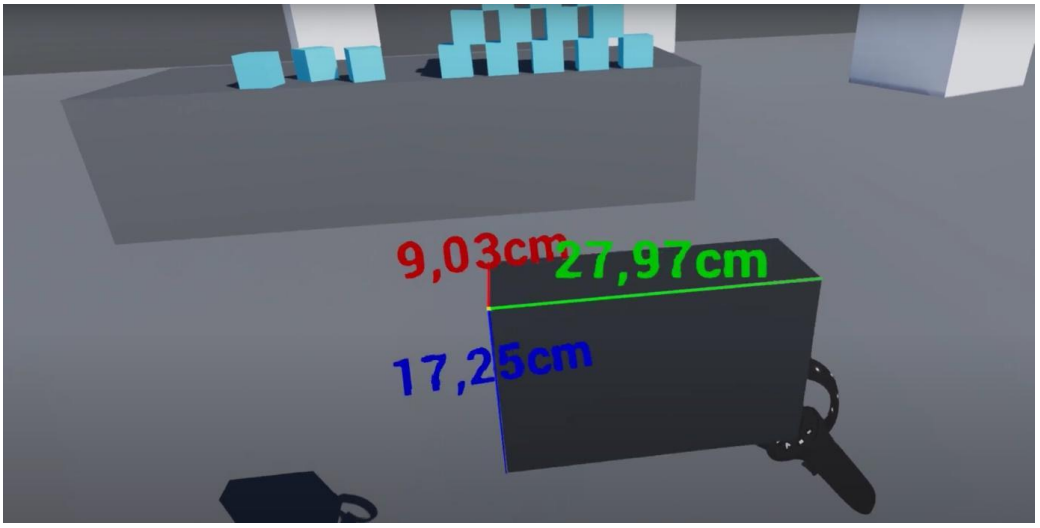


Figure 22: Creation of geometries inside Prismarch.

e.6 Design Support and Evaluation tool

It is a collection of support tools for design:

1. Measuring floor areas and volumes with indication of x,y,z values (Mindesk's annotation-style or tool can be called inside the platform), alternatives for the measurement tools are AVRF and Collabviewer (Figure 23);
2. Bounding box (the same logic to box selection tool in the Multi selection tool) but with the x, y, z values and area/volume annotations;
3. Circulation routing, (e.g. drawing spline route and user object follows the route);
4. Smart staircase modelling;
5. Toggling measurement resolution (mm, cm, m, km) to explore measurement resolution;
6. Toggling measurement system - decimal and imperial (feet and inches);
7. When measurement resolution changes, users can see the changes by using the floor grid size or any reference object size change.

This set of functions is foreseen to be addressed in the second year of the project since they provide auxiliary tools for design.

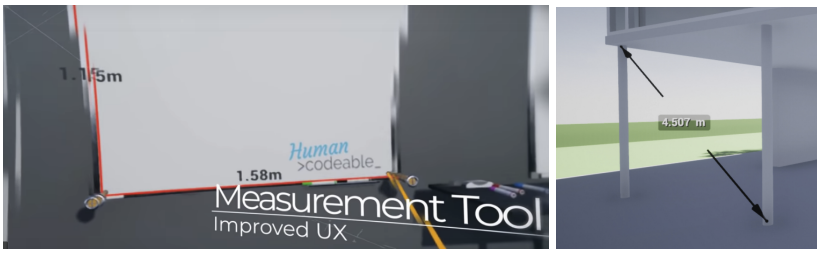


Figure 23: Measurement tool implementations.

e.7 AI tools for assisted design in VR

Based on the preparatory work that was performed in D2.1, which included an extended literature review and coordination with the AEC partners, an integration-ready computational framework for the traversal of the parametric design space has been developed by the UoM and is being reported in D2.2.

The framework includes a set of algorithmic variations which explore the parametric space guided by constraints, objectives and diversity measures, as well as a set of benchmarks that have been used to showcase the algorithms' functionality. It has been developed in a modular architecture, allowing for the application of any algorithm to any of the available benchmarks and facilitating the potential expansion of its functionality. The developed computational framework will serve as the back-end of an interactive design tool which will be completely integrated within the PrismArch platform. The front-end of the interaction will be based on a graph-based UI, currently being developed by CERTH (the 3D force directed Graph), which exposes design variations to the user, allowing them to select the ones they prefer and thus guide the search process. A detailed example of how this integration can be achieved is shown in the following Figure 24, while a detailed description can be found in D2.2. The implementation of this integration will be one of the first steps of D2.3, preparing the ground for further steps, such as the application of Designer Modeling AI methods to capture the designers' preferences.

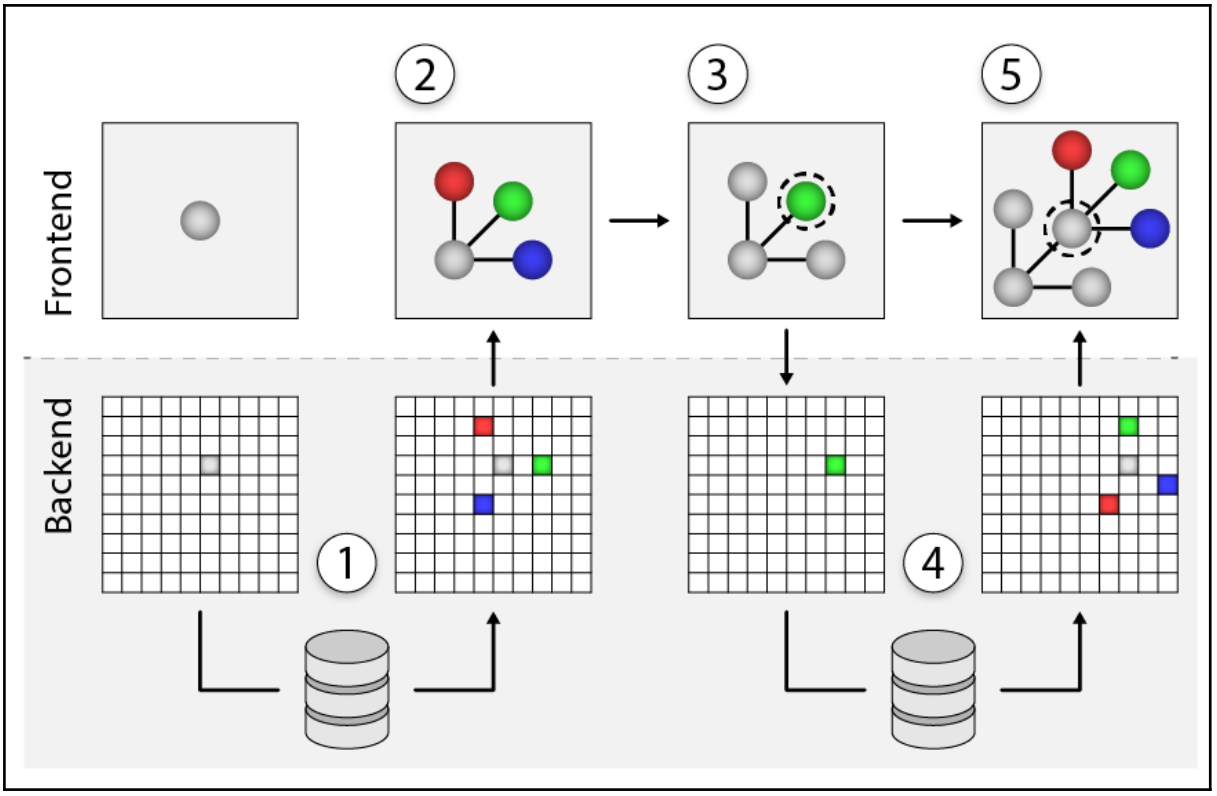


Figure 24: Visualization of the interactive process, between the designer and the Quality-Diversity algorithms which explore the parametric space (D2.2), through the graph-based UI.

e.8 Multi-Selection tools

Tools that allow for highlighting, grouping, isolating, showing/hiding objects. These tools follow the Rhino3D methodology of selecting objects such as:

- Box selection method;
- Users can assign and save tags for single or multiple selected objects for future reviews; Grouping objects;
- a single object or multiple objects are selectable or highlightable via the Unreal Engine custom-depth/post process, etc.);
- The highlighting colours are discipline specific;
- Multiple selected objects can be grouped, ungrouped, inverted, shown and hidden.

The developments on the multi-selection are foreseen for the second year of the project.

e.9 Semantics tool

The Semantics tool is a tool that allows to parse the whole Speckle database and perform queries aggregated to the database as a whole. As regards Speckle Objects, Figure 25 summarizes the hierarchy of the domain-specific ontology that has been developed to capture information pertinent to an architectural project.

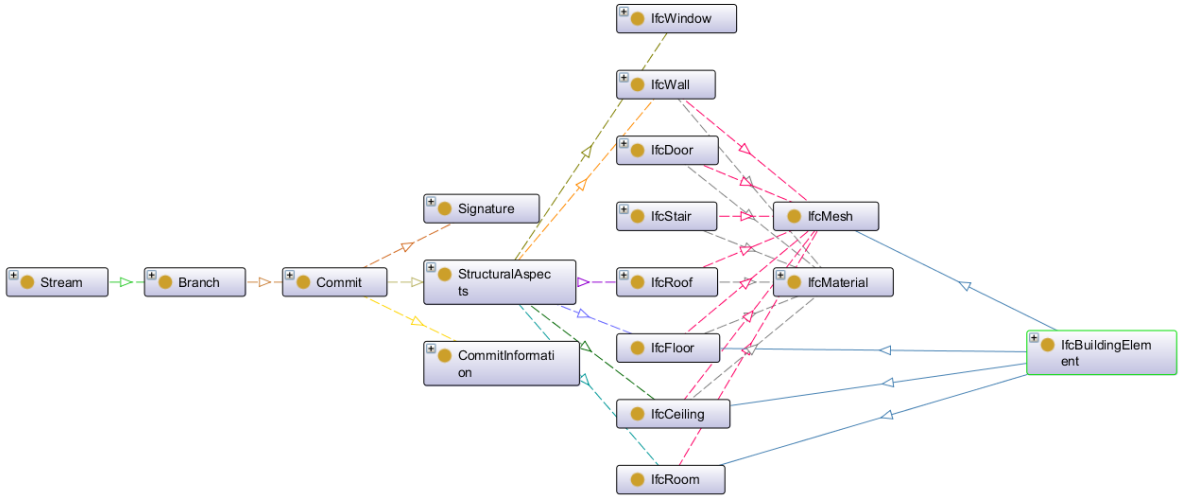


Figure 25: Hierarchy of the domain-specific ontology.

The above-mentioned ontological schema, expressed in Web Ontology Language (OWL) syntax, is used as a basis in order to semantically represent each of the project commits that are pushed into Speckle DB. Information in RDF format is stored in the triplestore, called Knowledge Base. This functionality is offered by the "Semantic Representation and Storage" component, which receives information coming from different architectural programs and is responsible for the generation and storage of high-level representation of various structural aspects and versioning information. The synchronization of the Knowledge Base with Speckle is achieved in specific time frequency (i.e. 1 minute).

The ontology inherits classes and properties from existing ontologies (ifcOWL and EXPRESS) and expands their properties to meet the needs of AEC. More specifically, the root of the developed ontology is the *Stream*, which corresponds to the generated stream in Speckle. Each Stream is connected to a *Branch* via a hasBranch property, while each Branch is connected to a *Commit* using a hasCommit property. Each Commit is related with the following classes:

- *Signature* that includes properties to represent information such as sphere level, creator role, etc
- *CommitInformation* that contains properties to capture information like id, date, latitude and longitude
- *StructuralAspects* that are related with all structural aspects of the commit (i.e. *IfcWindow*, *IfcWall*, *IfcDoor*)

Most of the StructuralAspects are related with *IfcMesh* and *IfcMaterial* individuals. IfcMesh, IfcFloor, IfcCeiling and IfcRoom are sub-classes of the *IfcBuildingElement* class. It is worth mentioning that the ontological schema has been extended to include information pertinent to Signatures and globals (namely latitude, longitude, height).

On top of the Knowledge Base, semantic queries are executed in order to retrieve information to support various usage scenarios using the "Semantic Retrieval" service. The internal semantic framework architecture is depicted in Figure 26.

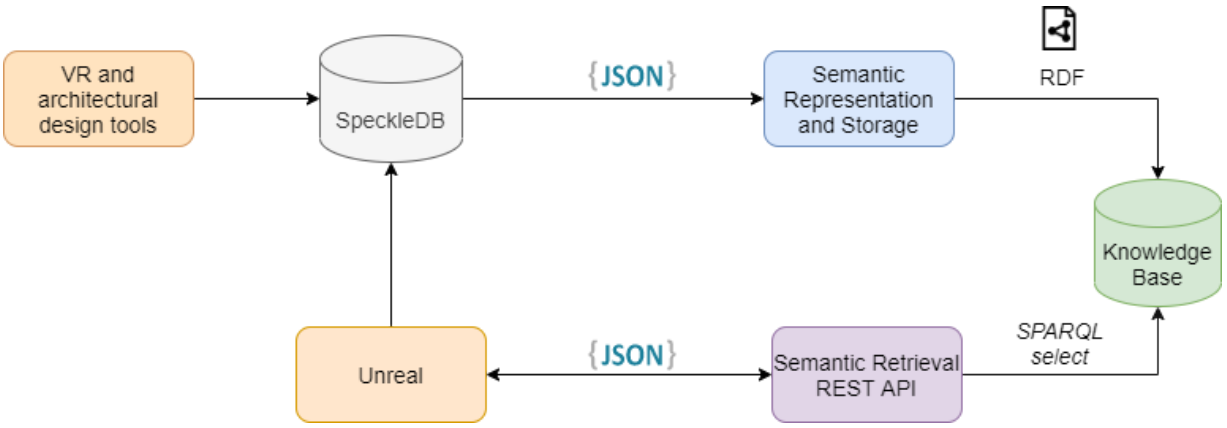


Figure 26: Semantic framework internal architecture

As already mentioned, the "Semantic Representation and Storage" component receives information in JSON format and transforms the data into Resource Description Framework (RDF) triples, following the defined ontological schema. An example of semantically representing metadata related to signatures is presented in Figure 27. The figure presents a signature instance (pm:Signature_1), the relationship between signature and the commit (pm:Commit_1) instances and the properties related to the signature (e.g. pm:hasStage, pm:hasSphereLevel, etc).

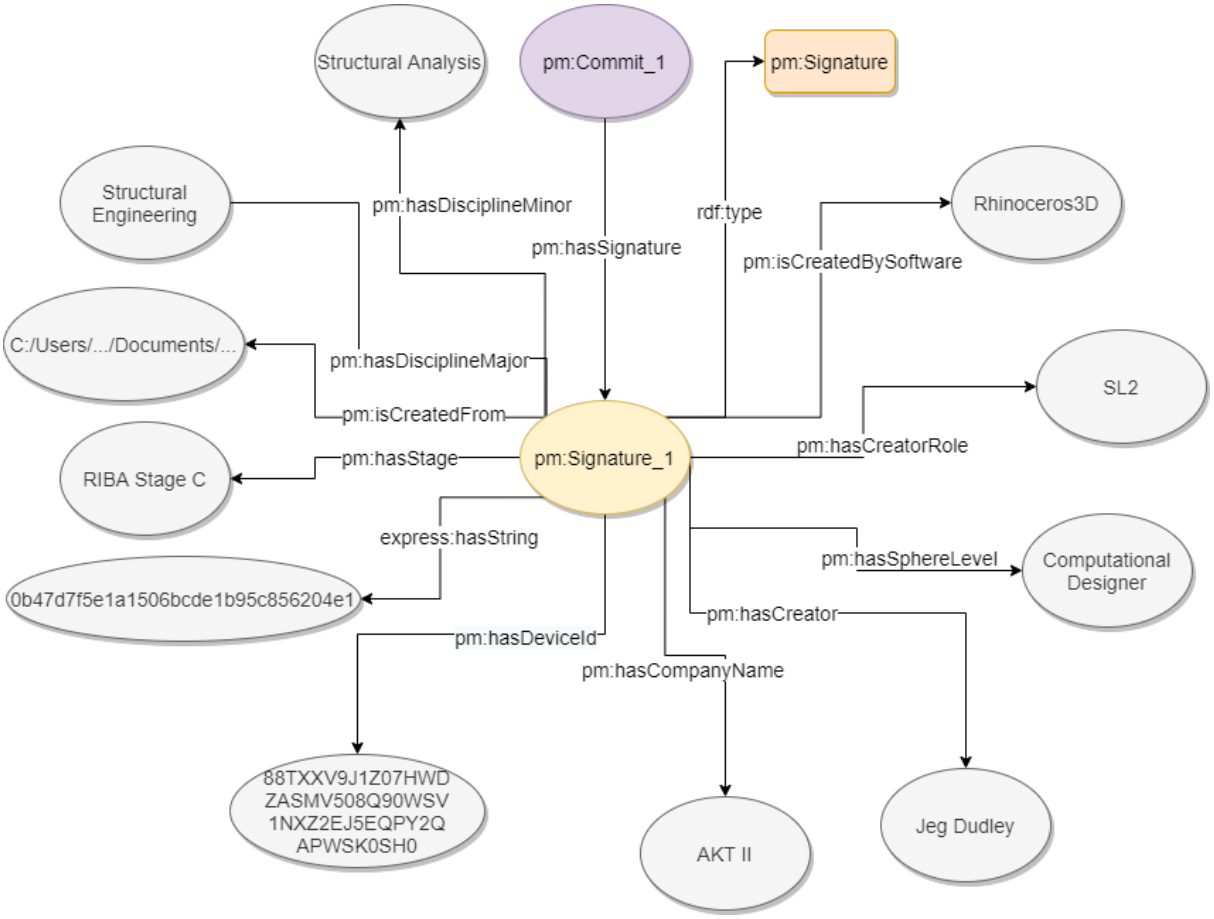


Figure 27: Example of semantic representation of Signature metadata

The semantic retrieval service is responsible for the execution of the appropriate queries in order to support the scenarios described in Appendix II. In all of the above mentioned

scenarios examples, we present the JSON input that the semantic retrieval service accepts, the SPARQL query that the service executes in order to retrieve the requested information and the JSON output that it provides. In the latest scenario, regarding object history, there are some optional fields which may not be present in all objects. These fields are unit, type, area, volume, cost, height, category, level and family.

It is worth mentioning that the ontological schema, the mapping and the semantic retrieval service will continue to be updated in order to cover the upcoming needs of representation, storage and retrieval of information, based on the user's needs. Scenarios, might for instance, be extended to detect commits with specific characteristics according to the Signature (i.e. sphere level).

Technically, semantics are based on the analysis of the information of Speckle JSON Objects. These objects are retrieved from Speckle's DB (Postgres) served with Apollo server through the GraphQL REST API[26] (see Appendix I for examples), then are processed in CERTH premises using a GraphDB[27] server and the semantically aggregated information becomes available for UE4 through a Tomcat SPARQL REST API. Examples for calling the Tomcat SPARQL REST API are found in Appendix II.

F. ORIENTATION AND SPACE ALTERATION TOOLS

Orientation and Space alteration tools are tools that allow the users to perceive the space and the information they are interested in. These are divided into five types.

f.1. Spatial Orientation Tool

It allows the user to navigate easily into space, e.g. into certain bookmarked spots, and see what others have done within a particular space. Developments are based upon the Advanced VR framework (AVRF[28]) as it provides a way to view the space as a map, to navigate to certain spots and view where the teammates are. The integration is foreseen for the second year of the project.

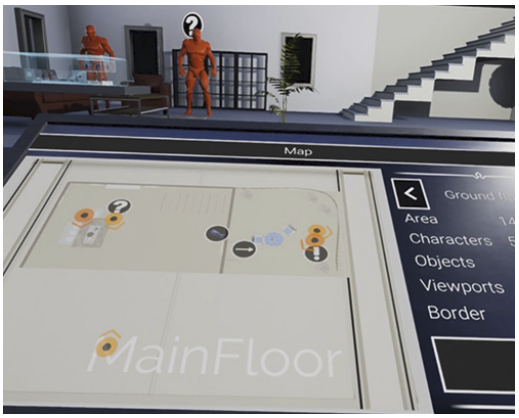


Figure 28: Spatial orientation baseline tool.

[26] https://speckle.xyz/graphql

[27] GraphDB - An enterprise ready Semantic Graph Database, compliant with W3C Standards https://graphdb.ontotext.com/

[28] https://www.unrealengine.com/marketplace/en-US/product/advanced-vr-framework

f.2 Toggle view mode

An interface was made in order to change the rendering mode. In Figure 29 and Figure 30, two examples are shown for the ZH Villa 3D model. Figure 29 is in full rendering mode with ray-tracing (called as Lit in UE4 terminology), whereas Figure 30a presents the clay mode, and Figure 30b the wireframe mode. The latter is called “Detailed Lighting” in UE4, because the textures are removed but the normals are kept. Other modes supported are the Wireframe, and the Unlit, where in the later all lights are removed and the materials emit their own basic color.

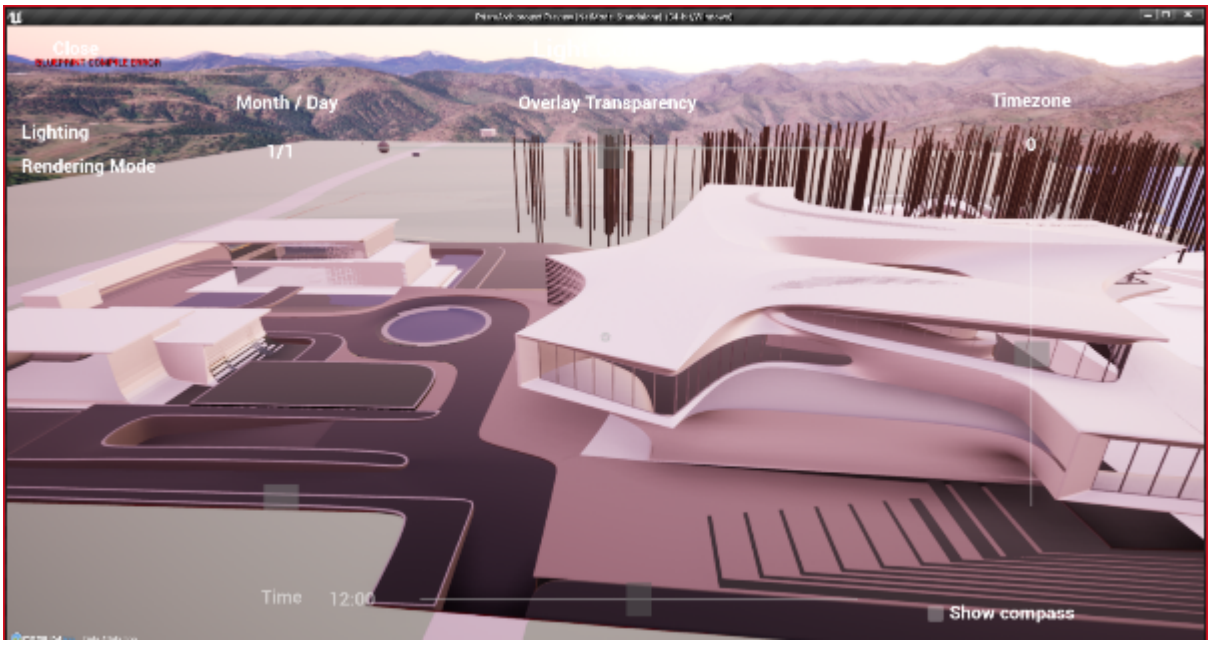
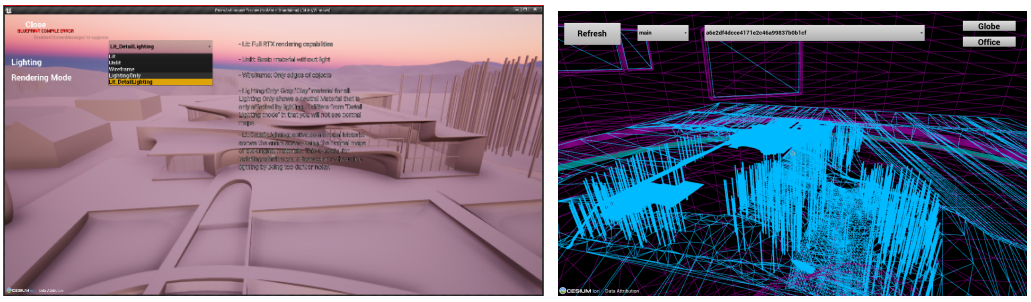


Figure 29: Full rendering mode.



(a) (b)

Figure 30: (a) Clay mode rendering (Detail Lighting); (b) Wireframe mode rendering

f.3 Toggle Camera Perspective Tool

This tool allows the user to view the project from several key perspectives repeatedly without having to travel to them each time. This can be achieved with the help of code extracted from the CollabViewer template of UE4 as it allows to place bookmarks in certain

spots and select these bookmarks via its VR Head Up Display as in Figure 31 “Bookmark”. This is foreseen to be integrated in PrismArch in the second year of the project.

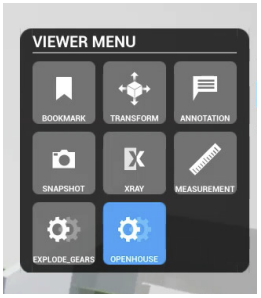


Figure 31: The HUD of Collabview. Upper left is the bookmark option to select spots for viewing.

f.4 Clipping Plane Tool

Provides ability to see and evaluate the cross-section of a 3D construction. This allows the user to view a cross-section of a building. It can be achieved with binary operations inside the Unreal Engine. To achieve this interpretation, a UX prototype was developed in Adobe XD (mock-ups, Figure 32 and Figure 33). It is a tool that will be embodied also in the tracing paper interface. Specific attention was given to the user’s navigating need while experiencing the section in a 3D plane and initial UI testing provided notes for further improvement in the direction of real-time section of more complex architectural geometries.

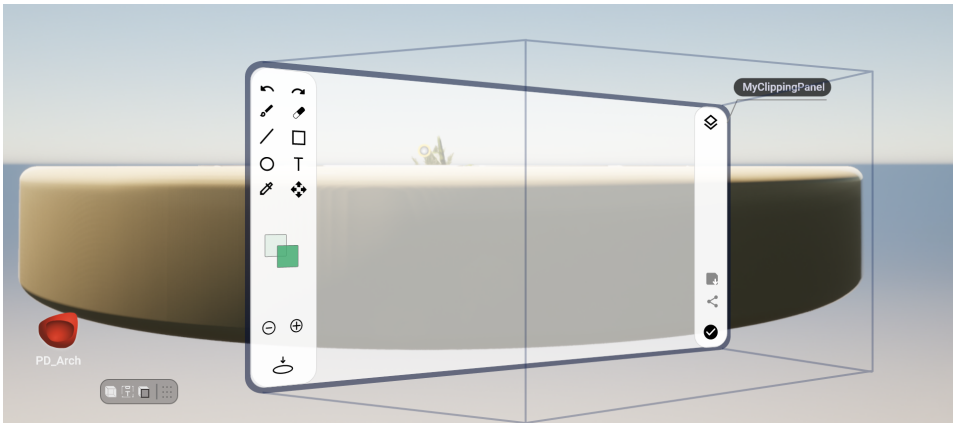


Figure 32: Clipping tool size selection mockup (cube).

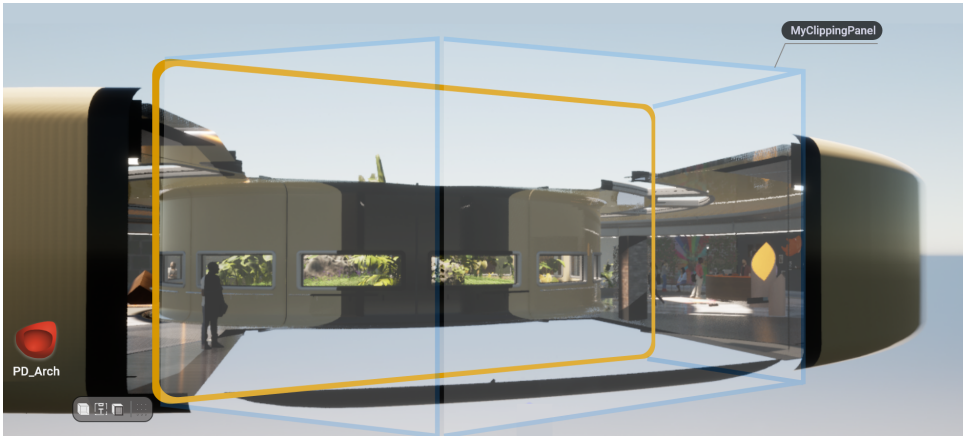


Figure 33: Clipping plane envisaged use in building section.

f.5 Map overlay tool and daylight simulation

A new requirement that was emerged during developments was the need of a geographical map that can be used:

a) for realistic daylight simulations due to the differentiation which is caused by the longitude and latitude of the building;
b) for browsing across projects in the globe; and
c) for retrieving the landscape in order to provide a higher level of realism.

Proper interfaces were developed for alternating month, time, and timezone as shown in Figure 34. Cesium provides the landscape in the form of a 3D mesh but allows also to insert custom photogrammetry data in order to update the landscape during architectural development changes.

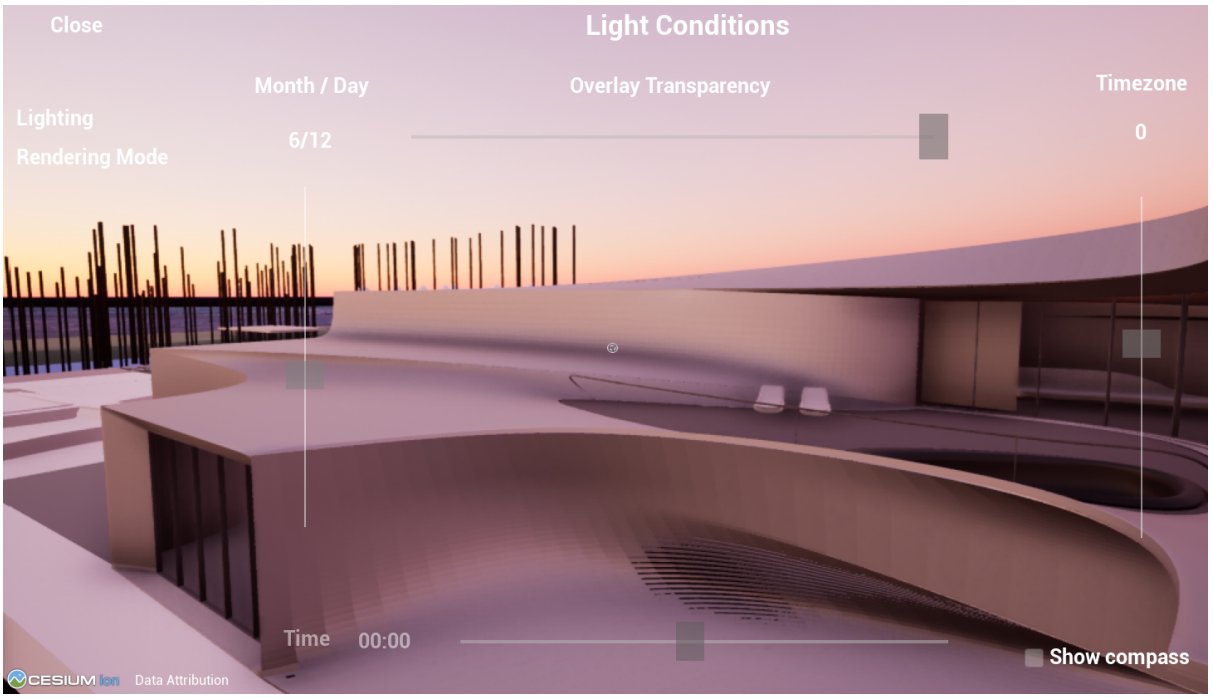


Figure 34: Map overlay allowed Dynamic sun position setting was feasible through Cesium maps.

3.3 Changing Speckle DB schema to store simulation information

Speckle Objects Schema is not covering all the possible information needed for PrismArch. Therefore, we have developed a C# library during PrismArch that allows to extend the schema with custom fields and store them in Speckle DB.

This initial C#-based library allows generic PrismArch data packages to be pushed to the Speckle database. These packages contain both generic PrismArch data - such as Textures, UV Mapping coordinates, and basic colours - and the bespoke PrismArch_Signature object (D1.1) required to uniquely identify and interact with different entities during a project.

Future development of the database interconnection will focus on forking and adjusting the default *Rhinoceros3D* and *Revit* Speckle Connectors, in order to generate custom PrismArch connectors for both of this software that generate PrismArch_Signatures by default.

This C# library is also used to implement a connector with SAP2000 (Structural analysis software). Work has progressed on the programming of a C#-based module that connects PrismArch to the structural engineering package SAP2000**.** This connection enables users within SAP2000 to export two sets of interrelated data to the PrismArch database:

- The underlying geometric definition of structural elements (Nodes, Beams, Columns, Meshes)
- The analysis results generated for those different elements, across multiple different types of analysis (displacement, utilisation, and so on).

In order to evaluate and visualise these structural models within the PrismArch VR platform, these models will be pulled from the database, and the geometric entities will be reconstructed, using standard Unreal mesh elements.

At this stage, the programmatic connection to SAP2000 is complete. There are still a few remaining structural Result Types that must be coded into the results 'package' sent to the database. Development must therefore now focus on the reconstruction of these structural entities inside PrismArch VR, and functionality for interacting with them, as outlined in *D1.2 Section 3.6.3*.

4. GITLAB REPOSITORY

The code is uploaded in a private Group in Gitlab.

https://gitlab.com/prismarch-h2020

It consists of the main PrismArch project (named temporarily as AdvancedVRFrameworkTemplate), the AI generative algorithms, the C# connector (AKT PrismArch), and the DeepSpeech models. The code is held privately due the confidential nature of the deliverables. Parts of the code such as the AI Generative Algorithms and DeepSpeech models for Architectural use cases will be made available as open source after the end of the project.

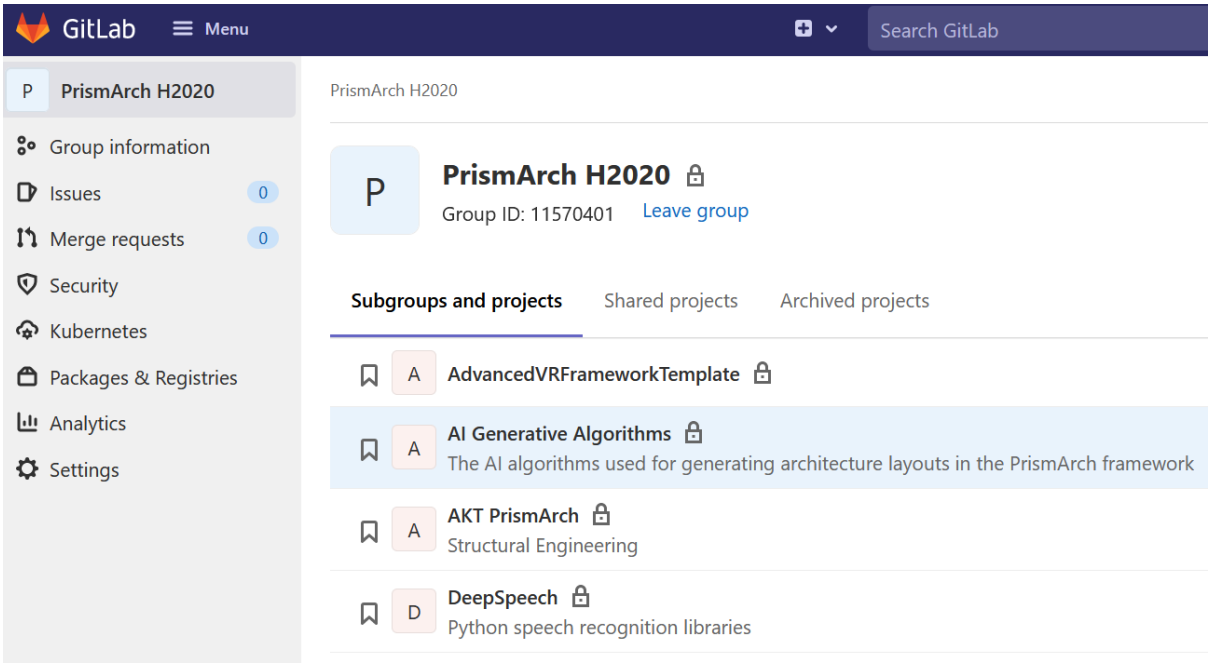


Figure 34: Indicative screen preview of the Gitlab PrismArch group.

The Unreal Engine 4.26 was used as it is the version that is supported by the majority of plugins used.

5. FUTURE DEVELOPMENTS ROADMAP

There are many PrismArch developments pending for the second year. However, there are 6 important functionalities that are higher prioritized. These can be found in Table 1.

Table 1: Prioritized tasks

#	Functionality	Comments
1	Push data from UE4 into Speckle (a.5)	Currently the interconnection with Speckle is unidirectional, i.e. from Speckle into UE4. We will collaborate with Speckle in order to allow committing from UE4 into Speckle for finalizing thus the asynchronous way of collaboration across UE4 and Rhino/Revit/SAP2000.
2	Develop more design interfaces (e.5)	The design tools in UE4 should be increased with nurbs and polysurfaces. The synchronous way of collaboration will be finalized for both directions, i.e. from UE4 to Rhino/Revit/SAP2000.
3	Multiplaying capability (Meeting Space)	Multiplaying has been integrated but more tests should be done in order to ensure the

		correct replication of actions across users.
4	C# library for extending DB schema and SAP2000 connector for transferring structural simulations information (a.5)	An initial C# was made for extending Speckle with PrismArch needed data. More developments should be made in order to commit the necessary data.
5	Register and Login (a.1)	The users should be able to login and register into Speckle DB through UE4.
6	Avatar configuration and representation (a.3)	Avatars representation as dummy robots is not sufficient. We foresee using more realistic human avatars that can be configured by users.

APPENDIX I

This Appendix contains examples for making queries into the Speckle database. It is used by the consortium in order to fetch data and develop API per platform used.

https://speckle.xyz/graphql

Query User	User info

query{
user {
name,
company
role
id
suuid
email
bio
avatar
}
}
	{
	"data": {
	"user": {
	"name": "Dimitrios Ververidis",
	"company": "CERTH",
	"role": "server:user",
	"id": "93c67be9d1",
	"suuid": "8f7f2b4d-4205-4f12-8f0c-d71ebacce054",
	"email": "ververid@iti.gr",
	"bio": null,
	"avatar": "...",
	
	}
	}
	}
	}

Query User Activity	Result

query{
user {
name
	{
	"data": {
	"user": {
	"name": "Dimitrios Ververidis",
	"activity": {
	"totalCount": 14,
	"items": [
	{

activity{
totalCount
items{
actionType
time
message
streamId
resourceType
resourceId
info
}
}
}
}
	"actionType": "stream_update",
	"time": "2021-09-17T01:53:45.059Z",
	"message": "Stream metadata changed",
	"streamId": "2ed9644a7c",
	"resourceType": "stream",
	"resourceId": "2ed9644a7c",
	"info": {
	"new": {
	"id": "2ed9644a7c",
	"isPublic": false
	},
	"old": {
	"id": "2ed9644a7c",
	"name": "One Thousand Museum Maimi",
	"isPublic": true,
	"createdAt": "2021-04-09T10:33:18.862Z",
	"updatedAt": "2021-09-03T06:25:39.030Z",
	"clonedFrom": null,
	"description": "No description provided."
	}
	}
	},
	{
	"actionType": "commit_create",
	"time": "2021-09-03T06:26:48.751Z",
	"message": "Commit created on branch globals: 4f41bdebf7 (333)",
	"streamId": "ddd17f1347",
	"resourceType": "commit",
	"resourceId": "4f41bdebf7",
	"info": {
	"id": "4f41bdebf7",
	"commit": {
	"message": "333",
	"objectId": "4623b01fa01fddc3690bd7ec9c8940a8",
	"streamId": "ddd17f1347",
	"branchName": "globals",
	"sourceApplication": "web"
	}
	}
	},
	{
	"actionType": "commit_create",
	"time": "2021-09-03T06:25:39.032Z",
	"message": "Commit created on branch globals: 2a49d5011b (111)",
	"streamId": "2ed9644a7c",
	"resourceType": "commit",
	"resourceId": "2a49d5011b",
	"info": {
	"id": "2a49d5011b",
	"commit": {
	"message": "111",
	"objectId": "522da64c9a4189bc853588bea352c0a6",
	"streamId": "2ed9644a7c",
	"branchName": "globals",
	"sourceApplication": "web"
	}
	}
	},
	{
	"actionType": "commit_create",
	"time": "2021-09-02T11:43:02.186Z",
	"message": "Commit created on branch globals: d1c25b8bfd (add lang-long)",
	"streamId": "2ed9644a7c",
	"resourceType": "commit",
	"resourceId": "d1c25b8bfd",
	"info": {
	"id": "d1c25b8bfd",
	"commit": {
	"message": "add lang-long",
	"objectId": "e14b9164d9fc38ee28d98273b83b09d9",
	"streamId": "2ed9644a7c",
	"branchName": "globals",
	"sourceApplication": "web"
	}
	}
	},
	{
	"actionType": "branch_create",
	"time": "2021-09-02T11:42:24.244Z",
	"message": "Branch created: 'globals' (d21bf72874)",
	"streamId": "2ed9644a7c",
	"resourceType": "branch",
	"resourceId": "d21bf72874",

Query for Streams	Results

query{
user {
name
streams{
totalCount
items{
id
name
	{
	"data": {
	"user": {
	"name": "Dimitrios Ververidis",
	"streams": {
	"totalCount": 16,
	"items": [
	{

description isPublic role createdAt updatedAt } } } }	"id": "2ed9644a7c", "name": "One Thousand Museum Maimi", "description": "No description provided.", "isPublic": false, "role": "stream:owner", "createdAt": "2021-04-09T10:33:18.862Z", "updatedAt": "2021-09-17T01:53:45.056Z" }, { "id": "ddd17f1347", "name": "One Park Drive", "description": "No description provided.", "isPublic": true, "role": "stream:owner", "createdAt": "2021-07-01T12:13:43.733Z", "updatedAt": "2021-09-03T06:26:48.747Z" }, { ….

Collaborators per Stream	Result

query{
user {
name
streams{
totalCount
items{
id
name
description
isPublic
role
createdAt
updatedAt
collaborators{
id
name
role
company
avatar
}
}
}
}
}
}
	{
	"data": {
	"user": {
	"name": "Dimitrios Ververidis",
	"streams": {
	"totalCount": 16,
	"items": [
	{
	"id": "2ed9644a7c",
	"name": "One Thousand Museum Maimi",
	"description": "No description provided.",
	"isPublic": false,
	"role": "stream:owner",
	"createdAt": "2021-04-09T10:33:18.862Z",
	"updatedAt": "2021-09-17T01:53:45.056Z",
	"collaborators": [
	{
	"id": "d21dd8ede8",
	"name": "Risa Tadauchi",
	"role": "stream:contributor",
	"company": null,
	"avatar":
	"https://lh4.googleusercontent.com/-f6PhhCT7NIQ/AAAAAAAAAAI/AAAAAAAAAAA/AMZuucm_4KLaisdQ9lbqUpqVpNiB94kIVA/s96-c/photo.jpg"
	},
	{
	"id": "190748faa2",
	"name": "Daria Zolotareva",
	"role": "stream:contributor",
	"company": "Zaha Hadid Architects",
	"avatar": null
	},
	{
	"id": "8d776c0d65",
	"name": "Konstantinos Sfikas",
	"role": "stream:contributor",
	"company": "University of Malta",
	"avatar": null
	},
	{ ….

Query for Commits information	Results

query{
stream (id: "2ed9644a7c"){
id
name
createdAt
updatedAt
commits{
totalCount
cursor
items{
id
referencedObject
authorName
authorId
createdAt
}
}
}
}
	{
	"data": {
	"stream": {
	"id": "2ed9644a7c",
	"name": "One Thousand Museum Maimi",
	"createdAt": "2021-04-09T10:33:18.862Z",
	"updatedAt": "2021-09-17T01:53:45.056Z",
	"commits": {
	"totalCount": 10,
	"cursor": "2021-04-09T10:35:04.885Z",
	"items": [
	{
	"id": "d3b1d1a4d6",
	"referencedObject": "7a181f7c0c427bc3436f498f593cf2bc",
	"authorName": "Dimitrios Ververidis",
	"authorId": "93c67be9d1",
	"createdAt": "2021-04-09T11:41:46.520Z"
	},
	{
	"id": "58e5c2cd70",
	"referencedObject": "89578aa6895622faf5c2055b382ff2a3",
	"authorName": "Konstantinos Sfikas",
	"authorId": "8d776c0d65",
	"createdAt": "2021-04-09T11:08:24.749Z"
	},
	{
	"id": "7b0b7be67e",
	"referencedObject": "197ef67478b9aed2782bed791a3f2469",
	"authorName": "Daria Zolotareva",
	"authorId": "190748faa2",
	"createdAt": "2021-04-09T11:00:04.077Z"
	},
	{
	"id": "3a9594e39e",
	"referencedObject": "26cdfb024af5d42e6a343ea1821af16f",
	"authorName": "Konstantinos Sfikas",
	"authorId": "8d776c0d65",
	"createdAt": "2021-04-09T10:58:13.074Z"
	},
	{
	"id": "15996c6dfe",
	"referencedObject": "55a9cad7cf667d0249ae05473ce2465a",
	"authorName": "Konstantinos Sfikas",
	"authorId": "8d776c0d65",
	"createdAt": "2021-04-09T10:47:00.811Z"
	},
	{
	"id": "88bb18fc03",
	"referencedObject": "100d9ba2de273bbbb6a2b6db8a6fe09b",
	"authorName": "Risa Tadauchi",
	"authorId": "d21dd8ede8",
	"createdAt": "2021-04-09T10:46:55.136Z"
	},
	{
	"id": "865c3b1a9a",
	"referencedObject": "100d9ba2de273bbbb6a2b6db8a6fe09b",
	"authorName": "Risa Tadauchi",
	"authorId": "d21dd8ede8",
	"createdAt": "2021-04-09T10:46:33.533Z"
	},
	{
	"id": "b305421a4c",
	"referencedObject": "100d9ba2de273bbbb6a2b6db8a6fe09b",
	"authorName": "Risa Tadauchi",
	"authorId": "d21dd8ede8",

	"createdAt": "2021-04-09T10:46:18.012Z"
	},
	{
	"id": "6c7d157889",
	"referencedObject": "2f3e808ee4e7e85f47c0f6ab9fb3520f",
	"authorName": "Dimitrios Ververidis",
	"authorId": "93c67be9d1",
	"createdAt": "2021-04-09T10:41:37.293Z"
	},
	{
	"id": "4f8ac02a61",
	"referencedObject": "09406b1532dd63f751972f881194e372",
	"authorName": "Dimitrios Ververidis",
	"authorId": "93c67be9d1",
	"createdAt": "2021-04-09T10:35:04.885Z"
	}
	]
	}
	}
	}
	}

APPENDIX II

Examples of SPARQL queries execution by the semantic retrieval service to support various scenarios

Commit history

Returns all commit history information including stream, commit and object identifiers, authors and dates.	
Input	{}

SPARQL query	PREFIX pm: <http://www.semanticweb.org/prismarch-ontology#>
	PREFIX express: <https://w3id.org/express#>
	select * where {
	?str a pm:Stream.
	?str pm:hasBranch ?br.
	?br a pm:Branch.
	?br pm:hasCommit ?s.
	?s a pm:Commit .
	?s pm:hasCommitInfo ?pi.
	OPTIONAL {?pi pm:hasAuthor ?author.}
	?pi express:hasString ?obj_id.
	?str express:hasString ?s_id.
	?pi pm:hasCommitId ?c_id.
	OPTIONAL {?pi pm:hasDate ?date.}
	}

Output	{
	"Commits": [
	{
	"date": "Issue Date",
	"streamId": "d725ef328c",
	"author": "Samuel Macalister",
	"commitId": "dcf5f6e3be",
	"objectId": "24cfb1e7f00a418b0e4aded2d979e102"
	},
	{
	"streamId": "d725ef328c",
	"commitId": "dcf5f6e3be",
	"objectId": "9439a781267858e500bc28ff9909acdf"
	}
	]

	}

Author history

Given a specific object identifier, returns the date and the author that generated it.	

Input	{
	"objectId": "9439a781267858e500bc28ff9909acdf"
	}

SPARQL query	PREFIX pm: <http://www.semanticweb.org/prismarch-ontology#>
	PREFIX express: <https://w3id.org/express#>
	select * where {
	?str a pm:Stream.
	?str pm:hasBranch ?br.
	?br a pm:Branch.
	?br pm:hasCommit ?s.
	?s a pm:Commit .
	?s pm:hasCommitInfo ?pi.
	?pi pm:hasAuthor ?author.
	?pi pm:hasDate ?date.
	?pi pm:hasSourceApp ?source.
	?pi express:hasString ?object_id.
	FILTER REGEX (?object_id, "9439a781267858e500bc28ff9909acdf")
	}

Output	{
	"Commits": [
	{
	"date": "2021-05-05T07:12:29.405Z",
	"author": "Dimitrios Ververidis",
	"source": "Grasshopper",
	"objectId": "9439a781267858e500bc28ff9909acdf"
	}
	]
	}

Cost information

Given a specific object identifier, detects the cost of it	

Input	{
	"objectId": "37902f721d1e6aff15d18274e67ee838"
	}

SPARQL query	PREFIX express: <https://w3id.org/express#>
	PREFIX pm: <http://www.semanticweb.org/prismarch-ontology#>
	PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	select * where {
	?s rdf:type ?type.
	?s express:hasString ?o_id .
	?s pm:hasCost ?cost.
	?s rdfs:label ?lbl.
	FILTER REGEX (?o_id, "37902f721d1e6aff15d18274e67ee838")
	}

Output	{
	"cost": "15"
	}

Room information

Given a specific room type, detects the object identifiers that correspond to it	

Input	{
	"type": "Bath"
	}

SPARQL query	PREFIX pm: <http://www.semanticweb.org/prismarch-ontology#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX express: <https://w3id.org/express#>

	select * where {
	?room a pm:IfcRoom .
	?room express:hasString ?id.
	?room rdfs:label ?room_type.
	?room pm:inLevel ?level.
	FILTER regex(?room_type, "Bath")
	}

Output	{
	"Room_ids": [
	"01c9a0304285d9e74a1d98bc49050288",
	"55c1772bfa5230f42486d7affaa87b09",
	"de1c9ceb6378064f57f92bf1c1e8b0ed",
	"e42e91f093b45f1776bbfe47cb8d1398"
	]
	}

Object history

Given a specific application id, detects all commits that contain the specific object along with available additional object information	

Input	{
	"applicationId": "a6aa132d-ccd7-408f-b2f9-ed67350c8c3a-0003b64a"
	}

SPARQL query	PREFIX pm: <http://www.semanticweb.org/prismarch-ontology#>
	PREFIX express: <https://w3id.org/express#>
	PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
	select * where {
	?str a pm:Stream.
	?str pm:hasBranch ?br.
	?br a pm:Branch.
	?br pm:hasCommit ?cmm.
	?cmm a pm:Commit .
	?cmm pm:hasCommitInfo ?l.
	?str express:hasString ?s_id.
	?l pm:hasCommitId ?c_id.
	?l pm:hasAuthor ?author.
	?l pm:hasDate ?date.
	?l pm:hasSourceApp ?source.
	?l express:hasString ?model_id.
	?a pm:hasStructuralAspects ?c.
	?c ?w ?s.
	?s pm:hasApplicationId ?app_id .
	?s express:hasString ?o_id .
	OPTIONAL { ?s pm:hasUnit ?unit . }
	OPTIONAL { ?s rdfs:label ?lbl. }
	OPTIONAL { ?s pm:hasArea ?area. }
	OPTIONAL{ ?s pm:hasFamily ?family .}
	OPTIONAL { ?s pm:hasVolume ?volume. }
	OPTIONAL { ?s pm:hasArea ?area. }
	OPTIONAL { ?s pm:hasCost ?cost. }
	OPTIONAL { ?s pm:hasHeight ?height. }
	OPTIONAL { ?s pm:hasCategory ?category. }
	OPTIONAL { ?s pm:inLevel ?level. }
	OPTIONAL { ?s pm:hasMesh ?mesh.
	?mesh express:hasString ?mesh_id. }
	OPTIONAL { ?s pm:hasMaterial ?material.
	?material express:hasString ?material_id.
	?material pm:hasOpacity ?opacity.
	?material pm:hasMetalness ?metalness.
	?material pm:hasRoughness ?roughness. }
	FILTER REGEX (?app_id, "a6aa132d-ccd7-408f-b2f9-ed67350c8c3a-0003b64a")
	}

Output	{
	"Commits": [
	{
	"date": "2021-05-05T07:12:29.405Z",
	"unit": "mm",
	"streamId": "1421b874ed",
	"modelId": "9439a781267858e500bc28ff9909acdf",
	"level": "Roof Line",
	"author": "Dimitrios Ververidis",
	"commitId": "92395333eb",
	"source": "Grasshopper",
	"applicationId": "a6aa132d-ccd7-408f-b2f9-ed67350c8c3a-0003b64a",
	"type": "SG Metal Panels roof",
	"objectId": "eefd8e082a53d3727d4579d285e6a52b"
	},
	{
	"date": "2021-04-29T10:37:01.682Z",
	"unit": "mm",
	"streamId": "1421b874ed",
	"modelId": "10a2e90d5ca463b12d6b6d7c2eb6920a",
	"level": "Roof Line",
	"author": "Dimitrios Ververidis",
	"commitId": "36f02a35ce",
	"source": "Revit2021",
	"applicationId": "a6aa132d-ccd7-408f-b2f9-ed67350c8c3a-0003b64a",
	"type": "SG Metal Panels roof",
	"objectId": "ee6c707f7fd8619dff2273383b06f856"
	}
	]
	}