PrismArch

Deliverable No D1.2

Elaborated report of cross-discipline principles-rules-constraints, and interfaces definition for cross-disciplinary and multi-simulation perspectives 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
Duration:	24 months
Due date of deliverable:	31st August 2021
Actual submission date:	18 September 2021
Version:	1.0
Main Authors:	Jeg Dudley (AKT II), Helmut Kinzler (ZHA), Arun Selvaraj (SWECO)

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

Deliverable title	Elaborated report of cross-discipline principles-rules-constraints, and interfaces definition for cross-disciplinary and multi-simulation perspectives in VR
Deliverable number	D1.2
Deliverable version	Alpha
Contractual date of delivery	31 August 2021
Actual date of delivery	18 September 2021
Deliverable filename	PrismArch_D1.2_MainPrinciplesInterconnections_v0.1
Type of deliverable	Report
Dissemination level	PU
Number of pages	175
Work package	WP1
Task(s)	T1.2 and T1.3
Partner responsible	AKT II
Author(s)	AKT II: Georgios Adamopoulos, Jeg Dudley, Joel Hilmersson, Naomi Lea. ZHA: Helmut Kinzler, Daria Zolotareva, Risa Tadauchi, Aleksandra Mnich-Spraiter. SWECO: Arun Selvaraj, Oussama Yousfi. UoM: Konstantinos Sfikas.
Editor	Jeg Dudley (AKT II)
Reviewer(s)	Spiros Nikolopoulos (CERTH) , Martin Brösamle (ETH)

Abstract	Document outlines the technical requirements, potential AEC ontologies and VR interfaces that must be implemented to realise the PrismArch platform described in previous PrismArch Deliverables.
Keywords	AEC, interoperability, data exchange, architecture, structural engineering, MEP engineering, VR, virtual reality, UI design, UX design, PrismArch.

Copyright

Deliverable history

Version	Date	Reason	Revised by
0.1	23/07/2021	Table of Contents	Dimitrios Ververidis (CERTH).
0.2	24/08/2021	First draft from AEC Partners	Jeg Dudley (AKT II). Helmut Kinzler, Risa Tadauchi and Daria Zolotareva (ZHVR).
0.3	08/09/2021	Additions from WP1, WP2 and WP4 Partners.	Konstantinos Sfikas (UoM). Dimitrios Ververidis (CERTH). Arun Selvaraj (SWECO).
0.4	10/09/2021	Editing of Partners comments complete. Sent for Internal Review.	Jeg Dudley (AKT II)
0.5	16/09/2021	Internal Review complete. Final edits based on Reviewers comments.	Jeg Dudley and Geogios Adamopoulos (AKTII).
1.0	18/09/2021	Formal Submission.	Jeg Dudley (AKT II).

List of abbreviations and Acronyms

Abbreviation	Meaning
AB	Advisory Board
BIC	Bank Identifier Code
CA	Consortium Agreement
CFS	Cost Financial Statement
DoA	Description of Action
DR	Deliverable Responsible
EC	European Commission
GA	Grant Agreement
GNU	GNU is not Unix
IBAN	International Bank Account Number
IoT	Internet of Things
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
ToC	Table of Contents
UML	Unified Modeling Language
QMR	Quarterly Management Report
WP	Work Package
WPL	WP Leaders
AEC	Architecture, Engineering and Construction
AR	Augmented Reality
BIM	Building Information Modelling
CAD/CAM	Computer-Aided Design & Computer-Aided Manufacturing
ICT	Information and communication technology
NDA	Non Disclosure Agreements
SME	Small and Medium-sized Enterprises
UG	User Group

VR	Virtual Reality

Executive Summary

Deliverable D1.2 builds upon the cross-disciplinary perspectives of the AEC industry discussed in D1.1, by providing a more detailed description of the specific software framework ('ontology') and functionalities that must be implemented to achieve the PrismArch platform.

These topics are grouped under two headings: *How data should be exchanged between different software packages and databases* (Section 2), and *How that data should be visualised and interacted with inside the VR environment* (Section 3).

Section 2 begins by first analysing in-depth several existing AEC ontologies, and evaluating their relative suitability for the PrismArch platform. Various PrismArch-specific requirements are then described in detail, and solutions are proposed for how they could be integrated into the chosen software ontology.

Section 3 describes a series of studies undertaken by the AEC partners to determine the most efficient interfaces that will allow architects, structural engineers and MEP engineers to collaborate within a shared VR environment, while respecting and enhancing the small number of discipline-specific UI requirements that also exist. These studies build upon the research undertaken in previous Deliverables - particularly D3.1 and D6.1 - and take the form of user interviews, followed by functionality maps, which together inform the design of the VR interfaces that are proposed at the end of this document.

Table of Contents

-

1.0 INTRODUCTION

The intention of this Deliverable is to elaborate further on the critical aspects of the PrismArch software framework and functionalities outlined across several previous Deliverables - such as D1.1 and D6.1 - and thus provide a detailed roadmap for their technical implementation.

Note: Throughout this document, we will refer to the software framework as the 'ontology'. We believe this word is more accurate than framework, as ontology is a standard term, used frequently in software development, that encompasses not only the categorisation and grouping of data types and objects, but also the underlying philosophy of how these entities interact within one another. [Liu, 2009].

Within an ontology are descriptions of all the different types of object that it contains. In an AEC setting, these include objects like Walls, Beams, Columns, and so forth. These objects are also sometimes called data types, elements, structures, entities, or schemas. Throughout this document we will primarily use the latter term, 'schema', especially when either discussing the development of these objects, or explicating the properties contained within them.

Establishing these critical aspects of PrismArch is no easy task, as the PrismArch project touches on a number of highly technical and non-trivial issues that exist within the digital domain of the AEC industry - namely, interoperability between software, the concept of model fidelity and data-rich BIM, authorship and attribution of intellectual property within multi-author environments, and lastly, the realisation and design of AEC within VR spaces. Even that final item, which might superficially appear straight forward, is complicated - for reasons that are described throughout in this document.

Therefore, to negotiate these topics, this Deliverable has been separated into two distinct sections:

Section 2 describes how the underlying data of PrismArch should be structured. This data includes the descriptions of typical schemas such as beams, walls and columns, but also of often-overlooked data such as architectural sketches, site photographs and drawing markups. It does so by first analysing in-depth several existing AEC ontologies, and evaluating their relative suitability for the PrismArch platform. This section then discusses requirements that are either PrismArch-specific, or that we feel have not been fully addressed by existing ontologies.

This section concludes by describing in detail how these requirements could be integrated into the chosen software ontology. Accompanying this section is a *Code Library*, containing initial prototypical C# versions of many of the PrismArch schemas outlined earlier in the document. This Code Library is available through the PrismArch GitLab account [GitLab].

Section 3 contains a series of studies undertaken by the AEC partners to determine the most efficient interfaces that will allow architects, structural engineers and MEP engineers to collaborate within a shared VR environment, while respecting and enhancing the small number of discipline-specific UI requirements that also exist. These studies build upon the research undertaken in previous Deliverables - particularly D3.1 and D6.1 - and take the form of user interviews, followed by functionality maps, which together inform the design of the VR interfaces that are proposed at the end of this document. As in Section 2, there are a series of 3D models and example code that accompany these studies, and which should support their technical implementation.

-

● 2.0 PRINCIPLES, RULES AND CONSTRAINTS FOR PRISMARCH

○ 2.1 Overview

■ 2.1.1 Introduction

PrismArch is a multidisciplinary platform for collaboration in virtual reality. In order for disciplines to communicate and exchange information about their specific domains, a common framework for information exchange within this platform must be established. This means a common, shared format for the exchange and schematisation of the associated knowledge domains.

The purpose of this part of the report is to examine various pre-existing approaches and formulations of ontologies and data structures, and eventually establish a framework for how the PrismArch platform will handle the information it needs to process, in order to facilitate a collaborative design process. In order to enable computers to directly exchange information, they need to be defined in a generic and computable format, i.e an ontology. Ontologies formalize and represent domain objects by mapping them to a common vocabulary or set of rules.[Issa et. al 2015]

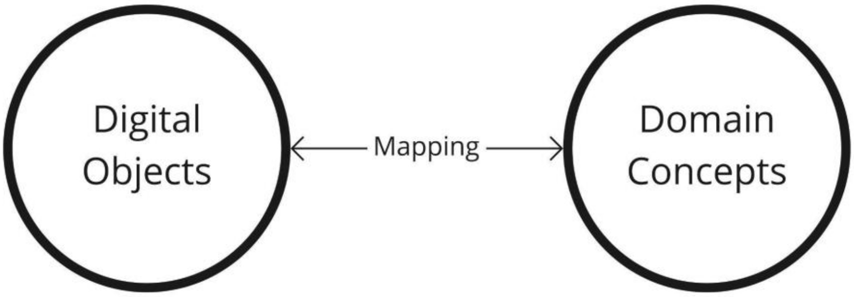


Figure 2.1.1a - Mapping between objects and concepts.

In the end, the outcome of this chapter is a single proposed ontology, which will form the backbone of the general exchange of information within the PrismArch environment. The structures within this ontology will feed into both the underlying algorithmic processes, as well as being injected into the VR environment for interrogation by humans in immersive 3D space.

■ 2.1.2 Approach

This study consists of three distinct parts, which together enabled us to determine the final concept proposed, and which explain the reasoning behind our decisions:

1 - Precedent Study

PrismArch will not exist in a vacuum, nor will it be the first platform to see the need for an ontological description of the domain that is Architecture, Engineering and Construction (AEC) [Tarandi 1998, Issa et al 2015]. This means that the project does not start from scratch when it comes to establishing a baseline for such principles, and an understanding of the current processes is needed. Further, this also requires the platform to position itself in relation to both present and potential future modes of data exchanges, in order to interact with them.

The precedent study examines and discusses previous approaches to ontological descriptions and interoperability. It describes some different solutions to this class of problem, spanning from exhaustive, all-encompassing descriptions conceived by industry consortia to more specific tools developed within companies to deal with their own, internal, disconnections and inefficiencies.

A selection of precedents is made to focus on a set of tools which, although they all fundamentally try to solve a similar type of problem, vary in scope, implementation, and philosophy. They will be examined and presented in regards to two aspects - namely their overall structure and the complexity of implementation, as the conceptual approach to structuring data in a digital context cannot be created, thus also not understood, in isolation from the technology that delivers it. [Laakso et. al 2012]

2 - Formulation of Requirements

Based on the lessons learned from the precedents, a reflection is made in relation to the needs and functionality envisioned for the PrismArch environment. Here the specifics of each precedent are evaluated based on their general suitability and applicability, and to what degree any pre-existing concept forms a natural base for the PrismArch platform to build upon.

This is synthesized into a set of requirements, to which the ontological approach for the PrismArch data organization or mapping must adhere, based on current state-of-the-art and reflection and input from all three disciplines in relation to their own work. Further, it also discusses and establishes what amendments, modifications or extensions potentially needs to be made to an existing framework in order for it to fulfill the desired functionality of the PrismArch environment.

3 - Proposed Object Structures

The chapter will then culminate in the proposed structures for the PrismArch platform. It will not result in exhaustive descriptions, but schematic outlines on how to deal with the expected complexity inherent in a multi-simulation, collaborative VR framework. Some examples are presented to concretize the explanations along with some reference snippets of code implementation.

These proposals describe three conceptually different approaches to the schematisation, based on the findings and integration with the reviewed precedents, and merged with the specific needs of the platform. As there is not necessarily a single way to structure and implement this, our set of proposals will provide a framework for discussion. As such, the advantages and disadvantages of the three proposals are listed and reflected upon.

Finally, a single concept is highlighted as the most promising one, which will form the base for more elaborate descriptions.

○ 2.2 Precedent Study of Existing AEC Ontologies

■ 2.2.1 Ontologies, Interoperability and the AEC industry

2.2.1.1 The Wider Problem

Before detailed elaboration on all the selected frameworks, a general introduction to the wider problem at hand is needed. This is not a problem in direct relation to the VR aspects, but to the nature of collaboration, and more specifically the processes of information exchange within the industry at large. Why do we need ontological descriptions of our knowledge domain?

The concept of ontologies are closely related to the topic of interoperability. This is a well known, and still remarkably unresolved, issue which has haunted the industry ever since the computer became a commonplace feature in the offices of architects and engineers alike. [Hamil 1994, Ekholm 2005] In the analogue days of building design, information was communicated through drawings. [Laakso et. al 2012]. Handling drawings means just handling geometric representation to which a human needs to assign some meaning, either through symbolic conventions, or simply through visual inspection of the geometry. In the absence of any interoperability approach, to transfer information from one package to another, the human will need to interpret the information present in one package, and type in the equivalent information in the receiving package. Interoperability is the concept of facilitating computer-to-computer exchanges, where this is done in an automated fashion. Ontologies provide a framework for digitizing this process, where each cluster of information is in the form of an object or schema, which both the sending and receiving software know how to handle. The contents of one software can automatically be made sense of by another.

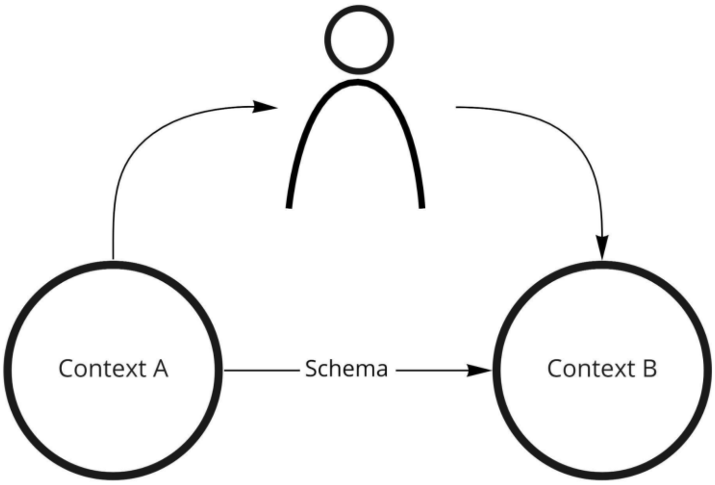


Figure 2.2.1.1a - Schemas should provide an intelligently structured format for data, that allows it to be smoothly transferred between different contexts, thus avoiding the requirement for human intervention.

This is often done in an object oriented fashion, [Rumbaugh et al. 1991] where objects in the knowledge domain are mapped to computer objects with properties that capture the nature of the domain object. An example could be a beam or a wall in a building which, based on their properties, can be rendered to the screen, but the underlying object is much more than only the geometric description [Hamil 1994]. This is what is often referred to as Building Information Modeling (BIM), and these objects can be used to facilitate the exchange of information from one package to another.

However easy this may sound, the waters get muddied as the industry consists of a wide range of disciplines, all providing their specific expertise and having their own sets of responsibilities, coupled with an ever-expanding set of objects and associated data. It has been noted that one of the reasons for the widespread problem of interoperability in the AEC industry is due to its fragmented nature, with each project containing a wide range of specialists with very varying adoption of software and IT processes. [Laakso et. al 2012]

In relation to this, each discipline also has their own objects of interest and data to support them in this process. A partial common ground can potentially be found, as there is some degree of overlap, but there is also a disconnection where multiple disciplines may share an object but relate to it in different ways, assigning different relevance or meaning to them. This concept has been previously highlighted and discussed in Deliverable 3.1 - see Section 2.1 regarding *boundary objects*.

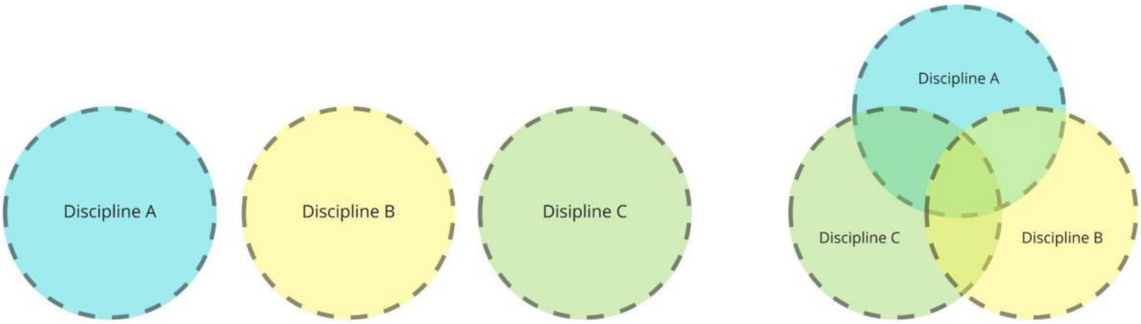


Figure 2.2.1.1b - Overlap between differing disciplines

But even within the local process of a single discipline, or even for a single person in their day-to-day work, a wide set of tools will be used for documenting, sketching, analysing, etc. This is a cause of great inefficiency, as data constantly has to be reinterpreted and reentered in various, subtly different ways in a wide range of software and documentation. [Tibuzzi 2016] An unambiguous, deterministic mapping from one to another is often not possible, as there is always a variation in the domain covered by each software.

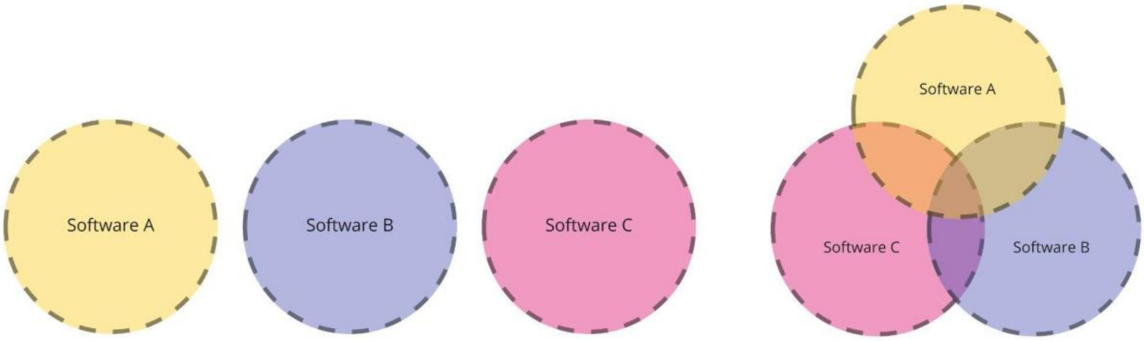


Figure 2.2.1.1c - Overlap between differing software. *Not* identical to the overlap between disciplines.

2.2.1.2 A Selection of Example Approaches

As the problems listed so far have been around for a long time, during this time a wide range of solutions have emerged. A non-exhaustive list of the ones most relevant for PrismArch are given below:

Name	Developer	Type / Domain	Software	Link
IFC	*buildingSMART*	ISO standard BIM	Various	Link
Speckle	*SpeckleWorks*	Various	Various	Link
Reakt	*AKT II*	BIM Structural	Rhino, Revit Structural	*Not available*

BHoM	*BuroHappold*	BIM Structural	Various	Link
Konstru	*Thornton Tomassetti*	BIM Structural	Rhino, Revit Structural	Link
Conveyor	*ProvingGround*	BIM	Rhino, Revit	Link
Beam	*MKS DTECH*	BIM	Rhino,Revit	Link

However, all of these examples are not necessary to study in further depth. The focus will mainly be on two open standards for interoperability which transcends disciplines, as this is in line with the nature of PrismArch, namely the IFC format and the Speckle data platform. Both of these represent very different approaches to a similar problem and attempt to cover the wider industry. Some smaller toolkits, which only exchange data between two different software, may also not really need a generic framework, or ontology, as it simply directly translates between them. Further, the study will also bring up two frameworks which are developed within AEC offices to tackle their own problems, as a counterweight to the open standards. Here, the first framework discussed will be that of the authors (AKT II), called *Reakt*, followed by a similar one developed at BuroHappold engineering, which is the *BHoM* toolkit. Thus, the precedent study will be composed of the following set of tools:

- IFC
- SPECKLE
- REAKT
- BHoM

■ 2.2.2 IFC and BCF

2.2.2.1 A brief introduction to IFC

The first topic in the reference study is the oldest, and probably most famous one, as it currently positions itself as the main exchange format for the AEC industry. [Berlo et al 2012] It is that of the Industry Foundation Classes, or the IFC format. The IFC was one of the first and remains most widely accepted large-scale models of object based thinking in the AEC industry, and traces its origins back to the 1990s and the early days of computer processes [Berlo et al 2012, Hamil 1994]. With the emergence of the computer aided design (CAD) paradigm, the first packages replicated the drawing based workflows, early CAD packages focused on the rendering of lines etc on the screen mimicking the information of analogue drawings. The IFC model brought the promise of an object-based approach, where things would not simply be geometries, but complex objects like a door or a wall or a window which, while still being rendered as some geometry on the screen, was based on a much more sophisticated classification. Underlying objects with a meaning would allow for intelligent systems with awareness of building constraints propagating across software and disciplines. A building part in one software could potentially be aware of certain constraints in the presence of another object, and throw warnings or adjust accordingly. [Hamil 1994]

The initiative is maintained by buildingSMART, whose work focuses on "*standardizing processes, workflows and procedures for openBIM enabling digital transformation"* [buildingSMART]. The ambition of the IFC model is a complete categorization of all the possible objects, constraints and properties necessary to capture and describe the entire building process. Currently IFC contains more than 1000 elements and describes a large set of objects spanning all the way from ducts, doors and beams to quantity definitions, persons and subcontract resources.

It should be clarified that IFC is neither a software nor a file format, but a standard for structuring information, and the intention was to create a high-level structure which sits above any software implementation. Today, some degree of support for import and export of IFC structured data can be found in most BIM software [ArchiCad, REVIT], and some other software [SAP2000] and is the most widely used BIM exchange format [Berlo et al 2012]. Despite this, and the fact that it has been present for a long time, the general adaptation of the format is still fairly low [Laakso et al 2012]. Reasons for this will be discussed later on.

2.2.2.2 Central Object Model Concept

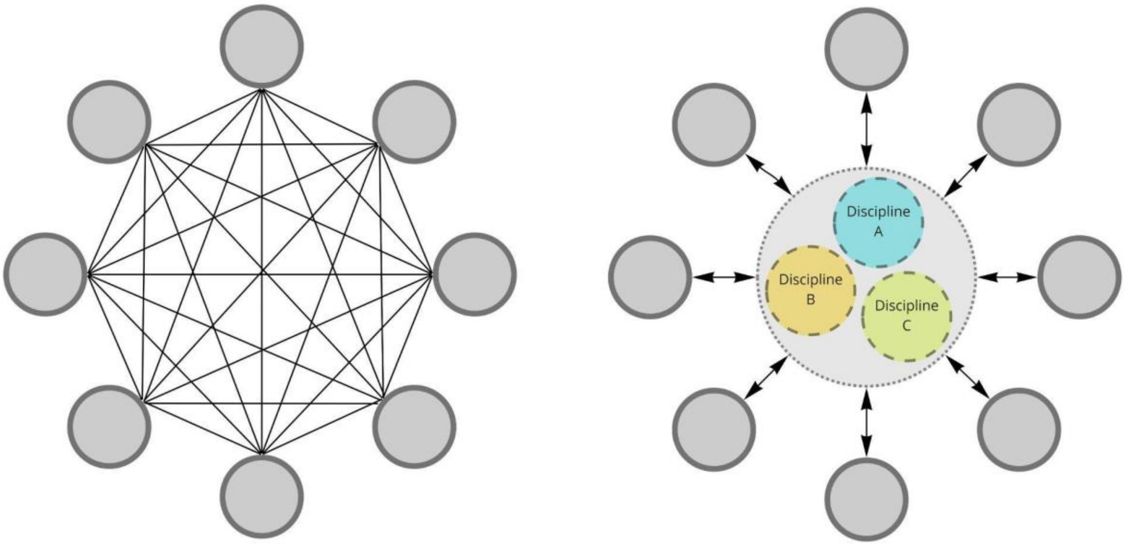


Figure 2.2.2.2a - Inefficient connection between software (left), versus the efficient shared object model (right).

The main principle of the IFC standard is the open, shared model. The picture, shown above, is now a familiar one in interoperability concepts (as we shall see later on). An open data format - into which all other software can read and write - acts as a middle man, indirectly connecting them all. An interface format, which theoretically can be in the form of a file, database or server. [Berlo et al 2012]

Direct translation from each software would create an immensely complex network of exchanges, unable to scale as each new connection would require additional translators to be written for each software. By adapting the shared model concept, this is reduced to two per software, as they all simply have to convert back and forth to the shared format.

2.2.2.3 The Structure of IFC

For the IFC model to handle the complexity of mapping the entire AEC domain, a strictly hierarchical, conceptual structure is used. At large the IFC data model is divided into four conceptual layers; which are *domain*, *interoperability*, *core* and *resource* layers.

The layers follow strict referencing hierarchies in an inheritance ladder, making objects within each layer referencing only allowed to reference the ones above, or within its own layer. A driving motivation is a modular structure, making sure that the model can be easily maintained and components can be reusable for software vendors or information modelers. [IFC Architecture Guide 1999].

The layers and their interrelationships are highlighted in figure [IFC Documentation].

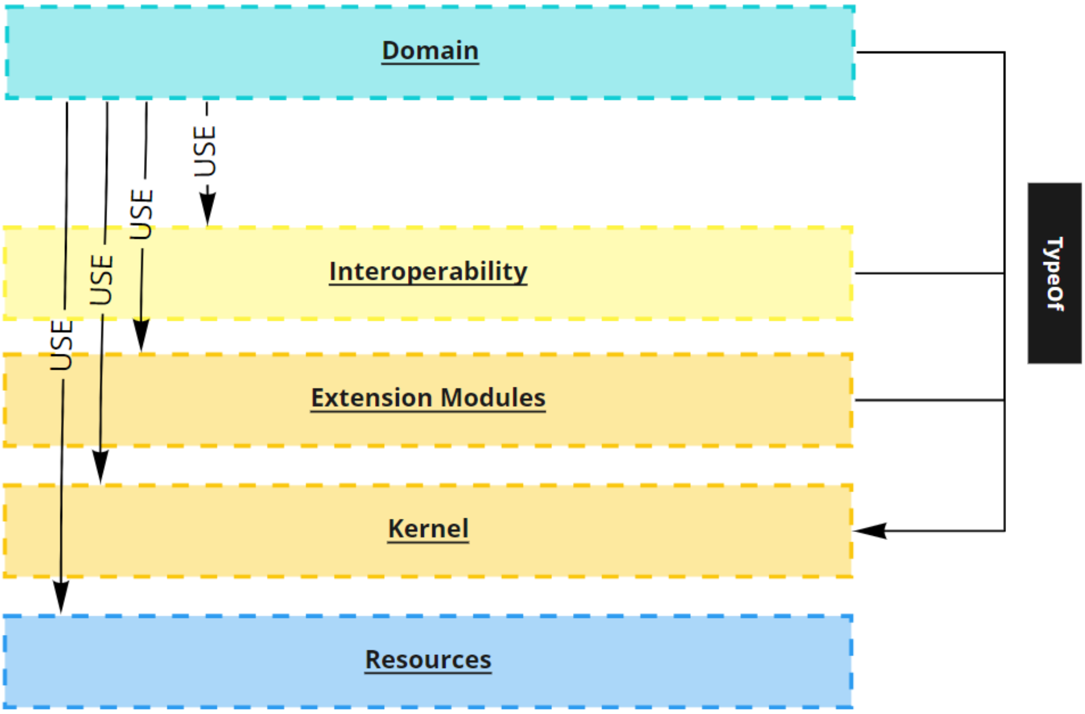


Figure 2.2.2.3a - Layers of the IFC data model

The main principles and function of the conceptual layers can be explained as follows:

Resource layer

The lowest level is the resource layer. This holds schemas to describe basic features of the more high level objects and can be referenced by all the other layers. Objects found in the resource layer can be considered general purpose which do not rely or inherit from anything else. This can be, for example, utility objects such as object history, identification and general purpose tables, or other things like measurement objects. Here you will also find the representation resource which forms the base for most geometric representation of the concrete entities in the higher levels.

Core layer

The next level, the core layer, provides the basic setups for object definition for the domain and interoperability layer, by defining the core abstract concepts to which the domains must adhere. This, in turn, consists of two sublayers, namely the *kernel* and *extension* modules.

The *kernel* is the foundation for all basic concepts within any IFC release, and defines model structure and decomposition. It should be highlighted that this is a type of template model with definitions for objects and relationships that are not AEC specific, despite this being their essential use-case. Anything in the kernel can reference definitions from the resource layer, but not the Core extension layer, as that is considered a higher layer than the kernel.

The second module, the core *extensions*, provide the specialization of the kernel in the form of a refinement of the largely abstract definitions into constructs pertaining to the AEC industry, which are further referenced by higher levels.

Interoperability layer

The interoperability layer provides the interface for domain models, thus providing an exchange mechanism for enabling interoperability across the domain objects.

Here objects and concepts shared across the disciplines are specified. These then form a basis for exchange of data across said disciplines. Further, this layer also contains model adapters to other, external non-IFC application models.

Domain layer

As the highest of all the layers is the domain layer, this contains models and objects for more specific contexts within the AEC industry, capturing domain objects for HVAC, structural design or architectural processes and software. These are concrete object entities used by the different disciplines in isolation. For exchange between disciplines, common structures defined in the interoperability layer must be used as an exchange mechanism.

The Structure of IFC Objects

As mentioned earlier, the IFC model explicitly defines over 1000 explicit objects. All the objects are defined using a certain convention, where IFC uses the EXPRESS as a data definition language which is based on an entity-relationship logic. The EXPRESS language definition is closely linked to the STEP model [STEP Format], containing the concepts *relationships*, *attributes*, *constraints* and *inheritance*. The final information models have the quality of being both human and machine readable. [Ekholm 2005]

The concepts of the EXPRESS language are adapted by the IFC model in a direct way, with the three basic types being *ifcObject, ifcPropertyDefinition* and *ifcRelationship* defined within the kernel layer mentioned above. All of these share a common base inheritance back to the *ifcRoot* definition, which is the most generic entity in the IFC model. A basic diagram of this is shown in figure. [Ekholm 2005, IFC Documentation]

The *ifcRoot* entity provides the basic properties name, ID, description and history, which are used across all entities in the model. [documentation]

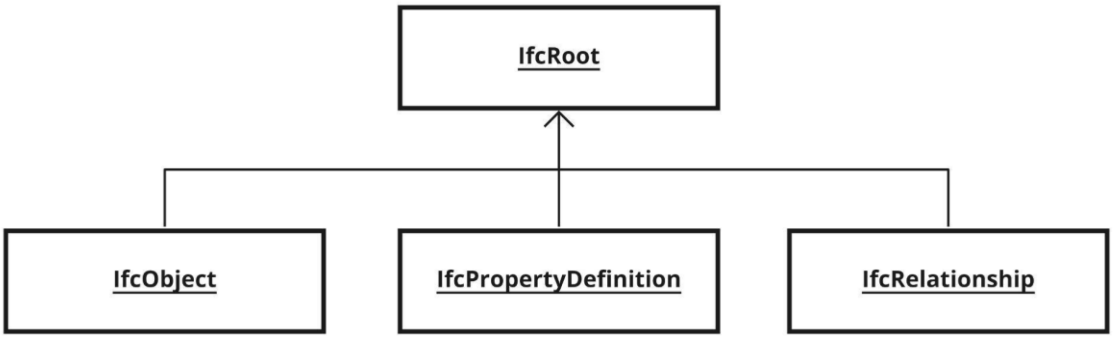


Figure 2.2.2.3b - IFC model basic object types

IfcObjectDefinition

The base *ifcObjectDefinition* entity is described in the buildingSMART documentation as:

"...*is the generalization of any semantically treated thing or process, either being a type or an occurrence. Objects are independent pieces of information that might contain or reference other pieces of information."*

This is in-turn a supertype of *IfcObject* and *ifcTypeObject*, where

"... *examples of IfcObject include physically tangible items, such as wall, beam or covering, physically existing items, such as spaces, or conceptual items, such as grids or virtual boundaries. It also stands for processes, such as work tasks, for controls, such as cost items, for actors, such as persons involved in the design process, etc.*

The object type (IfcTypeObject) defines the specific information about a type. It refers to the specific level of the well recognized generic - specific - occurrence modeling paradigm.

[official documentation]

IfcRelationship

"IfcRelationship" has a double role in that it both represents relations between members of "IfcObject", and relations between model classes.

Definition from buildingSMART:

"The abstract generalization of all objectified relationships in IFC. Objectified relationships are the preferred way to handle relationships among objects. This allows to keep relationship specific properties directly at the relationship and opens the possibility to later handle relationship specific behavior."

IfcPropertyDefinition

"*IfcPropertyDefinition*" represents different properties of domain objects.

Definition from buildingSMART: The IfcPropertyDefinition defines the generalization of all characteristics (i.e. a grouping of individual properties) that may be assigned to objects. Currently, subtypes of IfcPropertyDefinition include property set definitions, and property sets.

Example: ifcBeam

To give a more concrete example of an *IfcObject*, the following diagram shows the inheritance chain of an *IfcBeam*, one of the more commonly occuring architectural elements. All the objects follow a certain naming convention using the prefix *ifc* followed by the name in camel case. It's grouped together within the *IfcBuildingElement* category among other familiar objects such as *walls*, *windows* and *slabs*.

An example of this type of an object definition can be seen in the following image,which shows the EXPRESS definition of an *IfcBeam*.

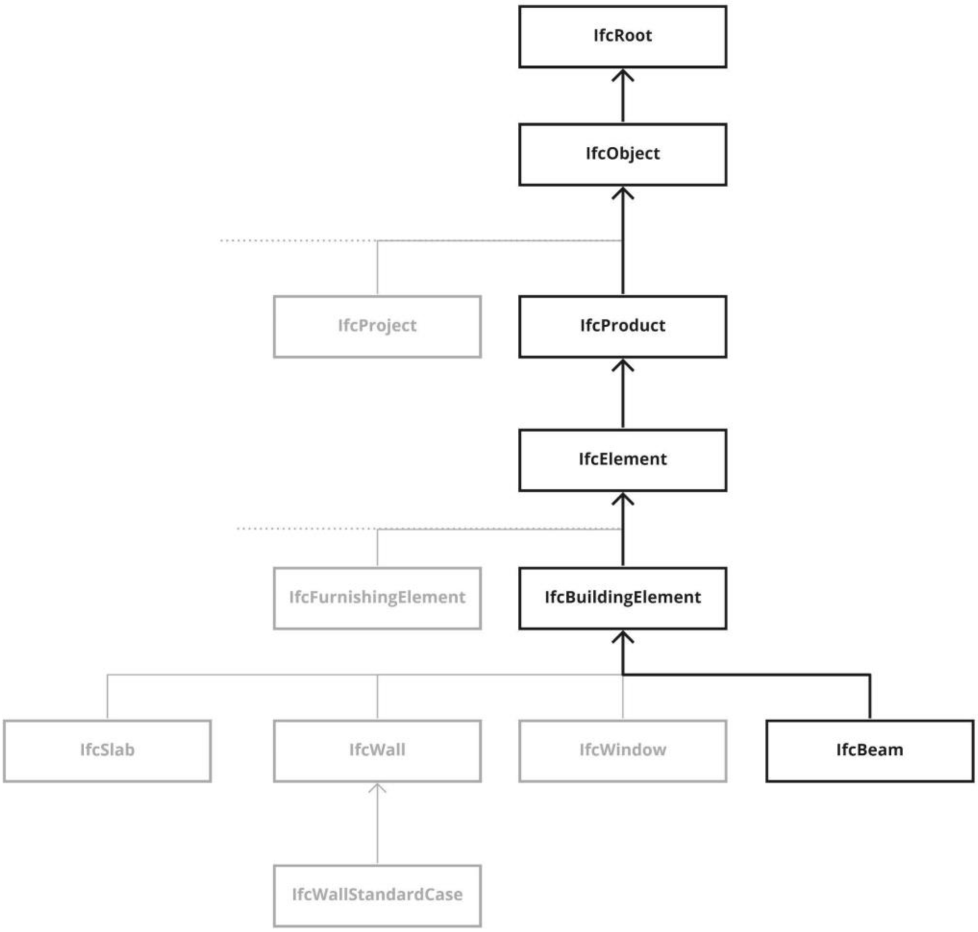
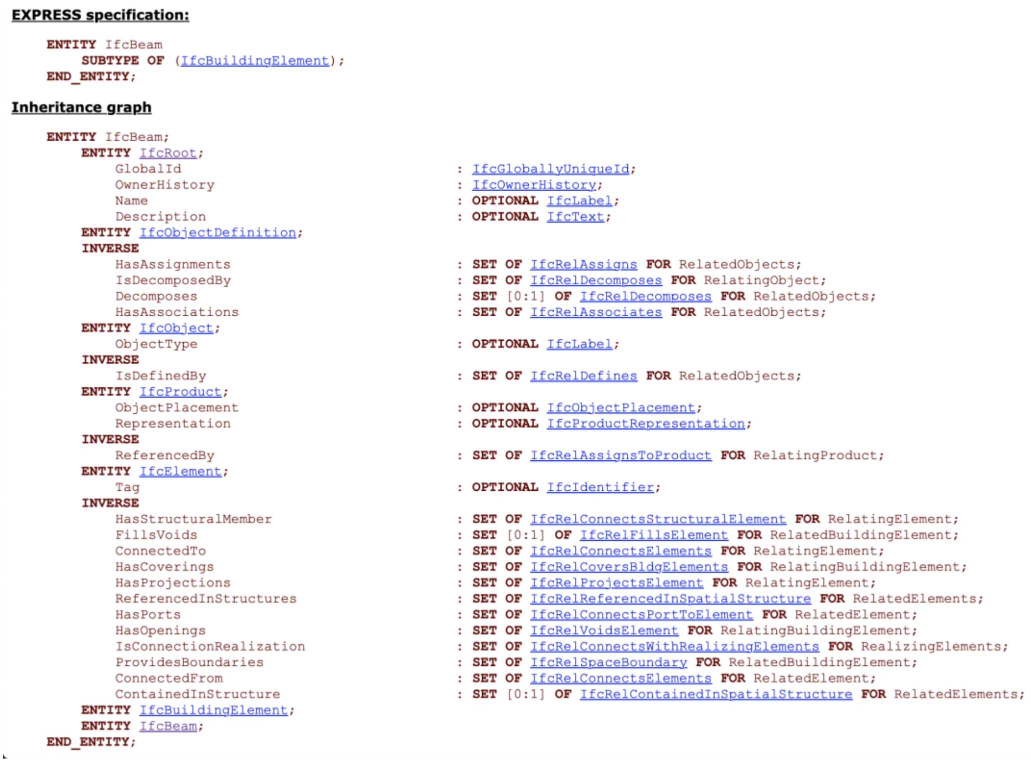


Figure 2.2.2.3c - Example of inheritance structure for some building elements



EXPRESS specification:

```
    ENTITY IfcBeam
        SUBTYPE OF (IfcBuildingElement);
    END_ENTITY;
```

Inheritance graph

```
    ENTITY IfcBeam;
        ENTITY IfcRoot;
            GlobalId                                : IfcGloballyUniqueId;
            OwnerHistory                            : IfcOwnerHistory;
            Name                                    : OPTIONAL IfcLabel;
            Description                             : OPTIONAL IfcText;
        ENTITY IfcObjectDefinition;
        INVERSE
            HasAssignments                          : SET OF IfcRelAssigns FOR RelatedObjects;
            IsDecomposedBy                          : SET OF IfcRelDecomposes FOR RelatingObject;
            Decomposes                              : SET [0:1] OF IfcRelDecomposes FOR RelatedObjects;
            HasAssociations                         : SET OF IfcRelAssociates FOR RelatedObjects;
        ENTITY IfcObject;
            ObjectType                              : OPTIONAL IfcLabel;
        INVERSE
            IsDefinedBy                             : SET OF IfcRelDefines FOR RelatedObjects;
        ENTITY IfcProduct;
            ObjectPlacement                         : OPTIONAL IfcObjectPlacement;
            Representation                          : OPTIONAL IfcProductRepresentation;
        INVERSE
            ReferencedBy                            : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
        ENTITY IfcElement;
            Tag                                     : OPTIONAL IfcIdentifier;
        INVERSE
            HasStructuralMember                     : SET OF IfcRelConnectsStructuralElement FOR RelatingElement;
            FillsVoids                              : SET [0:1] OF IfcRelFillsElement FOR RelatedBuildingElement;
            ConnectedTo                             : SET OF IfcRelConnectsElements FOR RelatingElement;
            HasCoverings                            : SET OF IfcRelCoversBldgElements FOR RelatingBuildingElement;
            HasProjections                          : SET OF IfcRelProjectsElement FOR RelatingElement;
            ReferencedInStructures                  : SET OF IfcRelReferencedInSpatialStructure FOR RelatedElements;
            HasPorts                                : SET OF IfcRelConnectsPortToElement FOR RelatedElement;
            HasOpenings                             : SET OF IfcRelVoidsElement FOR RelatingBuildingElement;
            IsConnectionRealization                 : SET OF IfcRelConnectsWithRealizingElements FOR RealizingElements;
            ProvidesBoundaries                      : SET OF IfcRelSpaceBoundary FOR RelatedBuildingElement;
            ConnectedFrom                           : SET OF IfcRelConnectsElements FOR RelatedElement;
            ContainedInStructure                    : SET [0:1] OF IfcRelContainedInSpatialStructure FOR RelatedElements;
        ENTITY IfcBuildingElement;
        ENTITY IfcBeam;
    END_ENTITY;
```

Figure 2.2.2.3d - Example EXPRESS specification for an IfcBeam

<u>The IFC domains</u>

Shared domains and specific domains. The shared domains are found in the *interoperability* layer and provide some basic functionality shared between the more specific domains. Two significant ones are

- *ifcSharedBldgElements*
- *ifcSharedBldgServiceElements*

ifcSharedBldgElements

Most of the shared and generic building objects are defined in the *kernel* layer, more specifically in the *ifcSharedBldgElements.* This data model contains most generic built elements and forms a solid base for the basic exchanges across the various domains.

The IFC documentation describes this model as:

“The shared building elements (IfcSharedBldgElements) define the subtypes of IfcBuildingElement, which is defined in the IfcProductExtension. Those subtypes are the major elements, which constitutes the architectural design of the building structure.

The elements (e.g. wall, beam, column, slab, roof, stair, ramp, window, door and covering) are the main components of the raw building (or carcass) which is central for the exchange of project data.“

IfcSharedBldgServiceElements

> *"The IfcSharedBldgServiceElements schema in the interoperability layer defines basic concepts required for interoperability primarily between Building Service domain extensions, notably IfcHvacDomain, IfcPlumbingFireProtectionDomain, IfcElectricalDomain and IfcBuildingControlsDomain. This schema includes concepts such as basic type and occurrence definitions for flow and distribution systems and fundamental properties commonly used in building service scenarios (such as fluid-flow properties, electrical properties, space thermal properties, etc.)"*

[Official Documentation]

Within the more specialised, and isolated domain models, one can find the following set of data models: [IFC Documentation].

- IfcArchitectureDomain
- IfcBuildingControlsDomain
- IfcConstructionMgmDomain
- IfcElectricalDomain
- IfcFacilitiesDomain
- IfcHvacDomain
- IfcPlumbingFireProtectionDomain
- IfcStructuralElementsDomain
- IfcStructuralAnalysisDomain

A set of these domains can be considered more relevant for the Prism platform given the current roster of disciplines involved. These will be elaborated in slightly more detail here:

IfcArchitectureDomain

Most architectural elements are already defined in the *IFCSHAREDBLDGELEMENTS* model mentioned above. The specific architectural domain contains only highly specific entities that have not been genericized enough to be present among the shared elements. This is in the end a fairly specific subset containing objects such as specific permeable covering properties for window or door openings, or for example space programs for design briefs.

IfcStructuralElementsDomain

The StructuralElements domain contains the necessary elements for parts that are of a structural nature. This contains specific objects such as footings, piles or reinforcement. Some of these are top level objects in their own right, whereas others, like reinforcement, are contained within other objects. [Official Documentation]

The schema IfcStructuralElementsDomain provides the ability to represent different kinds of building elements and building element parts which in general are of structural nature.

IfcStructuralAnalysisDomain

The StructuralElements domain does not contain analysis objects, but physical objects, and for the description of structural analysis objects the StructuralAnalysis domain is defined. This is used to tie structural element definitions to the shared building elements, with the overall aim to expose analysis results to the other domains. [Official Documentation]

Effectively, the model handles planar and spatial structural analysis models for the interface with structural analysis software, containing entities such as *ifcStructuralAnalysisModel, ifcStructuralConnection, ifcStructuralMember* and *ifcStructuralLoadgroup.*

IfcHvacDomain

> *"The IfcHvacDomain schema in the domain layer defines basic object concepts required for interoperability within the heating, ventilating and air conditioning (HVAC) domain. It extends concepts defined in the IfcSharedBldgServiceElements schema."*

[Official Documentation]

In general, the scope of the Hvac domain model can be said to include types like segments, fittings and connections like duct and piping used for various systems like water and air flow, along with relevant equipment such as boilers, fans and additional control devices like air vents. The scope is said to exclude industrial speciality equipment, provisions for dealing with chemical hazards, and control systems.

2.2.2.4 Schema vs. Implementation

As previously mentioned, IFC is not a file format but a standard specification, however, for the most part, IFC is designed for a file-based transaction system. [Laakso et al 2012] At the moment, the original STEP format is the most used across the industry, but there exists a wide range of ways to represent data of the IFC format.

An extensive list can be found at:

- https://technical.buildingsmart.org/standards/ifc/ifc-formats/

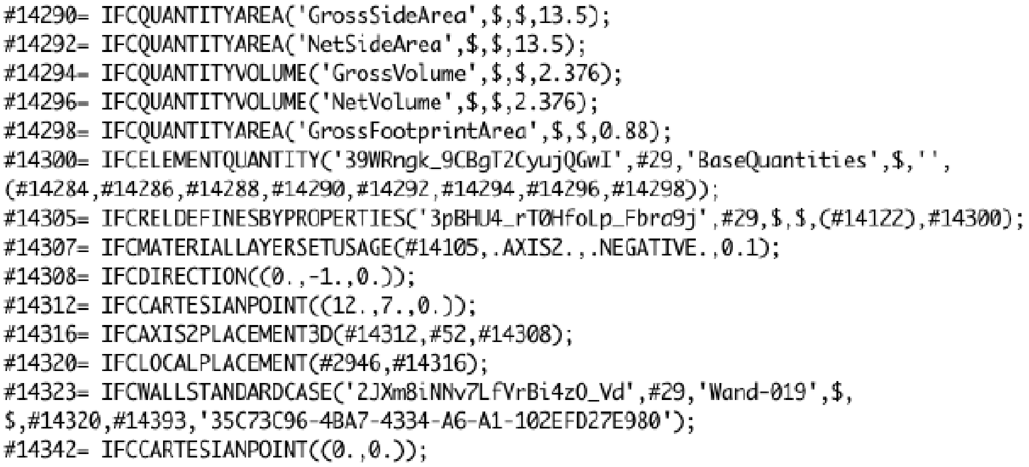
For the purpose of this study, three of those formats will be put under some more scrutiny. Along with the basic STEP format, those are:

- *XML format*
- *Json format*

The reason for those two specific formats being of particular interest lies in the fact that those are generic formats for data exchange which, for the purpose of the PrismArch environment, would open up possibilities of parsing ifc data using off-the-shelf or standardized ways. This would be difficult to do using a bespoke format as the STEP file. Below follows a brief comparison and examination of all formats, both the ifc and general xml and json.

STEP Format

The STEP format, being the earliest form of IFC data representation, is the most widely supported and encountered one in the industry today. A snippet of the way the file is structured can be seen below:



```
#14290= IFCQUANTITYAREA('GrossSideArea',$,$,13.5);
#14292= IFCQUANTITYAREA('NetSideArea',$,$,13.5);
#14294= IFCQUANTITYVOLUME('GrossVolume',$,$,2.376);
#14296= IFCQUANTITYVOLUME('NetVolume',$,$,2.376);
#14298= IFCQUANTITYAREA('GrossFootprintArea',$,$,0.88);
#14300= IFCELEMENTQUANTITY('39WRngk_9CBgT2CyujQGwI',#29,'BaseQuantities',$,'',
(#14284,#14286,#14288,#14290,#14292,#14294,#14296,#14298));
#14305= IFCRELDEFINESBYPROPERTIES('3pBHU4_rT0HfoLp_Fbra9j',#29,$,$,(#14122),#14300);
#14307= IFCMATERIALLAYERSETUSAGE(#14105,.AXIS2.,.NEGATIVE.,0.1);
#14308= IFCDIRECTION((0.,-1.,0.));
#14312= IFCCARTESIANPOINT((12.,7.,0.));
#14316= IFCAXIS2PLACEMENT3D(#14312,#52,#14308);
#14320= IFCLOCALPLACEMENT(#2946,#14316);
#14323= IFCWALLSTANDARDCASE('2JXm8iNNv7LfVrBi4z0_Vd',#29,'Wand-019',$,
$,#14320,#14393,'35C73C96-4BA7-4334-A6-A1-102EFD27E980');
#14342= IFCCARTESIANPOINT((0.,0.));
```

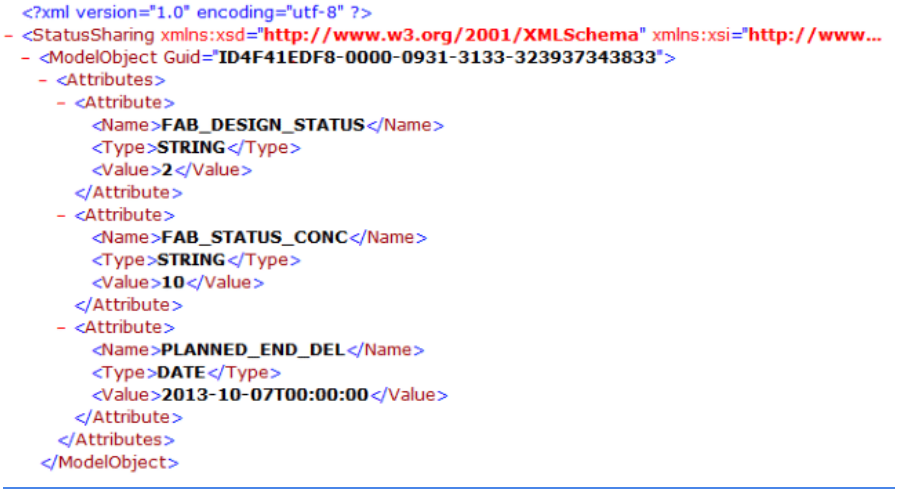
Figure 2.2.2.4a - Example of STEP format text file.

It is based on the ISO standard for clear text representation EXPRESS data models (ISO 103003-21) and each line documents a single object record with a global label. Although the format is technically human readable, cross-referencing labels of 100,000 objects is practically an impossible and nonsensical task for a human, hence the file is hard to make any sense of without a IFC STEP file parser of some sort. It has no hierarchies or indentation to help structure the data in a way that is easier for a human to make sense of, even if just for a local element.

A benefit of the STEP format is its fairly small size, making it efficient to store and send when handling large models, which is a common feature in AEC processes as buildings tend to be fairly large. However, no general off the shelf tools can be used to parse a IFC STEP file and IFC likewise provides nothing to help, so one has to rely on the work of third-party APIs, for example XBIM [XBIM homepage]. Likewise, a file can not be queried or partially read, but the entire thing has to be imported in order to get information out of it.

XML Format

An alternative way of storing data in the IFC schema is through the IFC-XML format. This format can be generated directly from the sending application per a structure called STEP-XML. An example of this format is shown below.



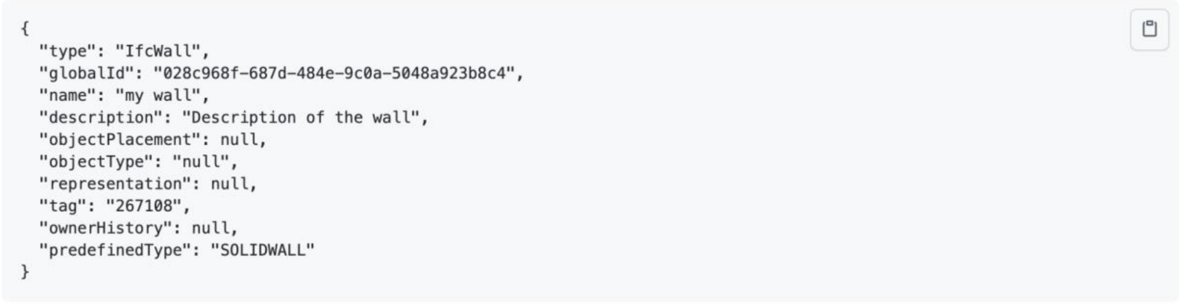
```
<?xml version="1.0" encoding="utf-8" ?>
- <StatusSharing xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www...
  - <ModelObject Guid="ID4F41EDF8-0000-0931-3133-323937343833">
    - <Attributes>
      - <Attribute>
          <Name>FAB_DESIGN_STATUS</Name>
          <Type>STRING</Type>
          <Value>2</Value>
        </Attribute>
      - <Attribute>
          <Name>FAB_STATUS_CONC</Name>
          <Type>STRING</Type>
          <Value>10</Value>
        </Attribute>
      - <Attribute>
          <Name>PLANNED_END_DEL</Name>
          <Type>DATE</Type>
          <Value>2013-10-07T00:00:00</Value>
        </Attribute>
      </Attributes>
    </ModelObject>
```

Figure 2.2.2.4b - Example of XML format text file.

This was developed for the exchange of partial building models and for direct integration with pre-existing and straightforward XML parsing tools. A drawback of the format is that the resulting files are 300%-400% larger for the same data, making it significantly less efficient compared to the STEP format. Because of the size implications, this format is less common in practice [ref].

IFC JSON

JSON is the de facto way of exchanging data within web-based mediums and is becoming increasingly popular within the AEC industry, as we will see later on in the chapter about Speckle.



```
{
  "type": "IfcWall",
  "globalId": "028c968f-687d-484e-9c0a-5048a923b8c4",
  "name": "my wall",
  "description": "Description of the wall",
  "objectPlacement": null,
  "objectType": "null",
  "representation": null,
  "tag": "267108",
  "ownerHistory": null,
  "predefinedType": "SOLIDWALL"
}
```

Figure 2.2.2.4c - Example of JSON format text file.

Software Implementation

Beyond this, IFC also does not provide any connections to existing software, meaning that the responsibility for setting up exchanges between proprietary software packages and data represented in the IFC format is solely down to the software package. IFC does provide an implementation agreement [IFC Implementation Agreement] which establishes the general way in which data should be represented and interpreted, but there is no mode of enforcing or controlling this. This can cause ambiguity and uncertainty in exactly how a certain object is represented in the IFC export, as that depends on the reasoning of any software themselves. Reports say that this may cause difficulties in the so-called “round tripping” of IFC models as different vendors might use the vast library of IFC structures in differing ways, essentially almost nullifying the whole premise of the effort. [Berlo et al 2012, Amoor 1997]

1.2.2.5 IFC today

Today, when it comes to BIM interoperability, IFC is the most widely recognised format. However, there are reports [Gielingh 2008, Laakso et al 2012] that the format still experiences a fairly poor industry uptake, despite having been around for a while. Gielingh believes there are three driving reasons responsible for this: business motivation, legal aspects and industry readiness.

However, a large portion of this can most likely be attributed to inconsistent and unreliability in exports. He further goes on to question the entire standardized model, stating that it will always be limited by these inconsistencies, stemming both from poor implementation but also the fundamental domain variations between software and disciplines.

A format initially conceived in the 1990s may carry some features and inherent logic that are not up-to-date with more current advances in technology. BuildingSMART themselves state that:

> *"The STEP standard is very efficient and has a rich set of advanced modelling techniques to create efficient file-based data exchange.* ***The definition of IFC has quite some STEP specific modelling techniques, and over time gathered some inconsistencies.*** *To work towards a future-proof IFC these issues need to be resolved..."*

From BuildingSMART GitHub

They further go on to list some issues to be tackled in order to future-proof the IFC standard. Among those things, they state that the geometry kernel is too big, and that it contains a large set of entities which, although they provide efficient storage, only have a few use cases. In relation to this they further state that many advanced modeling structures provide conflicting and ambiguous ways to build implementations. In total all these issues contribute to a lack of interoperability that has resulted in inconsistent implementations, as implementing the complete IFC models consistently across multiple isolated vendors is very difficult.

Further, they state that many advanced data modelling techniques, like linked lists, have no comparable equivalent in languages like UML (that aid in the conversion and translation of schemas between languages). Thus, trying to reconstruct IFC data types in languages like JSON and XML demands bespoke solutions, incompatible with their general libraries.

Finally concluding that:

> *"No one standard can/ should rule all use-cases.* ***IFC should have a clear focus and scope.*** *Although use-cases can be supported it does not mean they should be when there are alternative standards available.* ***Modularity helps in pairings with other standards and***

__reduces implementation effort when only relevant modules need to be implemented.__ This means that extensive relationships between distinct modules need to be minimized."

From BuildingSMART GitHub

2.2.2.6 The BCF Format

The BIM Collaboration Format (BCF) is a file format developed to enhance communication in construction projects. Development of the BCF started in 2009 by Solibri and Tekla, two members of the buildingSMART International Implementation Support Group (ISG). The desire to enhance and improve open communication technology between IFC based workflows led the ISG members to develop this format.

BCF mainly focuses on communication regarding model based issues. These issues are highlighted within a project file. BCF leverages the IFC files produced by appointed parties and produces a report which contains the issues to be discussed.

The data is formatted and transferred in XML. The data contains a PNG image and IFC coordinates for easier location of the issue. A description can also be added to give better insight of what the issue is, probable causes and proposed solutions.

BCF is an open file format similar to IFC and data dictionary (bsDD) which means that anyone may use it. The files can be exchanged via a web server or simple file exchange.

Since BCF is intended for use within a construction project we need to better understand the use of this file format in each phase of the construction. The four phases of construction and the use of BCF can be described as:

Design Phase

- Documenting quality assurance / quality checking (QA/QC) of BIM items.
- Identifying design coordination (aka clash detection) issues between BIM domains.
- Annotating design options, object substitutions, and material selections.

Procurement Phase

- Bidding coordination items and clarifications.
- Cost and supplier information for objects, assemblies, and/or systems.

Construction Phase

- Quality assurance / quality checking (QA/QC) records of installations vs. BIMs.
- Tracking availability of items/materials and coordinating substitutions.
- Collection of last-minute information for handover to owner/operator as part of the COBie deliverables.

Operations Phase

- Notations to handover models as changes are made to the facility and its many elements during occupation.
- Owners notes about needed upgrades.

As we mentioned previously, the BCF file format includes the PNG image, the IFC coordinates and the additional notes. The software needed to open the BCF file is called a

BCF Manager in general. BCF Managers may be open source or under license schemes. The most common open source BCF Manager is the BCFier for Windows. Proprietary software developers include BCF Managers either as stand-alone applications or as add-ins to a wider set of tools. The evolution of technologies, cloud solutions and the increase of the speed of internet connections led software developers to add cloud solutions to their development lists. Nowadays, BCF Managers exist in the cloud offering a wide range of additional services and enhanced collaboration between the different parties and stakeholders within a construction project. The issues can be handled locally and then uploaded to the cloud or directly developed and updated within the cloud solution.

The benefits of using the BCF file include :

- Better issue management
- Tracking history of issues
- One centralized environment for decentralized information
- Not losing comments and design solutions in the ocean of emails
- Saves time
- Increased level of collaboration
- Easy to understand and handle
- Local or cloud options
- Freeware or Payware solutions

The need for enhanced collaboration leads to solutions such as BCF. The data rich reports improve decision making, the way the format has been developed keeps track of all changes so at any given point we can revert back and inspect who make which decision and for what reason, and finally the open standard will enhance the multiple software intended to be used within PrismArch to have a centralised way of communicating the issues.

■ 2.2.3 Speckle

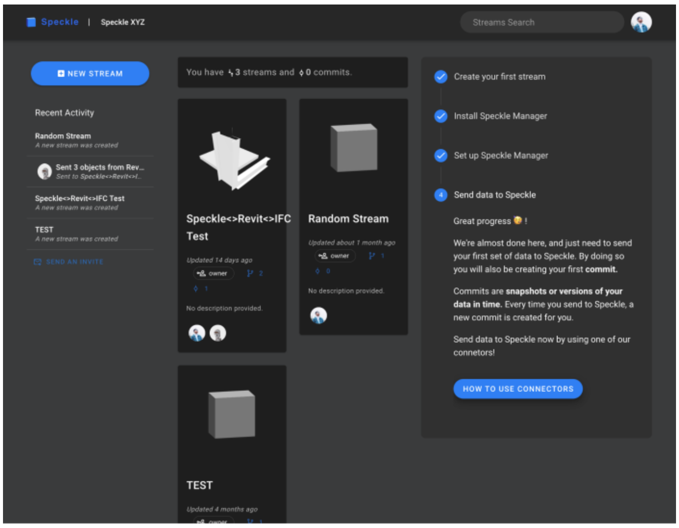
2.2.3.1 Background

Speckle is a more recent development within the AEC interoperability field. In its scope and approximate solution, it is superficially similar to IFC, and what that system was attempting to achieve twenty years earlier. However, Speckle takes a slightly different approach.
As they state:

> *"Bye bye legacy software and proprietary formats, you've been around for too long. There's a new way of working: it's based on open, accessible and secure data.*
> *We believe that open source, and in particular open source data infrastructure, is the only solution to this industry's stagnant, inefficient and fragmented modus operandi."*
>
> [Speckle]

So while Speckle is a set of data models, and a series of implementations across a range of software - both aspects applicable to IFC - it is *also* a web-based data storage platform and, beyond that, it is a philosophy for interoperability [Stefanescu 2020]. It emerged from a PhD project conducted at the Bartlett School of Architecture by Dimitrie Stefanescu, titled "*Alternate Means of Digital Design Communication",* and undertaken between 2015 and

2019. The thesis discusses digital data as a medium in the design process and covers topics like data classification, representation, and transactions from a collaborative and multidisciplinary perspective.
The outcome of this research was a start-up company called Speckle, and also a suite of tools and software connectors developed by them, given the same name. The 'mission statement' of the Speckle company is to build the first version control system for the AEC disciplines. Just as version control (in the guise of GitHub and similar projects) revolutionised certain aspects of software development, it is hoped that Speckle will similarly advance coordination and collaboration in construction.



You have 3 streams and 0 commits.

No description provided.

No description provided.

Great progress !

We're almost done here, and just need to send your first set of data to Speckle. By doing so you will also be creating your first **commit.**

Commits are **snapshots or versions of your data in time.** Every time you send to Speckle, a new commit is created for you.

Send data to Speckle now by using one of our connetors!

Figure 2.2.3.1a - Snippet of the Speckle web interface at *speckle.xyz*

As stated earlier, the Speckle tools contain a web-interface. However, the following review will not focus on this web platform or the nature of web-based collaboration, but will instead discuss the ontological concepts behind Speckle, and their underlying logic and philosophy.

2.2.3.2 The Speckle Interoperability Approach

Speckle takes some fundamentally different positions and assumptions about the nature of interoperability, data exchange and ontologies, compared to the IFC format. As previously discussed, IFC aims to explicitly define a structure for every single object one could possibly need as part of the AEC design process. Speckle, on the other hand, posits that such a thing cannot be done, and any such attempt will always be incomplete given the unpredictable and varied range of data associated with the wide range of objects within the industry [Stefanescu 2020].

Therefore, Speckle instead favours a more implicit and bottom-up driven approach. Based on a minimal set of objects, it allows users to customize them and assemble completely custom data structures tailored to their own needs.

In some scenarios the IFC model might fail, because it was developed by a centralized organization who did not include certain properties or sets of data that are now required, and the IFC model cannot flexibly respond. In comparison, Speckle allows users to append those properties to any object in an ad-hoc fashion. The rigid ontological formulation of IFC is here replaced by one that can change and morph given the current needs and, more importantly, the context.

Composable Objects

A composable object model, as proposed by Stefanescu, presents objects that are not explicit, but instead can be assembled or composed from a set of additional objects or data sets.

A set of low level objects can be composed to assemble higher level concepts, and likewise most high level concepts can eventually be reduced to a fundamental geometric or digital entity - such as a point, line, integer or boolean. [Stefanescu 2020]

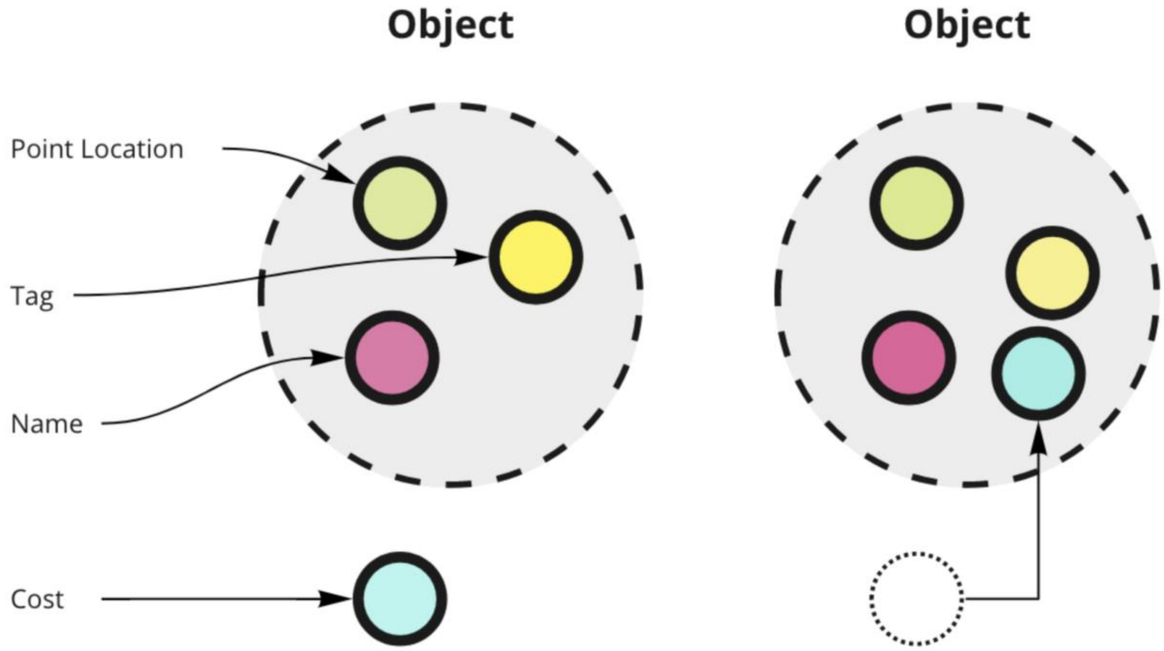


Figure 2.2.3.2a - Example of a composable object model.

Further, this approach extends to a fully flexible definition where objects can be made up of various sets of components, thus providing a mechanism for each discipline to project their specific meaning onto an object. What these compositions are can be formulated and revised as a project goes on, as new information and considerations naturally emerge throughout the design process. The objects are then flexible enough to adapt new meanings or hold additional information that suddenly becomes essential. This approach is therefore entirely consistent with the type of ‘wicked problem’ that AEC interoperability represents [Miller, 2016 & Rittel and Webber, 1973].

Ontological Revision

Another feature of these composable and extensible objects is that they can adapt and adjust based on the context. These objects are not required to hold all the data they might need at any point, or even placeholders for this data. Instead, objects naturally change what properties and features they contain as they move through the domain variations of different software applications, as discussed in chapter 1.2.1.

As D.Stefanescu writes :

> *"The actors involved in the design process may not share the same internal representations of a specific building element. For example, an architect may represent a beam in one way, whereas the structural engineer in a different way."*
>
> [Stefanescu 2020]

This brings us to the topic of *ontological revision,* as formalised by D.Stefanescu [Stefanescu 2020].

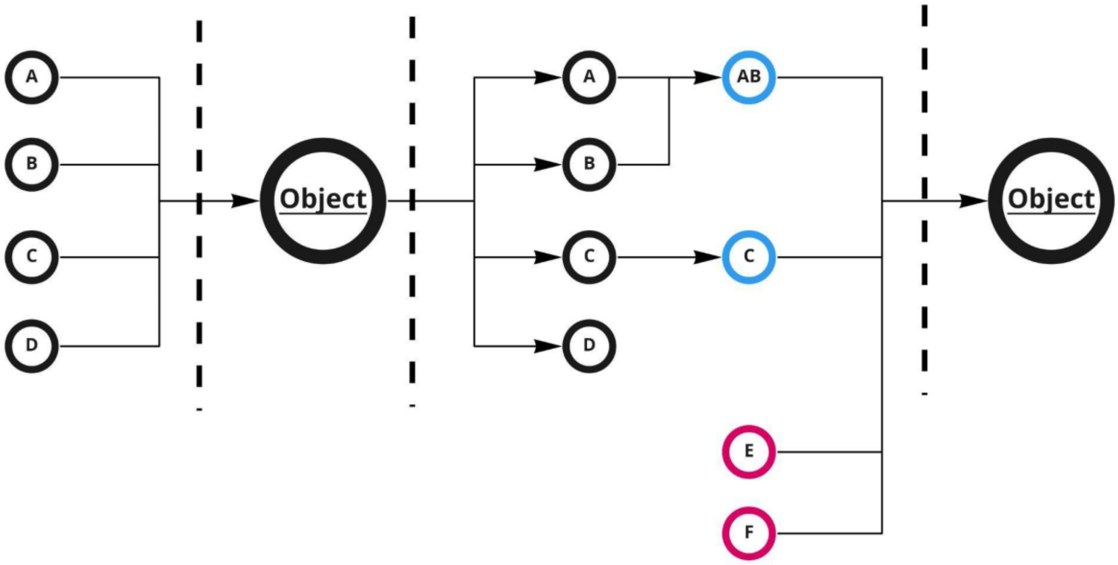


Figure 2.2.3.2b - Ontological revision, as proposed by Speckle.

Through this process an object can naturally emerge in a specific, specialist context, and within that context perform a particular function. As it is shared and moves into a different, more generic context, some of that information may no longer be relevant, thus naturally some of its data will and should be lost. Conversely, an object entering a context for which it was not explicitly designed can be assigned additional properties, making it suitable for use in new and unexpected contexts.

<u>Speckle Data Models</u>

As stated above, this flexible object structure clearly has many benefits. However, automating the process of translating and exchanging these flexible entities can be more complex than in the case of rigid data types, as there is less guidance for how to use the unexpected data that might be contained within any given object. Hence, some type of structure is necessary to project onto the data for software to be able to interpret it. This predicted structure should not be in contrast to the extensible and composable objects paradigm - instead, these schemas are used to enforce a certain set of sub components or properties to impose a certain meaning to it. This also does not negate the possibility to further extend a fixed schema with custom properties, although, these properties may have to rely on some human interpretation at the receiving end of an exchange.

Speckle acknowledges this with the concept of object models, or *Speckle Kits*. In contrast to IFC, however, Speckle does not set out to define an exhaustive object model, and instead adapts a more modular approach. The core Speckle object model, or *ObjectsKit*, only defines a handful of fundamental and very general objects, such as core geometric structures and a set of built elements. Speckle provides a minimum shared common ground, within which companies or individuals can amend, extend or replace. Users therefore have the option to either adapt an object if it is entirely suitable, or append additional user data to it if it requires only minor revision, or, as a last resort, create an entirely new data model of their own design - but which can still seamlessly transfer through the Speckle medium, thus extending the ecosystem.

A Speckle object model is composed of two parts, with the first part being the schemas themselves, and the second being how these schemas are interpreted in the particular sending and receiving contexts of different software applications. Together, these two components form the basis for Speckle Transactions.

This approach allows Speckle to move away from AEC entities that are hard-coded and centrally-defined schemas, into flexible schemas that can be adapted and expanded by the users themselves. And this in turn reinforces the *ontological revision* concept explained earlier: when an object moves from a more specialised software package into another, more generic one, there need not be compatibility issues - the object can instead simply shed the custom data pertaining to the specialist context and carry on as, for example, a simple geometric object. This concept is very powerful as it allows different disciplines to directly communicate across disparate software platforms, while still having the ability to retain and communicate their own unique properties within their own internal environments and exchanges.

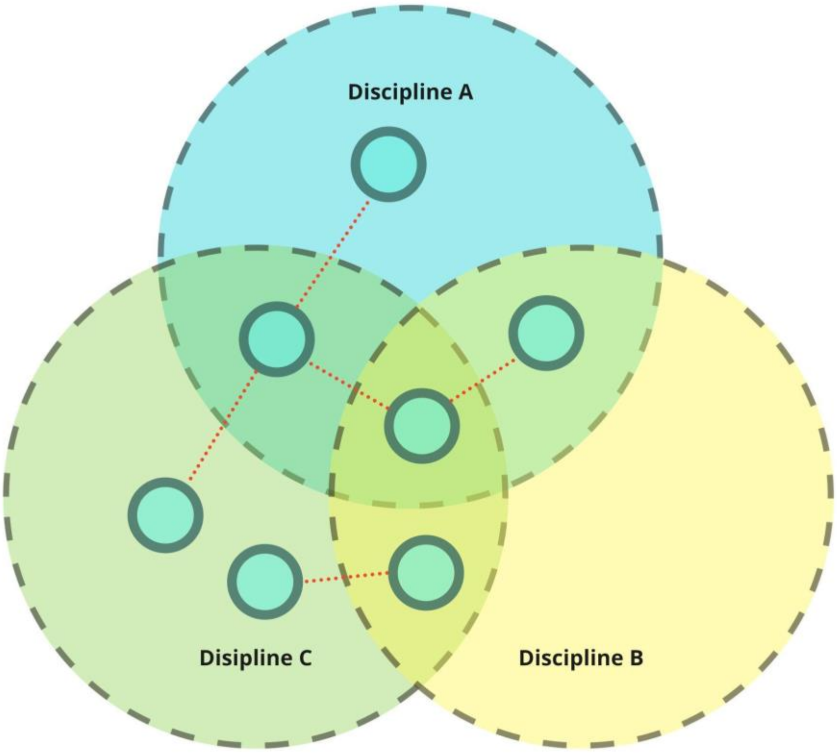


Figure 2.2.3.2c - Flexibility of schemas across disciplines.

Using this malleable ontology, a network of more- and less-specific schemas can be developed by different disciplines during the lifetime of an AEC project. This can occur with the critical assurance that overlapping data will be retained between parties, but without the burden of them all needing to conform to overly-complex overarching data types. A representation of this type of *networked object models* is shown in the diagram above.

Whereas IFC forces a global agreement on *all* aspects o*f all* object structures, here instead companies and disciplines share only a minimal common denominator, which each is free to expand upon and grow according to their own needs.

2.2.3.3 Implementation

In contrast to IFC, Speckle is a complete platform with objects and conversion defined within the .NET framework and API for the C# programming language.

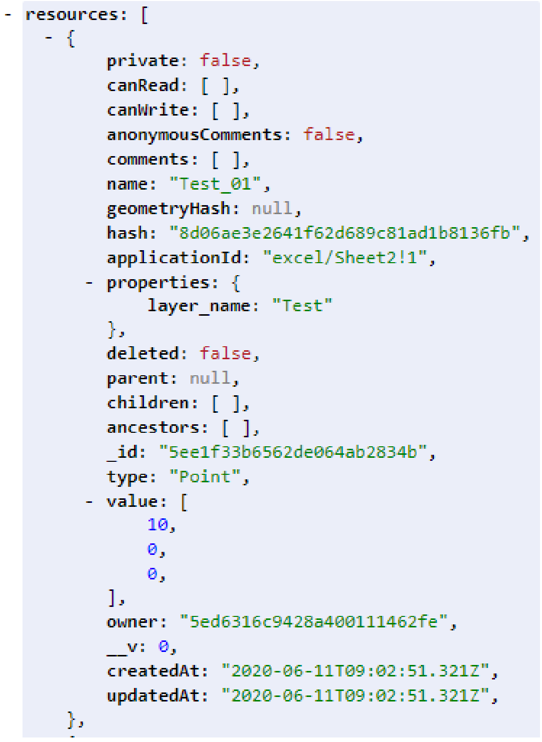
As Speckle is an open source initiative, all of the code is available at their GitHub account [Speckle GitHub].

JSON

Speckle, as a web-based platform, makes use of the JSON format for exchange of data. JSON nicely complements the composable and nestable data approach that Speckle proposes.

JSON is an unstructured format based on nesting keys and values. Standard tools can then be used to expand and query any Speckle object, no matter what structure it has.

An example of a serialized Speckle object with nested values can be seen here:



```
- resources: [
  - {
        private: false,
        canRead: [ ],
        canWrite: [ ],
        anonymousComments: false,
        comments: [ ],
        name: "Test_01",
        geometryHash: null,
        hash: "8d06ae3e2641f62d689c81ad1b8136fb",
        applicationId: "excel/Sheet2!1",
      - properties: {
            layer_name: "Test"
        },
        deleted: false,
        parent: null,
        children: [ ],
        ancestors: [ ],
        _id: "5ee1f33b6562de064ab2834b",
        type: "Point",
      - value: [
            10,
            0,
            0,
        ],
        owner: "5ed6316c9428a400111462fe",
        __v: 0,
        createdAt: "2020-06-11T09:02:51.321Z",
        updatedAt: "2020-06-11T09:02:51.321Z",
    },
```

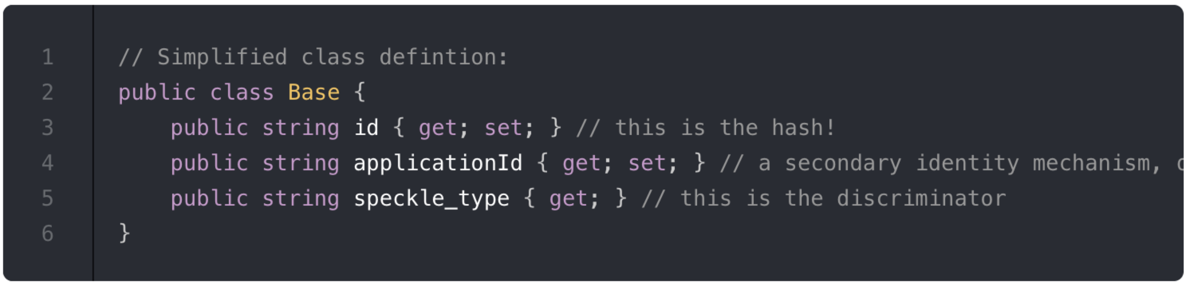
Figure 2.2.3.3a - Example Speckle object serialised to JSON format.

Objects

All Speckle objects share a common base object and are located in the Speckle.NET repository.

The latest release of Speckle (version 2) uses an underlying dynamic object [.NET documentation] as the mechanism to handle the unstructured nature of the data. This allows users to dynamically add properties through the dot notation, and these properties are then created at runtime. This ensures all Speckle objects can, at the most fundamental level, hold an arbitrary set of data. For reliable serialization, the underlying storage of this arbitrary data occurs within a *Dictionary<string, object>* structure, which nicely maps to the JSON format.

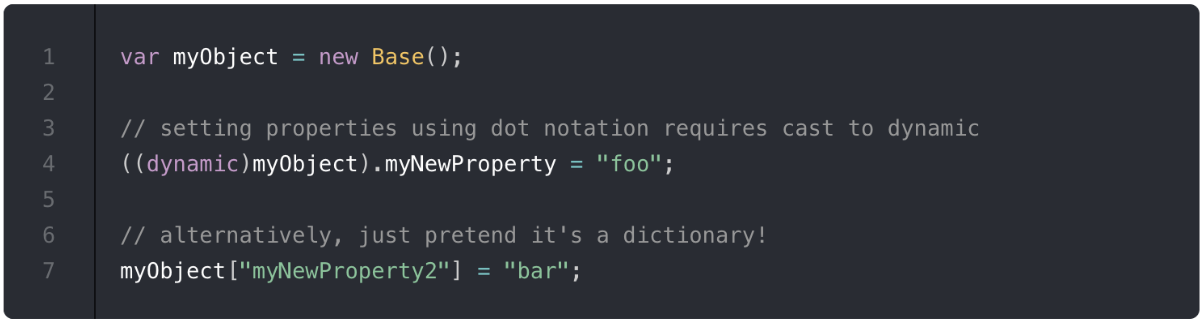
The base Speckle object therefore has the following basic properties.



```
// Simplified class defintion:
public class Base {
    public string id { get; set; } // this is the hash!
    public string applicationId { get; set; } // a secondary identity mechanism,
    public string speckle_type { get; } // this is the discriminator
}
```

Figure 2.2.3.3b - Speckle Base Object

This base object can be used out-of-the-box as shown below, by dynamically assigning properties and values.



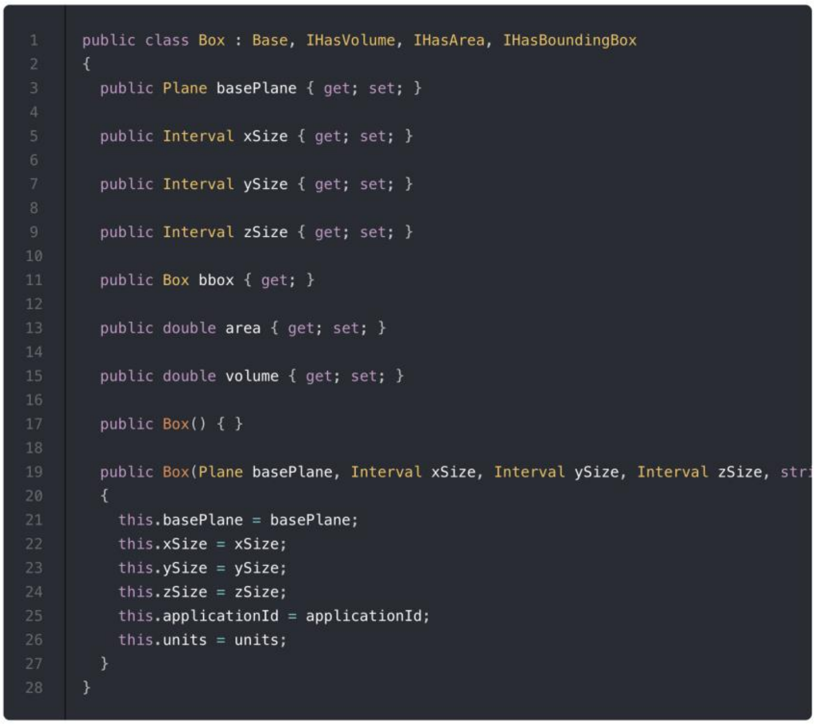
```
var myObject = new Base();

// setting properties using dot notation requires cast to dynamic
((dynamic)myObject).myNewProperty = "foo";

// alternatively, just pretend it's a dictionary!
myObject["myNewProperty2"] = "bar";
```

Figure 2.2.3.3c - Adding properties to the Speckle Base Object.

The Speckle base object handles tasks, like hashing, for interfacing with the Speckle database. Speckle objects are fundamentally immutable, thus changing any single property results in an entirely new object being created. This is controlled via the hash value, which is generated from all the object's properties.

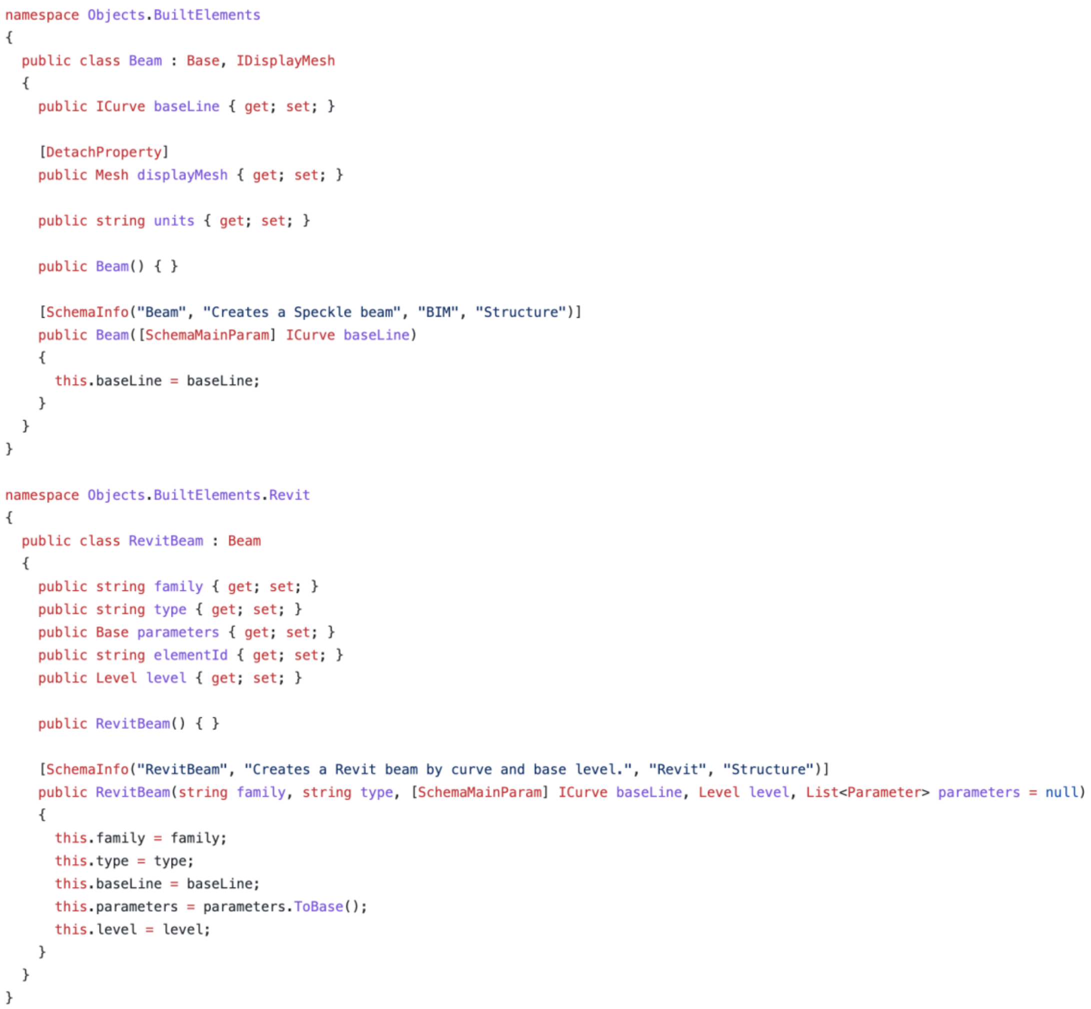
Even though Speckle objects can be used as they are, most object models are extended in one way or another, to generate more specific schemas. This is done by inheriting from the Speckle base object and adding additional properties. This is how the default Speckle objects model is created, and also how any user would create their own, custom model. An example is given below, that shows the extension of the base object into a *Box object*, by assigning properties such as a *Base Plane* and *Size* domains. Users do not have to start a new object model from the original base object, but can instead use any other object which extends the base object as their foundation.

A clear example of this is the default implementation of a *Speckle Beam*, and its child class *Speckle Revit Beam*, shown below:



```
public class Box : Base, IHasVolume, IHasArea, IHasBoundingBox
{
  public Plane basePlane { get; set; }

  public Interval xSize { get; set; }

  public Interval ySize { get; set; }

  public Interval zSize { get; set; }

  public Box bbox { get; }

  public double area { get; set; }

  public double volume { get; set; }

  public Box() { }

  public Box(Plane basePlane, Interval xSize, Interval ySize, Interval zSize, stri
  {
    this.basePlane = basePlane;
    this.xSize = xSize;
    this.ySize = ySize;
    this.zSize = zSize;
    this.applicationId = applicationId;
    this.units = units;
  }
}
```

Figure 2.2.3.3d - New Speckle Box Object, that inherits from the Base Object.



```
namespace Objects.BuiltElements
{
  public class Beam : Base, IDisplayMesh
  {
    public ICurve baseLine { get; set; }

    [DetachProperty]
    public Mesh displayMesh { get; set; }

    public string units { get; set; }

    public Beam() { }

    [SchemaInfo("Beam", "Creates a Speckle beam", "BIM", "Structure")]
    public Beam([SchemaMainParam] ICurve baseLine)
    {
      this.baseLine = baseLine;
    }
  }
}

namespace Objects.BuiltElements.Revit
{
  public class RevitBeam : Beam
  {
    public string family { get; set; }
    public string type { get; set; }
    public Base parameters { get; set; }
    public string elementId { get; set; }
    public Level level { get; set; }

    public RevitBeam() { }

    [SchemaInfo("RevitBeam", "Creates a Revit beam by curve and base level.", "Revit", "Structure")]
    public RevitBeam(string family, string type, [SchemaMainParam] ICurve baseLine, Level level, List<Parameter> parameters = null)
    {
      this.family = family;
      this.type = type;
      this.baseLine = baseLine;
      this.parameters = parameters.ToBase();
      this.level = level;
    }
  }
}
```

Figure 2.2.3.3e - Speckle Beam and Speckle Revit Beams, an example of multi-level inheritance.

■ 2.2.4 Reakt

2.2.4.1 Background

In addition to the development of the IFC format, a lack of uptake in the industry can be seen in the emergence of company-wide, bespoke interoperability approaches. These do not function as extensions or complements to the IFC format, but as completely separate ecosystems, which cater to the specific needs of the company, given project type and general company philosophy. Referring back to the Speckle chapter, D.Stefanescu writes that:

> *[company specific interoperability toolkits] ...result from the problematic nature of IFC, whose scope, while vast, is limited, and extensions of it are not easily shared…*
>
> *...These ad-hoc informed standards have scopes that vary from a single-project base to a company-wide norm. To a certain extent, the emergence of such standards results from the need to further articulate domain- and organisation-specific knowledge that accumulates over time: most observed examples underpin computational techniques, workflows and methodologies that quintessentially represent a given organisation market advantage, or a given domain's internal language constructs that allow for its effective functioning.*
> [Stefanescu 2020]

This chapter contains a detailed description of one of those, namely the *Reakt* toolkit, developed in-house at engineering firm AKT II, the main contributors of this report. But as can be seen in the table (1.2.1.2), there exists a wide range of similar initiatives across various larger companies in the industry. The *Reakt* toolkit is developed within the very specific needs of the company, as a design-led and collaborative practice. [Tibuzzi 2016]. As primarily a structural engineering firm, the scope and application of Reakt, as opposed to the IFC or Speckle, is smaller and more focused on a single discipline.

The Reakt schemas contains many non-engineering categories - such as Geometric entities, embodied carbon analysis and computational fluid dynamic (CFD) entities - however this study of the Reakt toolkit will mainly focus on just one of those aspects, namely the exchange of structural analysis models.

The scope of the Reakt structural toolkit emerges from the day-to-day work of the practise. Some examples of tasks carried out are:

- *Advanced geometry modeling.*
- *Exchanging geometry data, analysis data and BIM data between software.*
- *Structural results interrogation and visualisation.*
- *Variation studies and optimisation of structural systems.*
- *Quickly responding to, by understanding the structural implications of, changes to the architectural design.*

2.2.4.2 Structural analysis domain

The models that make up this domain do not hold the same set of information as their BIM or architectural counterparts. Therefore, the software designed for structural analysis does not cover the same ontological domain as the BIM software does. To highlight this, here will follow a breakdown of the ontological subdomain that constitutes the objects and data that are specific to the structural engineering domain. An overarching illustration of the domain interaction between BIM and analysis models is highlighted in the following diagram:

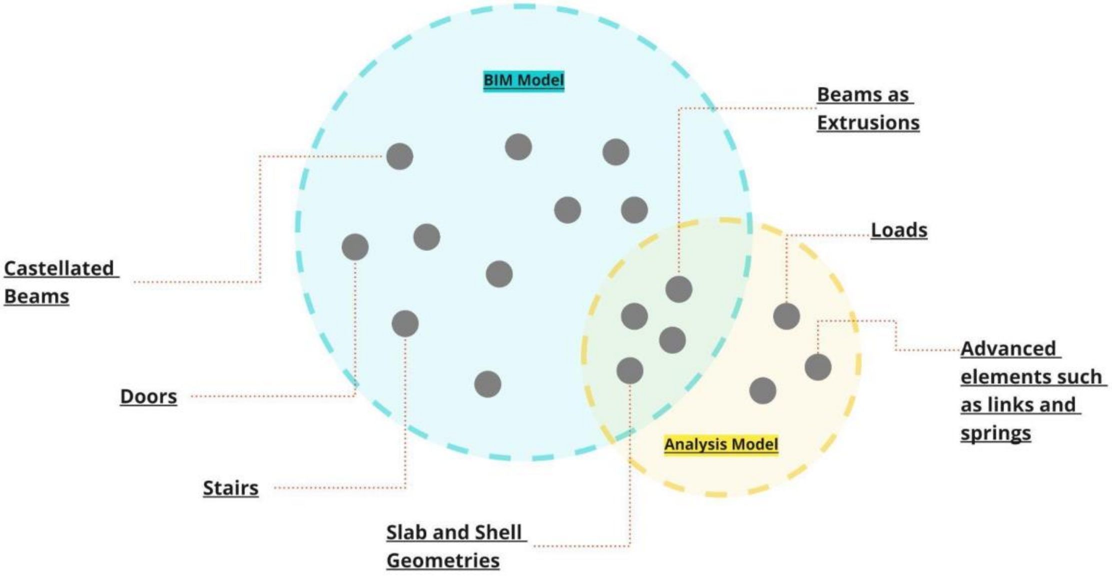


Figure 2.2.4.2a - overlap between structural analysis and BIM domains.

Before going into the workings and data structures of the Reakt toolkit, a discussion of the corresponding domain is necessary. As mentioned, this primarily deals with the exchange and categorisation of a structural model, which should be regarded as separate to (or at least a sub-domain of) the broader architectural domain. The structural analysis domain, as it deals with abstractions, is significantly smaller in scope than the architectural domain.

What needs to be stated is that the objects used in structural analysis largely follow a *behavioural* categorisation. Thus in the analysis world, the distinction or meaning of elements is slightly different than in the architectural world. For example: the word beam has different meanings across these disciplines. In engineering, a screw or column will be modelled as a “Beam” in the sense that it is a linear object that transfers normal, shear and bending forces. To a structural engineer, the actual physical nature or appearance is of lesser relevance. The analysis model simply deals with object behaviours as a means of understanding their physical behaviour.

Some of the objects have a physical relevance and a role to play in the BIM Model, some have not. Either because they are non-geometrical assignments such as support conditions or loads. Objects in the analysis context) might not, through their definition alone, given the macro view of what they represent in real world terms (here this refers to the BIM model, as it tries to document and represent the actual physical entities.) In a behavioural categorization, likewise, some elements in the architectural model might be geometrically excluded from the structural model, as their action is undesired and will be separate from the system as a whole. This might make them only represented as a line load together with other loads such as snow and wind loads. An example of some examples of object mapping between BIM and Finite Element Analysis (FEA) model contexts is shown below.

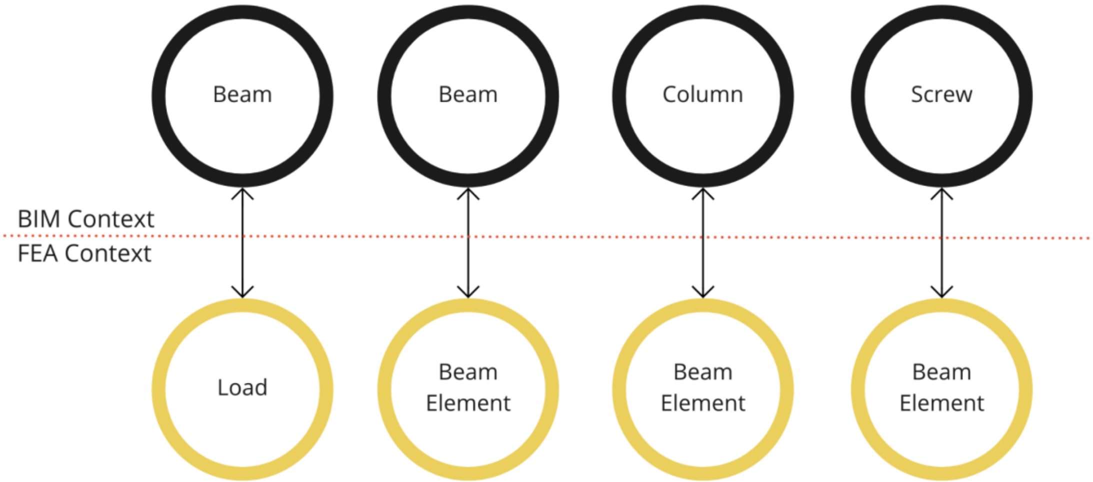


Figure 2.2.4.2b - Example of unintuitive objectmapping between analysis and bim contexts

2.2.4.3 Reakt Structural Object Model

The object model used for the structural part of the Reakt toolkit aims to cover the sets of objects and properties necessary for the exchange of a (primarily) structural model. Compared to the extensive IFC, and the more flexible model promoted by Speckle, Reakt represents a third and slightly different condition. The model is rigid and explicit, much like the IFC data model, however, the scope it is intended to cover is significantly reduced.

The general scope of elements used in a structural model creation can almost be reduced to the following subsets:

- **Structural Elements**
  - 0D
  - 1D
  - 2D
  - 3D
- **Assignments**
  - Boundary Conditions
  - Loads
  - Elements
  - Assignments
    - Materials
    - Cross Sections
    - Releases

Although these can be described in varying degrees of complexity, and there are some additional more advanced features, this scope covers the majority of analysis situations.

Finally, after a user has set up and run a model, this produces a set of analysis results. The majority of these results can be described as simple vector or scalar values for a specific element or location in the system. These simple data structures are then embedded within the particular elements to store data such as *nodal displacements*, *nodal forces*, *element forces* or *element stresses*.

In general Reakt defines only the following set of primary structural objects:

- Structural Node
- Structural Line
- Structural Mesh
- Structural Area
- Structural Link

2.2.4.4 Implementation considerations

The Reakt approach follows the same patterns as that of IFC and Speckle. The core concept of the interoperability approach uses a set of core, platform-agnostic objects to store data, and a library of conversions which reads and writes these containers from various structural analysis packages.

The general object model follows an inheritance chain based on a shared base object, which has the basic properties such as name and ID. Further this is extended either to *main objects*, which are the geometric entities such as beams and nodes, and general objects which can be things like loads, support definitions and cross sections.

The abstract main object has four components:

- Geometry
- Structure
- FEA
- BIM

which are responsible for different data sets needed. The *Geometry* component stores the underlying geometry logic used by most packages, the *Structure* component stores the necessary information for analysis packages, *FEA* stores analysis results and meshing and *BIM* is used to interface with BIM applications, such as Revit. To extend the objects with further functionality, more components could be added to associate more types of data to the object.

■ 2.2.5 BHoM (Buildings and Habitats Object Model)

2.2.5.1 Structure of the BHoM

The Buildings and Habitats Object Model is an initiative targeting interoperability within engineering and related disciplines started at the firm Buro Happold. It is described as:

“... a collaborative computational development project for the built environment. It is a collective effort to share code and standardise the data that we use to design, everyday – across all activities and all disciplines.”

[Official Documentation]

The aim of the BHoM is not to standardise processes and software, but the data to make the work more efficient and enable collaboration. Similar to Reakt, Speckle and IFC, BHoM is concerned with standardisation and establishing software-agnostic exchange formats. Eventually, the essence is the enabling of seamless transitions from code prototyping to final processes across all software used within the company.

2.2.5.2 Implementation Considerations

BHoM makes a fundamental distinction between the two dual components of computation, namely *data* and *manipulation.* Overall, the toolkit is structured around 4 code categories, which are:

- ***oM***
  The Object Model defines the base ontologies, which are grouped into a wide set of objects spanning from structural to acoustics, humans and lighting. Although the core object model is more explicit than Speckle, each object still has the ability to hold custom user data based on a *key value pair* type structure, which handles data local to specific contexts.

- ***Engine***
  The objects themselves define no particular operations, but the manipulations or methods are instead found in the engine. This code domain has all custom algorithms etc which the engineers define.

- ***Adapters***
  Similar to the conversions needed for a speckle kit, the logic for interfacing with proprietary software packages of the bhom is called adapters. These provide the conversion between objects or formats native to an application and the generic BHoM format.

- ***UI***
  The final category, the *UI*, is responsible for actually exposing the previous logic so that the users can interact with it. This could, for example, be in the form of a plugin, like the BHoM Grasshopper or Excel interfaces.

BHoM, like Reakt, does not have a fully-developed central database where the information is stored, as Speckle does. The object model simply handles objects at runtime. BHoM does, however, provide a link which allows reading and writing of data to a Mongo database (MongoDB). Mongo is a database with a structure based key-value pairs, similar to the json format explained in section 2.2.2 and 2.2.3. For more information, see the github repo.

■ 2.2.6 Summary and Discussion

Before establishing a set of criteria for the PrismArch ontology, we must identify and summarise the main characteristics of each of the precedent ontologies already discussed:

IFC

- Explicit / Static formulations for everything, to make direct Computer-to-Computer transactions.
- Myriad near-identical schemas create confusion and ambiguity, despite their explicitness.
- Inconsistency across implementations.
- Problem with a complete model: if objects don't suit users needs, revisions or updates to schemas is a slow and closed / centralised process.
- All responsibility of interpreting on the medium itself.
- Centralised standard - no way for users to customize.
- File based, not queryable.
- Not optimised for database interactions.
- Not optimised for fast web-based data exchange.

SPECKLE

- System designed to handle generic data structures which may or may not have any particular meaning for the particular software deserializing them. More responsibility for interpreting and verifying said information is assigned to the end-user.
- Late binding of data as objects are composable, both with predefined or custom data structures.
- A network of Object Models, shared base, but allowed to branch and expand.
- Ontological Revision.
- Combination of exchanges, object model and implementation /connectors.
- Web-based interface, easily queryable schemas.
- Closely tied to the .NET framework - the standard API language for AEC software.

REAKT

- Specialist / discipline-specific.
- Exhaustive but clearly limited in scope, due to single domain.
- Good overlap between target software platforms.
- Not optimised for database interactions.
- Not optimised for fast web-based data exchange.

BHoM

- Similar to Speckle in providing extendable objects.
- Decoupling of objects as containers and methods.
- Not optimised for database interactions.
- Not optimised for fast web-based data exchange.

What are some general trends that can be seen across the approaches?

- Both Speckle and IFC have distinctions between common or shared domains and specialist domains.
- Central object model used as a mechanism for connecting various software.

<u>What are some differences that can be seen? Which are favourable, and why?</u>

- **File-based vs. Object-based**
    - File-based requires exchange of complete models everytime.
    - Object-based approaches offer more flexible, atomised comparison between models and potential for database integration.
    - Object-based approach should be implemented in Prism.

- **Composed vs Complete**
    - Reakt Structural and IFC are complete models, necessitating a reduced scope to be manageable.
    - References made to other vast and all-encompassing explicit or structuralist ontological frameworks show that these types of formulations are bound to be incomplete. Further, it can be seen that the custom user data approach (i.e. the composed model) is widely used, as it is reported that, based on analysis of objects sent through the Speckle platform, 18% of the total objects made and sent are using custom sets of user data. [Stefanescu 2020]
    - A composed model should therefore be implemented in Prism, to ensure complete flexibility in element definition.

What are some core concepts to consider for the definition of PrismArch ontologies?

- **<u>Predictability vs Flexibility</u>**

  How generic can we make something for the automatic computer-to-computer exchanges to still make sense while retaining a customizable and inherently flexible system? A system that is able to respond to ad-hoc datastreams in relation to project specific demands.

  Essentially, this boils down to what degree the data should be interpretable by the computer or medium, or to what degree the computer is just responsible for the transaction and storing and the interpretation is instead down to the human consumer.

- **<u>Decentralisation of Models</u>**

  A fallacy of the central object model is that not all contexts have the same set of objects and properties. Likewise there will be an ambiguity when exporting an object from a reduced context, as to what form this object should take in the super-context, as there might be a series of possible answers.

  An example is a material: for the architect this may relate to finishes, visual qualities (which are rendered) or cost and sustainability aspects. However, for an engineer a material will also, in the structural context, have strength, weight, elastic, plastic, orthotropic qualities, etc. describing its predicted behaviour, which is irrelevant information for architects and MEP engineers.

Asking one model to do everything also make it rigid and slow to expand, difficult to modify. If your current case is not covered in the model, you have to do something else, and your information simply cannot be exchanged. A decentralised network of flexible models will be able to adapt accordingly.

- **What Data is needed to capture the design process as a whole, not just the design**
  There is a tendency of object models to highly focus on the representation of the actual built elements and their physical relations and properties. For data structures existing in the precedents for auxiliary information such as sketches, voice recordings, emails, preceding reports or design standards. However, the flexibility of the Speckle format would allow for the encoding of such data.

- **File-Based Technology vs a Database or Cloud-Driven**
  Data structures should be built around an ability to efficiently transfer information across web and similar mediums, where concepts like diffing should be preferred over always sending an entire model.

○

○ 2.3 Requirements for New PrismArch Ontologies

■ 2.3.1 Introduction

In the previous chapters we evaluated the characteristics of several significant existing AEC ontologies, and it is clear that Speckle's ontology and composable object model ((https://dimitrie.org/thesis/discussion/#71-schemas-and-standards) is highly advanced compared to many of the others, with the additional benefits of being both open source and supported by several extra functions, such as the web interface, and mature database integration.

We can therefore now introduce further functionality required specifically by PrismArch, which has been highlighted across prior Deliverables, and evaluate how well the existing Speckle framework supports them, then suggest minimal-effort solutions to aid integration where necessary. These functionalities are:

- *Interoperability Across AEC Disciplines*
- *Deltas & Database Integration*
- *PrismArch Asset Signature*
- *PrismArch Tags*
- *Architectural Requirements*
- *Structural Engineering Requirements*
- *MEP Engineering Requirements*

- *Texture Mapping*
- *VR Optimisation*
- *ML Optimisation* (titled 'Quality Diversity and Designer Modeling in PrismArch')

■ 2.3.2 Ontology Requirements Outlined in Previous Deliverables

Interoperability Across AEC Disciplines

A set of desired objects and data types have been previously outlined in deliverable D1.1 Section 4.2 page 109. Further, this deliverable also outlines the need for flexible structures and the need for appending custom or bespoke data at any point to any object. Both these specific and unknown data structures will be considered when the overall approach is outlined in the following chapter, based on the observations made in the precedent study. The wider problem of interoperability has been covered previously in this report in chapter 2.2.1, so for further reference see that.

Deltas & Database Integration

Database requirements have been mentioned and referenced in previous documents, both in relation to the intended use cases and from an implementation perspective. Speckle has previously, in Deliverable D5.1, been suggested as the primary choice of database solution, with a high prioritization stated for the asynchronous modes of collaboration and data exchange. Deliverable D1.1 emphasizes the necessity of recording the development of a project through all stages (e.g. using time-history documentation), as well as the need to thoroughly search and query all entities added to projects on the PrismArch platform. Additional requirements in D1.1 state the need for a singular, persistent database for all stages in the design process, which should be accessible for any partner at any point, assuming they have the requisite permission level.

With Speckle, every database insertion is a commit, and a history of all commits is stored for the project's lifetime. Speckle reinforces itself as the most suitable solution, as Speckle's JSON-based logic is also highly queryable.

Beyond the requirements stated above, PrismArch must also consider whether other efficient methods for transmitting data can be incorporated. As mentioned in chapter 2.2.2, AEC models are often saved in very large files, containing thousands or even hundreds of thousands of elements. Having to resend all of these elements to the database - just because a few among them have changed - is inefficient, and would reduce the likelihood that the database can seamlessly integrate with realtime technologies such as VR. In contrast, software programming uses highly efficient version control systems - such as GIT - that identify the changes (sometimes called '*deltas'* in software development) between the current and previous versions of a file, and ensure that only those *deltas* are sent to the repository. The process of determining the differences between versions of a file is called *'diffing'*.

In an AEC context, there are fewer examples of this process being attempted. The most useful example for PrismArch is a research project called **AECDeltas**, that was undertaken by representatives from both Speckle and BHoM, and which attempted to reconcile the "diffing" process within their respective ontologies [AECDeltas]. This initiative will not be covered in detail here, however the general problem is neatly formulated in this graphic, produced by the AECDeltas consortium:

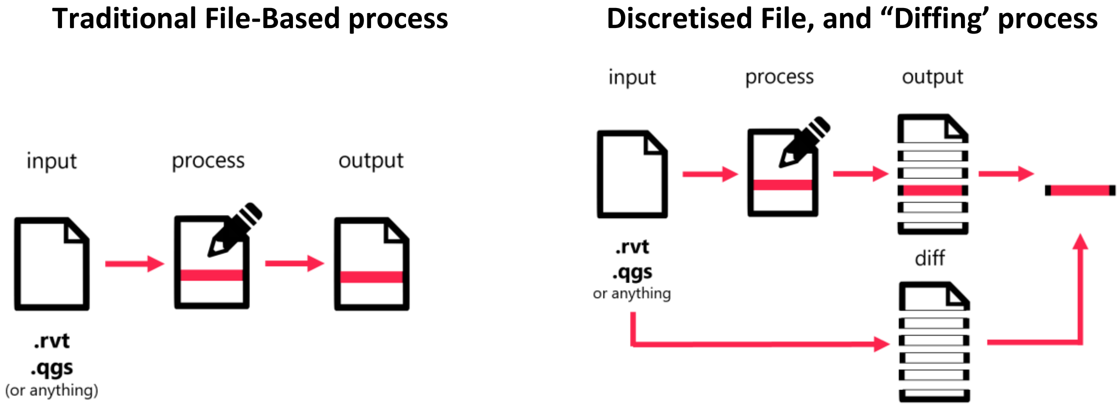


Figure 2.3.2a - AECDeltas Diffing process [Image credits: AEC deltas]

In a file-based exchange of information (left), any singular modification necessitates a re-export of the entire file. On the (right) however, the output is discretised, allowing the diffing process to isolate and export only the critical deltas.

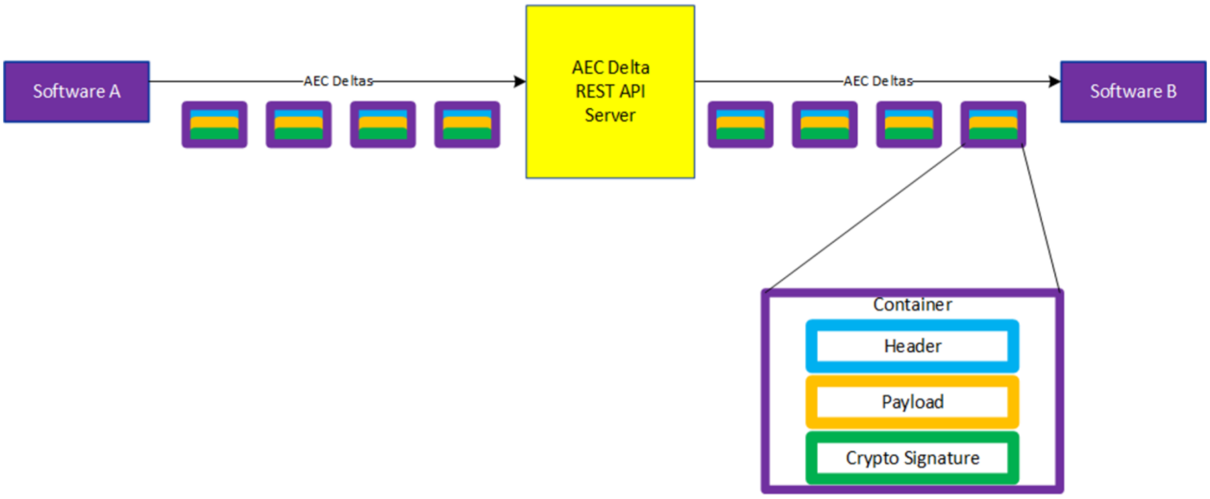


Figure 2.3.2b - Conceptual structure of the AECDeltas *delta* element [AECDeltas, Spec]

However, it is currently unclear how much of AECDeltas' desired diffing functionality has been integrated into *Speckle 2*. Some documentation exists that seems to suggest partial implementation within the previous Speckle 1 system [Speckle Diffing].

As part of the next phase of Work Packages 4 and 5, we should therefore establish the potential value of these processes for PrismArch, by testing first working prototypes and benchmarking the relative lag between generating design elements in external software, passing them to the database, and pulling them into the PrismArch VR environment. The outcome of these tests - as well as further research into the current implementation of diffing in Speckle 2 - will determine the value and viability of transmitting deltas within PrismArch.

PrismArch Asset Signature

As specified within *D1.1 Section 4.2 (C - Rules)*, there is a requirement for all PrismArch assets to be uniquely identified, and clearly labelled so that there can be absolute assurance about the authoring and attribution of every element - i.e. which individual and/ or company created an asset, at what time, the level of visibility/ privacy assigned to the asset, and so on.

Therefore, whenever any PrismArch asset(s) are created, a paired Prism_Signature element is created and integrated into the asset(s) prior to uploading to the PrismArch database. The method for integrating these Assets and Signature is described in *Section 2.4 Cross-Disciplinary Foundation for Ontology.*

The structure of the Prism_Signature has been kept intentionally basic: it must be a lightweight entity that excludes complex data types, and produces only small increases to the file size, thus ensuring that the creation of Signatures will not slow the flow of information to the Prism Database.

This small entity 'footprint' would also mean that there is the potential for these Signatures to be pushed to a wide array of other sources, beyond the Database. Automated alert systems could be set up to notify users - via email, text message, Slack or other messaging service - when new updates have been received by the Database, and could display the entirety of each Signature, providing users with a compressed history of the commits being made by different parties.

These Signatures can also be used to provide a thorough log of work created and changes made during the life of a project, for accountability and 'Golden Thread' requirements (outlined in *D1.1 Section 2.1*). However some consideration must be given to when notifications concerning new Signatures is shared with other parties: as these Signatures are generated for every asset created, it would be possible/ likely to overwhelm other users with messages on days when multiple teams (or a single large design team) is working within the PrismArch space.

See Specification Section (2.4.7) for proposed specification of the Prism_Signature Object.

PrismArch Tags

Alongside the requirement for the Prism_Signature, there must also be a flexible tagging system, that allows users to freely annotate assets, both within their discipline, and for the benefit of other design partners, clients and contractors. This requirement is discussed within multiple previous Deliverables, most notably within *D1.1 Section 4.2 (Taxonomies), D1.1 Section 4.4 (Item 2)* and *D3.1 Section 4.2 (Scenarios Developed as Part of Project Deliverables).*

In an AEC setting it is frequently necessary to add temporary tags to objects, which are removed or updated at a later date. For example, a "*For Review*" tag that must be deleted and replaced with a "*Change Accepted*" Tag once a model has been checked and agreed upon. Therefore flexibility and adjustment of tags is paramount.

We therefore believe that the existing tagging system present within Speckle is not highly suitable. Speckle Tags are assigned to Speckle Objects as they are created, and as such they are effectively 'read only' attributes of the parent object. They cannot be updated on their own, and the only available option would be to remove the original object and tag and re-commit the same object to the Speckle database, but with a new tag assigned. This approach would work, however it is inefficient, slow and unnecessarily expensive in terms of data-transfer, and increases the risk of data-loss, as the server connection could break during re-committing and the object would no longer be stored in the Database.

Therefore we propose the creation of two tag-related objects that together produce the necessary tagging functionality, while working within the existing Speckle 'Push-only' paradigm. These are self-contained entities that exist entirely outside the object that they reference - thus they can be deleted or superseded without deleting or otherwise affecting the object that they refer to.

The two objects are:

- **Prism_Tag** A unique tag name. Cannot be duplicated. e.g. the *"For Review"* Tag.
- **Prism_TagConnection** Assign a single Prism_Tag to a single Prism_Object.

  e.g. the *"For Review"* Tag is assigned to Beam_A. e.g. the *"For Review"* Tag is assigned to Beam_B.

The same Tag can be assigned to multiple objects using multiple TagConnectors. When a Tag should no longer be assigned to a specific object, the relevant TagConnector is either deleted or superseded, but the Tag itself is never deleted.

Just like all other objects, both the Tag and TagConnector objects must contain all relevant data to determine their origin, provenance, Sphere Level, etc. They are therefore assigned a Prism_Signature just like any other asset, and then bundled inside their own Prism_Objects like any other element.

Certain information tagged to an object must be perpetual and non-editable. For this reason, all Tags and TagConnectors have an 'Editable' property flag assigned to them, which states whether they can ever be edited or superseded in future, after their initial creation.

This unusual relationship between Objects, Prism_Tags and Prism_TagConnections is driven by the specifics of the Speckle Ontology and Database connections. These proposals could be revisited if our understanding of Speckle changes in future.

See Specification Section (2.4.7) for proposed specifications for both Tag Objects.

A) Architectural Requirements

Top level architectural information such as 'project categories', 'project types', and 'project scopes' are vital inputs to cover all the possible scenarios of architectural projects. The Rhino layering structure and both discipline-specific (internal) and cross-disciplinary task distribution reflects this information topology.

Listed below are the primary naming conventions for the architectural discipline.
Architectural naming conventions require flexibility, and the categories should be expandable.

- **Project Typology**
  - Civic

  - Cultural
  - Architecture
  - Education
  - Healthcare
  - Hotel
  - Industrial
  - Leisure
  - Mixed Use
  - Offices
  - Residential
  - Retail
  - Sports
  - Sustainable
  - Towers
  - Transport
  - Exhibition
  - Interior
  - Product
  - Fashion
  - Furniture
  - Infrastructure
  - Landscape
  - Urban Planning

- **<u>Project Scope Elements</u>**
  - Planning
  - Landscape
  - Architecture
  - Interior
  - FFE
  - Product Design

- **<u>Architectural Layers</u>**
  - **Arch_**
    - Context/ Site
    - Streets
    - Transport
    - Public spaces
    - Soft landscape (e.g. trees and grasses)
    - Hard landscape (e.g. water features, pools, walkways etc)
    - Walls
    - Slabs/ Floor Levels
    - Ceilings
    - Columns
    - Facade/ Envelope
    - Core
    - Staircase

  - Environmental (e.g. solar, wind etc)
  - Zoning
  - Areas
  - Survey

- **IntArch_**
  - Context/ Site
  - Finishes
  - FFE (Free-Standing Furniture and Equipment)
  - Integrated furniture
  - Environmental (e.g. solar, wind etc)
  - Areas
  - Survey

B) Structural Engineering Requirements

An overarching list of data for the structural engineering discipline, as pointed out in section 2.3.2, can be found in deliverable D1.1. Here further elaboration on some particular objects and data needed will be given. This will specifically focus on the model objects such as the components of analysis models, as opposed to reports or building codes. IFC has no good functionality for these types of objects, so no standardised schemas can be appropriated. The engineering models will share some overlap with the architectural domain, as some objects will have a direct architectural relevance, but it will also need some specific objects, which are only related to analysis models.

An engineering subdomain would need to extend the relevant architectural object model in two ways. One is the extension of the available object with the model specific inputs which are missing from a default architectural object model. Some of these would be *new non-geometric objects*, which would be non-existent in an architectural representation of a building. These are primarily:

- **Loads**
  - *PointLoads*
  - *LineLoads*
  - *Face Loads*
  - *Temperature Loads*
  - *LoadCases*
- **Advanced assignments**
  - *Release Conditions*
  - *Support Conditions*

Some are extensions of objects which are already existing in the architectural domain, but need further information to be useful for an engineer. This includes:

- *Beam profile properties*
- *Structural Properties for Materials*
- *Reinforcement specifications*

Most of these objects will be appended to a top level geometric object which will be an analysis specific geometric object. These are the 0D, 1D, 2D objects discussed in section

2.2.4. As these *may or may not map directly to an architectural element*, these will have to be separate entities compared to the architectural entities, due their abstract nature. The level of abstraction, as in: "*does an object represent an actual built element or simply capture a condition?"*, will have to be considered for interfacing with the architectural domain. In the engineering domain the distinction would only be:

- **0D:** Points for loads or support assignments
- **1D:** Linear elements such as beams, columns and cables
- **2D:** Area elements such as slabs, walls and shells

3-dimensional elements are considered so rare in building engineering applications that they can be excluded for the most general use cases.

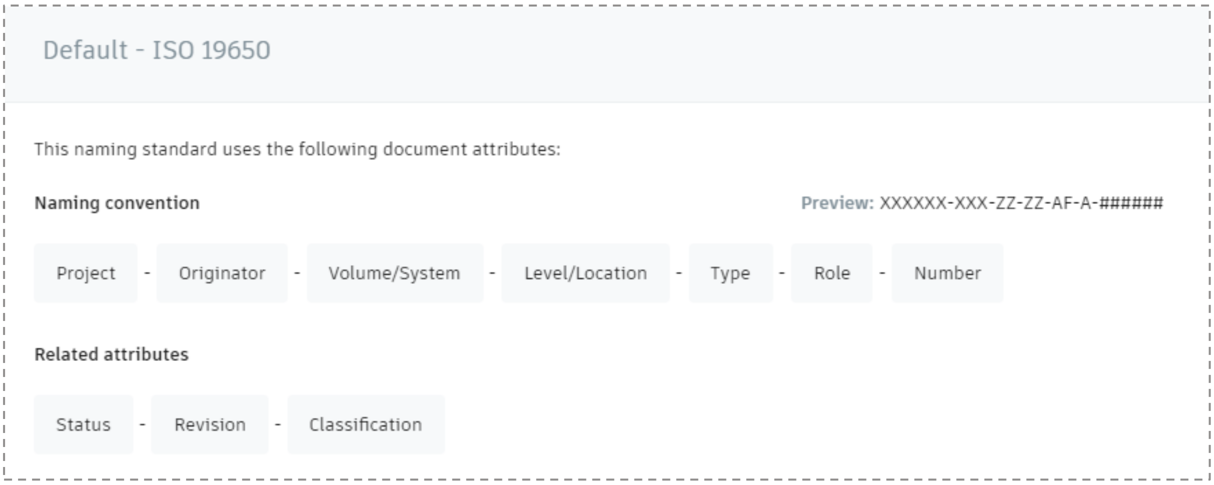
C) MEP Engineering Requirements

Due to the complexity and diversity of elements, tagging plays one of the most important roles in MEP data management. The different elements within a project file contain parameters which define different aspects of the elements. These parameters may contain common information such as length, width or height, information regarding the position the element is installed or other data such as installation or service date. The combination of the 3D element, the symbolic graphical representation, the parameters addition and even the calculation values aid in the wider Building Information Modelling process. The tagging process from an MEP perspective contain two parts:

Part 1 - Documentation Tagging:

The progress made over the years in digital information introduced new needs on data management. Therefore, manufacturers, consultants and other relevant parties started to request forms of classification to their models. This led to the development of two major classifications systems, the Omniclass and the Uniclass classification systems. Due to the complexity, broadness and diversity of elements within a construction project, each element may contain Uniclass or Omniclass data. This aids in construction and facility management. The development of BIM also introduced a set of documents which is used throughout the lifecycle of a construction project. Information stored in the models are driven by definitions and clarifications stated in these documents. These include the Employer's Information Requirements (EIR), the BIM Execution Plan (BEP), Task Information and Master Information Delivery Plans, Mobilization Plan etc. The BIM Documentation contains vital information for the project progress and everyone involved needs to follow whatever is stated for better collaboration and increased effectiveness. As an example, if the project coordinates are incorrect then the project file will not be placed properly and will be misaligned when linked into a file. This causes wasted time, inconsistency and reduced collaboration.

What started as a British Standard in BIM evolved to a series of ISO documents and became a worldwide norm. The ISO 19650 is the standard for Building Information Modelling properly defining the information exchange between stakeholders and ways to form data to better facilitate the information management within construction projects. To begin with, the naming convention and related attributes as defined in ISO 19650 is as follows:



Default - ISO 19650

This naming standard uses the following document attributes:

Naming convention

Preview: XXXXXX-XXX-ZZ-ZZ-AF-A-######

Project - Originator - Volume/System - Level/Location - Type - Role - Number

Related attributes

Status - Revision - Classification

Source: BS EN ISO19650:2-2018

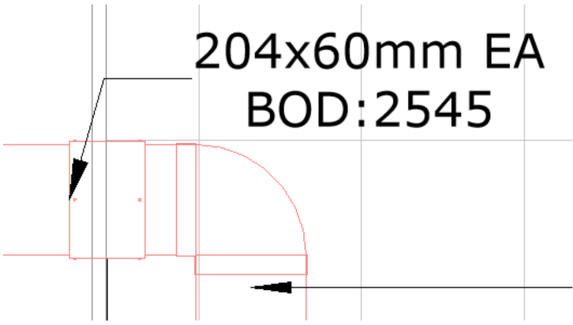
This information is defined in the EIR, BEP and other relevant documentation. These are crucial to be followed and needed as well within the VR environment. A search for a document would be identified by this naming convention and even before opening the actual document we can understand the spatial arrangement (volume), the floor (level/ location) the type of document (drawing, 3d model, sketch, specification etc) and the role (electrical, mechanical, public health). If there is an agreed numbering then even the number could define what type of document this is (for example 1000 for above ground drainage, 2000 for domestic water services etc).

All this information can be pulled directly from the ISO19650 documentation and applied in PrismArch. The documentation is intended to cover all disciplines. Furthermore, the BS EN ISO 19650:2018 series contain the National Annex and better clarify the approach to information management based on the British Standards.

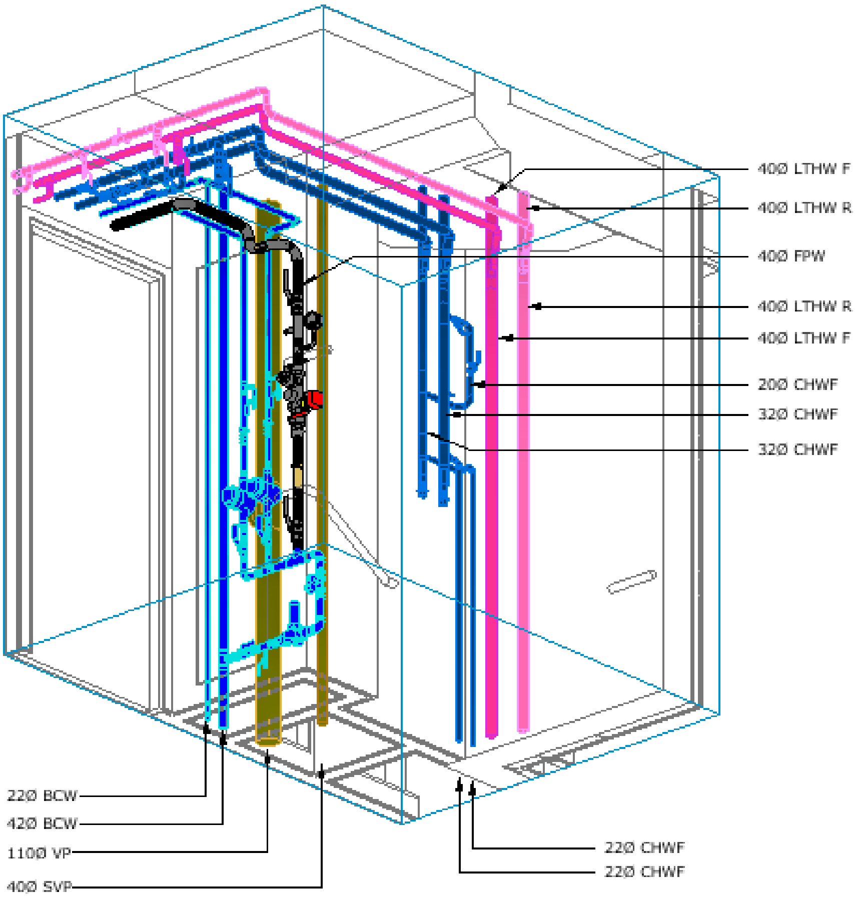
Part 2 - 3D Model Tagging:

The most common software used for BIM projects in MEP is Revit. Revit gives the ability to work with diverse forms of information which may contain 2D information, point clouds, 3D elements and manufacturer’s data. To facilitate such complex tasks Revit uses lots of data and pushes the information to the user’s monitor or sheets through tags.

Tagging in Revit is driven by Annotation Families. The Annotation Families are files which contain Parameters, labels and can also contain graphical elements such as arrows or 2D linework. There is also the ability for nested items within the Annotation Families such as white backgrounds. The Annotation Family pulls data from the actual element’s parameters and pushes it to the label of the Annotation Family. Then the tag is presented in the viewport and/ or sheet. Revit also gives the ability to tag elements within 3D views.

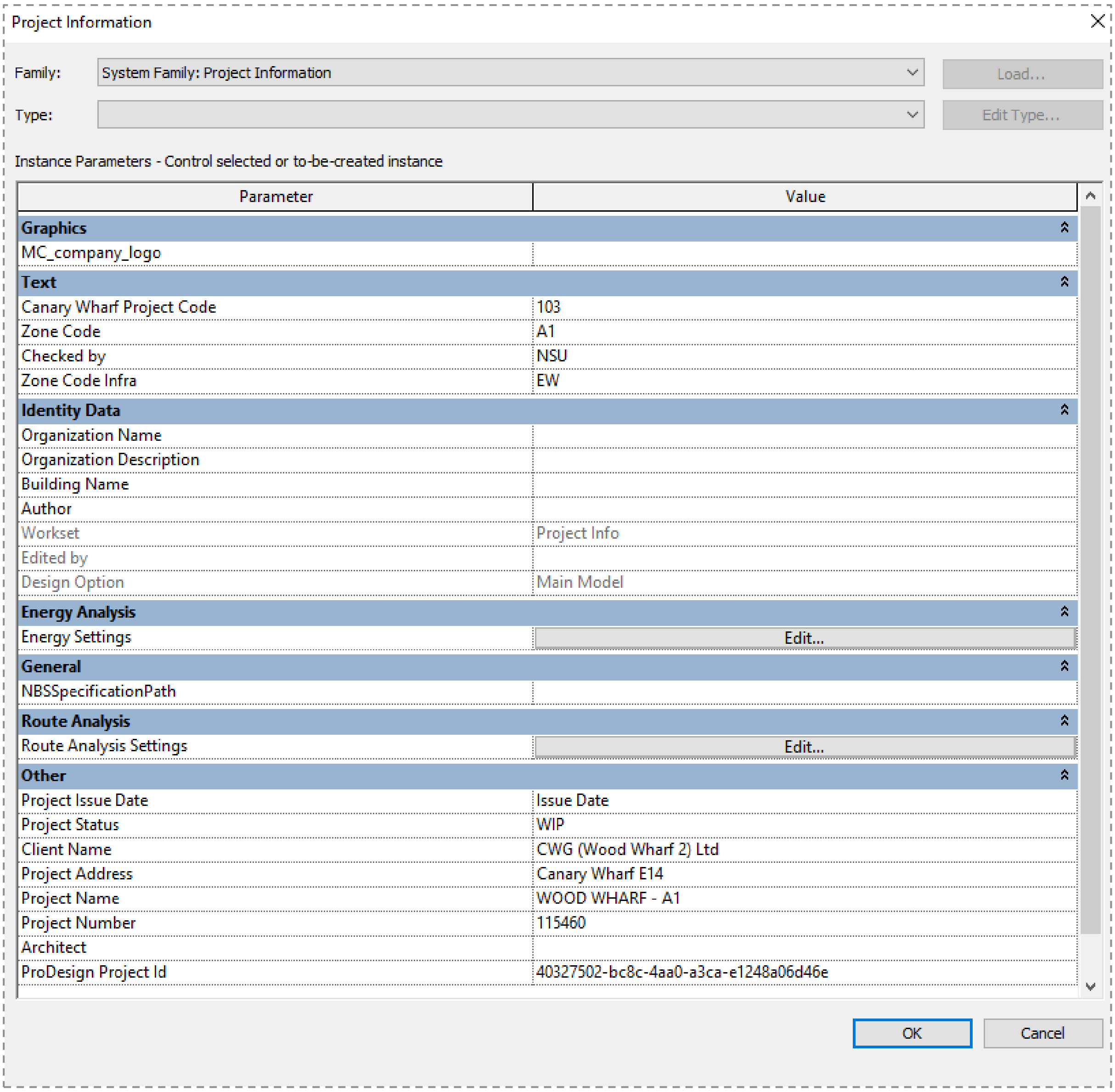


Example of tag in a 2d plan view within Revit



Example of multiple tags in a 3d view within Revit

The nature and amount of data required within the projects is driven by the agreements the appointing party makes with the appointed parties and is also standardized (e.g. RIBA Stages, ISO19650 Status and Suitability codes) to avoid confusion during the information exchange. Within PrismArch, the MEP project file should require at least the Project Information in the form of tags. The Project Information is the initial data required in every construction project and is covered in the form of System Family within Revit. This information should be transferred in PrismArch and be able to be viewed as tags through a widget which would be able to pull this information from the transferred file.



Family: System Family: Project Information

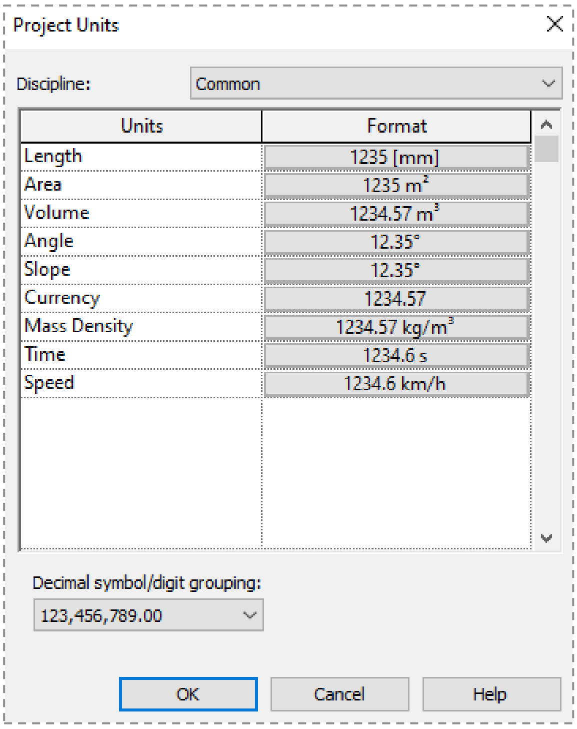
Type:

Instance Parameters - Control selected or to-be-created instance

Parameter	Value
Graphics	
MC_company_logo	
Text	
Canary Wharf Project Code	103
Zone Code	A1
Checked by	NSU
Zone Code Infra	EW
Identity Data	
Organization Name	
Organization Description	
Building Name	
Author	
Workset	Project Info
Edited by	
Design Option	Main Model
Energy Analysis	
Energy Settings	Edit...
General	
NBSSpecificationPath	
Route Analysis	
Route Analysis Settings	Edit...
Other	
Project Issue Date	Issue Date
Project Status	WIP
Client Name	CWG (Wood Wharf 2) Ltd
Project Address	Canary Wharf E14
Project Name	WOOD WHARF - A1
Project Number	115460
Architect	
ProDesign Project Id	40327502-bc8c-4aa0-a3ca-e1248a06d46e

Screenshot of the Project Information within Revit

Furthermore, the Project Units should also be transferred to ensure consistency. The Units are defined in the early stages of the project, all participants use the same unit system and this information should be also viewable within PrismArch. Revit has a specific command for Project Units which brings up the following window in the user's monitor.



Project Units

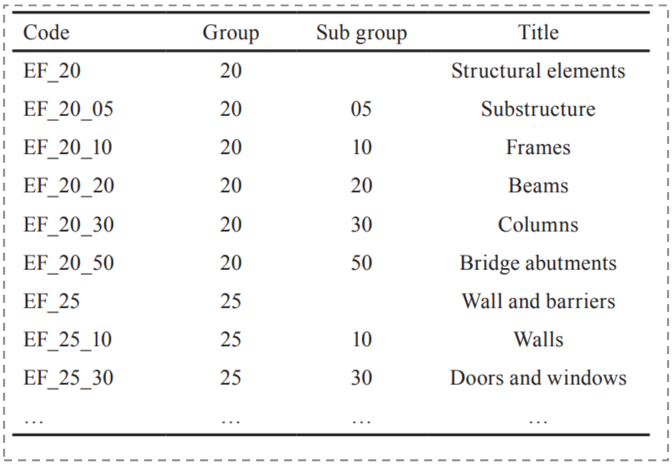
Discipline: Common

Units	Format
Length	1235 [mm]
Area	1235 m^2
Volume	1234.57 m^3
Angle	12.35°
Slope	12.35°
Currency	1234.57
Mass Density	1234.57 kg/m^3
Time	1234.6 s
Speed	1234.6 km/h

Decimal symbol/digit grouping: 123,456,789.00

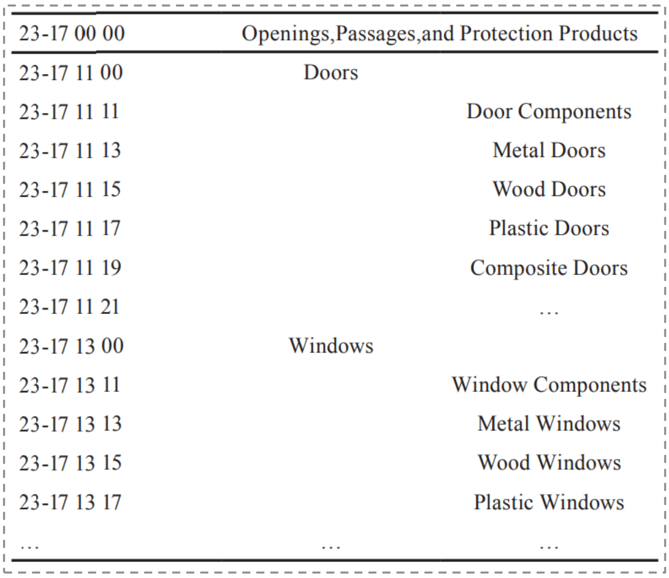
Screenshot of the Project Units Window within Revit

Additionally, the MEP elements will need to be covered based on hierarchical ontology within PrismArch. Earlier the Uniclass and Omniclass were mentioned. These classification systems can cover the hierarchical ontologies within PrismArch for MEP elements. The classification systems have the ability to categorize all elements which exist within a construction project.



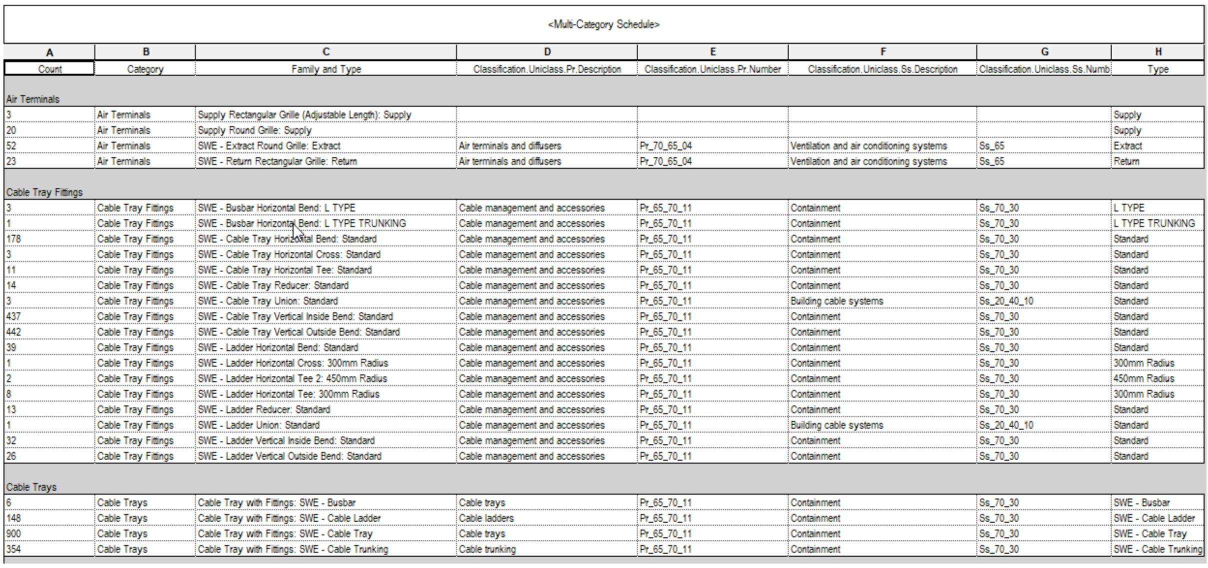
Code	Group	Sub group	Title
EF_20	20		Structural elements
EF_20_05	20	05	Substructure
EF_20_10	20	10	Frames
EF_20_20	20	20	Beams
EF_20_30	20	30	Columns
EF_20_50	20	50	Bridge abutments
EF_25	25		Wall and barriers
EF_25_10	25	10	Walls
EF_25_30	25	30	Doors and windows
…	…	…	…

Example of A Uniclass classification system. Source:https://www.thenbs.com/our-tools/uniclass-2015



23-17 00 00	Openings,Passages,and Protection Products	
23-17 11 00	Doors	
23-17 11 11		Door Components
23-17 11 13		Metal Doors
23-17 11 15		Wood Doors
23-17 11 17		Plastic Doors
23-17 11 19		Composite Doors
23-17 11 21		…
23-17 13 00	Windows	
23-17 13 11		Window Components
23-17 13 13		Metal Windows
23-17 13 15		Wood Windows
23-17 13 17		Plastic Windows
…	…	…

An example of Omniclass Classification System.
Source:https://www.csiresources.org/standards/omniclass/standards-omniclass-about



<Multi-Category Schedule>

A	B	C	D	E	F	G	H
Count	Category	Family and Type	Classification.Uniclass.Pr.Description	Classification.Uniclass.Pr.Number	Classification.Uniclass.Ss.Description	Classification.Uniclass.Ss.Numb	Type
Air Terminals							
3	Air Terminals	Supply Rectangular Grille (Adjustable Length): Supply					Supply
20	Air Terminals	Supply Round Grille: Supply					Supply
52	Air Terminals	SWE - Extract Round Grille: Extract	Air terminals and diffusers	Pr_70_65_04	Ventilation and air conditioning systems	Ss_65	Extract
23	Air Terminals	SWE - Return Rectangular Grille: Return	Air terminals and diffusers	Pr_70_65_04	Ventilation and air conditioning systems	Ss_65	Return
Cable Tray Fittings							
3	Cable Tray Fittings	SWE - Busbar Horizontal Bend: L TYPE	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	L TYPE
1	Cable Tray Fittings	SWE - Busbar Horizontal Bend: L TYPE TRUNKING	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	L TYPE TRUNKING
178	Cable Tray Fittings	SWE - Cable Tray Horizontal Bend: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
3	Cable Tray Fittings	SWE - Cable Tray Horizontal Cross: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
11	Cable Tray Fittings	SWE - Cable Tray Horizontal Tee: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
14	Cable Tray Fittings	SWE - Cable Tray Reducer: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
3	Cable Tray Fittings	SWE - Cable Tray Union: Standard	Cable management and accessories	Pr_65_70_11	Building cable systems	Ss_20_40_10	Standard
437	Cable Tray Fittings	SWE - Cable Tray Vertical Inside Bend: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
442	Cable Tray Fittings	SWE - Cable Tray Vertical Outside Bend: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
39	Cable Tray Fittings	SWE - Ladder Horizontal Bend: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
1	Cable Tray Fittings	SWE - Ladder Horizontal Cross: 300mm Radius	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	300mm Radius
2	Cable Tray Fittings	SWE - Ladder Horizontal Tee 2: 450mm Radius	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	450mm Radius
8	Cable Tray Fittings	SWE - Ladder Horizontal Tee: 300mm Radius	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	300mm Radius
13	Cable Tray Fittings	SWE - Ladder Reducer: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
1	Cable Tray Fittings	SWE - Ladder Union: Standard	Cable management and accessories	Pr_65_70_11	Building cable systems	Ss_20_40_10	Standard
32	Cable Tray Fittings	SWE - Ladder Vertical Inside Bend: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
26	Cable Tray Fittings	SWE - Ladder Vertical Outside Bend: Standard	Cable management and accessories	Pr_65_70_11	Containment	Ss_70_30	Standard
Cable Trays							
6	Cable Trays	Cable Tray with Fittings: SWE - Busbar	Cable trays	Pr_65_70_11	Containment	Ss_70_30	SWE - Busbar
148	Cable Trays	Cable Tray with Fittings: SWE - Cable Ladder	Cable ladders	Pr_65_70_11	Containment	Ss_70_30	SWE - Cable Ladder
900	Cable Trays	Cable Tray with Fittings: SWE - Cable Tray	Cable trays	Pr_65_70_11	Containment	Ss_70_30	SWE - Cable Tray
354	Cable Trays	Cable Tray with Fittings: SWE - Cable Trunking	Cable trunking	Pr_65_70_11	Containment	Ss_70_30	SWE - Cable Trunking

Screenshot of a multi-category schedule within Revit containing elements with Uniclass information

The tagging within PrismArch for MEP elements has a broad range of applications. The use of parameters and tags will aid in the information management and clarify the scope and purpose of the elements. Additionally, other forms of information can be exchanged through tags to assist in the design solution. Another example is the acronyms to clarify installation levels such as *FTS* (Fixed to Soffit), *BOD* (Bottom of Duct), *FA* (From Above) etc… This information should be included in the tags and selected appropriately to assist people who review the model. This will reduce time waste and add value to the project since the reviewer can verify the amount of work and the effort given for the best results.

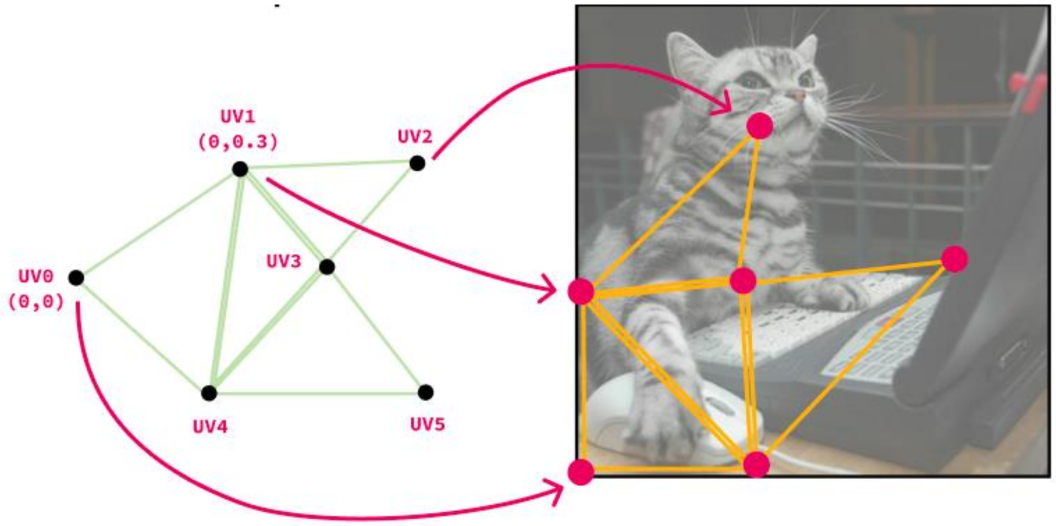
■ 2.3.3 Texture Mapping

The process of displaying 2-dimensional texture information on the surface of a 3-dimensional polygonal mesh is known as Texture Mapping: any arbitrarily chosen sample on the surface of the mesh needs to correspond or 'map' to a pixel of a 2D- rectangular image. There are various techniques that can achieve this kind of mapping, but the most common ones are *UV Maps* and *UV Projections*. Other methods, such as Walt Disney Animation Studios' Ptex format, are either uncommon outside large studios or impractical in real-time applications.

UV Maps / Texture Coordinate Maps

Introduction

A UV Map of a polygonal mesh, is a collection of 2-dimensional vectors or coordinates in the form of {***u,v***} where ***u*** denotes a normalized (0.0 to 1.0) position along the X-axis of a 2D Texture Space, and **v**, accordingly, a position along the Y-axis. A mesh with N number of vertices, typically stores N amount of UV coordinates.



3D Space **Texture Space**

Figure 2.3.3a - Mapping from 3D Euclidean space to 2D Texture space.

Modern graphics pipelines handle in the background (in the graphics shader's fragment stage) the smooth interpolation of UV coordinates across the surface of a triangle, using barycentric coordinates. Therefore, the definition of the UV Coordinates on just each triangle's corners (vertices) is enough for the whole triangle to be mapped successfully from 3D space to 2D Texture space.

The authoring of UV maps is typically performed at the modelling / texturing stage during the asset preparation process. Dedicated software packages or modules inside more general 3D software allow artists to fine-tune their UV Layouts according to their visual quality criteria. This process is crucial in the architectural visualization, entertainment and game-art industries, and typically performed by technical artists.

However, our user interviews with ZHA and AKT showed that it's not common for architectural designers, let alone structural engineers, to be aware of - or proficient in UV authoring workflows. This can result in exported models with missing or incorrectly set-up UV coordinates, and by extension, to unpleasing or erroneous texture display.

In such cases, meshes without UV coordinates should be *programmatically detected*, and their UV coordinates should be generated procedurally by the runtime, if a correct, faithful material appearance is desired.

UV Unwrapping

UV Unwrapping describes the process of “peeling” or “flattening” the 3D surface of a polygonal mesh so that all faces can lay flat and fit into a square area, while minimizing stretching and distortions. Once each triangle of the original mesh has its flat conceptual counterpart, then the mesh, no matter how complex, can be fully textured using a single texture.

In cases of highly angular architectural models or other types of ‘hard-surface’ geometries, the unwrapping can be reliably performed in an automatic way inside 3D software packages, or even then game-engine itself, and works by conceptually ‘splitting’ (the geometries themselves are not altered) the geometries either at their sharp creases, or depending on the direction they are facing, and afterwards flattening the resulting ‘UV islands’ inside the square 2D texture space.

In cases of more organic-looking geometries, with complex topologies and large variations of curvature and/or doubly-curved regions, the automatic, crease- or direction-based splitting might not always produce the desirable texture continuity across the surface in question (generating the so-called ‘texture seams’ in highly visible areas). In these cases, the UV author has to manually define where the ‘seams’ should really be, by graphically picking the borders of the conceptual UV islands. The UV Unwrapping algorithm will then attempt to split and flatten the faces while respecting the user defined seams.

UV Unwrapping is typically the first step of an automated UV mapping procedure. It is far from trivial to implement programmatically, but it can be skipped entirely if a geometry enters the application’s runtime with pre-defined UV maps from the asset preparation stage. Therefore, good quality geometry with clean UV maps is always preferred.

In cases where such a geometry is not possible to have beforehand, a high-quality C++ library handling this stage is recommended. Suggestions:

- Rizom UV
- xAtlas
- UV-Packer

UV Packing

UV packing corresponds to the procedure of fitting the individual UV islands as closely as possible, so that they achieve maximum coverage of the underlying texture space. Consider the following image, courtesy of the commercial C++ library UVPackmaster:

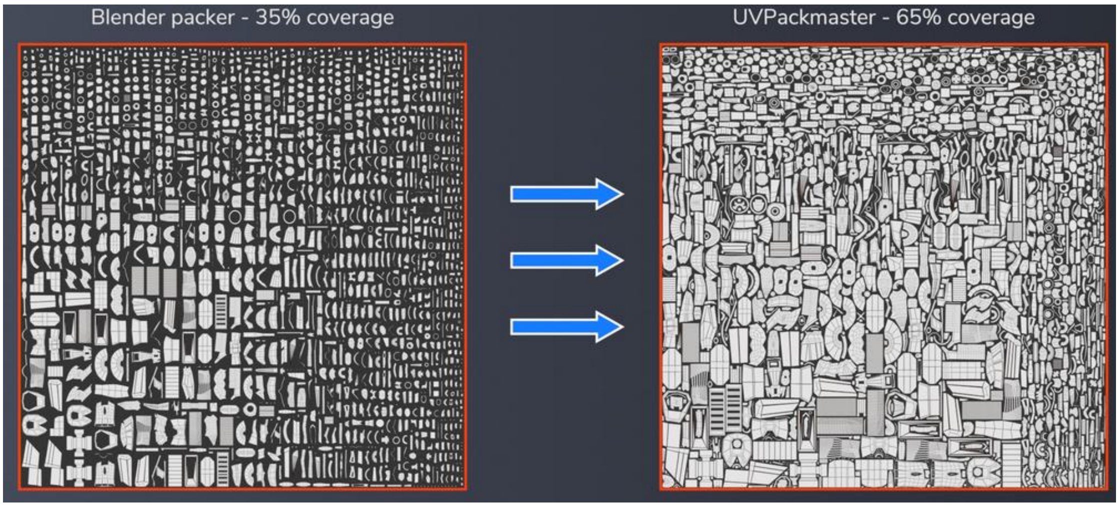


Figure 2.3.3b - Comparison between 2 different UV Packing algorithms

A denser packing, leading to better utilization of the texture space. In simple words, objects cover larger areas of their assigned textures, which results in higher texture detail of the final rendered object.

UV Packing is typically the second step of an automated UV mapping procedure, coming after UV Unwrapping. Like UV Unwrapping, it can be skipped entirely if a geometry enters the application's runtime with pre-defined UV maps generated during the asset preparation/modelling/texturing stage.

Otherwise, a high-quality C++ library handling this stage is recommended, since the problem of optimal UV packing is far from trivial. Suggestions:

- UVPackmaster 2
- Rizom UV
- xAtlas
- UV-Packer

Correct Relative Scale of UV Maps

In architectural visualizations, more often than not, the textures displayed on 3d objects correspond to physical entities with very specific dimensions, for example bricks, wooden planks, the specific grade of gravel, etc. Extremely common is also the re-use of a specific texture by multiple different objects, not necessarily of similar size, for example a really long concrete wall could share the same texture as a thin column. A problem that arises from the above two facts is the incorrect relative scale of the UV Maps of different objects sharing the same texture.

Imagine a simple case of *two rectangular meshes*:

One mesh is supposed to be 2m x 2m, and the other is supposed to be 10m x 10m. We desire to texture the 2 meshes using the same texture, let's say a square photograph of a concrete wall. The photograph/texture captures exactly an area of 2m x 2m of the original, physical wall that was photographed. Now, suppose that the 2 meshes have identical UVs, each one spanning to the extents of the square texture space. If we were to apply the texture to both *meshes, the first mesh would look correct, because the texture's 2m x 2m information would fit in a geometry that's 2m x 2m in 3D space. However, the second mesh would display an incorrect, greatly stretched texture, since the texture's 2m x 2m extents would be scaled to a 10m x 10m area. One solution to this problem would be to scale the UV coordinates of the second mesh and make them 5 times larger, so that the texture would occupy only 1/5th of the surface on each dimension. This would of course introduce texture tiling, which might or might not be desirable, but at least the apparent scale of the texture would be correct.*

What the aforementioned problem implies is that, at least for architectural visualizations, the UV coordinates of an object should be authored with consideration to the texture that is going to be applied to it, the theoretical 'real-world' size that the texture covers, and the object's dimensions.

Given that a PrismTexture object representing an architectural material can (and should) include information about its 'real-world' coverage, and that we can always query the largest / smallest side of a 3D object programmatically, what is suggested is an algorithmic procedure that scales the UV map of each 3D object according to the PrismTexture that is applied to it at any moment.

Application of a new texture during runtime, should trigger a rapid on-the-fly scaling of the object's UV coordinates.

UV Projections

There are cases where the manual / semi-automatic generation of UV maps is impractical, sub-optimal or simply unnecessary, depending on the geometric properties of the object to be textured, and the type of texture that is required. In these cases, a procedural mapping is preferred over an explicit one.

Such a 'UV Map-less' texture mapping can be achieved by utilizing one of many geometric projections, very much like the process of geographical projections which mathematically map the surface of ellipsoids (i.e. the Earth) into 2D maps.

Common uses of such procedural mapping are tileable, stochastic and near-stochastic textures, such as textures resembling random noise when viewed from a distance and typically natural-looking materials without precise geometric shapes. Such materials include gravel, sand, soil, scratched metal and other kinds of tear / wear, in-situ molded materials like concrete, fabrics etc.

Planar

This projection is similar in concept to a movie projector, but the associated image is projected onto the surface orthographically, meaning the projection rays travel perpendicular from the virtual projection plane onto the surface. This projection type is great for flat or nearly flat surfaces. Other instances are likely to cause undesirable stretching of the texture. You can specify the axis direction, either X, Y, or Z, that you want the projection to face.

Unreal Engine provides this functionality with the *LocalAlignedTexture* and *WorldAlignedTexture* nodes.

Cubic

Similar to planar, but the texture is planar-projected through a surface from all three axial directions, X, Y, and Z. A polygon receives a certain projection, based on its normal direction. This projection type is best on cube-shaped objects, and occasionally on detailed surfaces where texture seams are not of great concern. Unreal Engine provides this functionality with the *WorldCoordinate3Way* Node.

Cylindrical

The texture image is warped into a cylindrical shape and projected onto the surface. Very useful in texturing cylindrical shapes, for example, when labeling on various cans and bottles. Unreal Engine provides this functionality with the *Cylindrical UVs* node.

Spherical

The texture image is warped into a spherical shape and projected onto a surface. This is useful in texturing round objects, for example, planets and sports balls. Spherical projections are also useful in mapping 360° panoramic images for use as environment maps. Surfaces perpendicular to the projection produce undesirable stretching of the texture

Tri-Planar Mapping

In the last decade, tri-planar mapping has proved to be a versatile, high-quality technique for mapping textures on arbitrary geometries without UV maps, while avoiding the stretching & seaming problems of the classic projection types.

Tri-Planar mapping is similar in principle to Cubic mapping, but is way more sophisticated regarding the smooth blending of the texture projections coming from different directions. The end result tends to look like a uniform texture wrapping around the surface, with no visible start-end seams. This technique works best with stochastic, natural or noisy looking textures, although if implemented correctly it can produce adequate results with more technical, geometric textures. Ben Golus provides an excellent guide on the correct implementation of Tri-Planar mapping, applicable to any game-engine.

An example of how the Tri-Planar method can be built as a Material Function in Unreal Engine, is demonstrated here.

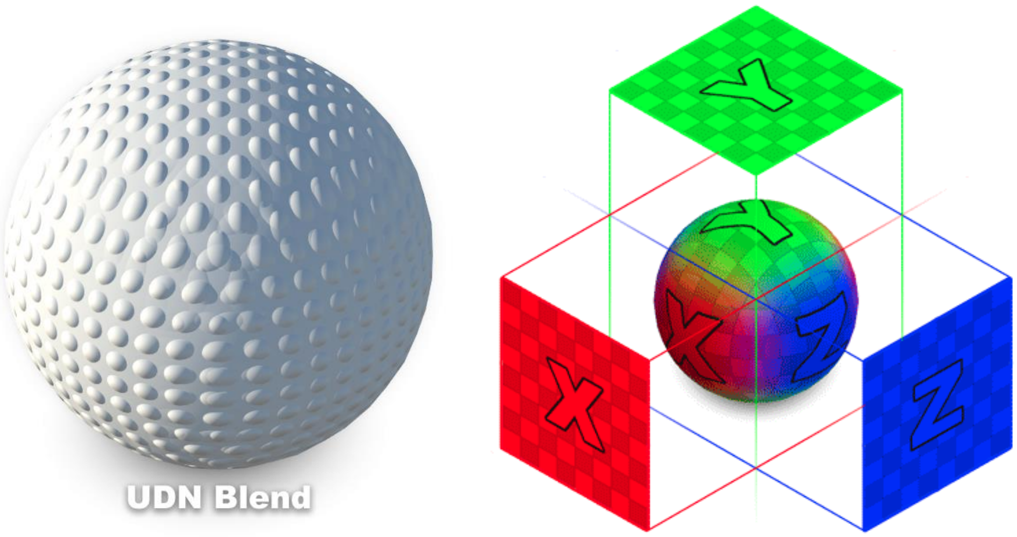


Figure 2.3.3c - Tri-Planar mapping result (left) and concept (right) (Ben Golus)

Procedural Stochastic Texturing by Tiling and Blending

A notable, high-end mapping technique that was first presented in 2018 by Heitz and Deliot is called Procedural Stochastic Textures by Tiling and Blending [Deliot and Heitz, 2018]. The algorithm is a *'simple tiling-and-blending scheme augmented by a novel histogram-preserving blending operator that prevents the visual artifacts caused by linear blending'*[Deliot and Heitz, 2018] . It works by sampling the original texture with hexagonal tiles and then making sure that each point on the textured surface is colorized by a blend of three tiles, as demonstrated here:

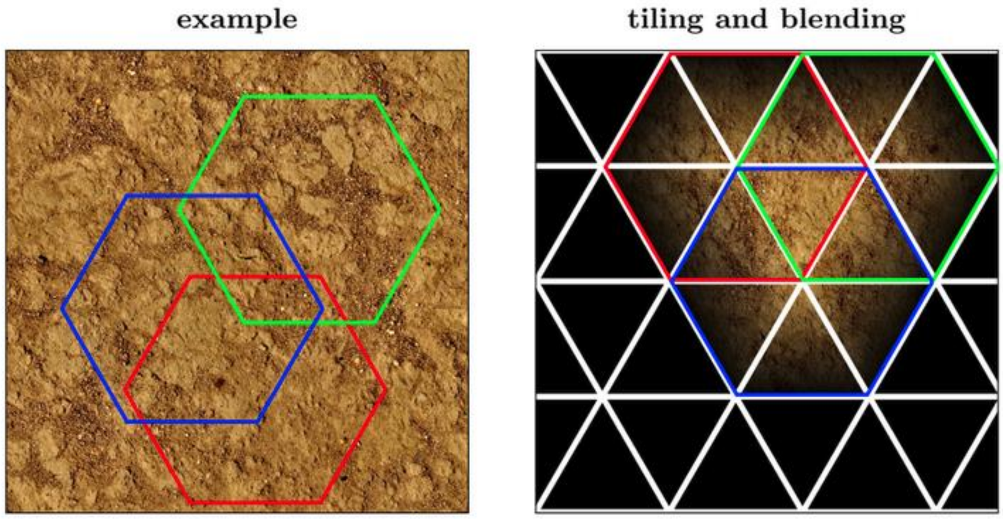


Figure 2.3.3d - Procedural stochastic texturing. Hexagonal "stamp" selection (left) and blending (right) [Deliot and Heitz, 2018]

This technique produces excellent, state-of-the-art results even on large surfaces, without signs of texture repetition. It works ideally if the input texture is of stochastic /noise-like nature, like rock or gravel textures, but fails miserably on inputs that present a strong pattern-like organization, like brick-walls.

Unreal Engine’s (4.26 and higher) *Texture Variation Node* produces similar results out-of-the-box, although with somewhat inferior blending of the randomized patches [Unreal TextureVariation].

Our proposed minimal specification for a Prism_Texture schema is detailed in the later Specification Section (2.4.6).

■ 2.3.4 VR Optimisation

2.3.4.1 Asset Optimisation

Mesh Merging / Batching

In a modern render pipeline, such as the one implemented in Unreal 4, the CPU carries out a significant workload, before the GPU takes over: working out which lights affect each object, setting up the shader and shader parameters, and sending drawing commands to the graphics driver, which then prepares the commands to be sent off to the graphics card.

All this “per object” CPU usage is resource-intensive, and in the case of a significant number of visible objects, it can add up. For example, if a model contains a thousand triangles, it is much easier on the CPU if they are all in **one mesh**, rather than in one mesh per triangle (adding up to 1000 meshes). The cost of both scenarios on the GPU is very similar, but the work done by the CPU to render a thousand objects (instead of one) is significantly higher.

Thus, a very common technique for reducing the amount of work the CPU needs to do, is the reduction of the visible object count. We can achieve that in three complementary ways. By:

- Combining close objects together, by ‘merging’ their meshes.
- Using fewer materials in our objects by putting separate textures into a larger texture atlas.
- Using fewer things that cause objects to be rendered multiple times (such as reflections, shadows and per-pixel lights).

An important consideration for the merging of objects is that only one *Material* has to be used for the entire mesh. Note that combining two objects which don’t share a material does not give any performance increase at all. The most common reason for requiring multiple materials is that two meshes don’t share the same textures; therefore to optimize CPU performance, asset designers need to ensure that any objects they combine share the same textures. In large models with complex assemblies and a lot of materials, the preparation of a single atlas for each merged mesh can be a very time-consuming process for the modellers.

Drawbacks of Mesh Merging

While the merging of close-by objects in singular polygonal meshes is a great way to reduce draw-calls and consequently greatly increase the performance of massive models, this technique poses significant drawbacks in terms of object manipulation flexibility. The core issue is that such a consolidation of individual objects into larger monolithic display assets, would immediately lead to individual objects losing their ontological autonomy, including but not limited to:

- Ability to be picked / selected / transformed by the user.
- Ability to be queried about their properties.
- Ability to be edited and saved.
- Ability to become hidden / colorized according to selection criteria or filters .

For example, a user might wish to see only the walls of the first floor of a building, all the slabs, and just a specific door in front of them. Afterwards, they might reset the filters, so that the whole building is visible, or filter out everything but a single element. Furthermore, they might decide to select a piece of geometry, then change its position and its material, only to completely delete it afterwards.

Such a de-coupling of display meshes from the data model of the objects they represent, means that an extra layer needs to exist between them, in order to keep them in sync.

Potential Solutions

Technical solutions to the above problem can certainly be devised, although their maintainability, scalability, performance and overall user-friendliness are subject to further research and not examined in this document. Nonetheless, such solutions should at the very least implement an encoding of each object’s ID *as well as its various categorization tags* onto the final merged mesh’s vertices, as arbitrary vertex attributes. This would ensure that any part of the mesh, if queried, could link back to the data model of the original object owning the queried point/triangle.

If a user wished to edit an object that is part of a merged mesh, then that model would need to be loaded temporarily as a separate entity, until the user’s operations are completed. The edited model would then need to be injected back into the merged mesh, replacing its previous triangles.

With regards to filtering geometries based on multiple selection criteria, a set of look-up tables / data textures would need to be maintained and updated in real-time, according to the selection criteria. All the objects would need custom shaders which would use each vertex’s attributes to look-up whether this part of the geometry should be shaded depending on the state of the look-up tables.

Another possible solution would be the on-the-fly merging and splitting of display meshes, based on the user’s desired action. Beyond the logistical nightmare of deciding which meshes should be combined and split, the performance and memory allocation costs of such a technique would need to be evaluated for VR frame-rates.

Unreal Engine’s ‘Nanite’ Virtualized Geometry System

Finally, Unreal’s recently announced Nanite system for the virtualization of mesh geometries appears to be a very promising solution for efficiently rendering hundreds of millions of polygons in interactive frame-rates, eliminating the need for techniques such as mesh merging and the manual authoring of multiple levels of detail for each mesh. Nanite, marked as Early Access at the time of writing, is responsible for procedurally generating a hierarchy of triangle clusters at multiple levels of polygon reduction, and then choosing the appropriate clusters to render depending on their visibility and distance from the camera. Geometries that are far from the user can afford to be rendered with only a few triangles, while the original high-resolution triangles are selected for parts that are very close to the user. That way, the total amount of triangles and the total number of draw calls per scene is kept more or less constant.

The investment in an early access technology carries significant risks, which should be weighed against the technical debt of the other potential solutions mentioned above.

Minimizing Polygon Count

Regarding the actual geometry of a model, there are two main optimization actions that should be considered by the modellers.

- The minimization of triangle count.
- The minimization of UV mapping seams and hard edges (doubled-up vertices).

Note that the actual number of vertices that graphics hardware has to process is usually not the same as the number reported by a 3D application. Modeling applications usually display the number of distinct corner points that make up a model (known as the geometric vertex count). For a graphics card, however, some geometric vertices need to be split into two or more logical vertices for rendering purposes. A vertex must be split if it has multiple normals, UV coordinates, or vertex colors.

Normal Flipping

An extremely common problem in workflows transferring CAD data to game engines, is the occasional incorrect orientation of surface normals. This problem is so notoriously hard to solve automatically for every possible case that it's usually deemed more practical to solve it manually, by bringing the model back into the 3d modelling package, identifying the flipped faces and manually applying a re-orientation command. A way to bypass this problem is through the use of double-faced shaders in the game-engine, sacrificing runtime performance (since the faces have to be rasterized twice) for workflow seamlessness. Otherwise, either a high-end normal-flipping SDK has to be integrated in the engine, or a Quality Assurance pass has to be completed before the models are officially registered in Sphere Level 2, 3 or 4.

Geometric LOD (Levels of Detail)

A widespread, decades-old technique that aims to reduce the amount of polygons being drawn without sacrificing detail is LOD, or Levels of Detail. Each entity is associated with a list of possible geometric representations of decreasing quality / fidelity, that are automatically swapped at render time depending on the virtual camera's distance from the object in question. This technique is prevalent in the visualization of forests or other scenarios where there are millions of elements to be drawn, the majority of which are very far from the user / player.

The drawback of this technique is that for each object to be displayed, the artist / modeller has to prepare more than one asset, increasing iteration times and making even simple changes inefficient.

That being said, a possible list of LODs is the following:

- **LOD 4:** Billboards or Impostors.
- **LOD 3:** Bounding Box / Convex Hull / Rough Voxelization.
- **LOD 2**: Polygon Reduction: Low-Quality.
- **LOD 1**: Polygon Reduction: High-Quality.
- **LOD 0**: Original CAD Geometry.

The above levels of detail (1-4) could in theory be generated automatically on model export, or even on-the-fly when a model is loaded in the game engine, thus reducing the pressure on asset designers / creators. Generative Adversarial Networks could be trained to generate models of minimum perceptual loss compared to the original, while trying to conform to a maximum triangle budget/ceiling.

Whether this extra geometric information is included in the model description or generated in-engine will determine whether the Data Model of Prism Objects (shared via Speckle) should make provisions for them. At this juncture, we believe that the most effective strategy for determining whether any of the asset optimisation approaches described above are necessary within PrismArch, is to carry out several tests of importing different AEC models into the PrismArch VR space. These models must encompass a range of detail levels, model scale, with and without element texturing, and so on. These tests should be undertaken as part of D5.2 - "first prototype of the VR-aided design platform".

Lightmap Pre-Baking

In situations where real-time raytracing is not possible due to performance / hardware constraints, all light interactions such as Soft Shadows, Global Illumination and Ambient Occlusion can be pre-computed and stored in 2D Textures called Light Maps, which are sampled in render-time and blended with the objects' assigned materials / textures, thus replicating the effect that light would have in the scene, without the cost of recalculating the ray bounces in every frame. A light map usually contains the lighting information of a big number of objects in the scene, and each object contains information about where exactly it should sample the light map in order to obtain its relevant lighting data.

The lightmapping stage itself has to be performed off-line, in the engine itself or through the use of a 3rd party rendering software able to produce light maps, but it is certainly not a procedure that can happen on-the-fly, since for large models it can take from minutes to hours to compute. See Unreal's *Lightmass* feature. [Unreal Lightmass]

Modern game engines provide ways to bypass the need for light maps, by using real-time Global Illumination which provides less accurate, but real-time results. However, these techniques cannot work with dynamically inserted / generated geometries, because they require a pre-processing step as well. See Unreal's *Lumen* feature. [Unreal Lumen]

If a client or user utilizes Nvidia RTX hardware, then photorealistic, real-time raytracing (including recursive reflections and refractions) is possible, which eliminates the need for any pre-processing stage.

2.3.4.2 Render-time Optimisation

GPU Instancing

Objects sharing the same *geometry* irrespective of differences in position, rotation or scale, and the same *shader properties* (commonly referred to as 'material') can be *instanced*.

What this means is that each one of these objects only needs to refer to a shared, singular geometry object in the system's memory, and all of them can be drawn using a single draw-call, as if they were a single object.

This technique, ubiquitous in computer graphics, follows the *Flyweight* [Nystrom, 2014] programming pattern, and reduces both the runtime memory footprint of the game/programme, since only one mesh can serve millions of instances, but also the runtime performance, since the CPU has to issue significantly less draw-calls to the GPU, a notoriously 'slow' operation.

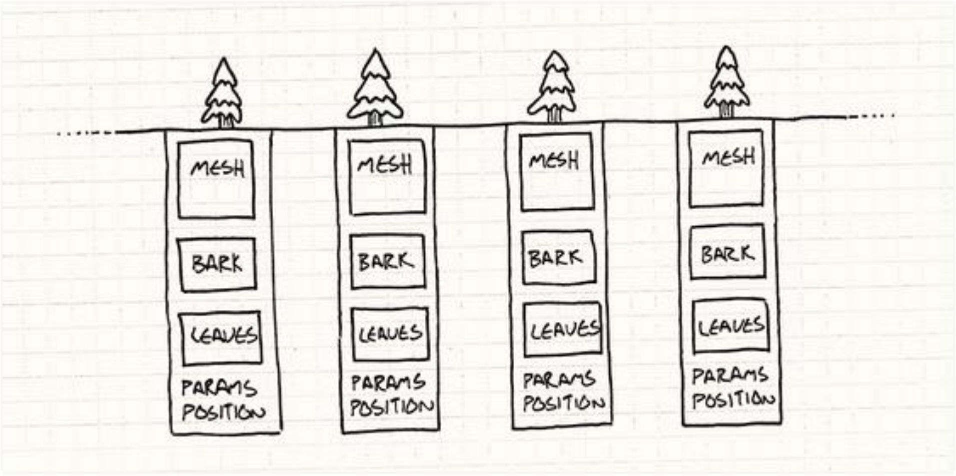


Figure 2.3.4.2a - Objects *without* instancing [Nystrom, 2014]

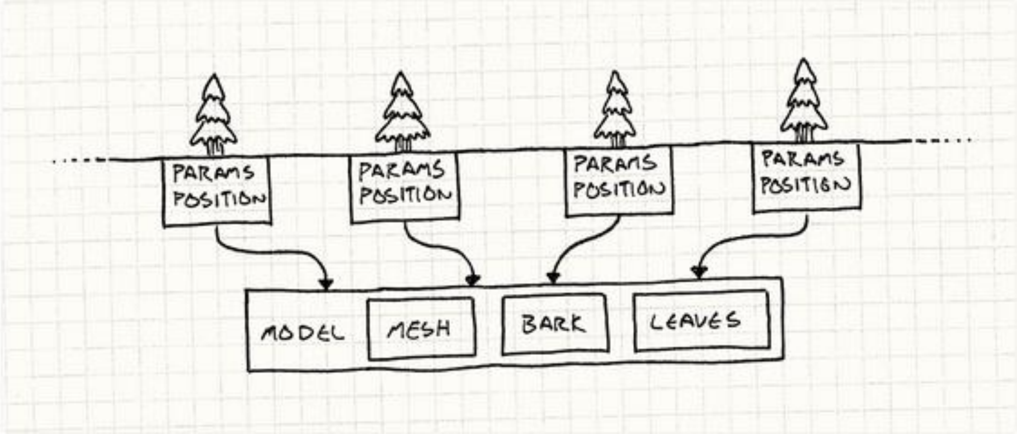


Figure 2.3.4.2b - Objects *with* instancing [Nystrom, 2014]

This technique has been known to CAD software for decades (commonly referred to as *'Blocks'* or *'Xrefs'*) since it suits reasonably well the types of geometries one commonly encounters in architecture and engineering: identical transformed copies of complex elements and assemblies (facade units, columns, staircases, detailing etc).

Problems arise when CAD models designed in a specific software package making heavy use of instancing are imported in another software which does not retain the instancing information. This problem used to be extremely common and a huge point of frustration in the architectural visualization industry during the last two decades, where heavily instanced

CAD models provided by architecture studios, were often naively converted by the importers used in visualization studios into near-impossible to work with plain polygonal meshes, that discarded any instancing information.

The problem can be exemplified quite easily:

Imagine a polygonal mesh -let's say a detailed representation of a building containing trees, facade elements, digital humans etc- which contains 100 MegaBytes worth of information. Now imagine that a designer decides to create 40 identical copies of that building, by simply duplicating the original geometry (no instancing). The total amount of memory required would be:

40 x 100 MB = ***4 GB***

easily saturating the Video RAM of a middle-range graphics card in 2021. If the designer chooses to instance the geometry instead, given that a 4x4 transformation matrix defining the position, scale & rotation of an instance weighs typically 64 bytes, and arbitrarily allowing an extra 960 bytes per-instance for extra information (reference to the original object, instance ID, custom per-instance data), then the memory footprint becomes:

100MB + 40 x 1MB = ***140 MB***

Without describing explicitly what this enormous difference means for the various parts of a rendering pipeline, it becomes immediately clear that Instancing alone can provide one of the most significant - if not the single most important - performance gains in a geometry visualization environment.

Therefore, any effort to design a high-performance, scalable geometry translator of CAD files to RT3D (real-time 3D) engines, should make sure to provide full support for reading Instanced Geometry and reconstructing it using the engine's instancing method. It follows that any Interchange Format / File Specification should provide full support for describing Instanced Objects.

Instancing is nowadays supported in every major 3D software package and file exchange format. Some notable examples, relevant to the present research programme include:

- Speckle Blocks
- Unreal Instanced Static Meshes
- FBX Instances
- USD Instances
- Rhino3D Blocks
- Maya Instances
- Sketchup Components
- Unity GPU Instancing

Further Reading in Unreal Art Optimization: GPU & Rendering Pipelines

- **Speckle Blocks**

  Since April 14 2021, Speckle 2.0 beta supports Blocks coming from Rhino and AutoCAD [Github commit]. As part of the same commit, Speckle's Unreal engine connector, currently in Beta, appears to support importing of instanced geometries by re-using objects it has already imported, when needed. [Github code]. A brief introduction to how the Speckle Block system works can be found at the [official blog].

- **Unreal Engine Mesh Drawing Pipeline**

  An advanced description of how Unreal's mesh drawing pipeline works, including the setup of Draw Call Batching, can be found at the [official documentation].

As with the prior *Asset Optimisation* section, we believe that testing must be carried out within the PrismArch VR platform to determine the necessity of including any *Render-Time Optimisation* strategies. These tests should be undertaken as part of D5.2 - "first prototype of the VR-aided design platform".

■ 2.3.5 Quality Diversity and Designer Modeling in PrismArch

Within this section we will provide an outline of the basic Artificial Intelligence functionality that we intend to provide within PrismArch. Having described this, we will then detail the type of schemas (here referred to as 'data structures') that must be implemented, to allow these processes to occur.

2.3.5.1 Functionality

As presented in [D2.1], Artificial intelligence in PrismArch aims to provide an assistive technology driven by the user in order to edit items collectively and in an informed manner. As described in Prismarch DoW, this assistive technology is envisioned through the algorithms of Quality-Diversity Search (QD) and Designer Modeling (DM).

D2.1 highlighted that MAP-Elites [Mouret and Clune, 2015] is considered the most suitable QD algorithm for the problem at hand, due to its operational simplicity and its relatively small number of hyper-parameters. Some existing applications of Map-Elites to design-related problems can be found in the works of [Gaier et al. 2018], [Sfikas et al. 2021], [Galanos et al. 2021].

As far as designer modeling is concerned, there are a variety of algorithmic approaches that can be applied to the available data. One of the earliest relevant studies [Liapis et al. 2014] showcases a design tool that incorporates computer generated suggestions which appeal to the human user. Other relevant approaches are also available, including Design Style Clustering, as shown in the work of [Alvarez et al. 2020], or using a Designer Preference Model for driving Evolution, as shown in the work of [Alvarez and Font, 2020].

The following subsections start by dealing with the problem representation in the context of QD search and the process of generating solutions. They then move on to explaining the process of user interaction with the QD system, as well as the process of data collection that can be used to support the DM process. Interdisciplinary constraints are also discussed as an important part in the context of QD. Finally, the last subsection discusses issues related to data formats related to both QD and DM.

Problem Representation

In the context of AI assistive design for PrismArch, we can define the problem representation as an abstract description of the solution that captures a number of topological characteristics such as, for example, connections between specific rooms, as well as a number of other important features, such as the target surface area of each space unit, but without specifically describing how those features are to be attained. Alternatively, the problem representation can be viewed as a set of hard constraints which differentiate solutions into feasible (the ones that satisfy them) or infeasible (the ones that do not satisfy them). By incorporating this approach, we consider that designers themselves should define the problem and negotiate it with a potential client, by analyzing the problem definition itself or potential concrete solutions that it produces.

QD System Processes

There are many approaches for the generation and transformation of solutions with a chosen representation. On one end of the spectrum, one could apply a completely random generation and mutation which would generate results with a very small chance of being feasible (i.e. satisfying at least the hard constraints). On the other end of the spectrum, one can design specially crafted generation and mutation methods that guarantee feasible solutions with the caveat of more computationally intensive processes and less variety in results. Our chosen approach lies somewhere in the middle of those edge cases, as an attempt to get the best of both.

The generation process starts by subdividing the space randomly. It then utilizes the generated regions (assigns them with a specific function) while considering some connectivity constraints and finally adds more specific details such as placing doors and windows. The mutation process is a bit more complex. It operates in a hierarchical fashion and can occur in one of three ways: mutating the spatial subdivision (which may affect the cells' utilization and placement of doors and windows), mutating the regions' utilization (which may affect the placement of doors and windows) or affecting the placement of doors and windows (which does not affect anything else). Both the generation and the mutation processes do not guarantee feasible solutions. The discovery of feasible solutions is ensured through the broader operation of the evolutionary algorithm.

Finally, the evaluation of the evolving content, as well as aspects of the genetic operators and initialization strategies, are dependent on an interim connectivity graph or adjacency matrix. As highlighted in [Section 3.1, D2.1], connectivity graphs are vital simplifications of the design problem and have often been used to assess the quality of floor plans in terms of proximity and doorways between rooms. As noted in [Section 2.1.9, D2.1], usually such connectivity graphs are provided by the client and are not open to negotiation; this is the assumption for the current prototype of PrismArch's AI algorithms. However, the graph itself could be evolved as discussed in [Section 2.1.3, D2.1] as a step prior to spatial tessellation. Some constraints on what could be evolved and the ability for human control over aspects of the graph (e.g. not allowing the removal of certain nodes/rooms or edges/connections) would be necessary in this case. These constraints can easily be encoded in the genetic operators, for instance by blocking mutations for nodes marked as "frozen" [Galanos et al. 2021].

Interfaces Definition for Cross-Disciplinary QD Assistance

This section presents several ways that a designer could become more involved in the design process, apart from describing the problem and the algorithm settings and receiving the results at the end.

Direct interaction with the solution representation: There are several ways in which the user may directly interact with the solution representation, in the context of the QD assistive system. First of all, they may generate their own solutions without even utilizing the evolutionary algorithm at all. Alternatively, they may choose to directly modify an existing solution generated by the system. While generating or modifying solutions, a designer can receive all the available "analytics", as described in [section 3.2 and 4.2, D2.1], during their design process and thus make more informed decisions or observations about their designs. Furthermore, user-generated designs can be used as the initial population of the evolutionary algorithm, thus providing a potentially better starting point for evolution instead of randomly generated solutions. Alternatively, a designer may interfere with the generative process by imposing specific restrictions on what parts of the solution representation the algorithm can modify.

Finally, the designer may even revisit the problem definition, after having examined a set of solutions. Slightly changing the problem definition can either be treated as part of a completely new process of search or, alternatively, as part of a continuous process of search that does not discard the previously generated solutions, despite the fact that they were generated through different feasibility criteria. Changing the problem definition is likely to cause infeasibilities in the existing set of solutions, however the evolutionary process (through mutation and selection) should be able to gradually correct the solutions, based on the new constraints.

Indirect interaction with the solution representation: Apart from a direct involvement of the user with the design process, their interaction may also be indirect. In this case the user does not actively alter or constrain the solution space, but is guiding the evolutionary process through their preferences.

The first way of doing so is through interactive evolution [Takagi 2001]. In this case, the user is presented with a set of possible solutions and asked to choose the ones that they prefer. Their choice may be based on any type of subjective criterion, or take into account the analytics and measurements that the system provides for each one of them. As soon as the user selects a subset of solutions, they are treated as the initial population for the generation of the next set. This way, the user's preference is directly treated as a form of fitness function, in the algorithm's operation.

In line with the work planned in WP2, the most ambitious and general way in which the designer can indirectly control the solution representation is through a designer model. The user's preferences may be captured in the form of a designer model [Liapis et al. 2013], via supervised machine learning (or, potentially, unsupervised learning when clustering groups of designers). After such models have been generated, they can be utilized as fitnesses, constraints, or custom representations [Liapis et al. 2014] for the evolutionary algorithm that adjusts various qualities of the generated content towards a specific style, or set of preferences that mimic the style of a designer persona.

Data Collection for Designer Modeling

Apart from the intrinsic aspects of the Quality Diversity assistive system, the algorithm will also exist within a specific context of operation, where human designers interact with it, controlling some or all of its parameters and interfering with its operation in various ways.

The data-records of this human-AI interaction will form the basis for the generation of designer models which can represent (predict, reproduce) the behaviour and/or preferences of the designers that have used the system for a period of time. Data collection will adhere to the overall data organization and ontology (see Section 2.3.2) of the PrismArch application, respecting issues of intellectual property and providing access to the data only to their respective owners. Any potential aggregation of data in the form of analytics or in the context of designer modeling that may depend on data of mixed ownership will have to be agreed upon between the respective owners.

As described in [Section 2.2.4, D2.1], there are several ways in which the designer can interact with the solution space and specifications, which also influences the type of data that can be collected regarding this interaction.

At its most basic, data collection will be in the form of timestamped events that describe instances of actions of specific subjects within the design space. In other words, every event describes “who did what, when”. For example, if a user is directly creating an asset through the design interface of PrismArch, the system may be automatically collecting all of the actions that a designer takes while designing a “solution” from scratch. Depending on the context of use, it is possible that this approach can generate an unnecessary amount of data. In this case, the data collection can occur in predefined time-intervals, so as to reduce their volume but still retain a compressed (lossy) overview of the design process. This mode of data collection could be then used in order to train neural networks that can “mimic” the designer’s behavior, i.e. design in a style that resembles that of a specific designer or a set of different designers.

In the second mode of interaction, instead of the hands-on design activity, the recorded data could include the designer’s preferences. In other words, the data collection could keep track of which solutions (out of a larger set of solutions) the designer found to be preferable. This mode of data collection could be used in order to model a specific designer’s “preference”, i.e. a model that can predict the selections that a specific designer would make. As soon as such a model is trained, it can then be “embedded” in the algorithm’s operation, effectively generating a hybrid AI that mimics some aspects of human preferences.

Cross-Discipline Principles, Rules, Constraints of QD and DM components

Interdisciplinary constraints are an aspect of the design process that is especially important in the context of large-scale design projects. The examples of evolutionary design that can be found in the relevant literature [Gaier et al. 2018], [Sfikas et al. 2021], [Galanos et al. 2021], however, are usually focused on problems of relatively small scale or on problems of a larger scale but from a relatively narrow perspective (within the bounds of a single discipline). Feedback from the AEC industry partners of PrismArch on cross-discipline principles, rules, constraints was collected in preparation of D2.1, adding an important dimension to the literature review carried out.

As extensively described in [Section 3.2.1, D2.1], all AEC partners were able to recognize the fact that their design activity is constrained by external factors (including the activity of other disciplines), as well as the fact that their domain knowledge and responsibility introduce constraints that other disciplines must recognize and respect. Furthermore, all disciplines recognize the importance of effectively communicating those constraints throughout all the stages of the design process and, in some cases, a large emphasis is placed on the initial stages of design, where a number of important and high-level design decisions are made.

2.3.5.2 Data formats for DM and QD

The main prerequisite for performing any type of Designer Modeling is the access to data that describe the activity of designers in the context of PrismArch. An example of designer activity that could be a starting point is the designers’ interaction with elements of the VR space, such as selecting and deselecting 3D elements. Having access to the logs of such simple actions, accompanied by meta-data like time-stamps, user IDs and other meta-information about the selected objects could be a starting point for an initial prototype of Designer Modeling.

As far as QD is concerned, a special kind of data-structure is required. Some of the most important characteristics of this data-structure include the following: 1) the ability to represent a topological description of architectural designs, including the existence of discrete space units (rooms), their interrelations (connection, proximity) and other properties. 2) the ability to represent geometry in a way that corresponds to the current topological representation, i.e. a specific geometrical implementation of a given topological description. 3) The ability to transform (mutate) solutions holistically, affecting both topology and geometry and being able to inspect and control their proper relation. Such a data-structure has already been prototyped by the UM development team and is currently in a process of refinement, so as to be as flexible as possible and not tied to specific geometric or other constraints. Converting this data-structure to a typical 3D model format (such as an ".OBJ" geometry file, or a Rhino file ".3DS") can and will be undertaken by the UoM team. Converting it to a BIM file (for example Revit) may also be feasible through an intermediate file-format, possibly through the Speckle data processing layer.

■ 2.3.6 Auxiliary Data

A typical AEC project quite often needs to be complemented by a collection of extraneous data references beyond BIM geometries, such as image-based media (image and video files), audio recordings, text-based media (e-mails, reports), source code, datasets (binary data, excel tables, databases) and other arbitrary files. This concept has been previously outlined in several Deliverables, such as *D1.1 Section 3.1a "Case Study 1"* and *Deliverable 6.1 Section 1.1 d.3 "PrismArch Core Functionalities".*

The exact format of some of these external files can be reasonably predicted due to their widespread adoption and standardization. For example, .JPG, .PNG, .GIF and .TIFF formats cover the majority of the image file types that one would encounter in a typical AEC workflow.

It is fair to assume, however, that only a subset of all possible files that might be attached during the lifetime of a project can be predicted and accounted for in the initial design of the PrismArch Library.

Although it is impossible to cover the extent of all future additions, the principles of extensibility and scalability highlight the need to approach such auxiliary data in the future without major restructuring of the original class hierarchies.

Ideally, each new format addition should only require the definition of a new Parser, a new Renderer and a new Handler, in accordance with the Visitor pattern. These three objects are responsible for providing the new operations that should be applied on the file type in question.

- The **Parser** would take care of translating the binary data of the file in question, to the actual object structure it should represent. That external object would eventually be distilled into a known type or a collection of known types.

- The **Renderer** would take care of visually representing the object that the parser creates inside the 3D scene of the Prism VR application.

- The **Handler** would define all the user interactions that can be applied to the visual representation of that object.

When it comes to what file formats one could encounter, we could separate them in a few logical categories:

- **Image-based media**, typically compressed and encoded using some protocol. These could be static images, or video files, and can be represented as selectable flat geometries in the VR scene or as flat, full-frame imagery.
- **Audio**, typically compressed and encoded using some protocol. Audio files can exist in a 3D scene and be experienced spatially. In certain cases, the exact location of an audio file is crucial, as it might be related to acoustics analyses performed at exact locations.
- **Text-based media**, such as e-mails and simple text documents. These files don't typically correlate to a specific location in 3D space, and can be represented in VR through a full-frame text reader interface, with text-selection and text-editing capabilities.
- **PDF documents**, which due to the specificity & versatility of the format, can be considered a category of their own. They could be represented in a way similar to other text media, or through a dedicated PDF Viewer which could harness the multi-media aspects of pdf files.
- **1D, 2D or 3D datasets**, which could be represented using a variety of charts / graphs in the 3D scene, or as flat 2D plots.
- **Other scientific formats**, which are probably experienced best through dedicated, file-specific viewers, such photometric IES data, 5-dimensional reflectance data produced by gonio-photometers, or even X-Ray CT scans of structural elements.
- **Raw datasets** such as .csv or Excel files, which are best when represented as structured sheets / tables.

The generic data type that would hold these auxiliary data should be one that can be serialized to Speckle and attached to any Prism_Object.

○ 2.4 Core Foundation for Ontology

■ Overview

Now that all of the requirements are established, the following chapter will explore a series of schematic prototypes for how the final object could be structured. The core aim and questions raised regarding this set of proposals will be the previously discussed balance between *generic* and *explicit* formulations. Essentially, finding the right degree of interpolation between flexibility and predictability based on the reconfigurable paradigm of Speckle. The core question is: how rigid should the underlying PrismArch ontology be to achieve the required functionality, while still accommodating:

- *Efficient exchanges*
- *Queryability*
- *Interdisciplinarity*
- *Freedom of users to add custom user data*
- *Bottom-up emergence of object models*

In general software-design terms, we investigate here the right balance between *Inheritance* and *Composition* for Prism-objects. In the coming chapters, three different concepts will be proposed to examine the approach for how to enforce predictability on the objects. The proposals range from explicit Inheritance structures, where most data is stored within clearly defined properties of the objects, to objects whose description is composited purely from arbitrary user data. All of these proposals are formulated as extensions of Speckle objects and the study mainly focuses on how each approach works specifically in this context, and not necessarily in a general sense. It is supported by a set of code examples developed using the C# programming language and the Speckle.NET API -- see the [GitLab] code library.

The three Options proposed are:

Option 1 - Rigid Data
Objects are explicit, and defined using an *Inheritance*-based approach.

Option 2 - Rigid Components
An approach mixing *Inheritance* and *Composition*.
Objects are composable and specific data structures are encoded into rigid modules, which are appended to the objects user data.

Option 3 - Flexible Data
A purely *Composition*-based approach.
Objects rely mainly on the user specifying their own structures in order to tailor it to their needs.

All three options will be laid out in a schematic sense and explained regarding the way they would hold and enforce constraints on supplied data of various types - for example metadata or structural engineering-specific properties.

Key:

Red	PrismArch schema
Blue	Speckle schema
Green	Unstructured Data

■ 2.4.1 Option 1: *Rigid Data*

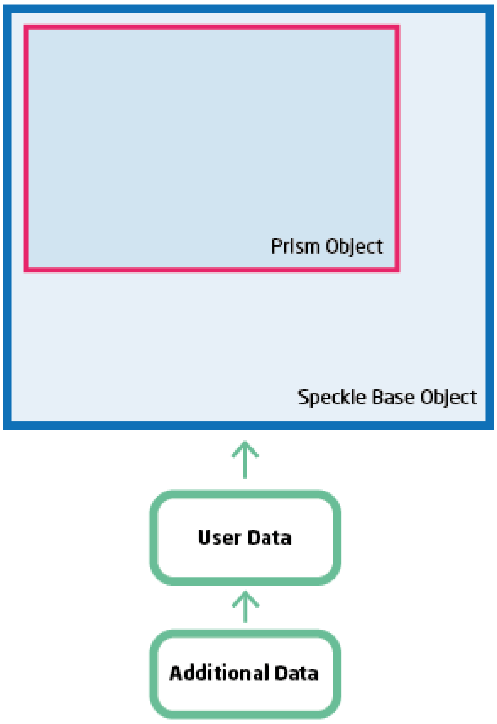


Figure 2.4.1a - Schematic diagram of Option 1 - an inheritance-based approach

Schematic Approach

The first approach uses a fairly conventional *inheritance* approach, where any new PrismArch schema is created by inheriting from a Speckle base object (or other 'higher level' Speckle objects). This provides objects with a high degree of predictability, by enforcing clear principles on how data should be structured through the object definition itself. This makes it clear for all users what an object does and how it expects input data. Additional user data can still be appended if necessary though, to complement the core functionality. These objects should not be sealed, and a hierarchy of PrismArch objects can be defined - similarly to how the IFC model is structured, where some generic layers are extended through the use of new objects intended for specific disciplines or purposes.

Speckle Integration

The creation of this type of object means inheriting from a Speckle base class in one or more levels, depending on the complexity of the objects. The example shows a generic PrismArch object which extends the Speckle Base object to establish the baseline for the PrismArch inheritance chain. This base object can be defined as the first extension of the Speckle object, which can serve as a container for all the specific prism properties presented in chapter 2.3.2 , which are shared across all objects.

Further, this base object will be extended with more specific objects which are needed in the PrismArch environment, such as objects to handle engineering entities with, for example analysis data, or MEP specific objects. Based on the type of the object the PrismArch platform will be able to interpret it and handle accordingly.

Summary

Advantages:

- Explicit structure of Objects ensures data is always ordered correctly.
- Easy for any platform implementation to be programmed to handle the objects, due to their explicit nature.
- By using inheritance to create frequently-needed PrismArch entities on top of the Speckle base, this option is conforming to Speckle's conceptual approach. Increases likelihood of schemas conforming to future Speckle updates.

Disadvantages:

- Potentially too rigid, if numerous discipline-specific schemas are generated - as has been seen in the IFC format. Must ensure that the minimal number of PrismArch schemas are created.
- Creating a PrismArch-specific base object (or adjusting the existing Speckle base object) to include the Prism_Signature and/ or Signature properties is not efficient, as it would necessitate the creation or alteration of numerous inheritance chains for existing 'multi-level' Speckle schemas (SpeckleBeam, SpeckleWall, etc). *In this Option there is no solution for how to achieve the Prism_Signature functions.*
- Potential versioning issues when deserializing complex objects made with older versions.
- Serialised objects *could* become larger - if PrismArch schemas inherit from multi-level Speckle entities - as potentially null or unassigned fields would be stored. Can be avoided by inheriting from the Speckle base object wherever possible.

■ 2.4.2 Option 2: *Rigid Components*

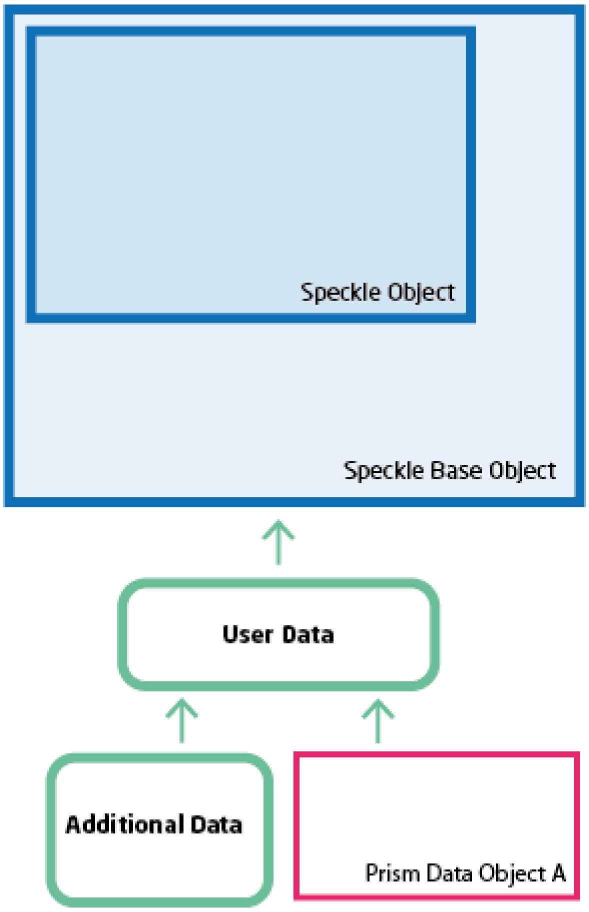


Figure 2.4.2a - Schematic diagram of Option 2 - rigid data structures as modules

Schematic Approach

The second approach is to enforce less structure through inherited objects, and instead provide this same conformity using ‘orphan’ (i.e. non-inheriting) objects. In this scenario we avoid the necessity to adjust every entity within large inheritance trees, and instead PrismArch data is contained within explicit data objects which are passed into the Speckle user data. Here, these objects are responsible for enforcing the correct format of their supplied data.

Because these entities are stored within the expandable Speckle user data attached to an object, it would be the responsibility of the PrismArch platform to search this unstructured user data ‘container’, looking for relevant entities to display/ search/ query, and determine how they should operate upon their parent object. In this scenario, entire object types can be generated on-the-fly just by combining pre-existing components, without the need for these objects to be defined at the software design stage. Importantly, the addition of one type does not require the re-compilation of the library, it can be a runtime operation.

The creation of modules can be delegated to the parties to which they are relevant, and modules that are irrelevant for any particular user can then be ignored, and redundant explicit properties will not complicate the general object in contexts where they are not present.

Speckle Integration

For this option, existing Speckle objects would require minimal or no adjustment, and the orphan PrismArch schemas would similarly be simple to generate.

How many user-data modules exist can emerge from the user base as the modification or addition of more modules does not interfere with other modules already present, as they all live separate from one another. A certain discipline can then develop them as they see fit, using a relevant Speckle object as a common denominator.

Summary

Advantages:

- Serialised objects remain lightweight, as only existing data is included.
- Prism_Signature can be successfully stored in this Option. Must be attached to user data, to avoid write-order issues with Speckle base object metadata.

Disadvantages:

- Potential versioning issues when deserializing complex objects made with older versions.
- Frequently-needed PrismArch entities are attached as seemingly ‘optional’ or unstructured user data to potentially empty (or unneeded) Speckle objects. Would be cleaner and more in line with the Speckle ontological philosophy to make these into standalone Speckle-inheriting classes.
- The creation of validation checks/ unit tests upon the PrismArch schemas is slightly more complicated than Option 1, as their presence would need to be determined within the flexible user data entity.

■ 2.4.3 Option 3: *Flexible Data Structures*

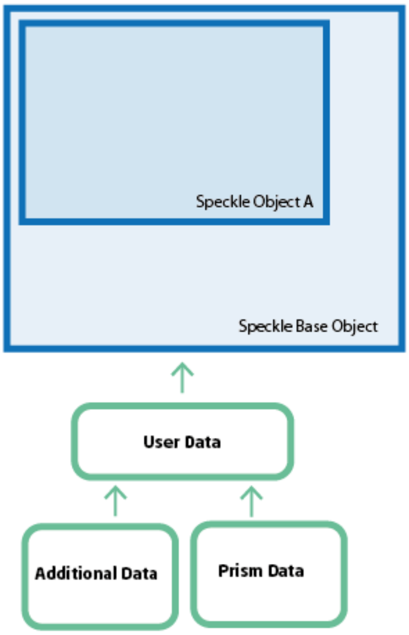


Figure 2.4.3a - Schematic diagram of Option 3 - an approach using custom data

Schematic Approach

The third Option represents a ‘minimal intervention’ approach which is completely flexible, due to the fact that it implements a pure composition pattern. Here, no specific formats are defined, but information is generated from sending sources based on ad-hoc defined user data structures, which eventually consist of a series of known “blittable” basic types and/ or arrays of them. This approach delegates all responsibility on how data is structured to each user, giving them complete freedom over what is or isn’t transmitted. This, however, forces the primary task of interpreting this data onto the consumer, as there is no way for the PrismArch platform to predict or know exactly what type of data it will receive. The PrismArch platform could be adapted to respond to this genericness and reduce user interpretation, however this is presumed highly difficult.

Speckle Integration

The implementation of this approach is very straightforward, as the predefined Speckle Object kits and user data functionality can be implemented ‘off-the-shelf’. PrismArch data is simply appended to objects just like any other user data, with no need for PrismArch-specific schemas. The platform would then be tasked with parsing and processing these data structures from the Speckle objects. Speckle’s approach of developing custom kits with custom structures and connectors could be used as it is as well, in case particular partners or disciplines would like to customize their objects.

Summary

Advantages:

- Maintains complete Speckle flexibility.
- Serialised objects remain lightweight, as only existing data is included.

- Very little development required, as no Wrapper functions or objects created to pass PrismArch objects.

Disadvantages:

- No explicit format for prismArch data types, making it highly likely that unstructured PrismArch data will be misinterpreted within the database/ VR environment/ etc. *This disadvantage alone is enough to render this Option unacceptable.*
- No indication by the ontology of which data is useful or connected.
- No ability to create validation checks/ unit tests upon the PrismArch data, as not contained within established schemas.

■ 2.4.4 Comparison and Selection

Comparison

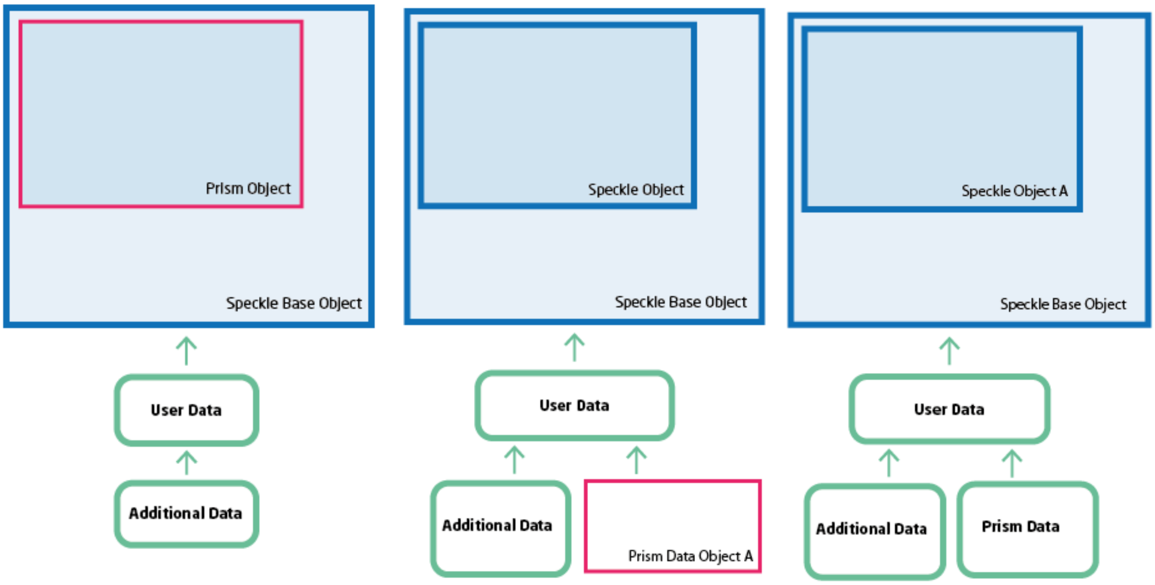


Figure 2.4.4a - From left to right, Options 1, 2 and 3.

Given the advantages and disadvantages listed in the previous chapter and the general requirements established in Chapter 2.3, the following conclusions can be drawn for each of the options:

- **<u>Option 1</u>** is a highly promising approach - however it must ensure that minimal hierarchy is applied to PrismArch schemas, to ensure it does not exhibit similar inflexibility and proliferation of data types as exemplified by IFC. The precedent study has demonstrated that this would be very problematic and would hinder uptake of the PrismArch platform. The inability to incorporate mandatory PrismSignature data is a problem.

- **Option 2** is a promising approach as it allows for both the ability for automated computer-based data processing together with the ability for users to flexibly extend and modify within some reasonable constraints. Critically, this option would successfully accept PrismSignature information - i.e. PrismArch metadata.

- **Option 3** is too unstructured to achieve the vital PrismArch functionalities previously agreed upon. Although it would make development of external software connectors extremely straightforward, it would also significantly increase the complexity of data interpretation and application, and push all of these difficulties onto operations occurring within the VR connector and environment. This in turn could potentially undermine the platform's real-time responsiveness. Lastly it would likely limit the level of automation possible on the platform.

2.4.5 Selected Approach

From the summaries above, it is clear that a hybrid of Options 1 and 2 would be the ideal. This new Option (4) is shown below:

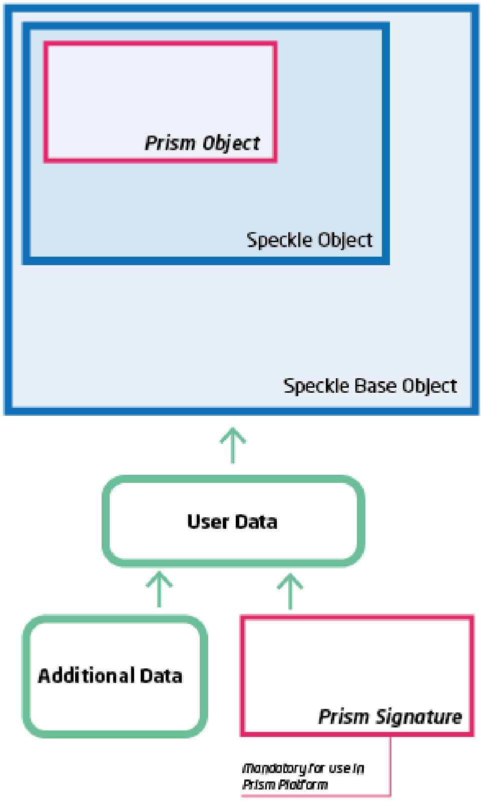


Figure 2.4.5a - Schematic diagram of new Option 4 - the selected approach

In this new Option, all PrismArch entities (with one exception) inherit from Speckle entities. The vast majority of them inherit from just the Speckle base object, thus ensuring they conform to Speckle Transporter and DB requirements, but avoiding the need for substantial rewrites to the constructors for multiple levels of inheritance chains.

The exception is of course the Prism_Signature entity, which will still exist as a clearly defined class, but which can be most easily stored within the user data.

Earlier studies considered expanding the Speckle base object with properties matching the Signature. While this would have been a 'conceptual pure' solution, it would also have necessitated the same types of significant and onerous rewrites to inherited classes, connectors just described - these must be avoided.

■ 2.4.6 Specifications for Core PrismArch Schemas

Prism_Signature

The following is a minimum specification for the 'Meta Data' or DNA object, that must provide authorship and attribution information for every asset produced:

Property	Type	Description
AssetId	GUID OR string OR CheckSum	Unique GUID or HASH string for this asset. Still TBD.
CreationDate	DateTime	
SphereLevel	int	1 - 5.
PersistenceLevel	enum	Selected from a pre-existing **PrismPersistenceLevel** type.
CreatorName	enum	
CreatorRole	enum	Selected from a pre-existing **PrismArchRole** type.
CompanyName	string	Must be empty if user is within SL1. Must be assigned for all other SLs.
Discipline	enum	Selected from a pre-existing **PrismDiscipline** type. e.g. *Architecture*
Sub-Discipline	enum	Selected from a pre-existing **PrismSubDiscipline** type. e.g. *Facade Studies*
Stage	string	Project Stage.
CreatedBySoftware	enum	Selected from a pre-existing **PrismSoftware** type.
CreatedFromFile	string	Text string of original file *Path* - thus includes original folder location, original file name and file extension.
DeviceName	string	Automatically read from the device: PC, laptop, etc.
DeviceId	string	Automatically read from the device.

Prism_Tag

Property	Type	Description

Name	string	The Tag text, examples: Status *"Draft", "In Progress", "For Review"* Flexible comments: *"Is this detail correct?"*

Prism_TagConnection

Property	Type	Description
ObjectId	GUID or Hash	The unique identifier of the Asset this Tag is assigned to. Refers to the **Asset_Id** property of the Asset's **Prism_Signature**.
TagId	GUID or Hash	The unique identifier of the **Prism_Tag** assigned. Refers to the **Asset_Id** property of the Tag's **Prism_Signature**.
Status	Enum	i.e. *Active*, *Superseded*, etc.

Prism_Texture

The following is a minimum specification of a Texture object suitable for the scope of PrismArch:

Property	Type (C++)	Size	Description
Id	GUID	16 B	Unique identifier for the texture
Name	string	-	Human-readable name of the texture
Width	int	4 B	Width of the texture in pixels
Height	int	4 B	Height of the texture in pixels
Channels	uint8	1B	The number of channels per pixel: Can be 1,2,3 or 4 corresponding to R, RG, RGB, or RGBA
BitDepth	uint8	1B	The total number of Bits Per Pixel. - For an 8-bit / chPannel RGB image the Bit Depth is 24. - For a 32-bit / channel (HDR) RGB image, it is 96, etc
Ppm	float	4 B	Pixels Per Meter - the correlation of the texture to real-world units of the surface it depicts. A 2048 x 2048 image that represents a 1m x 1m wall would have a Ppm = 2048.
Data	uint8[]	-	The actual information of the texture, as a byte array

User Data	map<string, Prsim_Object>	-	Dictionary of arbitrary data attached to the texture

Prism_UVMap

The following is a minimum specification of an object holding information about the UV Mapping of a mesh:

Property	Type	Size	Description
UV_Type	uint8	1 B	Whether this object contains UV coordinates or makes use of one of the procedural Projection Methods.
Texture Coordinates	float[]	-	An array of floating point numbers in the form of [x1, y1, x2, y2 … xN, yN] representing the per-vertex texture coordinates associated with the object.
User Data	map<string, Prsim_Object>	-	Dictionary of arbitrary data attached to the texture

Prism_InstanceDefinition

The following is a minimum specification of an object serving as an Instance Definition, an object that does not exist in a 3D scene, but various Instances reference it:

Property	Type	Size	Description
Definition	Prism_Object	-	The actual object
User Data	map<string, Prsim_Object>	-	Dictionary of arbitrary data attached to the Instance Definition

Prism_Instance

The following is a minimum specification of an object serving as an Instance, an object that references a Prism_InstanceDefinition:

Property	Type	Size	Description
PrototypeId	GUID	16 B	The ID of the object that this Instance refers to
Transformation	Matrix4x4	64 B	The transformation matrix of the instance
User Data	map<string, Prism_Object>	-	Dictionary of arbitrary data attached to the Instance Definition

Prism_AuxiliaryData

Example class outlines:

```
// Example Generic Auxiliary Object
class Prism_AuxiliaryObject : Prism_Element
{
        byte[] raw_data ;

        virtual void Parse(Prism_Parser parser)
        {
                parser.Parse(this);
        }

        virtual void Render(Prism_Renderer renderer)
        {
                renderer.Render(this);
        }

        virtual void Handle(Prism_Handler handler)
        {
                handler.Handle(this);
        }
}

// Example Bitmap Parser
class Prism_BitmapParser: Prism_Parser
{
    virtual  void Parse(Prism_AuxiliaryObject obj);
}

// Example Bitmap Renderer
class Prism_BitmapRenderer: Prism_Renderer
{
    virtual void Render(Prism_AuxiliaryObject obj);
}

// Example Bitmap Handler
class Prism_BitmapHandler: Prism_Handler
{
    virtual void Handle(Prism_AuxiliaryObject obj);
}
```

○ 2.5 Specifications for Discipline-Specific PrismArch Schemas

■ 2.5.2 Structural Engineering Schemas

Alongside the cross-disciplinary ontology described above, we must also provide discipline-specific schemas for certain classes of object - for example objects to contain the results of structural engineering analysis, that will not follow the generic PrismArch data types. The two classes related to output results are intentionally loosely structured, and thus more readily able to store results from a range of structural engineering software.

Prism_StrEng_Results

The following is a minimum specification for a set of objects that can hold structural engineering-specific input geometries and output results:

The objects described here are:

Classes

- Prism_StrEng_Results (the parent entity for the other objects)

- Prism_Base_Point3D

- Prism_StrEng_Line

- Prism_StrEng_Area

- Prism_StrEng_Results_Scalar

- Prism_StrEng_Results_Vector

Enumerators

- *Prism_StrEng_Results_Types*

- Prism_StrEng_Results_SubTypes

- Prism_StrEng_Results_OutputCaseTypes

The two classes related to output results are intentionally loosely structured, and thus more readily able to store results from a range of structural engineering software.

Property	**Type**	**Description**
Joints	Prism_Base_Point3D[]	Array of all unique 3D Points within the structural model - can be used for SupportPts, Nodes, Beam/ Column End Pt or Mesh Vertices.
Frames	Prism_StrEng_Line[]	Line elements that connect pairs of 3D Points. Stored as Indices that refer to the Joint property.
Areas	Prism_StrEng_Area[]	Area elements that connect sets of 3 or 4 3D Points. Stored as Indices that refer to the Joint property.
Results_ByJoint_Scalar	List<Prism_StrEng_Results_Scalar>	Generic Scalar value storage. For results per Structural Node.
Results_ByJoint_Vector	List<Prism_StrEng_Results_Vector>	Generic Vector value storage. For results per Structural Node.
Results_ByFrame_Scalar	List<Prism_StrEng_Results_Scalar>	Generic Scalar value storage. For results per Frame element.
Results_ByFrame_Vector	List<Prism_StrEng_Results_Vector>	Generic Vector value storage. For results per Frame element.

Results_ByArea_Scalar	List<Prism_StrEng_Results_Scalar>	Generic Scalar value storage. For results per Area element.
Results_ByArea_Vector	List<Prism_StrEng_Results_Vector>	Generic Vector value storage. For results per Area element.

Prism_StrEng_Line

Property	**Type**	**Description**
PtStart	int	Start Point. Stored as an Index that refers to the Joint property.
PtEnd	int	End Point. Stored as an Index that refers to the Joint property.

Prism_StrEng_Area

Property	**Type**	**Description**
A	int	Index of 1st Point in the MeshFace. Stored as an Index that refers to the Joint property.
B	int	Index of 2nd Point in the MeshFace. Stored as an Index that refers to the Joint property.
C	int	Index of 3rd Point in the MeshFace. Stored as an Index that refers to the Joint property.
D	int	Index of 4th Point in the MeshFace. Stored as an Index that refers to the Joint property.
IsTri	bool	Flag stating whether the MesFace is a Tri or Quad, and thus whether the “D” index should be used.

Prism_StrEng_Results_Scalar

Property	**Type**	**Description**
ResultType	Prism_StrEng_Results_Types	Type of results - i.e. *Displacement*, *Stresses*, etc.
ResultSubType	Prism_StrEng_Results_SubTypes	Sub-Type of results - i.e. *Translation*, *Rotation*, *Max Stress*, *Min Stress*, *Utilisation*, etc.
OutputCaseType	Prism_StrEng_Results_OutputCaseTypes	Load Case Type - i.e. *Dead* Load, *Live Load*, etc.
StepNum	double	Structural Mode for the given results.
Results	double[]	Flattened array of all results.

Prism_StrEng_Results_Vector

Property	Type	Description
*Same Properties as **Prism_StrEng_Results_Vector**, however **Results** property is different:*		
Results	Prism_Base_Vector3D[]	Flattened array of all results.

Prism_StrEng_Results_Types

Value	Description
Unset ● Joint ○ Displacements ○ Reactions ○ Restraints ● Area Shells ○ Joint forces ○ Element forces ○ Element Stresses ● Frames ○ Joint Forces ○ Element Forces	

Prism_StrEng_Results_SubTypes

Value	Description
Unset	
Translation	
Rotation	
Base Reactions	
Global FX	Reaction Force in the X-direction
Global FY	Reaction Force in the Y-direction
Global FZ	Reaction Force in the Z-direction
Global MX	Reaction Moment in the X-direction
Global MY	Reaction Moment in the Y-direction
Global MZ	Reaction Moment in the Z-direction
Joint Reactions	
F1	Reaction Force on the 1-axis
F2	Reaction Force on the 2-axis
F3	Reaction Force on the 3-axis
M1	Moment Reaction around 1-axis

M2	Moment Reaction around 2-axis
M3	Moment Reaction around 3-axis
Joint Displacement	
U1	Displacement on the 1-axis
U2	Displacement on the 2-axis
U3	Displacement on the 3-axis
R1	Rotation around the 1-axis
R2	Rotation around the 2-axis
R3	Rotation around the 3-axis
Joint Restraints	
U1	Restraint on the 1-axis
U2	Restraint on the 2-axis
U3	Restraint on the 3-axis
R1	Rotation around the 1-axis
R2	Rotation around the 2-axis
R3	Rotation around the 3-axis
Area Shells - Element Stresses	
S11Top	Stress in Top of Shell in 1-Direction
S22Top	Stress in Top of Shell in 2-Direction
S12Top	Stress in Top of Shell in 12-Direction
SMaxTop	Max Stress in Top of Shell
SMinTop	Min Stress in Top of Shell
SAngleTop	
SVMTop	
S11Bot	Stress in Bottom of Shell in 1-Direction
S22Bot	Stress in Bottom of Shell in 2-Direction
S12Bot	Stress in Bottom of Shell in 12-Direction
SMaxBot	Max Stress in Bottom of Shell
SMinBot	Min Stress in Bottom of Shell
SAngleBot	
SVMBot	
S13Avg	
S23Avg	
SMaxAvg	
SAngleAvg	
Area Shells - Element Forces	
F11	Internal Force along 1-axis
F22	Internal Force along 2-axis
F12	Internal Force normal to 1 or 2 axis
FMax	Max Force
FMin	Min Force
Frames - Element Joint Forces	
F1	Force along 1-axis
F2	Force along 2-axis
F3	Force along 3-axis
M1	Moment around 1-axis
M2	Moment around 2-axis
M3	Moment around 3-axis
Frames - Element Forces	

P	Axial Load
V2	Shear along 2-axis
V3	Shear along 3-axis
T	Torsion
M2	Moment around 2-axis
M3	Moment around 3-axis
Frames - Element Stresses	
S11	Stress on the face in the 1-axis direction
S12	Stress on the face in the 1 and 2axis direction
S13	Stress on the face in the 1 and 3axis direction
SMax	Max Stress
SMin	Min Stress

Prism_StrEng_Results_OutputCaseTypes

Value	Description
Unset	
Dead	Dead Load Case
MODAL	Modal Case
LL	Live Load Case
SIDL	Super Dead Load Case
SW	Self Weight Case
ULS	Ultimate Limit State
SLS	Serviceability Limit State

■ 2.5.3 MEP Engineering Schemas

The MEP Engineering Schemas need to be based on hierarchical ontology and driven by the classification systems. Uniclass and Omniclass classification systems provide tables which can drive the schemas.

Omniclass contains 15 tables, some of them focusing on buildings while others focus on landscapes or civil and process engineering. The tables can be used independently and focus on a particular area of a construction project.

The Omniclass tables can be downloaded by clicking this **link**

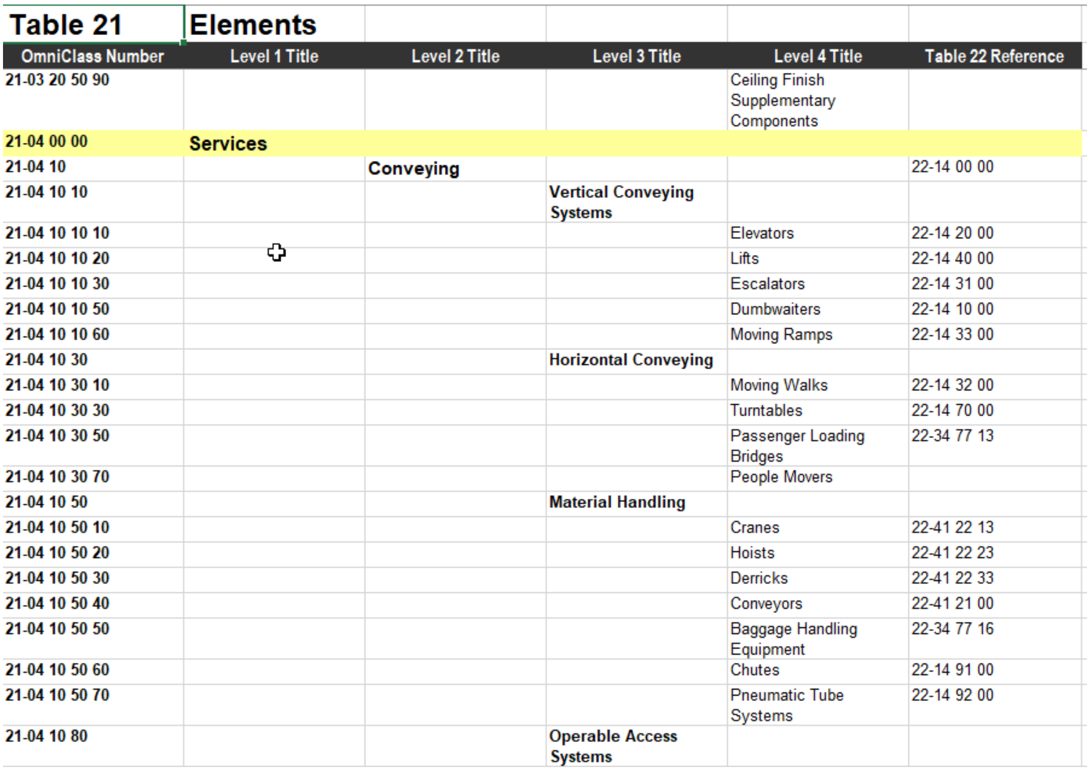
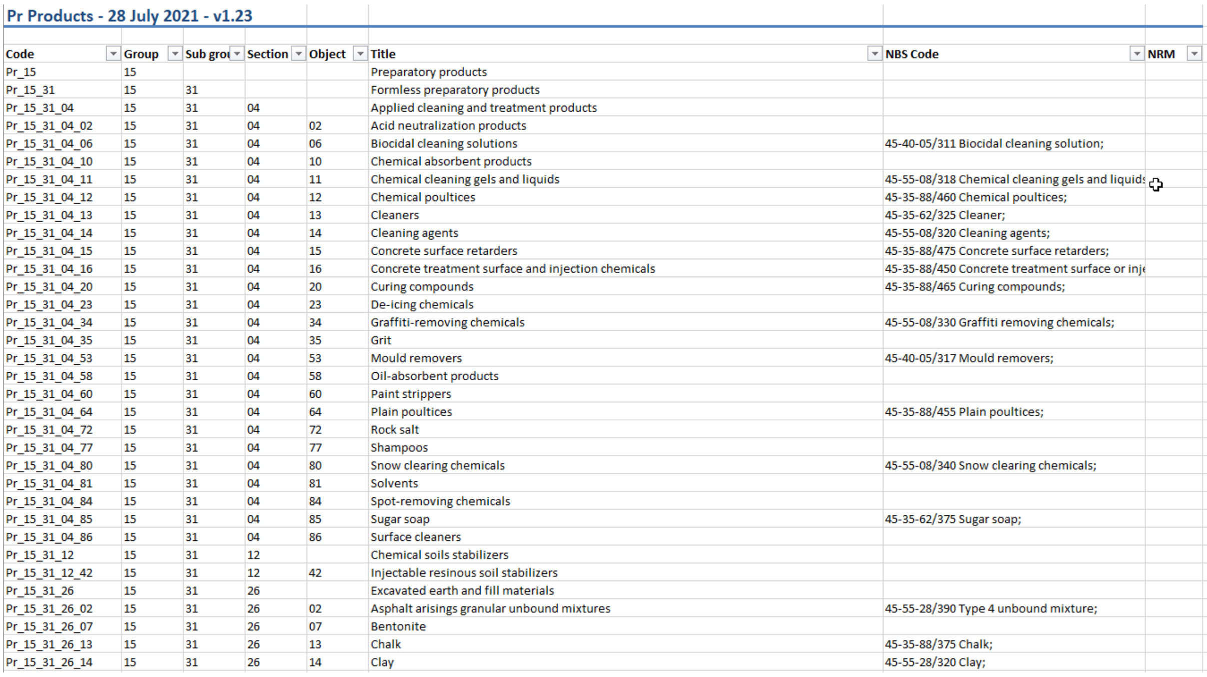


Table 21	Elements				
OmniClass Number	Level 1 Title	Level 2 Title	Level 3 Title	Level 4 Title	Table 22 Reference
21-03 20 50 90				Ceiling Finish Supplementary Components	
21-04 00 00	Services				
21-04 10		Conveying			22-14 00 00
21-04 10 10			Vertical Conveying Systems		
21-04 10 10 10				Elevators	22-14 20 00
21-04 10 10 20				Lifts	22-14 40 00
21-04 10 10 30				Escalators	22-14 31 00
21-04 10 10 50				Dumbwaiters	22-14 10 00
21-04 10 10 60				Moving Ramps	22-14 33 00
21-04 10 30			Horizontal Conveying		
21-04 10 30 10				Moving Walks	22-14 32 00
21-04 10 30 30				Turntables	22-14 70 00
21-04 10 30 50				Passenger Loading Bridges	22-34 77 13
21-04 10 30 70				People Movers	
21-04 10 50			Material Handling		
21-04 10 50 10				Cranes	22-41 22 13
21-04 10 50 20				Hoists	22-41 22 23
21-04 10 50 30				Derricks	22-41 22 33
21-04 10 50 40				Conveyors	22-41 21 00
21-04 10 50 50				Baggage Handling Equipment	22-34 77 16
21-04 10 50 60				Chutes	22-14 91 00
21-04 10 50 70				Pneumatic Tube Systems	22-14 92 00
21-04 10 80			Operable Access Systems		

Screenshot of Omniclass Table 21 (partial information).

Source: https://www.csiresources.org/standards/omniclass/standards-omniclass-about

Uniclass is a classification system initially developed for the UK market, however due to evolution in data management the system is now aligned with ISO19650 aiding in several aspects from CAD layering to costs, annotation etc. The Uniclass system contains 12 tables which can be downloaded by clicking this **link**



Pr Products - 28 July 2021 - v1.23

Code	Group	Sub grou	Section	Object	Title	NBS Code	NRM
Pr_15	15				Preparatory products		
Pr_15_31	15	31			Formless preparatory products		
Pr_15_31_04	15	31	04		Applied cleaning and treatment products		
Pr_15_31_04_02	15	31	04	02	Acid neutralization products		
Pr_15_31_04_06	15	31	04	06	Biocidal cleaning solutions	45-40-05/311 Biocidal cleaning solution;	
Pr_15_31_04_10	15	31	04	10	Chemical absorbent products		
Pr_15_31_04_11	15	31	04	11	Chemical cleaning gels and liquids	45-55-08/318 Chemical cleaning gels and liquids	
Pr_15_31_04_12	15	31	04	12	Chemical poultices	45-35-88/460 Chemical poultices;	
Pr_15_31_04_13	15	31	04	13	Cleaners	45-35-62/325 Cleaner;	
Pr_15_31_04_14	15	31	04	14	Cleaning agents	45-55-08/320 Cleaning agents;	
Pr_15_31_04_15	15	31	04	15	Concrete surface retarders	45-35-88/475 Concrete surface retarders;	
Pr_15_31_04_16	15	31	04	16	Concrete treatment surface and injection chemicals	45-35-88/450 Concrete treatment surface or inje	
Pr_15_31_04_20	15	31	04	20	Curing compounds	45-35-88/465 Curing compounds;	
Pr_15_31_04_23	15	31	04	23	De-icing chemicals		
Pr_15_31_04_34	15	31	04	34	Graffiti-removing chemicals	45-55-08/330 Graffiti removing chemicals;	
Pr_15_31_04_35	15	31	04	35	Grit		
Pr_15_31_04_53	15	31	04	53	Mould removers	45-40-05/317 Mould removers;	
Pr_15_31_04_58	15	31	04	58	Oil-absorbent products		
Pr_15_31_04_60	15	31	04	60	Paint strippers		
Pr_15_31_04_64	15	31	04	64	Plain poultices	45-35-88/455 Plain poultices;	
Pr_15_31_04_72	15	31	04	72	Rock salt		
Pr_15_31_04_77	15	31	04	77	Shampoos		
Pr_15_31_04_80	15	31	04	80	Snow clearing chemicals	45-55-08/340 Snow clearing chemicals;	
Pr_15_31_04_81	15	31	04	81	Solvents		
Pr_15_31_04_84	15	31	04	84	Spot-removing chemicals		
Pr_15_31_04_85	15	31	04	85	Sugar soap	45-35-62/375 Sugar soap;	
Pr_15_31_04_86	15	31	04	86	Surface cleaners		
Pr_15_31_12	15	31	12		Chemical soils stabilizers		
Pr_15_31_12_42	15	31	12	42	Injectable resinous soil stabilizers		
Pr_15_31_26	15	31	26		Excavated earth and fill materials		
Pr_15_31_26_02	15	31	26	02	Asphalt arisings granular unbound mixtures	45-55-28/390 Type 4 unbound mixture;	
Pr_15_31_26_07	15	31	26	07	Bentonite		
Pr_15_31_26_13	15	31	26	13	Chalk	45-35-88/375 Chalk;	
Pr_15_31_26_14	15	31	26	14	Clay	45-55-28/320 Clay;	

Screenshot of Uniclass Product Table (partial information)

Source: https://www.thenbs.com/our-tools/uniclass-2015

○ 2.6 Non-Ontological Requirements for PrismArch

■ 2.6.1 Security and Authentication

As outlined in several of the previous Deliverables, there is a critical requirement for the PrismArch platform to ensure the security and privacy of all data shared between parties. These issues are clearly demonstrated in *D1.1 Section 4.2 (Data Authority / Ownership of Data)*, *D3.1 Section 3.2 (Data and Document Management)*.

As described in *D4.1 Section 3.2.1 (Speckle system and developments towards integration),* and below, the existing Speckle framework contains most of the requirements needed for PrismArch. It is a highly secure data distribution platform, the developer state that:

"Enterprise Speckle Servers (as well as our hosted offering, speckle.xyz) are deployed with end-to-end security:

- *The Speckle Server uses https (TLS) to encrypt all incoming data from all clients.*
- *All data is stored in a managed PostgreSQL database cluster.*
- *The DB is only accessible from the Kubernetes cluster that runs your server and its other components.*
- *DB credentials are securely stored in a Kubernetes secret.*
- *SSL is always used to communicate with the DB.*
- *Data in the DB is encrypted at rest with LUKS.*
- *The DB will have a standby failover node, & PITR (point in time recovery)."*

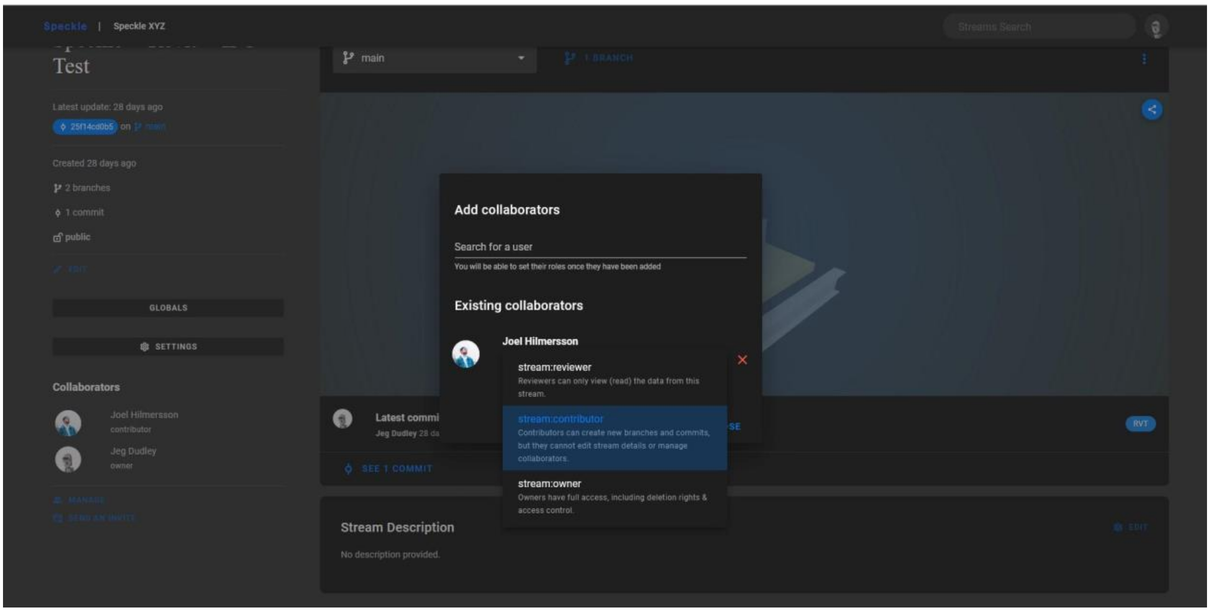
From **https://speckle.systems/security/**

The same online document also confirms that Speckle complies with the EU's GDPR policy. See https://speckle.systems/privacy/ for additional information.

The security arrangements above are for Speckle's *Enterprise* solution - however it confirms that their platform can achieve the level of data security necessary for PrismArch, and the above should be treated as our standard requirement for PrismArch.

Data Visibility and Confidentiality

However, while Speckle generates highly secure data connections, it does not provide the functionality necessary to isolate incoming data from different parties, or provide granular permission levels that control which users can see or operate upon different data sets. Speckle currently only provides differentiation between *Reviewer*, *Contributor* and *Owner.*



Add collaborators

Search for a user

You will be able to set their roles once they have been added

Existing collaborators

Joel Hilmersson

stream:reviewer
Reviewers can only view (read) the data from this stream.

stream:contributor
Contributors can create new branches and commits, but they cannot edit stream details or manage collaborators.

stream:owner
Owners have full access, including deletion rights & access control.

Stream Description

No description provided.

Figure 2.6.1a - Current Speckle Permission Levels.

As specified in *D1.1 Section 4.2 (B - Constraints)* a pivotal requirement is that the PrismArch platform knows exactly which organisation each user belongs to, as well as what Sphere Level they are currently operating within, in order to ensure that they can only share (or have shared with them) data appropriate to their Role, Organisation, Sphere Level, and so on.

To provide this functionality we therefore need to look beyond the 'default' implementation of Speckle.

Options for Database Integration

With the above issues in mind, there are a range of database integration options that span between *highly aggregated* and *highly distributed* solutions. Note that the term *distributed* does not equate to *'siloed'*, as is usually the case in AEC.

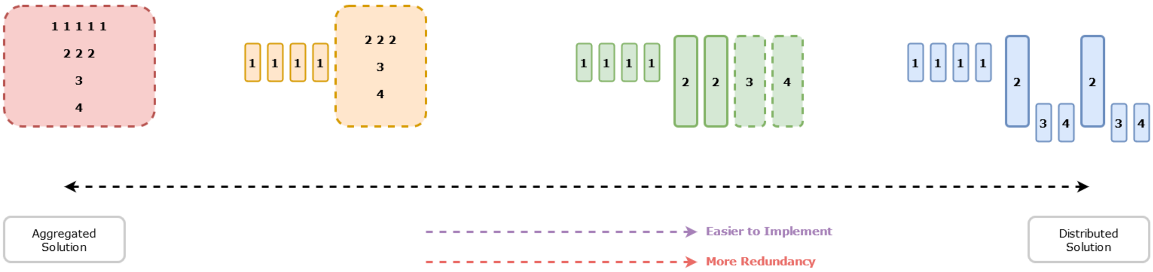


Figure 2.6.1b - Options for Speckle Database deployment. The numbers shown (1-4) refer to the Sphering Level information contained within.

We believe that all of these options (described in detail below) are *technically* feasible, by adapting the existing Speckle Server architecture. However we also believe that the earlier options would be very complex to achieve, with increasingly simplicity for latter options.

Additionally we believe the system is increasingly secure from failures for latter options. Most significantly, we think that *only* the final Option (4) fully achieves the permission levels and access control requirements requested in prior PrismArch reports.

Option 1 - Highly Aggregated

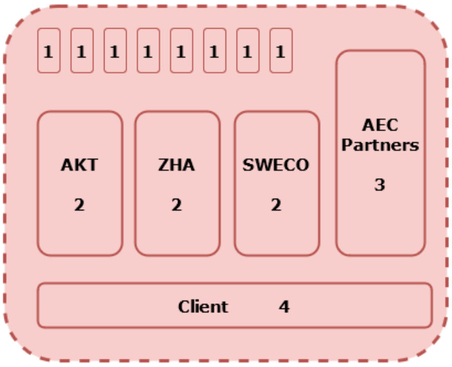


Figure 2.6.1c - Option 1 for Speckle deployment.

This solution is the default Speckle approach.

In this option, *all* data (SL1-SL4) is stored in a single Repository. This Repo could be a local Server within an AEC Partner's office, or a Cloud-based Server (*AWS*, *Google Cloud*, *Azure*, etc).

This has the advantage of no duplication of data, which in turn ensures no issues with data synchronisation.

However there is significant risk of data loss, due to no duplication of data (i.e. single point of failure) for SL1-SL4. This could be somewhat mitigated if the Server is cloud-based and includes backups.

There is unacceptable functionality with regards to Permission Levels: All users can access all the data of others (from SL1-SL4). AEC Partners would be rightly wary of unpermitted data extraction.

A minor issue is that even personal SL1 data is stored in the same Repo, thus being unnecessarily costly for whoever pays for this Server - the client, the architect, etc.

Option 2 - Minor Distributed

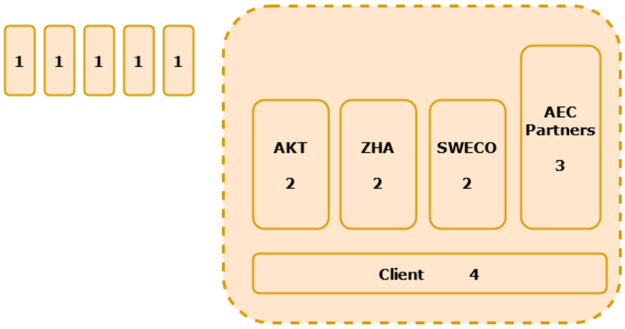


Figure 2.6.1d - Option 2 for Speckle deployment.

In this Option, *most* data (SL2-SL4) is stored in a single Repository. This Repo could be a local Server within an AEC Partner's office, or a Cloud-based Server (*AWS*, *Google Cloud*, *Azure*, etc).

This has the advantage of reduced duplication of data, which in turn ensures less issues with data synchronisation.

However there is significant risk of data loss, due to no duplication of data (i.e. single point of failure) for SL2-SL4. This could be somewhat mitigated if the Server is cloud-based and includes backups.

There is unacceptable functionality with regards to Permission Levels: All users can access all the data of others from Company-level upwards (from SL2-SL4).

In this scenario SL1 data is now stored in *personal Repos* (again, could be client-based, local Server, or Cloud Server), which avoids unnecessary cost for the client, the architect, etc.

This level of distribution is arguably the minimum required, and already necessitates *two* different data transport systems to be set up.

Option 3 - Major Distributed

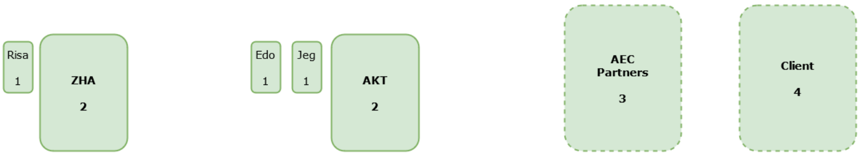


Figure 2.6.1e -Option 3 for Speckle deployment.

In this Option, *minimal* data (SL3-SL4) is stored in a single Repository. This Repo could be a local Server within an AEC Partners office, or a Cloud-based Server (*AWS*, *Google Cloud*, *Azure*, etc).

There is some advantage of reduced duplication of data, which ensures slightly less issues with data synchronisation, but only for SL3-SL4.

There is a reduced risk of data loss, due to no duplication of data (i.e. single point of failure) for SL3-SL4. This could be somewhat mitigated if the Server is cloud-based and includes backups.

Even with this arrangement, there is still an unacceptable functionality with regards to Permission Levels: All users can access all SL3 and SL4 data of others - i.e. Clients can access work distributed between Companies (but supposedly not shared with them yet) within SL3, and all Companies can access all SL3 data, regardless of whether it was shared with them.

In this scenario SL1 data is now stored in *personal Repos* (again, could be client-based, local Server, or Cloud Server), while SL2 data is stored in *company Repos* (again, could be local Server, or Cloud Server). This change slightly Increases the risk of data loss, but would be localised to specific companies.

This level of distribution necessitates *three* different data distribution systems to be set up.

Option 4 - Fully Distributed

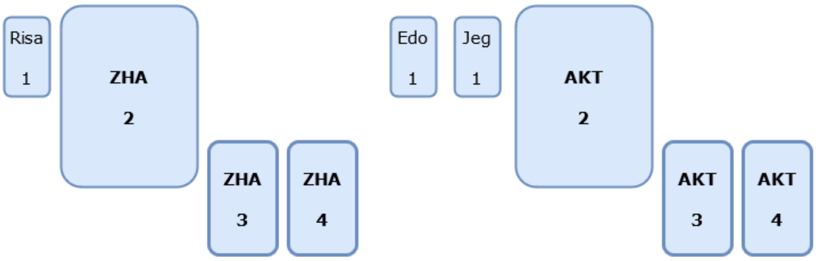


Figure 2.6.1f - Option 4 for Speckle deployment.

In this Option, *no* data is stored in a single Repository. In general, it is comparable to a git-style distribution of private branches in private instances (streams/forks), with only the more public branches being pushed to more public instances of the database.

This has the slight disadvantage of duplicating the same shared data across multiple locations.

There is now a greatly reduced risk of data loss, as all partners now have an identical copy of the SL3 and SL4 Repos. Most critically, this is likely the only option which can guarantee that no party can access information they are not permitted to see.

SL1 data is stored in *personal Repos* (could be client-based, local Server, or Cloud Server).

SL2 data is stored in *company Repos* (could be local Server, or Cloud Server).

Synchronised copies of SL3 and SL4 data are stored in *Company Repos.* (could be local Server, or Cloud Server).

This level of distribution still only necessitates *three* different data distribution systems to be set up.

Unique Aspects of Option 4

Due to the fully distributed nature of SL3 and SL4, all AEC Partners must formally agree - before a project begins - what level of security is acceptable for *all* synchronised Repos.

For example:

- *ZHA stores SL3 data in 2-Factor Authenticated cloud server.*
  - ✓acceptable to all parties.
- *AKT stores the same data in an open internet database.*
  - ✗unacceptable to some parties.

Because SL3 and SL4 are duplicated, we must ensure that all AEC Partners can only possess *synchronised* copies of these Repos. An ideal way to do so would be to wrap the functions for pushing to different Sphere Levels inside a parent function, that ensures the commit is distributed to all of the relevant databases simultaneously.

Summary of Options, and Potential Implementation Solution

Our analysis of the potential options has determined that only option (4) achieves all of the significant PrismArch Database requirements. This is because 'default' Speckle databases only have a limited set of permission options: once a user has joined a Speckle project they can *see* all data, and the permission levels just control what they can *edit* or *delete*.

That is not sufficient for PrismArch. We need the ability to entirely exclude certain users/ groups from seeing or even knowing about the existence of certain data, as specified in the Sphering Levels discussions throughout D6.1 and D1.1. There may be the potential to greatly expand the permission system for Speckle, that could in turn potentially remove some of the impediments from utilising a more shared data source (such as options (1) or (2)). However, there would still be questions about which party holds the admin level controls for that shared database (and thus can access all data, regardless of permission levels). Given how strongly the AEC Partners have suggested that data access and sovereignty is an issue, and given the current level of understanding of the Speckle systems, we felt that option (4) is the best compromise at this juncture.

Conveniently for this project, we believe that it is possible to customise the 'default' Speckle platform to generate a bespoke arrangement very akin to the option (4) shown above. Speckle employs entities called *Transports*, that define how it writes to and from mediums. As stated below, it is possible to set up multiple Transports in parallel that could write to a range of different Servers (for example, Servers housing SL2, SL3 and SL4) simultaneously.

According to their own documentation:

"Transports: Different storage systems have various characteristics that make them better (or more ill) suited for different scenarios. This is why, rather than employ a unique storage system, Speckle uses an intermediary abstraction layer: transports. A transport defines the way Speckle writes to, and reads from, a given persistence layer.

One such transport is the Speckle Server Transport. Another transport is an SQLite Transport. Speckle comes with a couple of other transports too: a MongoDB transport, an In-Memory Transport, as well as a Disk Transport. Other transports, such as an S3 transport, a MySQL transport could easily be developed.

Moreover, send operations are no longer restricted to one single location: Speckle allows you to send data, in parallel, to multiple transports. For example, data can be sent to two different Server Transports at the same time, one being an internal server and one being an external one - a different stakeholder involved in the process."

From https://speckle.guide/dev/architecture.html

Further documentation on this subject is available in the "Writing Your Own Transports" section of https://speckle.guide/dev/transports-dev.html.

Example of Multiple Synchronised Speckle Transports

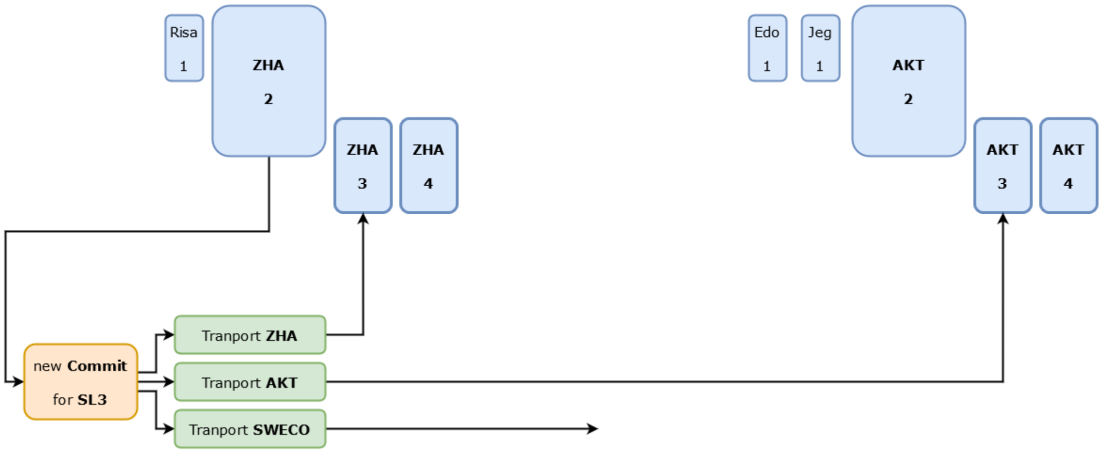


Figure 2.6.1g -Conceptual process for utilising multiple transports.

PrismArch should therefore implement multiple *Transports* - not just 1x Transport per SL, but instead 1x Transport per Company per SL - that are written to simultaneously. To avoid the risk of not pushing to all relevant SL Servers, all of these different Transports must *not* be exposed to PrismArch users. Instead, groups of these Transports should be wrapped within custom PrismArch Speckle Connectors, that use a combination of the User, Company, and SL meta-data to determine which Transports should be activated in any given commit.

For example:

- An **SL1 object committed by Helmut (ZHA) up to SL2** creates 1x Transports:
  - *ZHA SL2 Repo*

- An **SL2 object committed by ZHA up to SL3** creates 3x Transports:
  - *ZHA SL3 Repo*
  - *AKT SL3 Repo*
  - *SWECO SL3 Repo*

- An **SL3 object committed by AKT up to SL4** creates 4x Transports:
  - *ZHA SL4 Repo*
  - *AKT SL4 Repo*
  - *SWECO SL4 Repo*
  - *Client's SL4 Repo*

There must be no functionality exposed that allows commits to pass to a *single company's* SL3 or SL4 Repo, as this would break synchronisation between all the distributed Speckle servers.

Transports, Commits and Validation

Given that the PrismArch ontology that we are proposing attaches the critical Prism_Signature to the Speckle 'optional' user data, it could be sensible to include a validation function within all commits or transport functions, which would ensure that a Signature is attached to all entities before sending them.

● 3.0 CONCEPTUALISING THE VR EXPERIENCE

○ 3.1 Introduction

The PrismArch platform will differentiate itself from existing VR collaborative softwares in the way it investigates the commonality and symbiosis between different building design practices. Drawing from the particular characteristics of building design, virtual reality provides value through not only the three-dimensional aspects of user interaction but also the confluence of intricate project information in one virtual space.

The research in this section provides both a general understanding of how the AEC VR experience could be conceptualised from literature reviews and user interviews, followed by domain specific requirements developed through case studies and design exercises.

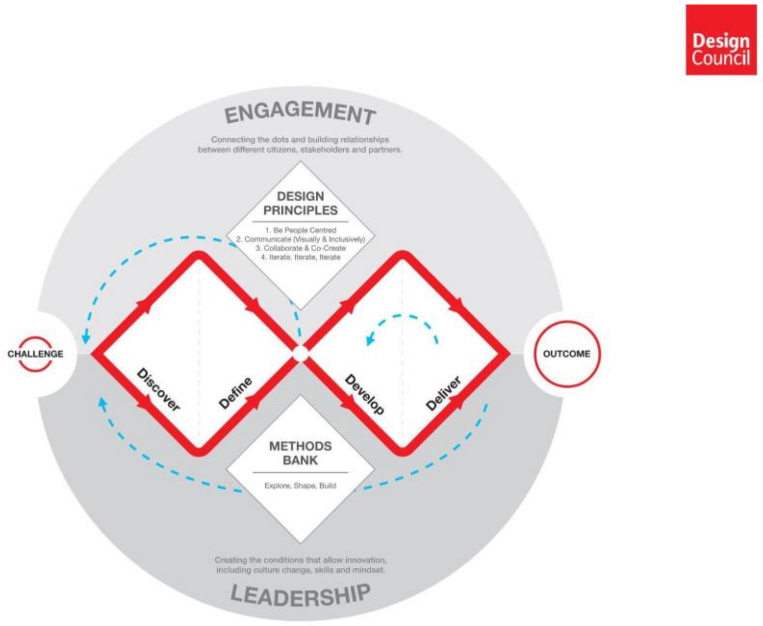
○ 3.2 VR Experience Requirements Outlined in Previous Deliverables

Before this point, there has been a tremendous amount of discussion on the prototype requirements and data classifications of the PrismArch platform. The complexity of creating a working prototype is evident not only in the multidisciplinary nature of the project scope but also in the multitude of stakeholders involved at each stage of the building design lifecycle.

Previous deliverables have outlined sixteen base-level user requirements (D1.1, sct. 4.4) that respond to design needs emerging from the convergence of AEC disciplines. These requirements collectively offer a solid framework for the more general features required during a range of AEC activities. The PrismArch consortium has also identified four distinct data classifications (D6.1, sct. 1.1), where a clear definition was given to the Personalisable User Interface (PUI). This section of the work assumes the PUI to be a configurable interface that can be brought across different spheres to facilitate activities undertaken during individual design work and reviews, as well as design team meetings. Many of the functionalities proposed henceforth will fall under sub-section *d. 'Toolkit'* category.

Translating Requirements into Software Features

At the current stage of research, it is paramount to recognise that many of the collected requirements have to be translated into software features ahead of development. This is no small feat, as was demonstrated by the strenuous efforts in both D1.1 and D6.1. For an innovation project such as PrismArch, where development happens mainly in uncharted waters, we need to acknowledge that the development activities are often less structured and harder to anticipate. However, setting out instructions as clearly as possible will still increase development efficiency and improve quality assurance further down the line.


Design Council
ENGAGEMENT
Connecting the dots and building relationships between different citizens, stakeholders and partners.
DESIGN PRINCIPLES
1. Be People Centred
2. Communicate (Visually & Inclusively)
3. Collaborate & Co-Create
4. Iterate, Iterate, Iterate
CHALLENGE
Discover
Define
Develop
Deliver
OUTCOME
METHODS BANK
Explore, Shape, Build
Creating the conditions that allow innovation, including culture change, skills and mindset.
LEADERSHIP

© Design Council 2019

Figure 3.2a - Design Council's framework for innovation

A constructive guideline for innovation is the Double Diamond launched by [Design Council], a much-referenced design methodology in the industry. The diagram captures an iterative quality of the design process and the divergent and convergent flow of exploration. The methods documented in this section fall roughly within the 'Define' phase. They are set out to accurately generate the problem definition, significantly scoping down the focus from previous deliverables. Some of these include discipline-specific user interviews, user story mapping, card sorting, etc. More details will be provided in later parts of this section.

3.3 User Experience in the 3D Environment

3.3.1 Considerations for User Interaction Design

Despite the slow adoption of immersive technology, the AEC industry is in fact a prime candidate for innovation in 3D interaction design. As shown by [Zaker 2018], the recent integration of BIM in construction projects indicates a collaborative use of semantically rich 3D models for design tasks. This transition could be the starting point of a paradigm shift where the traditional screen-based interactions no longer suffice user needs.

When designing for user interaction in the 3D environment, two topics deserve more attention in the current research context: interaction fidelity and interaction precision. [Bowman 2014] eloquently describes the contemporary design trends in both of these domains.

The ongoing debate for interaction fidelity falls between results that are *'realistic'* vs. those that are *'magical'*. [Bowman 2012] defines interaction fidelity as 'the objective degree with which the actions (characterized by movements, forces, body parts in use, etc.) used for a task in the UI correspond to the actions used for that task in the real world'. One might argue that realistic interactions require a shorter learning curve, as long as their affordance is explicit in the user interface. On the other hand, the trade-off is missed opportunities to attempt 'enhanced' interactions that might achieve better efficiency. This decision will fundamentally fall upon the interaction designers, depending on the user's individual tasks.

Hardware limitations of the spatial tracking system have restricted the level of interaction precision virtual reality could attain. This will prove challenging for the AEC disciplines, especially if design reviews rely on users carrying out tasks such as measurements and alterations on the fly. Recommendations by [Bowman 2014] such as the 'progressive refinement' method deserve some lengthy examinations to reach optimal precision.

In addition to the above discussion, [Bowman 2001] accumulated a well-researched list of options for universal 3D interaction tasks that will provide concrete points of reference during the prototype stage:

- ***Navigation***
  - *Travel: motor component of viewpoint motion*
    - *Gaze-directed steering*
    - *Pointing*
    - *Map-based travel*
    - *"Grabbing the air"*
  - *Wayfinding: cognitive component; decision-making*
- ***Selection****: picking object(s) from a set*
    - *Simple virtual hand*
    - *Ray-casting*
    - *Sticky finger (occlusion)*
    - *Go-go (arm-extension)*
- ***Manipulation:*** *modifying object properties (esp. position/orientation)*
    - *Simple virtual hand*
    - *HOMER (Hand-Centered Object Manipulation Extending Ray-Casting)*
    - *Scaled-world grab*
    - *World-in-miniature*
- ***System Control:*** *changing system state or mode*
    - *Virtual menus*
    - *Tool selectors (belts, palettes, chests)*
    - *Speech commands*
    - *Pen & tablet technique*

Case Study: Gravity Sketch

An effective example for universal 3D interaction techniques is [Gravity Sketch], a virtual reality powered 3D design platform funded by the Horizon 2020 programme. Upon testing the latest application on the Oculus Rift headset, we discovered a few instances that clearly encapsulated the benefits of interacting in a 3D environment.

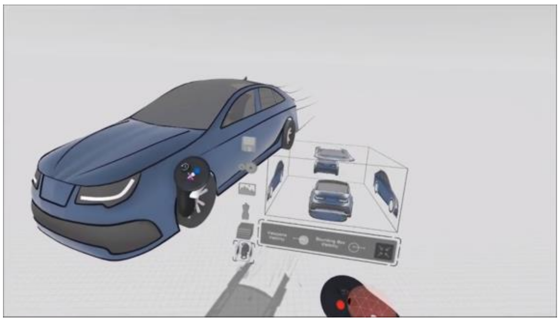


Figure 3.3.1a - The Orthographic Viewport

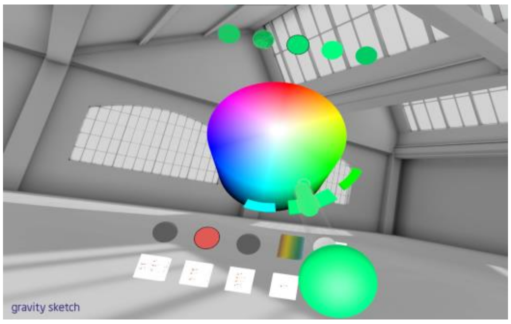


Figure 3.3.1b - The 3D Colour Picker

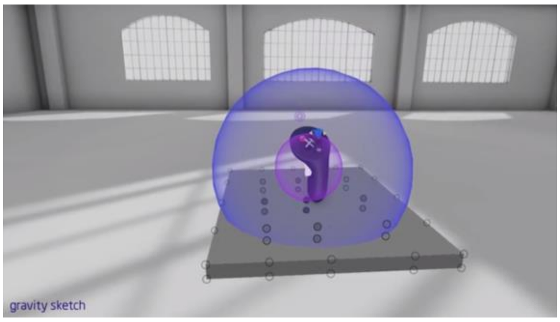


Figure 3.3.1c - Weighted selection for a SubD object

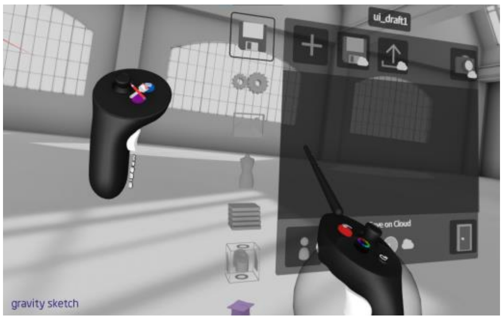


Figure 3.3.1d - Pointing at a docked menu panel

The toolkit comes with an ‘Orthographic Viewport’ that shows all the model side views projected onto a miniature box. The user can hold up the viewport like a solid object and closely inspect the views by moving it closer to their face. It also provides a 3D ‘Colour Picker’ which is cylindrical shaped, with a colour wheel on the top representing the hue and its height representing the value of a chosen colour. Furthermore, the toolkit incorporates weighted selection for a SubD object so that the user could grab onto a surface and sculpt it by hand. Some other effective 3D interactions include being able to point precisely with a finger-like stick and the ability to dock menu panels in mid-air. All of these are incredibly intuitive gestures created for a 3D workspace. They enhance what can be achieved traditionally through screen-based interactions.

■ 3.3.2 Considerations for a Dynamic Interface Design

Given the open-endedness of the proposed platform, it is fair to assume that at any given point there’s a chance that a user will encounter some purely novel elements that they have never interacted with before. This open-endedness is reflected in the underlying data-structures, as discussed in Section 2.4. If the PrismArch VR world can contain complex, 'exotic' elements that are not specified at the platform's design stage, but are generated by the users, what is the appropriate type of UI that can accommodate such extreme ontological variability ?

To answer this question, we cannot help but consider an alternative UI paradigm that shifts away from fixed toolbox-like or palette-like examples like Gravity Sketch, and is more similar in nature to modular, node-based tools like Scratch or Grasshopper3D, where the composition of simple elements leads to novel, ad-hoc complex behaviours.

One could theorise a very high-level example of this concept in the ability to install and uninstall toolsets according to their discipline-specific preferences, as part of a plug-in system. Such a system, as useful and necessary as it may be, does not constitute an example of a modular and dynamic UI system, since all it enables is the addition and removal of fixed UI elements/functionalities, but not the synthesis of new ones.

Closing in on a more granular level, however, one can start breaking down and abstracting away existing UI features into very simple atomic operations, that can then be recombined by the user in novel and creative ways. These "UI atoms" could have a physical, tangible 3D appearance in a VR context, allowing the user to treat them as functional building blocks, gradually constructing an inventory of bespoke UI inventions, for complex element filtering, geometry generation, navigation, querying, visualization, etc.

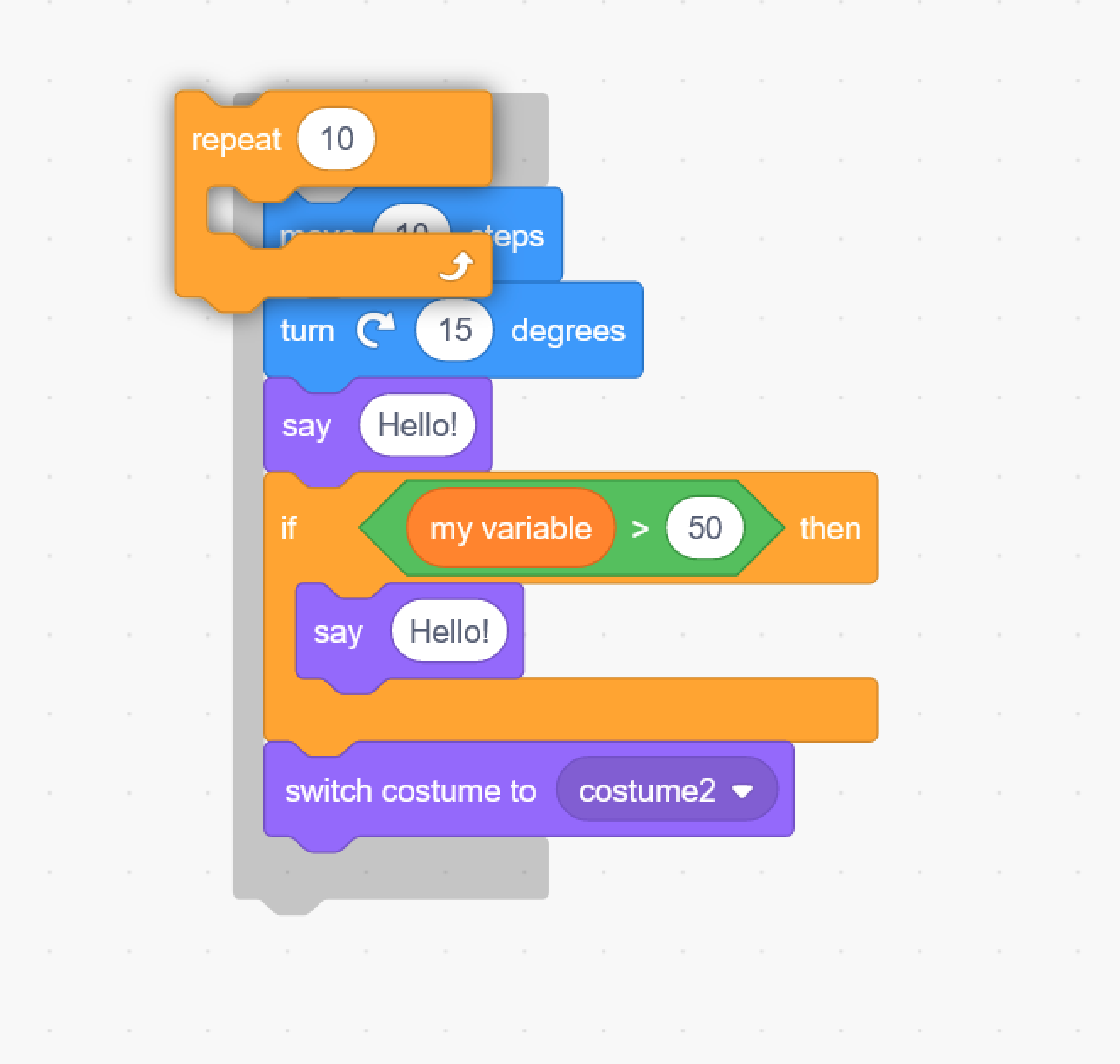
These assembled UI "molecules" could be short-lived for very project-specific operations, or end up becoming versatile, industry-standard "swiss-army knives", through their continuous tweaking and combination with tools of other users. The Prism project could finally decide to formalize and optimize some of these user tools based on their popularity and usefulness.

Reflecting upon the ontology and schemas described in previous sections, we could start seeing some synergies between such an interface and the data structure (sct 2.4.7, sct 2.5). Each object from the ontology could be represented in the interface with a series of nested properties, and being able to connect functions to these objects according to their properties would contribute to the dynamic quality of the interface.

Case Study: Scratch

Scratch is a visual programming language originally developed in MIT that is used by millions of children and adults worldwide. It uses a block-based UI paradigm, with users "clicking" blocks together like lego bricks. Different types of blocks allow for boolean logic, conditional statements, loops, sensor inputs, event triggering, geometry drawing and multimedia playback, among others.

The compact nature of each block synthesis allows users to conceptualize their logic as tangible contraptions, which can be mashed together or deconstructed with ease. What is of particular interest is that there's no way a block synthesis can become non-executable. What can slot into what is handled by the UI (shape system) and quite simply, one cannot produce invalid or erroneous "code", only malfunctioning at the worst case scenario.



```
repeat 10
move 10 steps
turn ↻ 15 degrees
say Hello!
if my variable > 50 then
  say Hello!
switch costume to costume2 ▾
```

Figure 3.3.2a - An example of a loop containing a conditional statement in Scratch.

Case Study: Grasshopper 3D

[Grasshopper 3D] is a graphical algorithm editor - part of the Rhinoceros CAD modelling software - that enables designers to generate 3D geometries or perform complex data operations through a node-based interface. The user interacts with the interface by dragging components onto a digital 'canvas', where inputs and outputs are connected dynamically via wires to produce the final results.

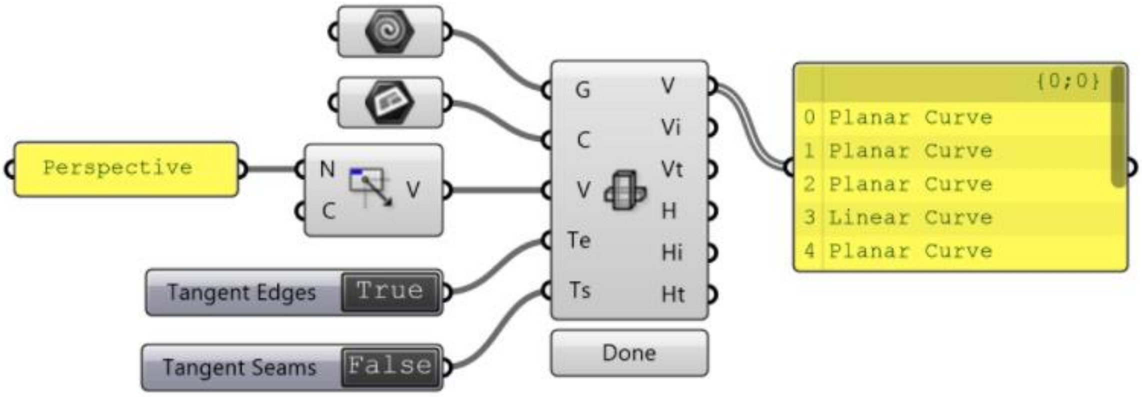
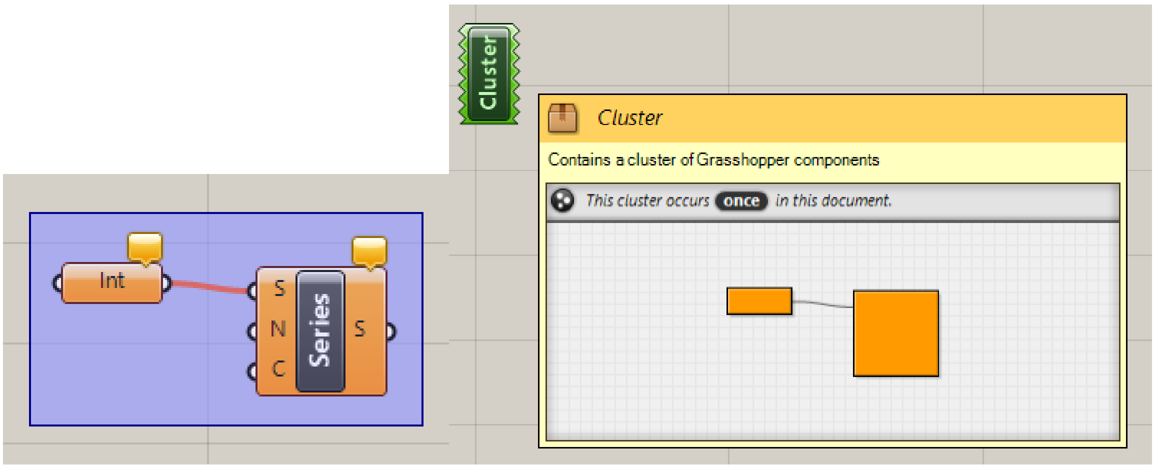


Figure 3.3.2b - Grasshopper node-based visual programming interface.



Cluster

Contains a cluster of Grasshopper components

This cluster occurs once in this document.

Figure 3.3.2c - Grouping and clustering nodes with the Grasshopper UI

The modularity of Grasshopper is exemplified not only in its excellent plug-in system that developers can author programmatically, but more importantly in the ability to cluster networks of nodes and nest them inside singular nodes or *'Clusters'* in Grasshopper terminology. These clusters are the equivalent of Functions/ Methods in traditional programming languages: re-usable, containerized pieces of logic with clear inputs and outputs. Clusters themselves can become parts of larger projects, or be embedded into more complex, higher-level clusters. A user can maintain an easily accessible library of user-objects/ clusters and share them with peers or re-use them in other projects.

Although we certainly don't propose a direct translation of a 2D UI paradigm like Grasshopper or Scratch into our VR platform, we trust that there are valuable lessons to be learnt from such tools in terms of designing a UI system able to handle a great degree of ontological uncertainty, while providing users the ability to navigate, analyze and author this world in their own preferred, discipline-specific ways.

○ 3.4 Core Requirements for the VR Interface

3.4.1 The PrismArch Four Root Classes

The earlier deliverable D6.1 outlines the four fundamental classes in the PrismArch VR platform, User Interface (UI) being one of them. The submitted report clearly highlights the requirement of visually distinguishing each class inside the singular VR space. Building on the previously-submitted material, this section studies core requirements for the PrismArch VR User Interface, proposing its structure as well as the control/interaction methods for both individual and collective scenarios.

The four fundamental classes in the PrismArch VR platform are:

(1) **Immersed Humans (IH)**
(2) **User Interface (UI)**
(3) **PrismArch Design Objects (PADO)**
(4) **PrismArch Metadata (PAM)**

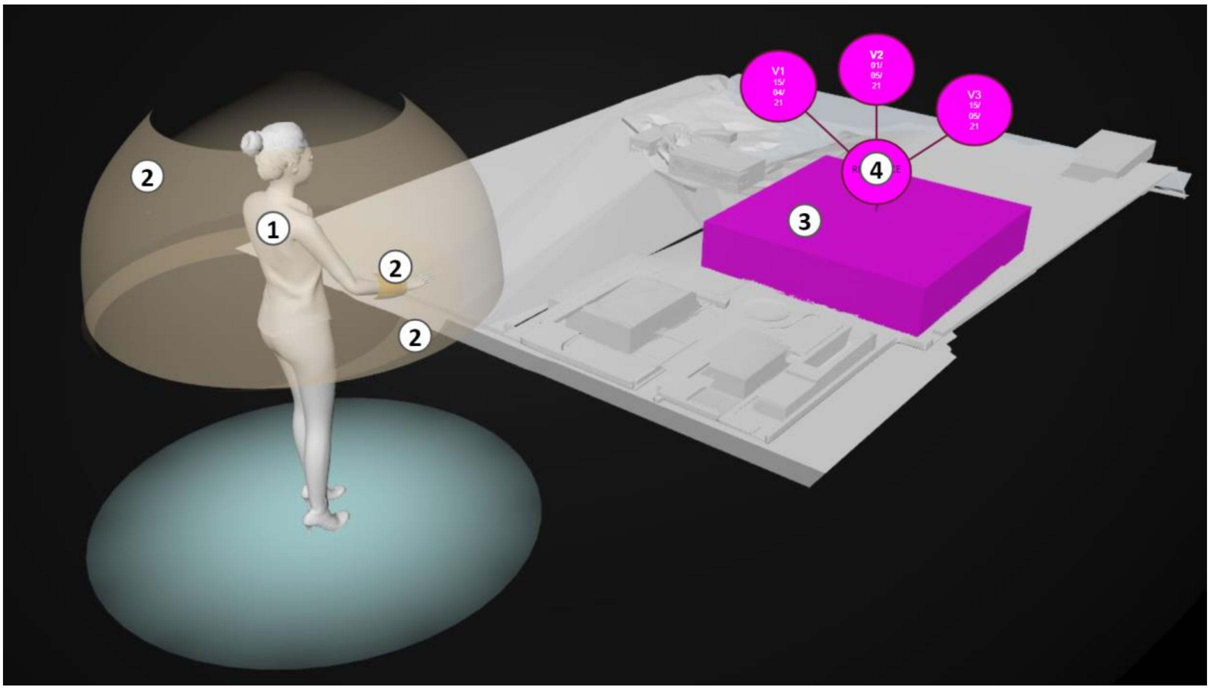


Figure 3.4.1a - The Four Root Classes of PrismArch Platform Content

(1) **Immersed Human**

In the deliverable D6.1, an Immersed Human is described as a '*digital representation of an immersed individual who is on-boarded to work inside the PrismArch World (PAW)*' (D6.1, p.14).

Requirements: Every on-boarding member should be allowed to configure their personal avatar, which resides inside their Personal Work Sphere (PWS). Inside the PWS, the user can complete the avatar configuration and customisation for the VR environment with the help of a virtual mirror to increase cognitive self awareness (Figure 3.4.1b). Inside VR, the user's main view is a first person view, mostly seeing their hands and arms when immersed. In order to configure the avatar, the user needs to recognize him/herself, and having a mirror inside VR is helpful for the purpose.

By using full size human figures, we enable an accurate representation of scale. It is important to humanise the avatars, to uphold the ethos of human-to-human interaction (as opposed to perceiving our collaborators as monsters or robots). Adding facial characteristics and facial expressions also minimises the need for name tags and other forms of identifiers (Figure 3.4.1c and Figure 3.4.1d).

While we think it is important to have a full-body tracking photorealistic avatar, we are aware of the current limitations in precision of the motion tracking and restrictions in VR performance. The default PrismArch avatar can therefore be simplified to develop a proof of concept, with lower resolution of human body geometries and head and motion controller tracking at a minimum. PrismArch default avatars would also need to include the user’s company logo, user’s name and user’s project role (e.g Company Directors, Admins, Management, Designers, Engineers) and these can be colour coded in a discipline registered company colour.

The user will need to set the correct eye height and tracking environment for the best immersive experience (see NVIDIA Holodeck example, Figure 3.4.1e). The calibration optimization UI is required as part of the PrismArch on-boarding process. Navigation sensitivities and field of view should be adjustable across the platform, and also require the relevant UI.

Note: Each IH is always created inside the dedicated Personal Work Sphere (PWS), which is contained within a discipline-specific Collective Work Sphere (CWS).

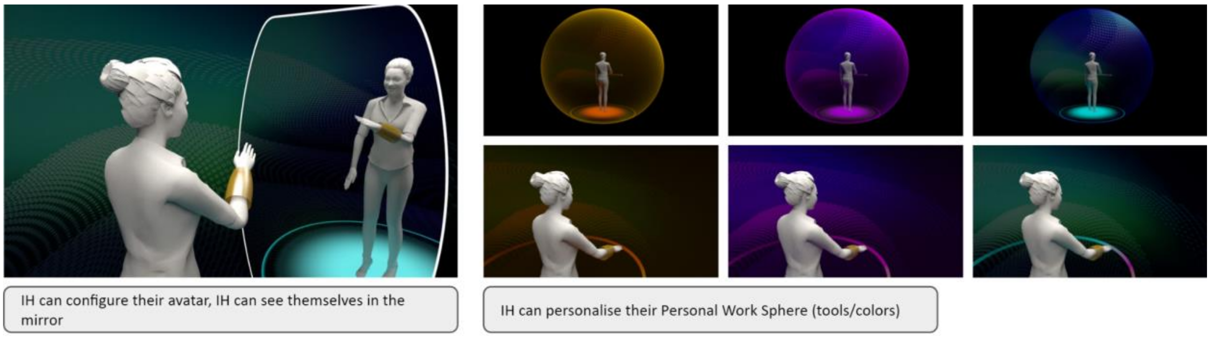

IH can configure their avatar, IH can see themselves in the mirror
IH can personalise their Personal Work Sphere (tools/colors)

Figure 3.4.1b - Avatar Configuration inside PrismArch Personal Work Sphere

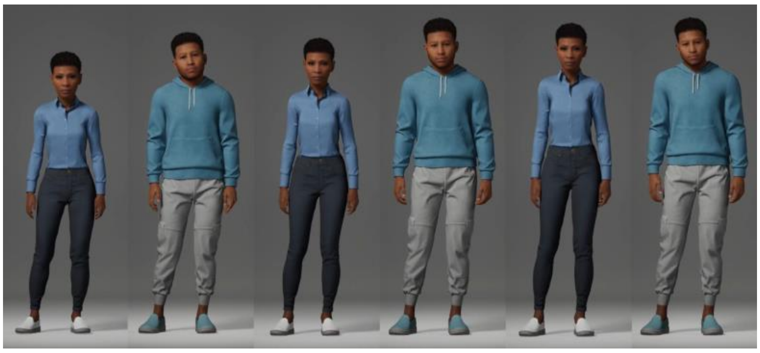


Figure 3.4.1c - Examples of the User Interface to configure **avatar height**, the existing plugin MetaHuman inside Unreal Engine 4

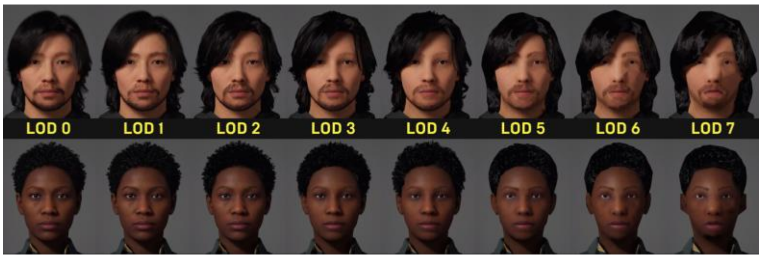
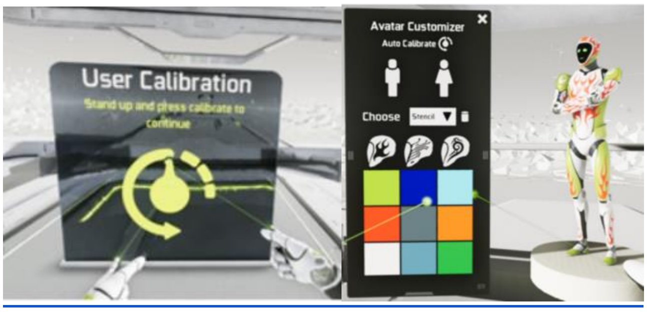


Figure 3.4.1d - Examples of the User Interface to configure avatar **LOD**, the existing plugin MetaHuman inside Unreal Engine 4



User Calibration

Stand up and press calibrate to continue

Figure 3.4.1e - Example of the existing application to calibrate an avatar inside VR, Holodeck User Guide

(2) **User Interface**

Below are three basic User Interface types that can be accessed inside the PrismArch VR platform:

- **Personalisable and Customisable User Interface** (with sub-widgets and shortcuts)
- **Billboard User Interface**
- **Query User Interface (with Metadata Nodes)**

Personalisable and Customisable User Interface

The Personalisable and Customisable User Interface is attached to the Personal Work Sphere (PWS). *'Personalisable'* is defined as a system tailored platform content for individual users, while *'customisable'* is defined as the content being arranged *by the user (D6.1, sct.1.1).*

Requirements: The User Interface should be able to be minimised and maximised according to different usage scenarios. Whenever the user interacts with different PrismArch class types, views and modifies content with different user scales, or seeks to avoid occlusion issues, the UI can be adapted to increase work efficiencies (Figure 3.4.1f). The UI size and user display distance can be adjusted according to the ergonomic information registered for that specific user.

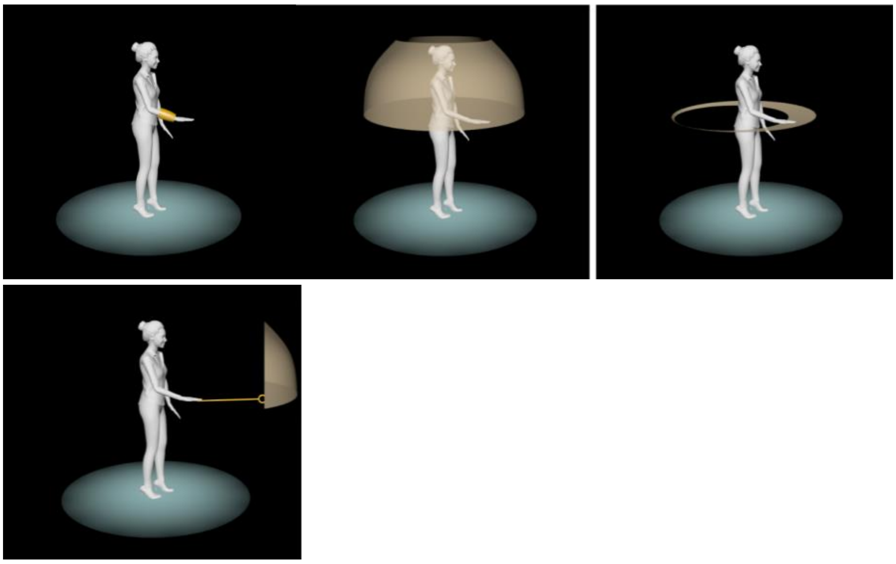


Figure 3.4.1f - PrismArch User Interface types and Interaction modes

Top (from left): the user avatar hand interacting with small UI, medium UI or large UI

Bottom: the user avatar laser/ray interacting with the medium UI

The Personalisable and Customisable User Interface will need to contain the following tools and functionalities (D6.1, sct. 1.1):

- Task bar widget

- Dashboard widget
- Preferences widget
- PrismArch Functionalities and Proprietary Functionalities
  (e.g. Administration Tool, Commenting and Markup Tool, Clipping Plane Tool, Toggle Camera Tool, Toggle View Mode Tool, Specialist and Proprietary Tools)

The user interacts with these widgets with the avatar hand. Both Personalisable and Customisable User Interface can be scaled based on the user's preference. In the condition that the UI is enlarged, the distance between the user and the UI should be adjusted automatically considering the user control-display ratio. The user interaction mode can switch to a ray mode to interact with the widgets if the UI is located outside of the user's reaching region. The switch, however, must be intuitive and the transformation should be smooth enough to avoid user distraction.

"*At a higher level of platform operation, the platform offers system embedded and persistent functionalities. These PrismArch CORE functionalities enable all the IHs to co-exist and work collaboratively inside the PrismArch World (PAW)*". (D6.1, sct.1.1)

The functionalities listed and requested in D1.1 and D5.1 also require interfaces to enable/disable/execute the content. For frequently accessed functionalities, it would be helpful to have shortcut icons, similar to the examples in Figure 3.4.1g.

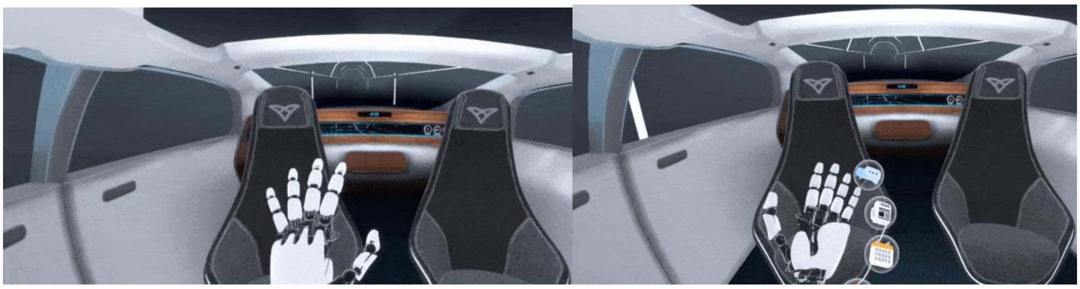


Figure 3.4.1g - the next generation of XR cabin experiences in autonomous cars, UltraLeap

PrismArch Core Functionalities

- Administration functionality
- Commenting and Markup functionality
- Clipping Plane functionality
- Toggle Camera functionality
- Toggle View Mode functionality

Specialist and Proprietary Tools

- Mindesk (as Rhino, Grasshopper and Revit connectors)
- Zoom, Skype and/or alternative Steam or other Voice Chat functionalities

- Outlook Email
- Specialist tool _ architect
- Specialist tool _ structural engineer
- Specialist tool _ MEP engineer

Please visit D5.1 and D6.1, sct 1.1 for the full list of functionalities.

Billboard User Interface

The Personalisalbe and Customisable User Interface outlined above is connected to the Personal Work Sphere. In addition to this user specific UI elements, the PrismArch VR platform requires project specific information display, or Billboard type UI, loaded into a 'world'[1] inside the PrismArch Project Sphere. The Billboard User Interface has a range of applications. For example, it could be a display to visualize a specific or queried project information, and/or it could function as a shortcut to another UI. Note that any interacted content should reflect the registered discipline colours when the content is hovered, highlighted or selected (Figure 3.4.1h).

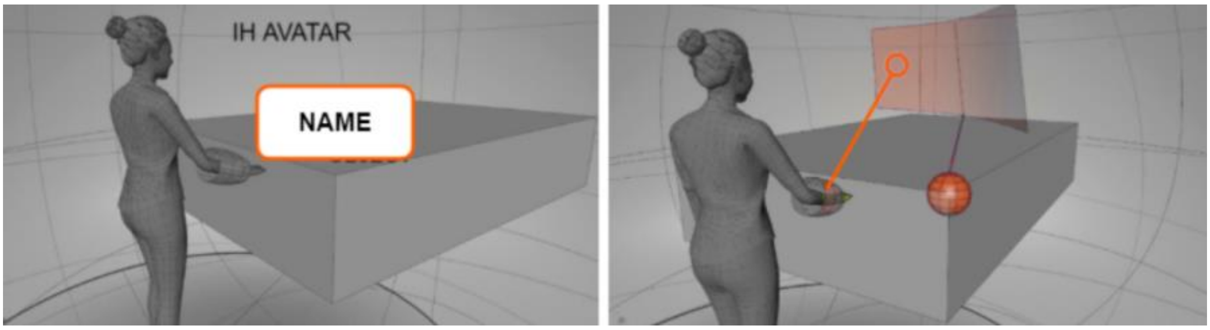


Figure 3.4.1h - Interacted UI colour reflecting to the user registered discipline specific colour code
Left: the user interacts with a Billboard UI with the hand and highlighted in a discipline specific colour
Right: the user interacts with a ray attached to the hand and highlighted in a discipline specific colour

Example of colour codes for each discipline are suggested as follows:

- Guest/ Default: White or a random colour from a list, #ffffff, (255,255, 255)
- Arch: Orange, #ff6600, (255,100, 0)
- Struct: Green, #00ff00, (0,255, 0)
- MEP: Blue, #00ffff, (0,255, 255)

Figure 3.4.1i - Examples of PrismArch discipline specific colour codes

[1] [Unreal Engine] Unreal Engine Documentation, "Unreal Engine 4 Terminology", URL:https://docs.unrealengine.com/4.26/en-US/Basics/UnrealEngineTerminology/

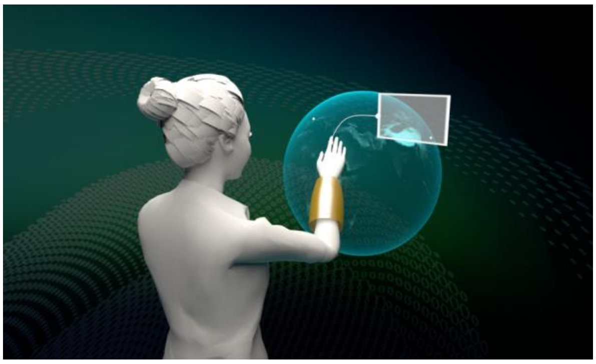


Figure 3.4.1j - the Immersed Human and the PrismArch Billboard User Interface

The Billboard User Interface can also be attached to visualized Metadata. PrismArch Metadata can be compared to 3D and immersive folder structure. Metadata nodes, which are explained later in this section, are representations of diverse content (formats) and the structure enables the users to view and interact with large volume and complex information in an organised manner and to describe project information from a holistic point of view. The Billboard User Interface can solely exist to project text and thumbnail information and/or to interact to load another UI (Figure 3.4.1i).

Please visit *3.6.1.a User Interface Design for On-boarding* to see the full storyboard.

Query Interface (with Metadata Nodes)

Please visit *3.6.1.d User Interface Design for Content Query and Demarcation for more info*.

In summary, it is recommended to have at least three basic types of User Interface Designs inside the PrismArch VR platform; Personaliable and Customisable UI, Billboard UI and Query UI. These User Interfaces can be anchored to either the Personal Work SPhere or PADO. We would like to highlight that the user experience must be consistent regardless of what type of User Interface the user is interacting with.

(3) PrismArch Design Objects (PADO)

A PrismArch Design Objects (PADO) is defined as a project-related object produced inside the PrismArch World by the on-boarded users (D6.1, sct.1.1). PADO is a project specific item with a tag containing information about the author, their organisation and discipline.

Requirements: A user should be able to scan PADO information and read the tag content by highlighting the object. The information will need to pop up at a suitable location based on the distance between the user and interaction target. This may be near the hand of the avatar, in front of the user camera (HUD), within the user's Personalised UI, or by the metadata node depending on the control distance ratio. For a more detailed explanation and design proposal please visit *3.6.1.b User Interface Design for Personal Work Sphere*

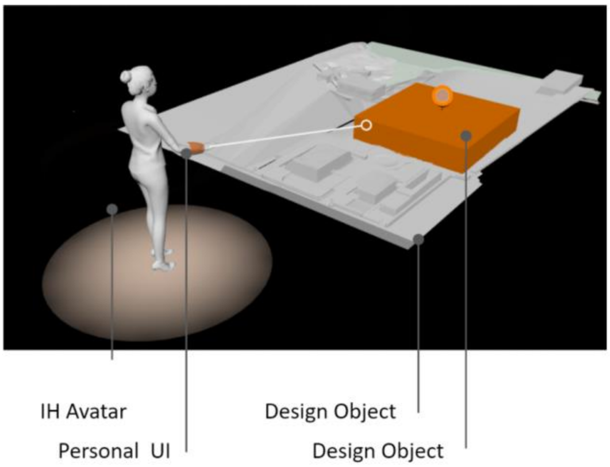


Figure 3.4.1k - the Immersed Human interacting with a PrismArch Design Objects (PADO) using a transformable user interaction tool - ray mode

(4) PrismArch Metadata

PrismArch Metadata (PAM) is defined as a set of project information and data that is referenceable to other sets of project information and data (D6.1, sct.1.1). Metadata is created by authorised users or through PrismArch automated functions. It is located inside the PrismArch Singular Database (PASD).

Requirements: Metadata can contain reference information or information associated with PADO. The associated PADO metadata can be visualised as a simplified geometry in order for the Immersed Humans to access and examine the data inside VR (Figure 3.4.1l). The users should be able to query and preview the metadata information inside VR (if they have access privileges) by highlighting a node or multiple nodes. The search results should then be displayed to the user for further manipulation and examination. For further study about user interaction with the Metadata Nodes, please visit *3.6.1.d User Interface Design for Content Query and Demarcation*.

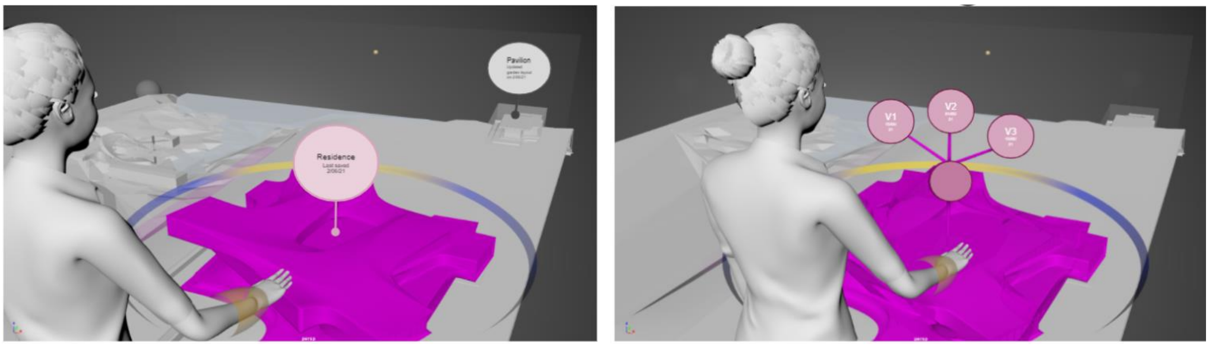
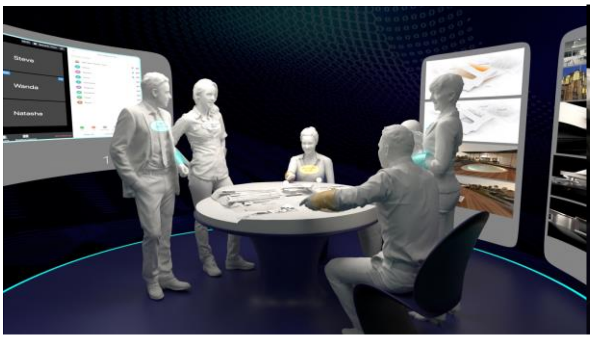


Figure 3.4.1l - the Immersed Human interacting with a Design Object and Metadata Nodes

3.5 Discipline-Specific Requirements for the VR Interface

3.5.1 Architecture



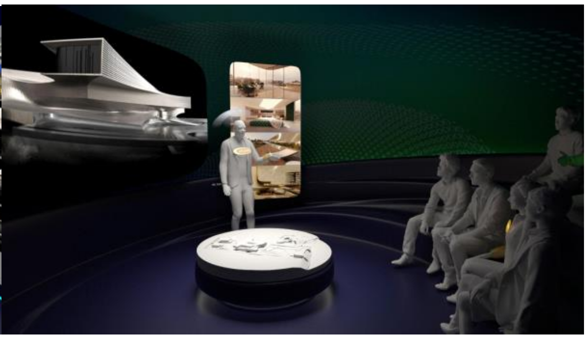


Figure 3.5.1a (left) - Meeting Sphere for an internal discipline specific or cross disciplinary project **review** with curated PrismArch Design Objects (PADO)
Figure 3.5.1b (right) - Meeting Sphere for a project **presentation** with curated PrismArch Design Objects (PADO)

The UI interface requirement of the Architecture discipline follows closely the Core Requirements for the VR Interface. This is due to the wide range activities undertaken by designers at various design stages, described in greater detail in previous deliverables D.1.1 and D.6.1. These activities include, but are not limited to the below:

(1) Design Activities: sketching, 3D modeling, drawing, Informed by references to the project context (existing conditions), regulatory framework, cultural references

(2) Collaboration between members of the team internally within a discipline as well as with the members of the wider design team.

(3) Coordination of input of all disciplines - Implies a requirement for clear demarcation of different disciplines' scopes, maintenance of the authorship and IP protection.
A single model within one environment will help to develop a common holistic understanding of design by all parties; enable the team to identify collisions.

(4) Arranging meetings with the Client. Involving preparation of comprehensive presentations documenting current status of the design progress, using photorealistic polished visualisations, using cultural references, coordinated multidisciplinary input)

(5) Maintaining record of design decisions made during the lifecycle of the project. (a requirement to be able to retrieve historical data from the project archive for a period of up to 15 years).
Adequate security is necessary to safeguard project records, taking full account of data protection legislation, and safeguarding clients’ confidential information.

(6) Practice and project management.
Overview of all projects, managing division of work within teams; monitoring progress of individuals and project teams.

Please visit 3.6.1 Cross-Disciplinary Interface and 3.6.2 Architecture-Specific UI Elements for full scope of what the architectural discipline would like to request for the PrismArch platform.

■ 3.5.2 Structural Engineer

Despite the rapid development in immersive technologies including more and more compact HMDs, larger Powerwalls and CAVEs, we encounter scepticism in the structural engineering discipline for introducing VR into the workflow. [McCabe 2015] attributes such resistance to the industry's 'extensive use of Revit and other 3D-modelling programs', and concludes that converting existing BIM models should only require modest resource commitment. Reflecting on the previous sections outlining the challenges with VR optimisation, we hesitate to draw a similar inference (sct 2.3.4).

In a few research papers examining VR-integrated workflow, such as one by [Zaker 2018], we discover that the structural engineering practice has been omitted from case studies. [Steed 2017] presents a series of engineering use of VR during design reviews and operational training, yet goes on to comment that 'despite the opportunities demonstrated, engineering is still a challenging area for VR, as engineering models are large and complex.'

The lack of effective examples for interrogating structural engineering results in VR assures us that there are significant opportunities in this area, but also signifies the amount of research needed to identify the most effective interface design. These upcoming sections document closer examinations on this topic.

Structural Engineering Workflows

We have initially outlined the following steps as an assumption of how structural engineers will incorporate the PrismArch VR toolkit into their existing workflow:

1) *Outside or Inside* PrismArch - The structural engineer examines an architectural model and produces an abstract model. If the Rhino modelling tools are used, this process could be undertaken either outside or inside the PrismArch environment (via the Mindesk API). This abstracted model will form the geometric input for the structural model.
2) *Outside* PrismArch - The abstracted geometric model is loaded into external Structural Engineering software, and structural engineering-specific properties are assigned, to generate the complete structural model.

3) *Outside* PrismArch - This structural model is analysed and run in the same external software.
4) *Outside* PrismArch - The structural engineering results generated by this analysis (along with the original properties, or 'inputs' of the model) are exported from this external software and imported into the PrismArch Database.
5) *Inside* PrismArch - These results can now be visualised and interacted with using the PrismArch VR Toolkit.

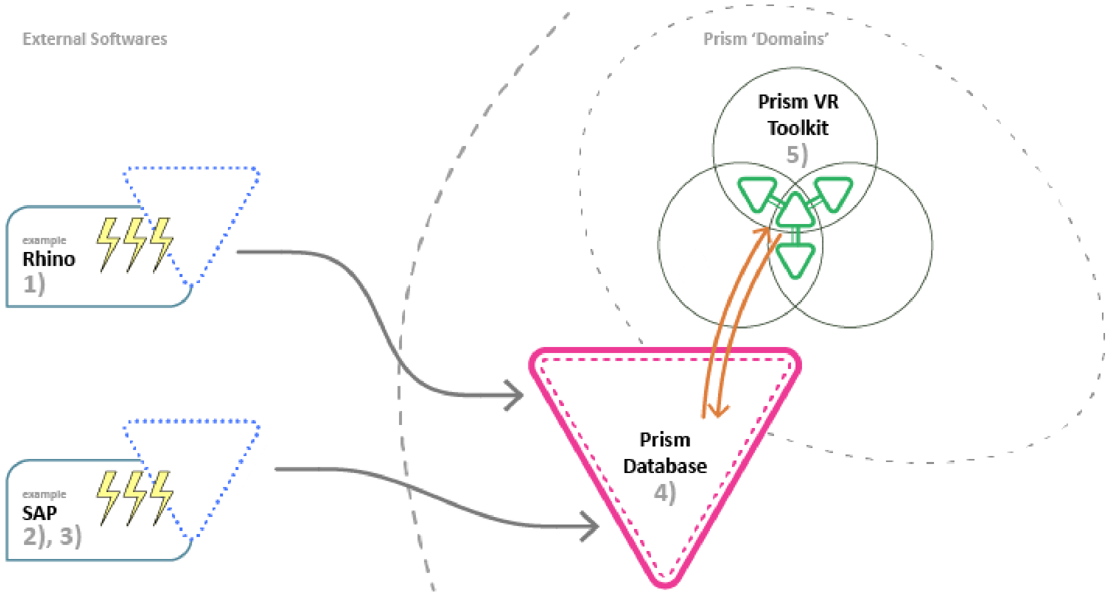


Figure 3.5.2a - Diagram illustrating how PrismArch could be incorporated into structural engineer workflows

This identified workflow is only a hypothetical analysis, however, and does not explain the structural engineering processes to a suitable level of detail. Therefore, we undertook a range of studies to interrogate these assumptions. This section of research builds upon and complements the prior analysis undertaken in D3.1, by contributing the next level of detail to this process analysis research.

Structural Engineering Interviews - Phase 1

We organised a series of semi-structured interviews with engineers at different experience levels in AKT. The interview goal was to learn how structural engineers visualise, interact with, and export structural models currently with the digital tools they have, and where they feel there are challenges and opportunities.

Phase 1 - Methodology

The planning of these interviews follows the 'jobs-to-be-done' technique. As [Laubheimer 2017] highlights, such a method focuses on user problems and needs with the additional ‘behavioral and attitudinal details’. It presumes that “*whenever users “hire” (i.e., use) a product, they do it for a specific “job” (i.e., to achieve a particular outcome). The set of “jobs” for the product amounts to a comprehensive list of user needs*.”

To synthesise software features from these problems and needs, we carried out design exercises to determine the most essential few functions to be included in the minimum viable product (MVP https://en.wikipedia.org/wiki/Minimum_viable_product) development. We determined specific use case scenarios and arranged workshop sessions to map out user stories which help us understand how all the features should fit together.

As explained by [Kaley 2021], user story mapping is a widely used ‘lean’ method in user experience design. It ‘outlines the interactions that the team expects users to go through to complete their goals in a digital product’. A detailed user story map usually depicts three levels of user actions at increasing granularity from top to bottom. Even though the levels are often called by varying names - such as ‘activities’, ‘tasks’, and ‘stories’ - they are stacks of actions displayed in sequential order across the map.

This ‘lightweight representation’ of the software features not only presents a clear picture of the structural engineering design activities but also provides a valuable overview of how activities within different disciplines flow together and intersect each other. This has greatly informed the design process for the aforementioned cross-disciplinary functions in the toolkit.

Phase 1 - Outcomes

Based on the varying structural engineering use cases, we have generated four scenarios where structural results will be shown inside the PrismArch VR. Each scenario is then broken down into specific actions the engineer undertakes to perform their job.

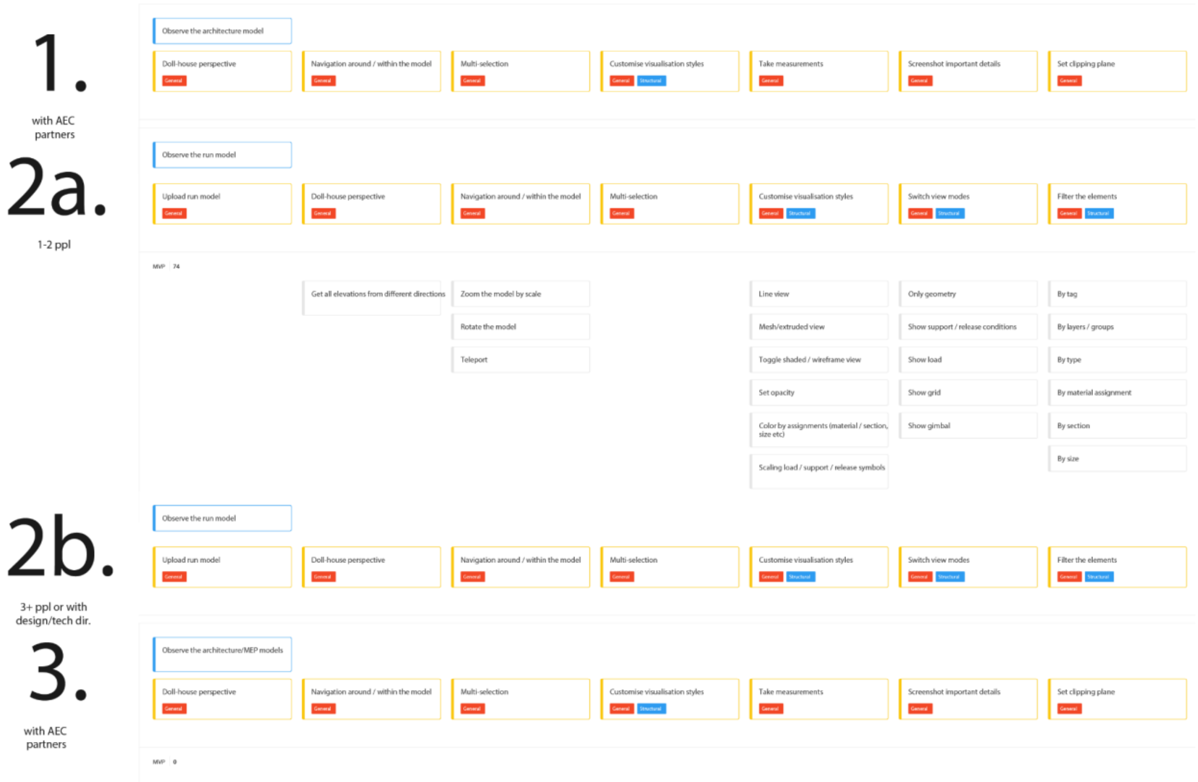


Figure 3.5.2b - Structural engineer user stories (See detailed views within **PrismArch_D1.2_ADDENDUM**)

1 Briefing with the architect during early-stage design. The engineer would likely show an overview of the model without going into too many details. This is to demonstrate a rough draft of what they are about to analyse.

2 During a structural engineer design review.

a. Reviewing solo or with another team member. Visualising the structural results would help the engineer build up an all-in-one mental model for how the structure behaves. This is a form of sanity check. They would examine if the structure was built correctly to the initial assumptions and that the assumptions themselves were correct.

b. Presenting internally to other engineers and design/technical directors, e.g., for mini-milestones. The engineer would present a compiled model where one can interrogate how it is modelled and how it is acting. They would review the calculated results by grouping them and everything would be shown together as a designed system.

3 Presenting to external AEC partners, e.g., during a design team meeting. The engineer would produce a condensed package explaining the issues or providing a solution. There is a high level of abstraction in these presented results. They are not literal but have been selected and filtered corresponding to a specific review objective.

Structural Engineering Interviews - Phase 2

We have conducted the second round of interviews with the same group of engineers as above. These workshops delved deeper into detailed actions identified from the user story map. Looking at these actions again from a fresh perspective, the engineers re-sorted their processes into categories, which will help formulate the information architecture supporting the interface design (Figure 6).

Phase 2 - Methodology

A powerful tool to uncover the mental model of users from a specific knowledge domain is card sorting. It is widely practised during the stages leading up to MVP development, where an information architecture is being solidified to show how the user interface could be structured.

[Sherwin 2018] argues that a card sorting exercise is often delivered to participants ‘according to criteria that make sense to them’. Establishing the suitable grouping of user actions could later ensure that we create the interface features matching with users’ expectations. Observing participants undertake this task can also reveal to us some concealed user habits and attitudes.

Phase 2 - Outcomes

The procedure we implemented in this section belongs to the type of method called ‘open card sorting’. The engineers were each instructed to group actions in categories that seem appropriate to them. Some similarities of grouping are starting to emerge as we repeat this exercise.

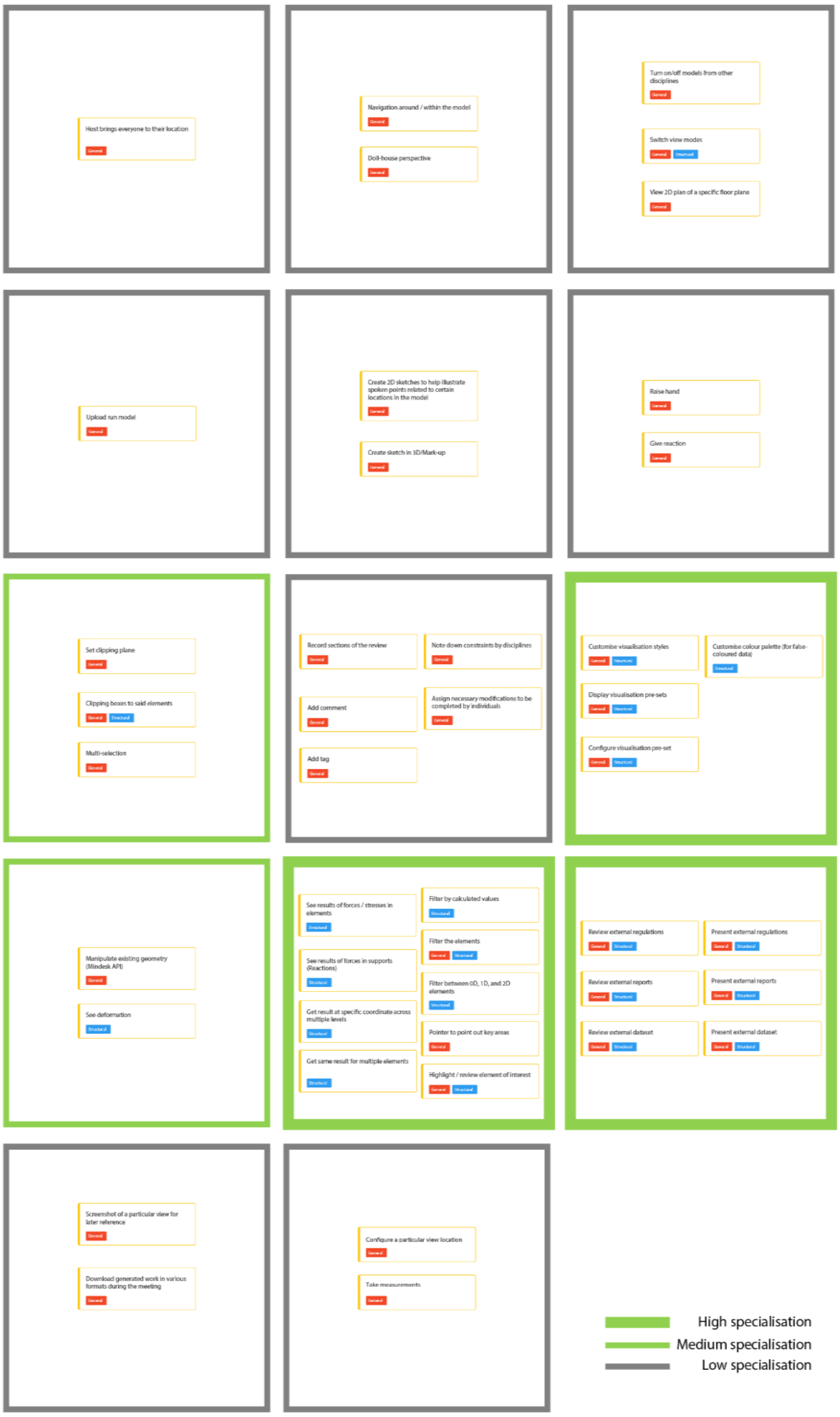

Host brings everyone to their location
General
Navigation around / within the model
General
Doll-house perspective
General
Turn on/off models from other disciplines
General
Switch view modes
General
Structural
View 2D plan of a specific floor plane
General
Upload run model
General
Create 2D sketches to help illustrate spoken points related to certain locations in the model
General
Create sketch in 3D/Mark-up
General
Raise hand
General
Give reaction
General
Set clipping plane
General
Clipping boxes to said elements
General
Structural
Multi-selection
General
Record sections of the review
General
Note down constraints by disciplines
General
Add comment
General
Assign necessary modifications to be completed by individuals
General
Add tag
General
Customise visualisation styles
General
Structural
Customise colour palette (for false-coloured data)
Structural
Display visualisation pre-sets
General
Structural
Configure visualisation pre-set
General
Structural
Manipulate existing geometry (Mindesk API)
General
See deformation
Structural
See results of forces / stresses in elements
Structural
Filter by calculated values
Structural
Filter the elements
General
Structural
See results of forces in supports (Reactions)
Structural
Filter between 0D, 1D, and 2D elements
Structural
Get result at specific coordinate across multiple levels
Structural
Pointer to point out key areas
General
Get same result for multiple elements
Structural
Highlight / review element of interest
General
Structural
Review external regulations
General
Structural
Present external regulations
General
Structural
Review external reports
General
Structural
Present external reports
General
Structural
Review external dataset
General
Structural
Present external dataset
General
Structural
Screenshot of a particular view for later reference
General
Download generated work in various formats during the meeting
General
Configure a particular view location
General
Take measurements
General
High specialisation
Medium specialisation
Low specialisation

Figure 3.5.2c - The groupings and priorities after card sorting (See detailed views within **PrismArch_D1.2_ADDENDUM**)

Structural Engineering Operations

To formulate the discipline-specific features required in the user interface, we have to first understand what types of results the structural engineers hope to extract from them.

Inside the PrismArch VR, three types of operations are imperative for structural engineers to engage with structural results: *displaying*, *interacting*, and *exporting*.

Displaying	**Input Conditions**	Cross-sections		
		Different material types		
		Support conditions	Fixity	
			Rotation	
		Load conditions		
	Results	3D results *(‘abstracted’/’externally represented’)*	Forces	Bending moment diagram
				Shear diagram
				Axial
				Torsion
		3D results *(‘on geometry’)*	Colouring	Forces *(e.g., Bending, Axial Torsion, as above)*
				Utilisation
		Numeric results	Reactions	
			Number values from interaction	
Interacting	**Filtering the Results** *with physical criteria*	Clipping planes		
		Clipping boxes		
	Filtering the Model *with numeric criteria*	e.g., Only see Utilisation data, in the range of X to Y		
		e.g., Only see Deformation data, in the range of X to Y		
	Scaling Results	Deformation/ Displacement *(with variable scaling)*		
Exporting		Screengrabs		
		Tabulated Data		

Structural Presets

One of the primary user needs we have uncovered through this round of interviews was the concept of structural 'presets' to help engineers arrive at pre-selected results quickly.

As indicated by the previous workflow section, structural engineers perform complex calculations and prepare structural models within specialised desktop software. The results of these calculations are then imported into the PrismArch VR platform to be visualised and interacted with. It would significantly increase work efficiency if the commonly occurring user selections are configured as presets. For instance, through the click of a button, the engineer will be able to promptly present some of these refined results during later design review (see P.143 for descriptions of this scenario). This functionality is also in support of the critical 'Tagging' and 'Query' functions described in D1.1, Section 4.4 - *Requirement Collected*.

Precedent Studies

We have discovered two precedent examples that showcase potential presets beneficial for structural engineering operations. Both of these interfaces are still at early stages in their development, but they are compelling instances for demonstrating the user needs.

Case Study: EulerVR

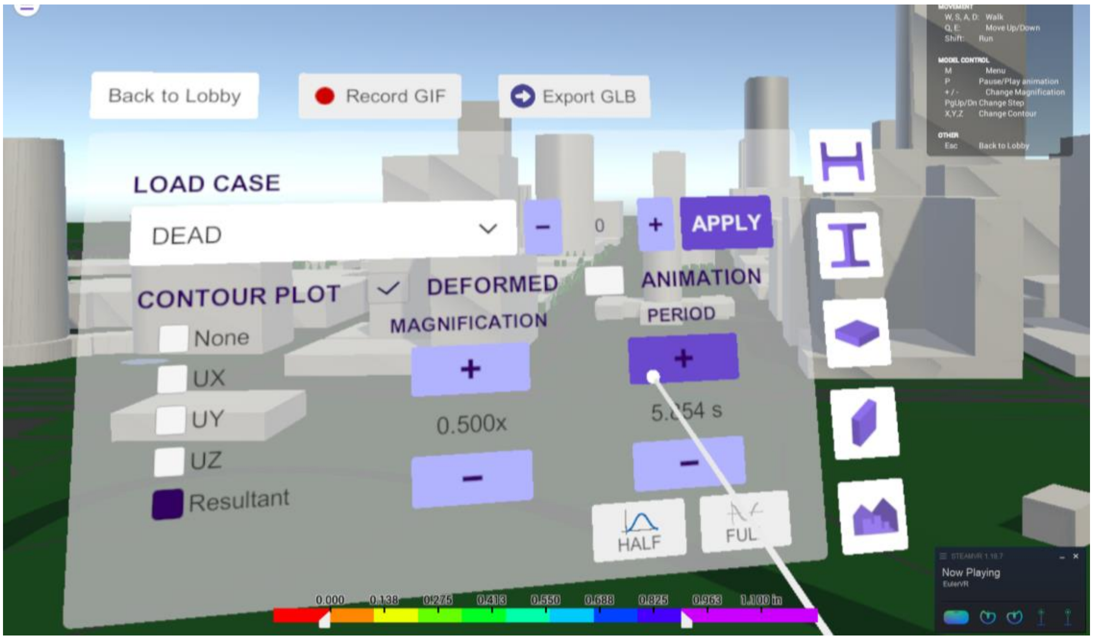


Figure 3.5.2d - the user interface in EulerVR

[EulerVR] offers an incredibly straightforward toolkit for structural engineers to visualise and animate some of their calculation results. The user interface in their beta access is stripped back to a set of most basic functionalities, including:

- *A colour gradient bar with a customisable range.*
- *Filtering elements by type (i.e. cross-sections, beams, slabs, walls, background, etc).*
- *Load cases (i.e. Dead Load, Live Load, etc).*
- *Contour plot.*
- *Deformation scale.*
- *Animation speed.*

Some of the user feedback we have received from initial usability testing are:

- The user would like to better customise the colour gradient feature (e.g., colour, range).
- They stressed the importance of seeing the value ‘0’ indicated on the colour contour.
- They wanted to be able to highlight multiple points/elements on the model, locking in place the info panel attached to each.
- The animation of the deformation should last roughly 8 seconds for optimal understanding.
- They mentioned a desire for more options to filter and hide elements (e.g. by groups, layers, typology, 0D, 1D, 2D, etc.)

Case Study: SolidVR

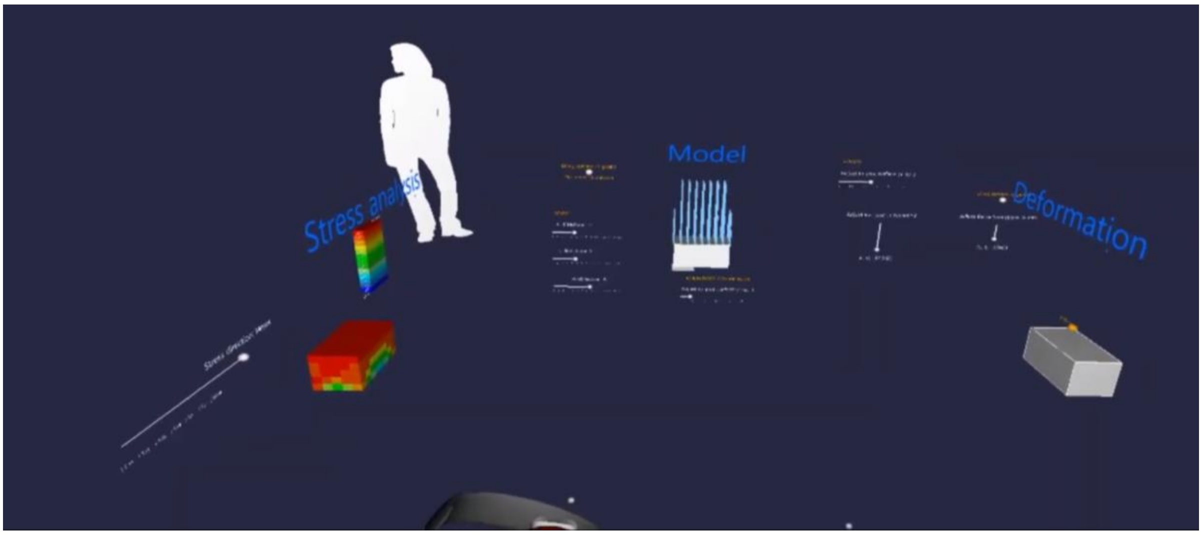


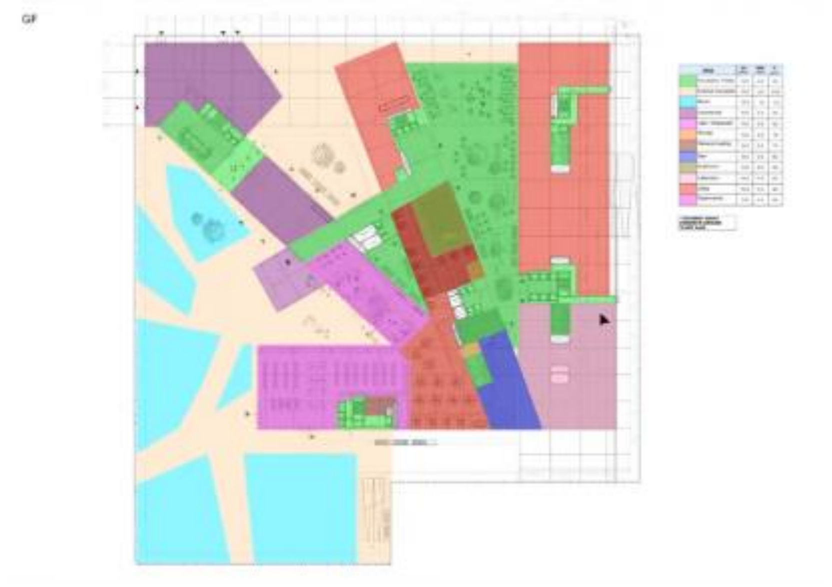
Figure 3.5.2e - the user interface in SolidVR

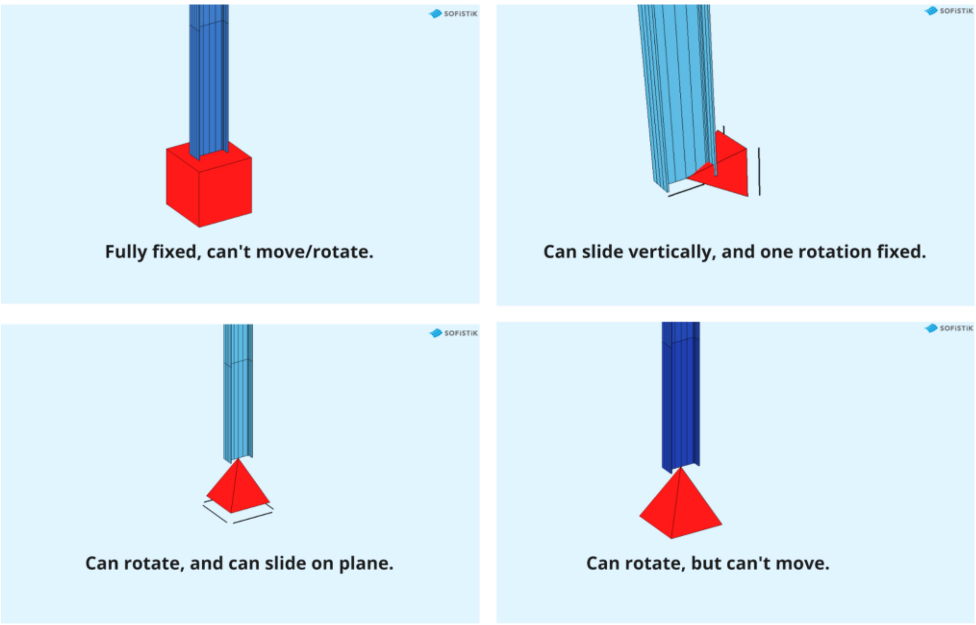
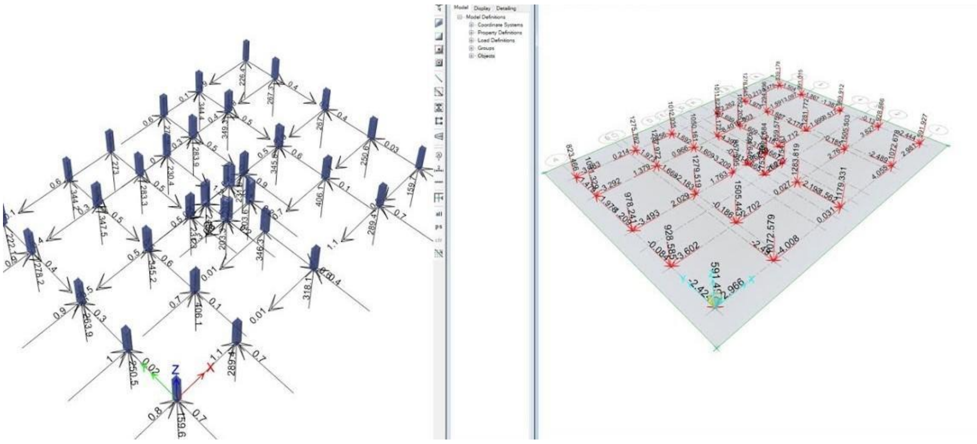
SolidVR is a prototype interface by [Akselsen 2019] based on Mindesk API to post-process components from Grasshopper in VR. To investigate the benefits of VR for structural design, they created ‘an analysis program meshing solids into 8-node hexahedron elements’. Their interface focuses on chosen use cases including stress analysis, model manipulation, and deformation, which are some common analytical needs for structural engineers.

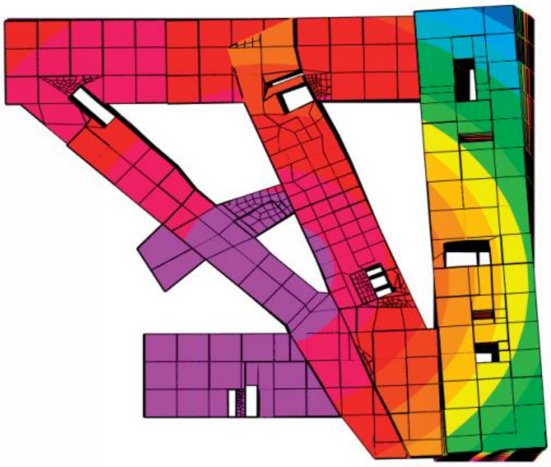
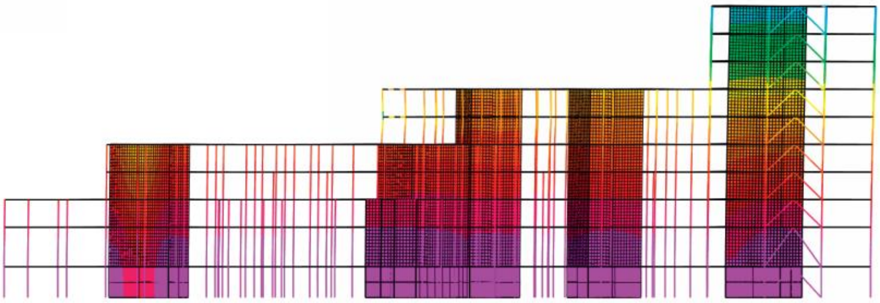
Proposed PrismArch Presets

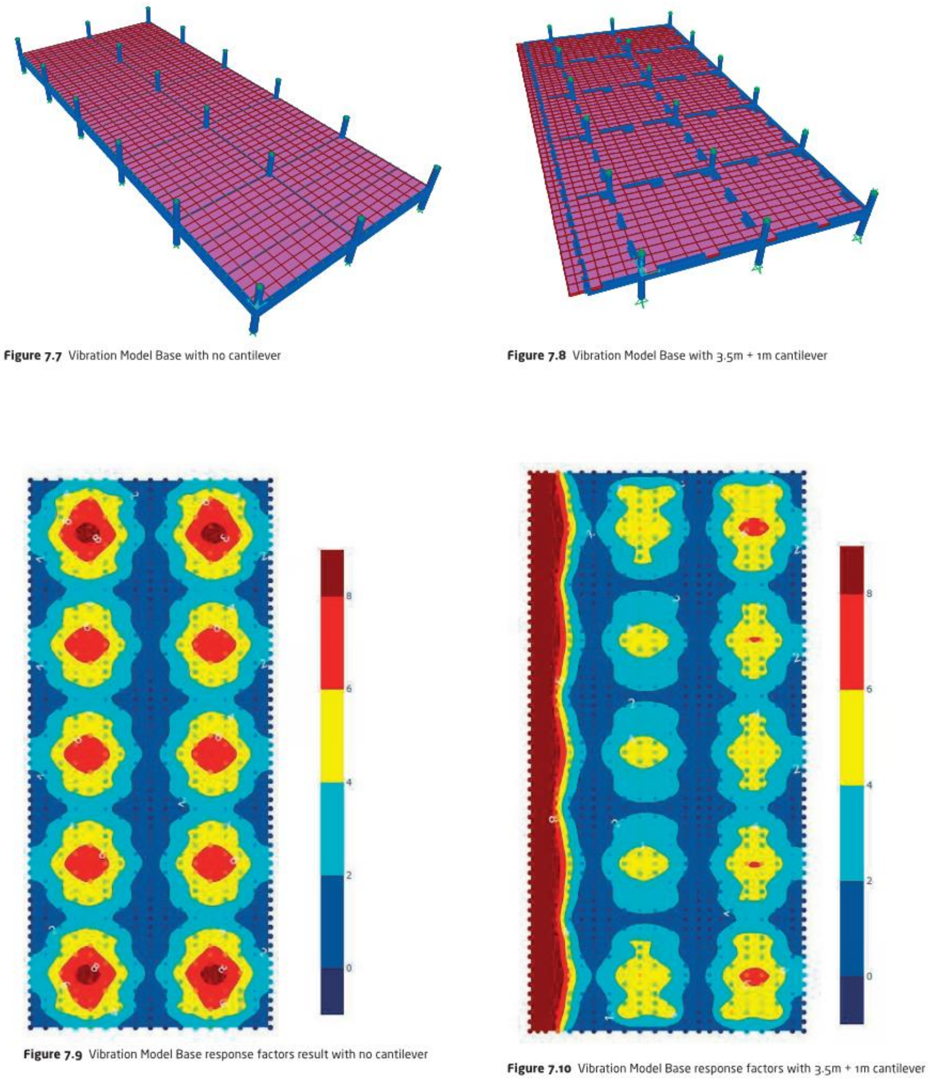
We propose two categories of structural engineering presets on different abstraction levels:

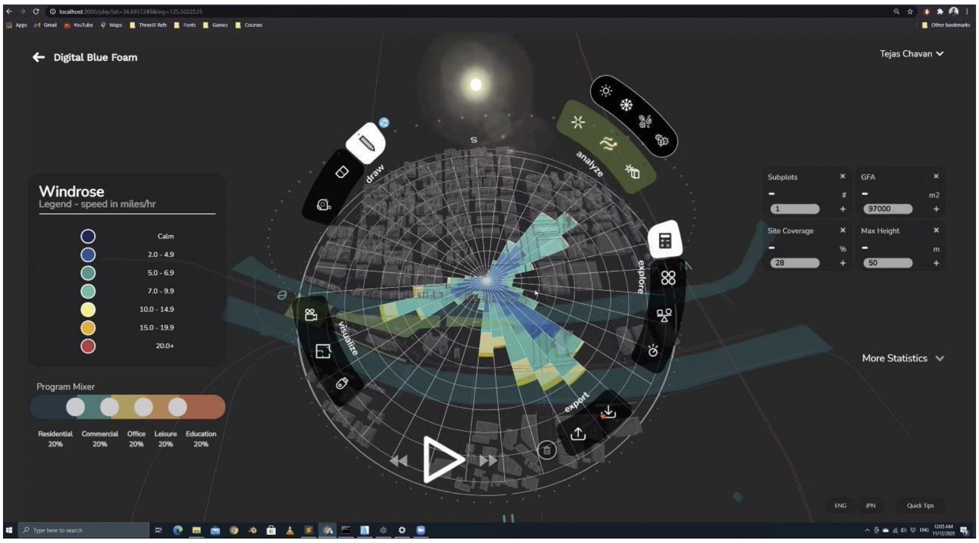
Macro Presets

Preset Type	Visual Style	Filter Elements	Select Conditions	Numeric Values	Scaling Factor
Loading Diagrams	Customised colours to differentiate area	Filter by pre-defined area	Select which load case	Customised range	
	 Figure 3.5.2f - example of a loading diagram				
Support Conditions		Filter by condition	Select which support condition		

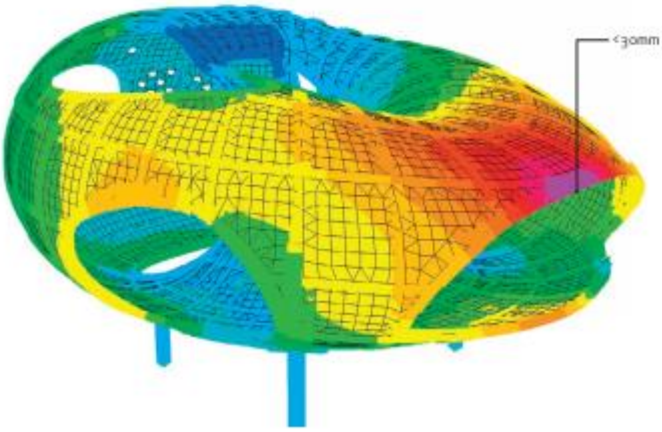
	 Fully fixed, can't move/rotate. Can slide vertically, and one rotation fixed. Can rotate, and can slide on plane. Can rotate, but can't move. Figure 3.5.2g - the support conditions illustrated in SOFiSTiK				
Support Reactions		Filter by range			
	 Figure 3.5.2h - example of support reactions				
Modal Cases for the Overall Stability	Customised colour gradient for cross-referencing, from 0 to max or custom threshold	Show all elements	Select which modal case		

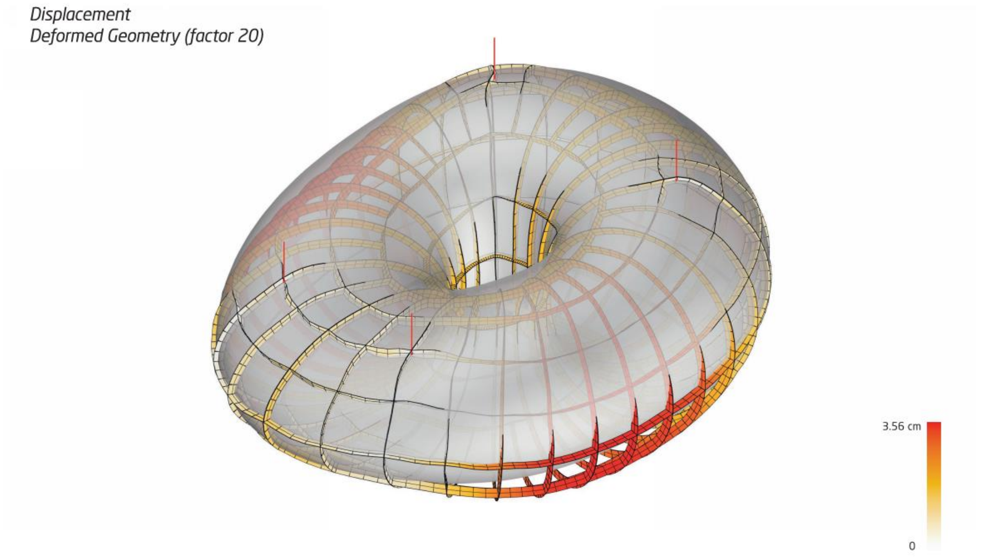
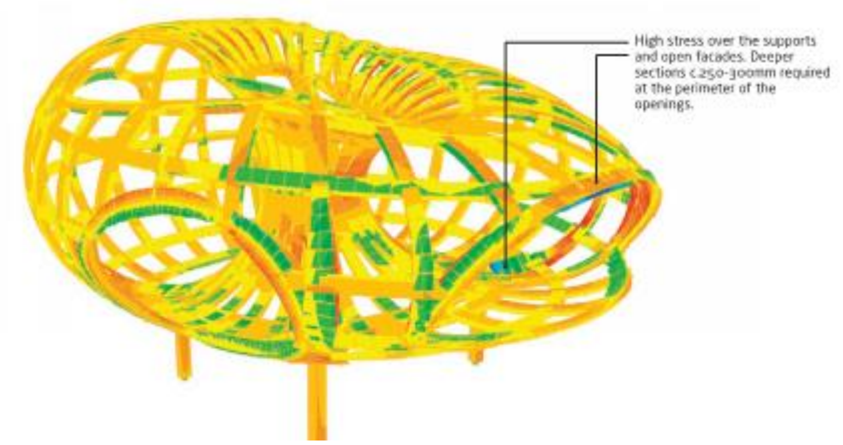
	 **Figure 8.6** Modal response of structure  **Figure 8.7** Lateral displacement due to earthquake response spectrum analysis Figure 3.5.2i - example of modal cases				
Vibration Plots for Specific Portions	Customised colour gradient for cross-referencing, from 0 to max or custom threshold	Filter by level	Select which designed option		

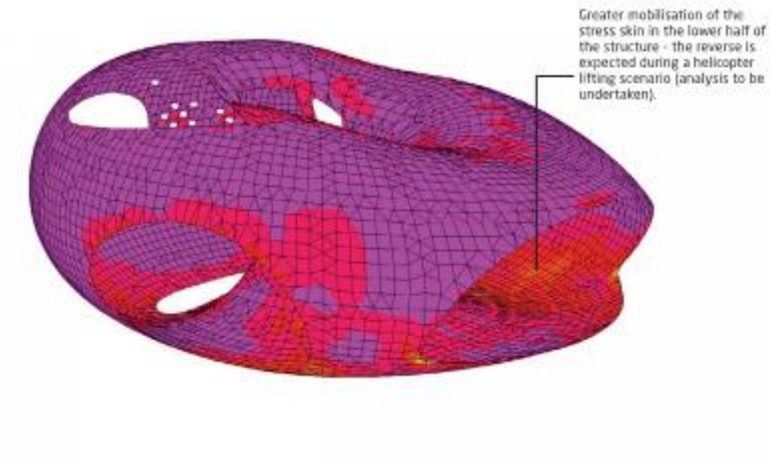
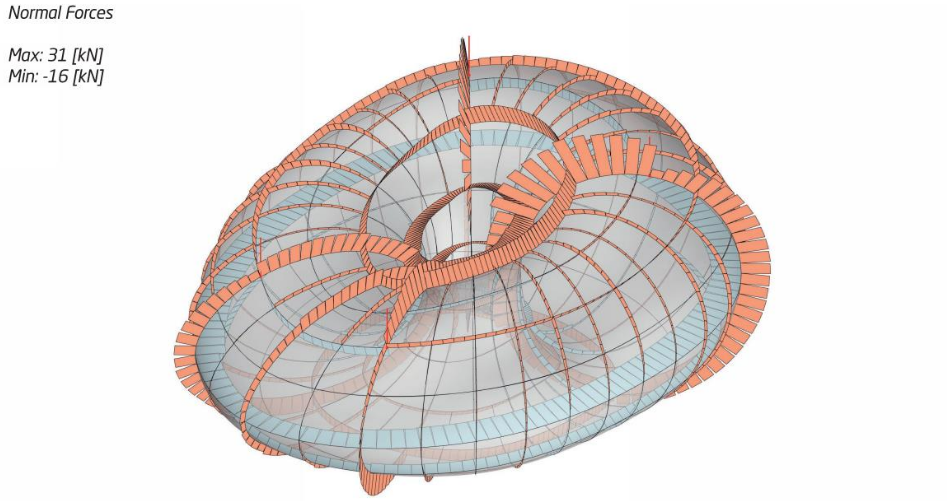
	 Figure 7.7 Vibration Model Base with no cantilever Figure 7.8 Vibration Model Base with 3.5m + 1m cantilever Figure 7.9 Vibration Model Base response factors result with no cantilever Figure 7.10 Vibration Model Base response factors with 3.5m + 1m cantilever Figure 3.5.2j - example of vibration plots for two design options				
Overall Building Conditions (e.g., Material Quantity, Cost, etc)	Customised colour gradient for cross-referencing, from 0 to max or custom threshold	Show all elements	Select which building condition		

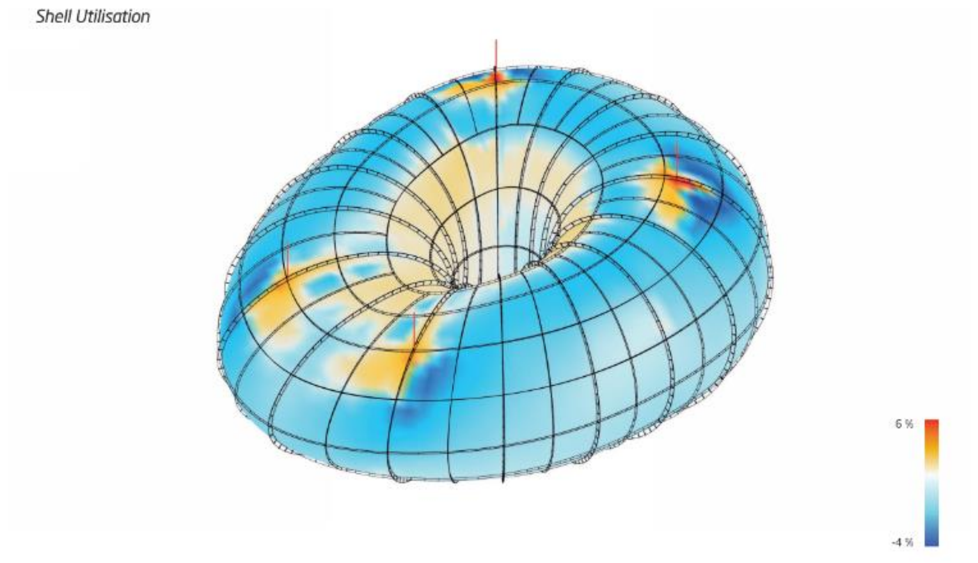
	 localhost:3000/play?lat=34.6937249&lng=135.5022535 Apps Gmail YouTube Maps ThreeJS Refs Fonts Games Courses Other bookmarks Digital Blue Foam Tejas Chavan Windrose Legend - speed in miles/hr Calm 2.0 - 4.9 5.0 - 6.9 7.0 - 9.9 10.0 - 14.9 15.0 - 19.9 20.0+ Program Mixer Residential 20% Commercial 20% Office 20% Leisure 20% Education 20% S draw analyze explore visualize export Subplots 1 GFA 97000 m2 Site Coverage 28 % Max Height 50 m More Statistics ENG JPN Quick Tips Type here to search Figure 3.5.2k - example of a windrose diagram in Digital Blue Foam

Micro Presets

Preset Type	Visual Style	Filter Elements	Select Conditions	Numeric Values	Scaling Factor
Deformation	Customised colour gradient for cross-referencing	Show all elements		All plots show the same type of values for different loading conditions	
	 Figure 3.5.2l - example of a vertical contour plot of the deflected shape				
Deformation	Customised colour gradient for cross-referencing, from 0 to max or custom		Select which load case		Plot on scaled deformed geometry

<table>
<tr><td rowspan="2"></td><td>threshold</td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="5">
Displacement
Deformed Geometry (factor 20)
3.56 cm
0
Figure 3.5.2m - example of displacement on deformed geometry</td></tr>
<tr><td rowspan="2"><b>Material Stress</b></td><td>Specific colour gradient based on limit stress values</td><td>Showing only 1D elements</td><td></td><td></td><td></td></tr>
<tr><td colspan="5">
High stress over the supports and open facades. Deeper sections c.250-300mm required at the perimeter of the openings.
Figure 3.5.2n - example of stresses in the ribs under gravity loading scenario</td></tr>
<tr><td><b>Material Stress</b></td><td>Specific colour gradient based on limit stress</td><td>Showing only 2D elements</td><td></td><td></td><td></td></tr>
</table>

<table>
<tr><td></td><td colspan="5">Greater mobilisation of the stress skin in the lower half of the structure - the reverse is expected during a helicopter lifting scenario (analysis to be undertaken).
Figure 3.5.2o - example of shell stresses</td></tr>
<tr><td><b>Beam Forces</b></td><td>Specific color / style for diagram</td><td>Select specific elements to show (by group for example)</td><td>Select which load case</td><td></td><td>Plot the diagram at scale</td></tr>
<tr><td></td><td colspan="5">
Figure 3.5.2p - example of beams under normal forces</td></tr>
<tr><td><b>Stresses in 2D Elements</b></td><td>Custom gradient colours (in this case white = 0, red is > 0 and blue is < 0)</td><td>Select specific elements to show (by group for example)</td><td>Select which load case</td><td></td><td></td></tr>
</table>

	 Figure 3.5.2q - example of shell utilisation

■ 3.5.3 MEP Engineer

From an MEP perspective, the use of VR has been investigated as a solution and the benefits were recognised since the first attempt. Initially there were pilot projects set up in VR to better understand the things that VR offers as a working environment, the quick wins and the more complex information to handle. A plant room was exported from Revit and added in Unreal Engine with VR navigation controls giving us the ability to move around the plant room, investigate the equipment and the first thing we noticed was the advantage of reviewing the elements in the physical dimensions. Quickly we understood errors such as clearance areas, valve heights, clashes which were missed or potential health and safety risks.

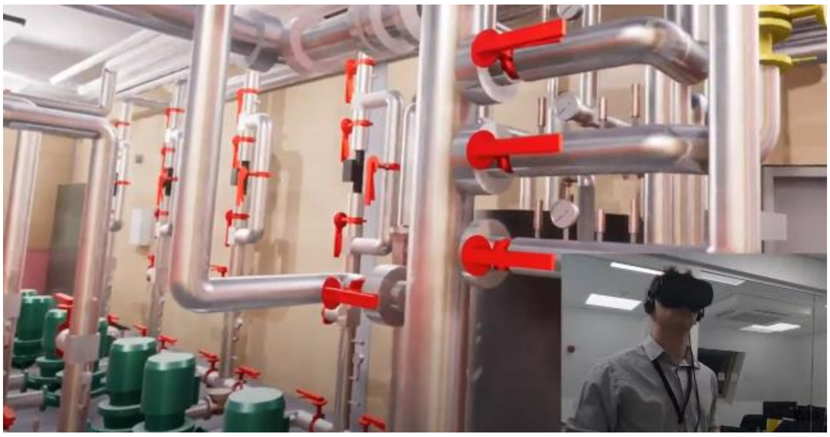
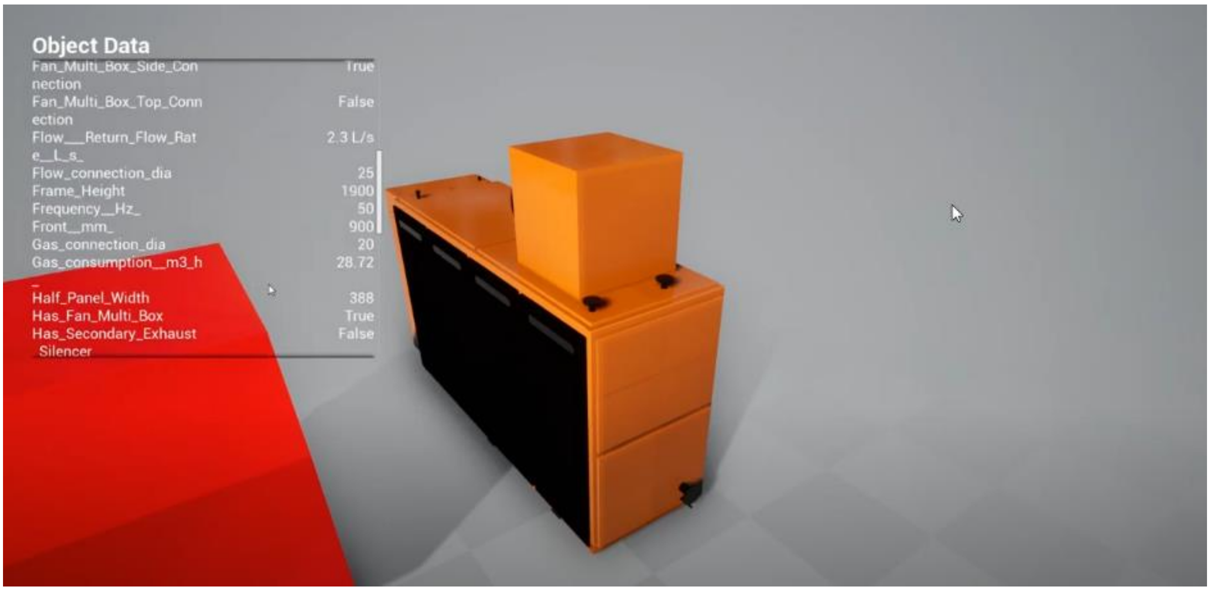


Figure 3.5.3a - Footage from video captured while using the VR headset. Source:https://youtu.be/_jftWo850eE

This aided tremendously to the design solution since we were able to improve our design and recognise additional errors which were missed while working in the computer monitor.

We also wanted to enhance the use of VR and wanted to visualise the data of the elements. This led us to further develop our skills and ended up with a widget which demonstrated the data the element contained.

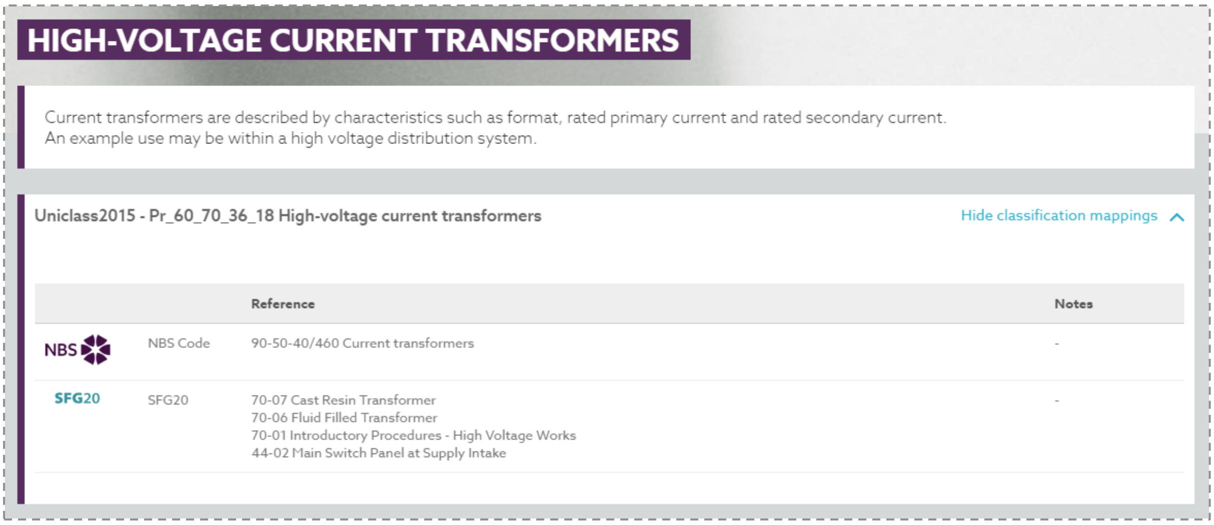


Object Data

Fan_Multi_Box_Side_Connection	True
Fan_Multi_Box_Top_Connection	False
Flow__Return_Flow_Rate__L_s_	2.3 L/s
Flow_connection_dia	25
Frame_Height	1900
Frequency__Hz_	50
Front__mm_	900
Gas_connection_dia	20
Gas_consumption__m3_h_	28.72
Half_Panel_Width	388
Has_Fan_Multi_Box	True
Has_Secondary_Exhaust Silencer	False

Figure 3.5.3b - Screenshot of footage demonstrating the data visualised in real time environments. Source:https://youtu.be/M6OlHWJuDCE

Due to the innovation that PrismArch is intended to offer, we would require this data to be demonstrated within the VR environment. There are different levels of data addition within a construction project as the project matures. The different RIBA Stages define the amount of data which an element needs to have. The example below demonstrates the data added to a transformer from Stages 2 to 5:



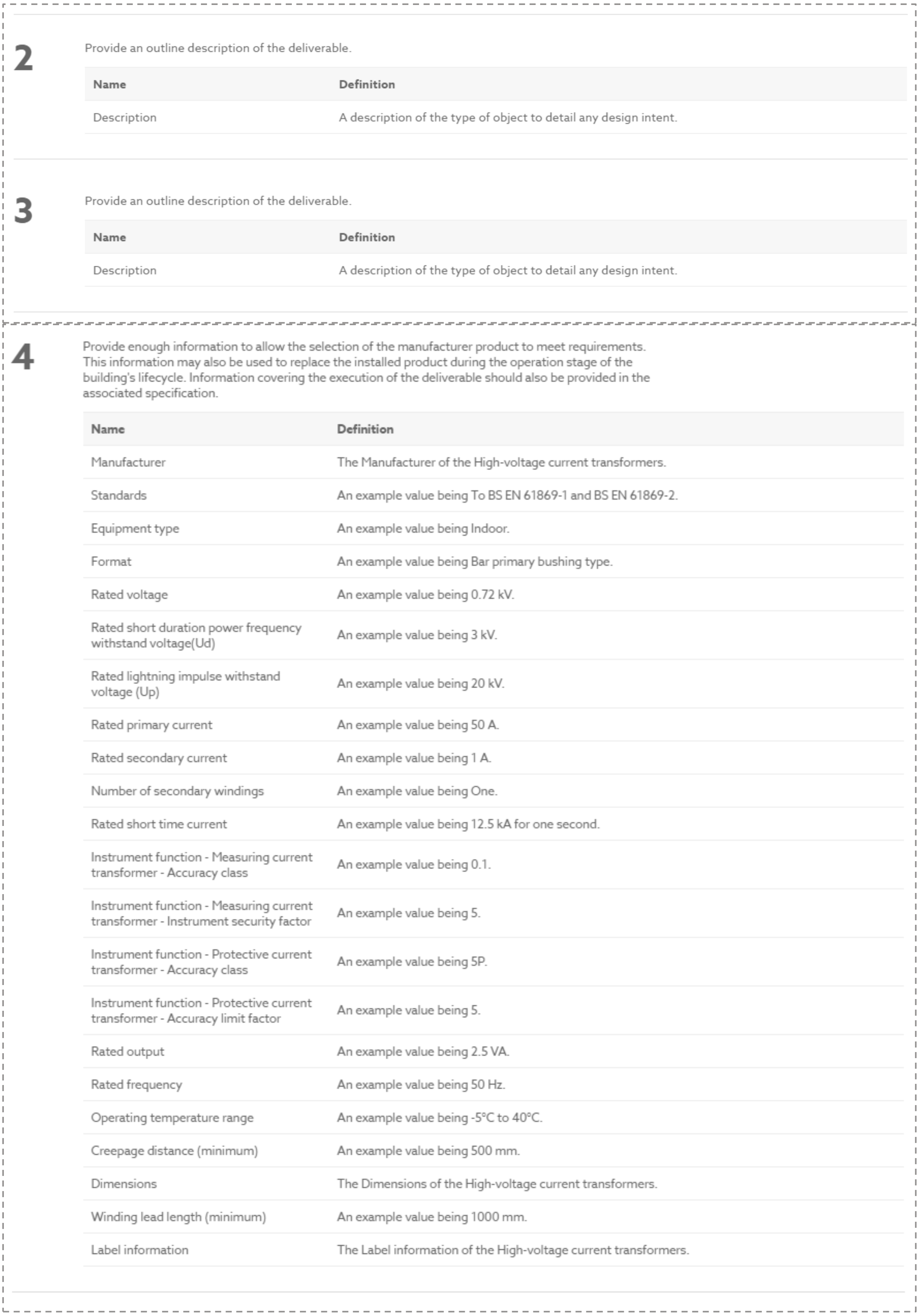
HIGH-VOLTAGE CURRENT TRANSFORMERS

Current transformers are described by characteristics such as format, rated primary current and rated secondary current.
An example use may be within a high voltage distribution system.

Uniclass2015 - Pr_60_70_36_18 High-voltage current transformers

		Reference	Notes
NBS	NBS Code	90-50-40/460 Current transformers	-
SFG20	SFG20	70-07 Cast Resin Transformer 70-06 Fluid Filled Transformer 70-01 Introductory Procedures - High Voltage Works 44-02 Main Switch Panel at Supply Intake	-

Figure 3.5.3c Basic information of a transformer. Source:https://toolkit.thenbs.com/uniclass/ss



2

Provide an outline description of the deliverable.

Name	Definition
Description	A description of the type of object to detail any design intent.

3

Provide an outline description of the deliverable.

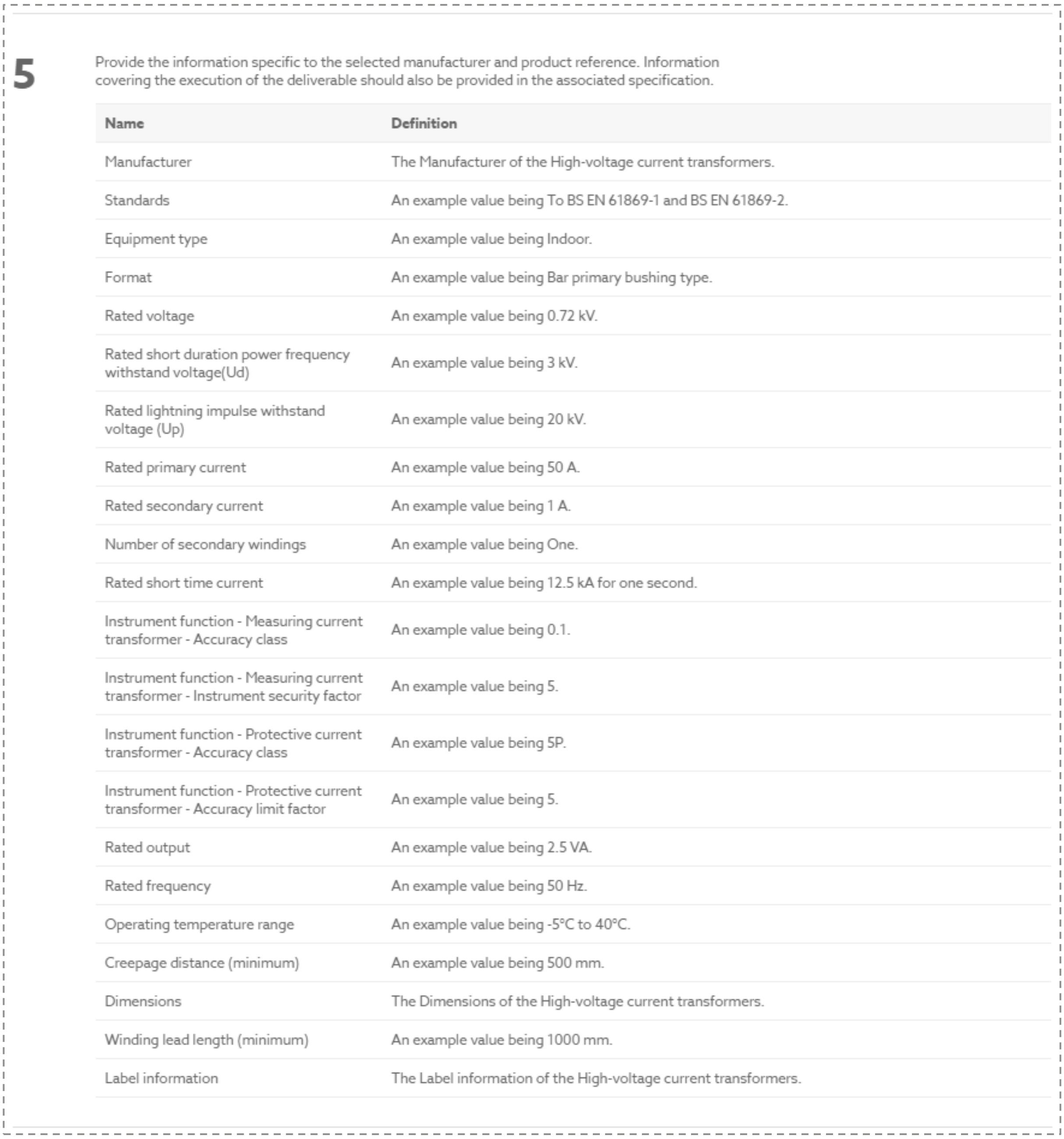
Name	Definition
Description	A description of the type of object to detail any design intent.

4

Provide enough information to allow the selection of the manufacturer product to meet requirements. This information may also be used to replace the installed product during the operation stage of the building's lifecycle. Information covering the execution of the deliverable should also be provided in the associated specification.

Name	Definition
Manufacturer	The Manufacturer of the High-voltage current transformers.
Standards	An example value being To BS EN 61869-1 and BS EN 61869-2.
Equipment type	An example value being Indoor.
Format	An example value being Bar primary bushing type.
Rated voltage	An example value being 0.72 kV.
Rated short duration power frequency withstand voltage(Ud)	An example value being 3 kV.
Rated lightning impulse withstand voltage (Up)	An example value being 20 kV.
Rated primary current	An example value being 50 A.
Rated secondary current	An example value being 1 A.
Number of secondary windings	An example value being One.
Rated short time current	An example value being 12.5 kA for one second.
Instrument function - Measuring current transformer - Accuracy class	An example value being 0.1.
Instrument function - Measuring current transformer - Instrument security factor	An example value being 5.
Instrument function - Protective current transformer - Accuracy class	An example value being 5P.
Instrument function - Protective current transformer - Accuracy limit factor	An example value being 5.
Rated output	An example value being 2.5 VA.
Rated frequency	An example value being 50 Hz.
Operating temperature range	An example value being -5°C to 40°C.
Creepage distance (minimum)	An example value being 500 mm.
Dimensions	The Dimensions of the High-voltage current transformers.
Winding lead length (minimum)	An example value being 1000 mm.
Label information	The Label information of the High-voltage current transformers.

Figure 3.5.3d - Different levels of information of a transformer as the project matures.
Source:https://toolkit.thenbs.com/uniclass/ss

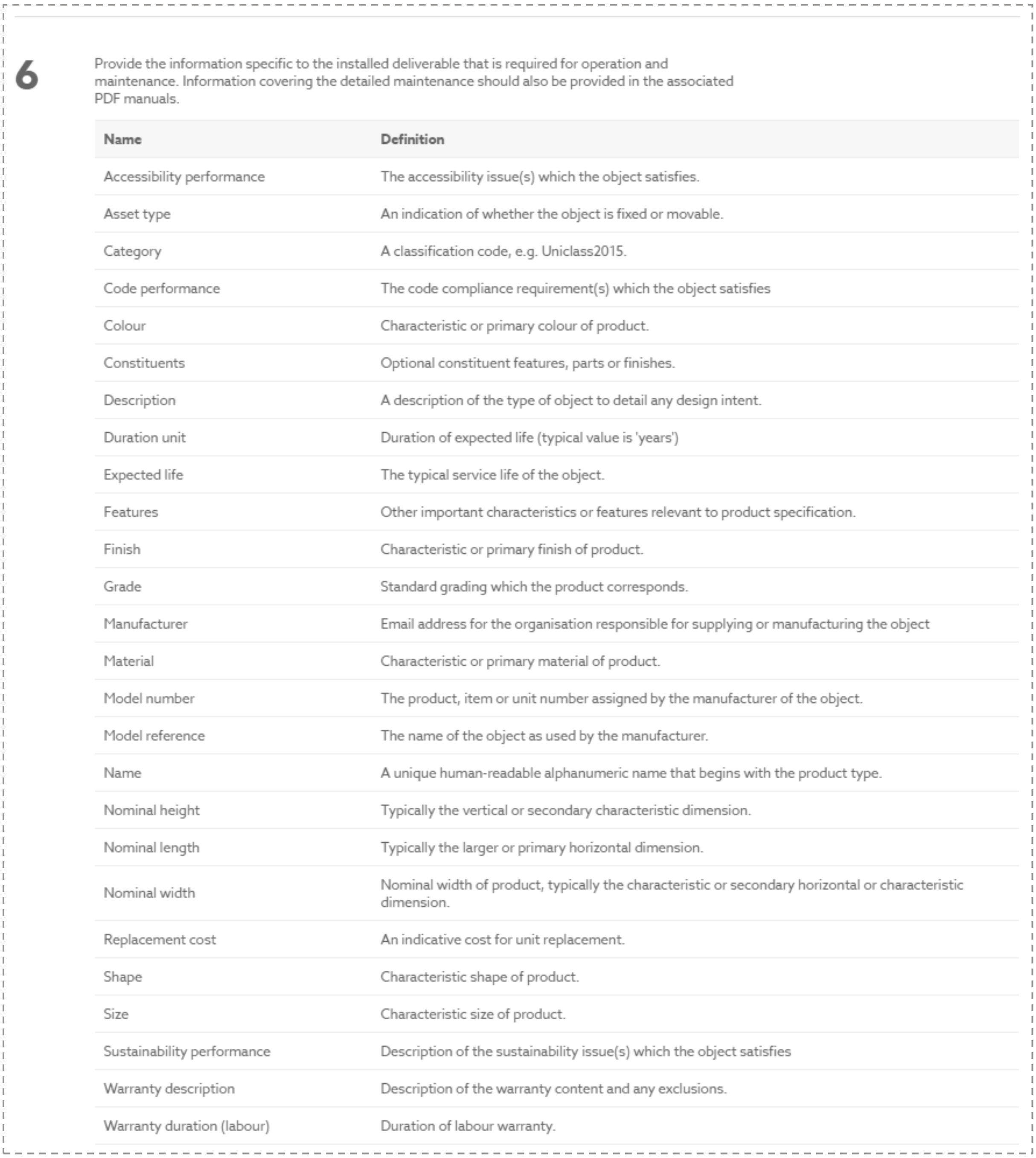


5

Provide the information specific to the selected manufacturer and product reference. Information covering the execution of the deliverable should also be provided in the associated specification.

Name	Definition
Manufacturer	The Manufacturer of the High-voltage current transformers.
Standards	An example value being To BS EN 61869-1 and BS EN 61869-2.
Equipment type	An example value being Indoor.
Format	An example value being Bar primary bushing type.
Rated voltage	An example value being 0.72 kV.
Rated short duration power frequency withstand voltage(Ud)	An example value being 3 kV.
Rated lightning impulse withstand voltage (Up)	An example value being 20 kV.
Rated primary current	An example value being 50 A.
Rated secondary current	An example value being 1 A.
Number of secondary windings	An example value being One.
Rated short time current	An example value being 12.5 kA for one second.
Instrument function - Measuring current transformer - Accuracy class	An example value being 0.1.
Instrument function - Measuring current transformer - Instrument security factor	An example value being 5.
Instrument function - Protective current transformer - Accuracy class	An example value being 5P.
Instrument function - Protective current transformer - Accuracy limit factor	An example value being 5.
Rated output	An example value being 2.5 VA.
Rated frequency	An example value being 50 Hz.
Operating temperature range	An example value being -5°C to 40°C.
Creepage distance (minimum)	An example value being 500 mm.
Dimensions	The Dimensions of the High-voltage current transformers.
Winding lead length (minimum)	An example value being 1000 mm.
Label information	The Label information of the High-voltage current transformers.

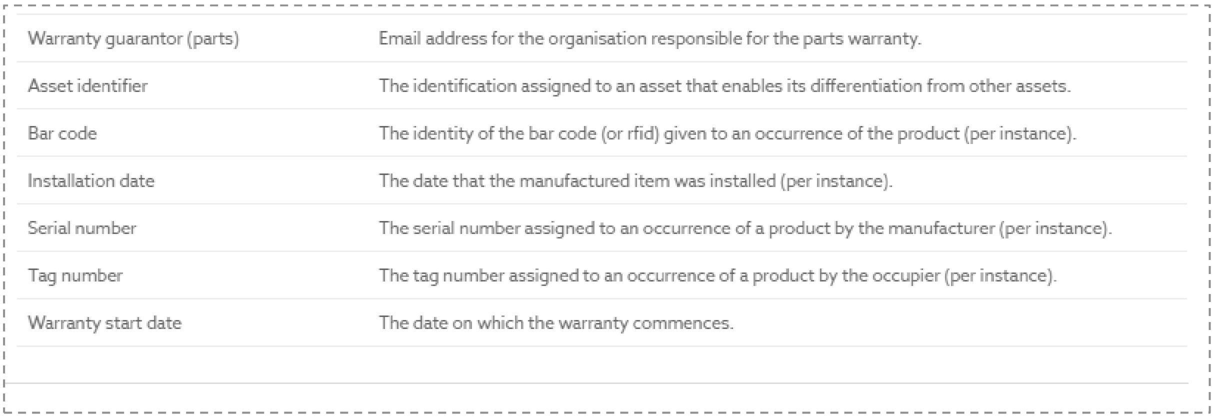
Figure 3.5.3e - Advanced level of information of a transformer. Source:https://toolkit.thenbs.com/uniclass/ss



6

Provide the information specific to the installed deliverable that is required for operation and maintenance. Information covering the detailed maintenance should also be provided in the associated PDF manuals.

Name	Definition
Accessibility performance	The accessibility issue(s) which the object satisfies.
Asset type	An indication of whether the object is fixed or movable.
Category	A classification code, e.g. Uniclass2015.
Code performance	The code compliance requirement(s) which the object satisfies
Colour	Characteristic or primary colour of product.
Constituents	Optional constituent features, parts or finishes.
Description	A description of the type of object to detail any design intent.
Duration unit	Duration of expected life (typical value is 'years')
Expected life	The typical service life of the object.
Features	Other important characteristics or features relevant to product specification.
Finish	Characteristic or primary finish of product.
Grade	Standard grading which the product corresponds.
Manufacturer	Email address for the organisation responsible for supplying or manufacturing the object
Material	Characteristic or primary material of product.
Model number	The product, item or unit number assigned by the manufacturer of the object.
Model reference	The name of the object as used by the manufacturer.
Name	A unique human-readable alphanumeric name that begins with the product type.
Nominal height	Typically the vertical or secondary characteristic dimension.
Nominal length	Typically the larger or primary horizontal dimension.
Nominal width	Nominal width of product, typically the characteristic or secondary horizontal or characteristic dimension.
Replacement cost	An indicative cost for unit replacement.
Shape	Characteristic shape of product.
Size	Characteristic size of product.
Sustainability performance	Description of the sustainability issue(s) which the object satisfies
Warranty description	Description of the warranty content and any exclusions.
Warranty duration (labour)	Duration of labour warranty.

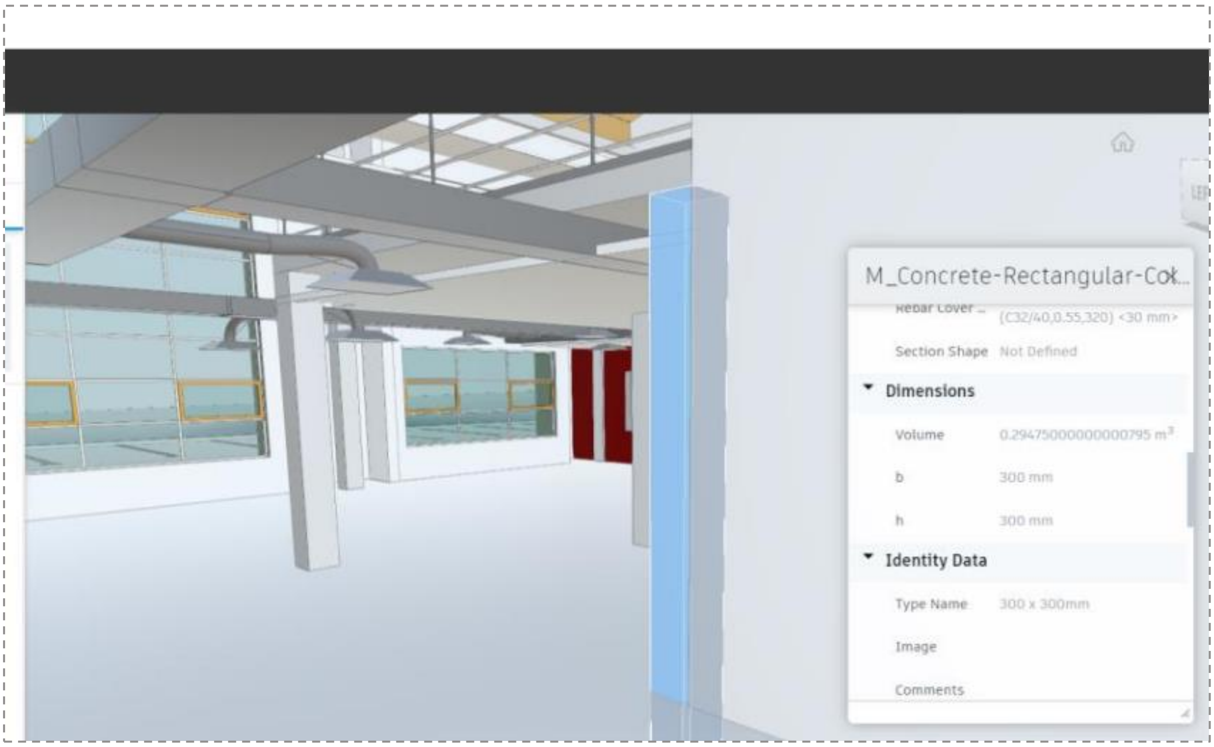


Warranty guarantor (parts)	Email address for the organisation responsible for the parts warranty.
Asset identifier	The identification assigned to an asset that enables its differentiation from other assets.
Bar code	The identity of the bar code (or rfid) given to an occurrence of the product (per instance).
Installation date	The date that the manufactured item was installed (per instance).
Serial number	The serial number assigned to an occurrence of a product by the manufacturer (per instance).
Tag number	The tag number assigned to an occurrence of a product by the occupier (per instance).
Warranty start date	The date on which the warranty commences.

Figure 3.5.3f - Complete level of information of a transformer. Source:https://toolkit.thenbs.com/uniclass/ss

We would need to include this data into the VR environment and through tag visualisation to be able to bring them to the user interface.

The data within the models is not something new. However, the latest web technologies brought this data into the web environments. A similar approach can be followed in PrismArch.

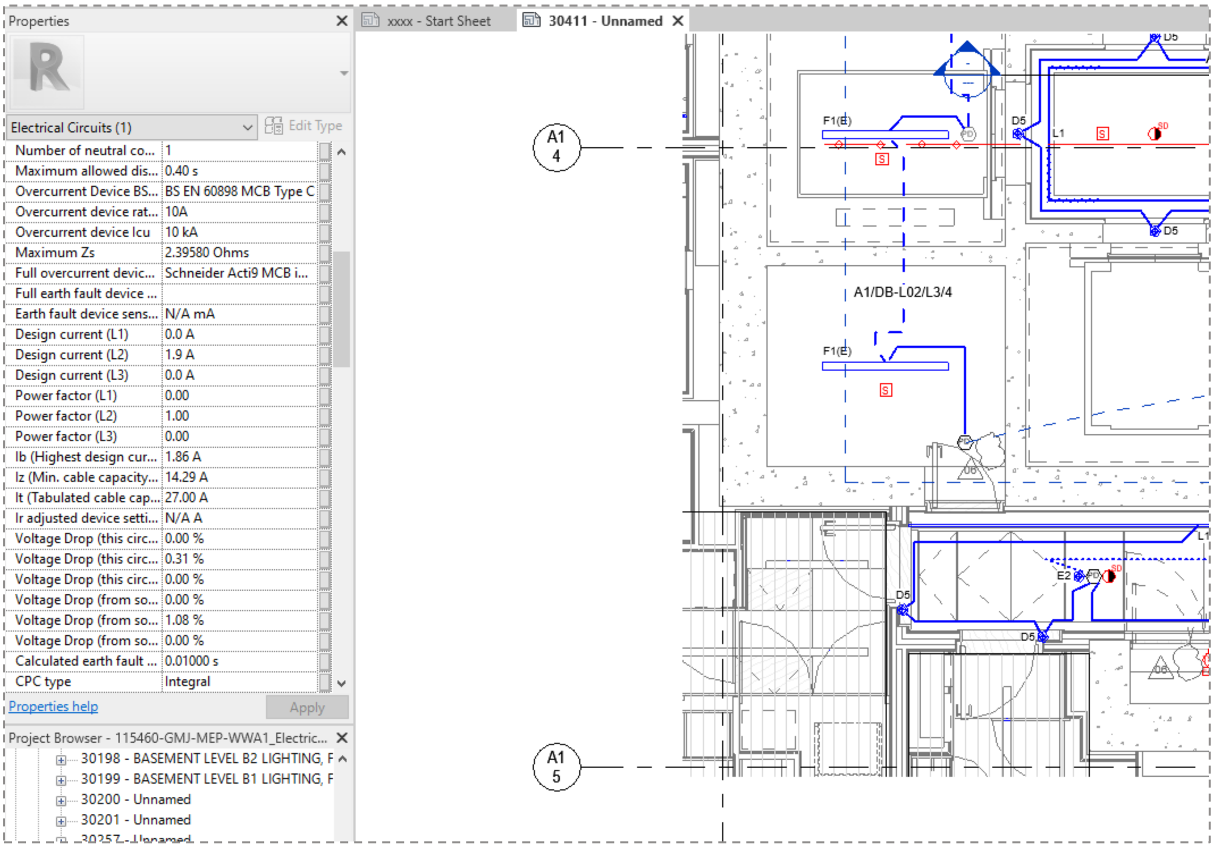


M_Concrete-Rectangular-Col...

Rebar Cover ...	(C32/40,0.55,320) <30 mm>
Section Shape	Not Defined
Dimensions	
Volume	0.29475000000000795 m³
b	300 mm
h	300 mm
Identity Data	
Type Name	300 x 300mm
Image	
Comments	

Figure 3.5.3h - Example of 3d model data within a web based environment. Source: Autodesk BIM360 web environment.

This information is added in the initial specialist software and transferred to PrismArch. The parameters are set up in the project file and the data can be added either manually or automatically based on the commands the user is executing.



Properties

Electrical Circuits (1)

Number of neutral co...	1
Maximum allowed dis...	0.40 s
Overcurrent Device BS...	BS EN 60898 MCB Type C
Overcurrent device rat...	10A
Overcurrent device Icu	10 kA
Maximum Zs	2.39580 Ohms
Full overcurrent devic...	Schneider Acti9 MCB i...
Full earth fault device ...	
Earth fault device sens...	N/A mA
Design current (L1)	0.0 A
Design current (L2)	1.9 A
Design current (L3)	0.0 A
Power factor (L1)	0.00
Power factor (L2)	1.00
Power factor (L3)	0.00
Ib (Highest design cur...	1.86 A
Iz (Min. cable capacity...	14.29 A
It (Tabulated cable cap...	27.00 A
Ir adjusted device setti...	N/A A
Voltage Drop (this circ...	0.00 %
Voltage Drop (this circ...	0.31 %
Voltage Drop (this circ...	0.00 %
Voltage Drop (from so...	0.00 %
Voltage Drop (from so...	1.08 %
Voltage Drop (from so...	0.00 %
Calculated earth fault ...	0.01000 s
CPC type	Integral

Project Browser - 115460-GMJ-MEP-WWA1_Electric...

30198 - BASEMENT LEVEL B2 LIGHTING, F
30199 - BASEMENT LEVEL B1 LIGHTING, F
30200 - Unnamed
30201 - Unnamed
30257 - Unnamed

Figure 3.5.3i - Example of parameters and data within an electrical project file developed in Revit

The data can include engineering solutions such as electrical calculations, mechanical flows, pressure drops etc.

To better understand the needs, two examples for an ideal usage of PrismArch are given below:

Example 1:

An MEP user is viewing level 3 of the building within the PrismArch environment. The architectural and structural models are loaded and the user is drawing electrical elements for space allocations. After the user finishes the task, they must check for any physical clashes between elements.Either by using the PrismArch UI tools (or ideally by using the proposed PrismArch speech recognition tools) the user is able to view the mechanical model initiate the clash detection process.s. The user is also able (through either UI or speech interactions) to toggle the visibility of the the mechanical model within the overall design..

Example 2:

The MEP user is navigating within a plant room. The user selects an element by pointing it with the VR controllers and uses the speech tool to bring information within the user interface. For example, the user might say “bring up the user manual” or “bring up the data sheet” and the platform reveals the relevant data.

○ 3.6 Proposals for the VR Interface

■ 3.6.1 Cross-Disciplinary Interface

3.6.1.a User Interface Design for On-boarding

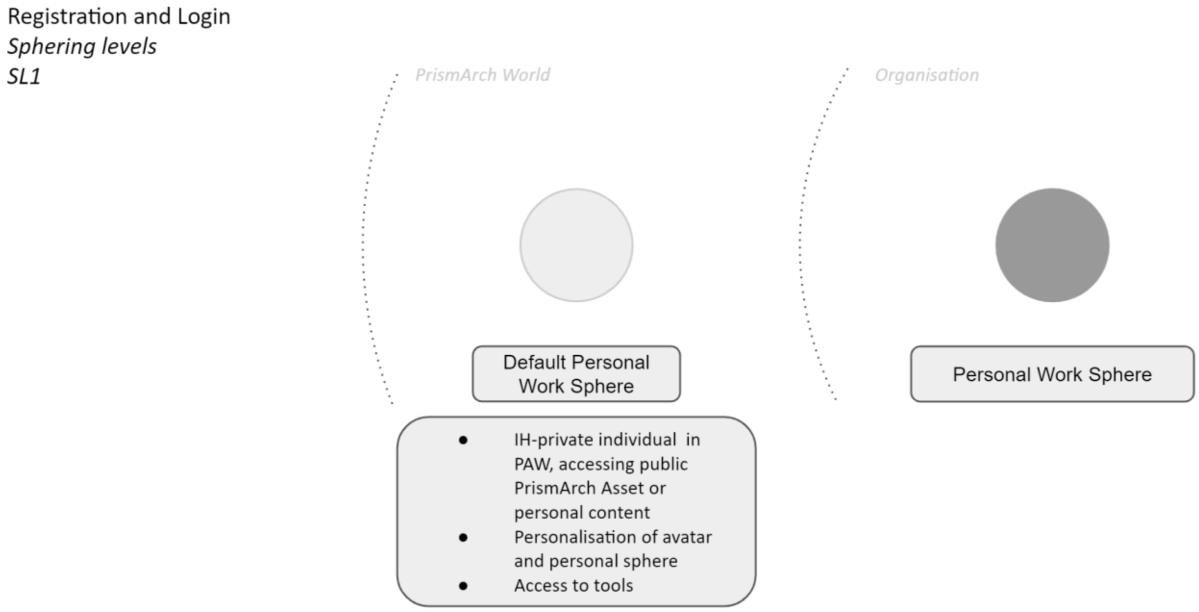

Registration and Login
Sphering levels
SL1
PrismArch World
Organisation
Default Personal Work Sphere
Personal Work Sphere
IH-private individual in PAW, accessing public PrismArch Asset or personal content
Personalisation of avatar and personal sphere
Access to tools

Figure 3.6.1a - Diagram illustrating on-boarding usage scenario

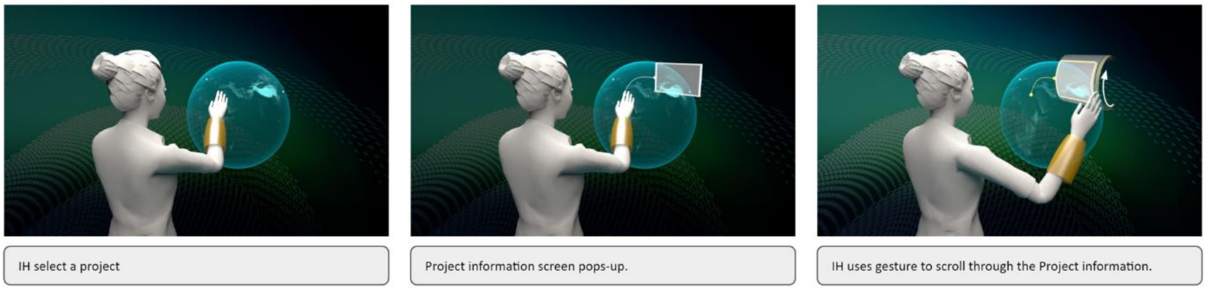
By default an individual can access the PrismArch platform as a “private entity”. The user will have to register and Login to company branded Personal Work Sphere within one of the AEC disciplines organisations. With company credentials, IH gains access to the information about projects, as well as companies proprietary tools etc. Any content and IP created by employee IH is a property of the company and will remain within its sphere.

The deliverable D6.1 outlines the purpose of sphereing a immersed human unit including the Personalisable and Customisable User Interface, by explaining as below;

“*PWS is a customisable personal work environment that contains a digital representation of the immersed human and his/her Personalisable User Interface (PUI). PWS is an individual unit for each IH and this enables individual recognisable and interactable inside the PAW. With the registered personal information about discipline and role types, the system allocates each IH to one of the collective units (most likely a company name), Collective Work Sphere (CWS)*.”
(D6.1, sct. 1.2)

On-boarding is a process to compound user tailored project space and every participant, therefore, is asked to configure and confirm its personal settings in order to access the PrismArch platform embedded system, core as well as user proprietary tools and functionalities. A usage scenario described below, aims to illustrate an on-boarding situation where the user choses a project and loads project PrismArch Central Model (PCM) with the associated project information. How the interaction works can be compared with the folder explorer on PCs. This self referencing VR User Interface and Interaction logics can be applied throughout the platform.

Below illustrates the user on-boarding storyboard, starting from selecting a project location and until entering the Project Sphere:



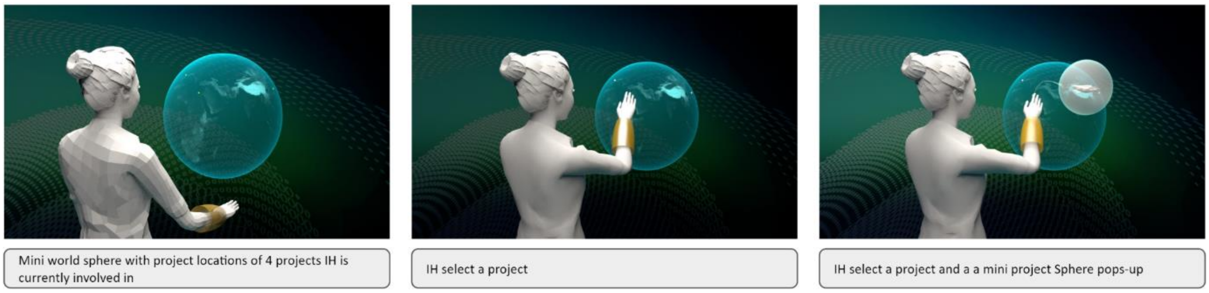
IH select a project

Project information screen pops-up.

IH uses gesture to scroll through the Project information.

Figure 3.6.1b - On-boarding Storyboard Part 1/4

First, the user sees a hologram like a globe with different project locations. The user hovers a location to read the project description. The popup Billboard UI with an annotation line is scrollable to see the entire panel content. The projected content could include a project thumbnail, project information, year, stage etc.



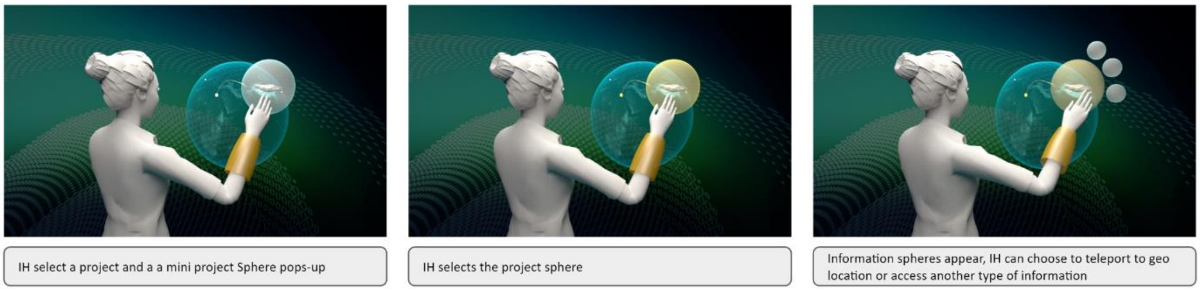
Mini world sphere with project locations of 4 projects IH is currently involved in

IH select a project

IH select a project and a a mini project Sphere pops-up

Figure 3.6.1c - On-boarding Storyboard Part 2/4

After the user confirms the selection, a new sphere pops up at the annotated location. Now the user sees a low resolution model or the thumbnail image.



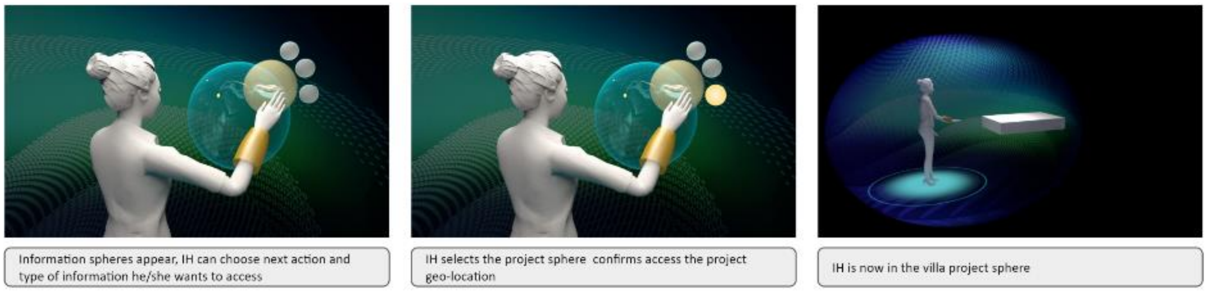
IH select a project and a a mini project Sphere pops-up

IH selects the project sphere

Information spheres appear, IH can choose to teleport to geo location or access another type of information

Figure 3.6.1d- On-boarding Storyboard Part 3/4

Next, the user confirms the selection and sees another set of spheres popping up from the selection source. When the user hovers the sphere, the glow colour changes. The highlighted colour can be a discipline-specific registered colour. Each sub-spheres would include different content such as a site image (jpeg), diagrammatic floor plan (pdf), technical drawing (dwg), commentary (txt), structural data (txt).



Information spheres appear, IH can choose next action and type of information he/she wants to access

IH selects the project sphere confirms access the project geo-location

IH is now in the villa project sphere

Figure 3.6.1e- On-boarding Storyboard Part 4/4

Lastly, the user interacts with the sub-sphere(s) to visit the relative PADO. After a confirmation message, there is a blank loading screen, and then the user is teleported to the suggested system location.

Below are images from a VR experience that demonstrates the user on-boarding experience explored in the above:

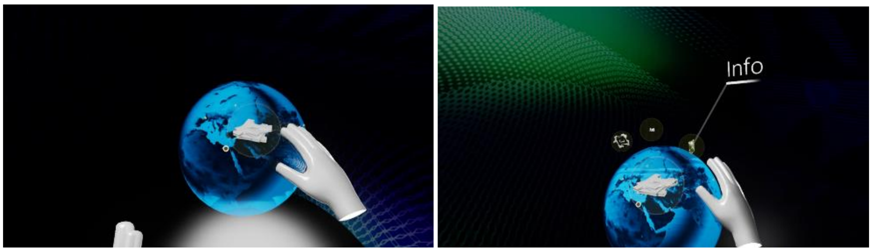


Figure 3.6.1f - On-boarding Storyboard VR mock up

3.6.1.b User Interface Design for Personal Work Sphere

The PrismArch Central Model (PCM) is located at the fixed location and real scale, this is essential for multidisciplinary collaboration and coordinating design input by all disciplines. Therefore in order to interact with the PADO at different scales, the IH as well as PWS needs to adjust their scale as required.

As the scale of projects varies depending on their typology and scope agreed in professional service contracts with clients, navigation by IH in VR space is therefore an important consideration. IH needs to be able to move intuitively and comfortably, sometimes covering vast physical distances in order to be able to interact with any of the core PA classes.

The Level of detail (LOD) of the PADO visible to IH from different distances and scales, should be adaptable, and adjusted accordingly. Similarly the same adjustment should apply to the metadata information, and other UI elements such as tags, labels etc.

In order for IH to be able to orientate themselves within the project overall scale e.g. while working on different scales, the provision of a project 3D/2D minimap is an essential part of the user interface.

Based on the range of tasks undertaken by design teams on construction projects, as documented in D1.1, ZHA has developed a storyboard. This includes use case scenarios, which describe situations the PA users are likely to experience frequently. This allowed the team to identify and illustrate top level interactions occurring within the platform.

IH Interaction with PADO
A User scenario described below, aims to illustrate an example of basic interaction between IH and PADO:

First, as a user accesses a private villa project, he/she has a general oversight of the DO, and can see top level project information (e.g. project name, current version, last update date...). As IH moves closer to the PADO and adjusts their scale, more information is displayed - the designer can see individual parts of the project with relevant information e.g. about other designers assigned to particular parts).
Last, IH select the part he/she wants to work on. The PADOs are highlighted and displayed at the highest level of detail, so that the designer can now manipulate design features using design tools such as Mindesk or review models at 1:1 scale.

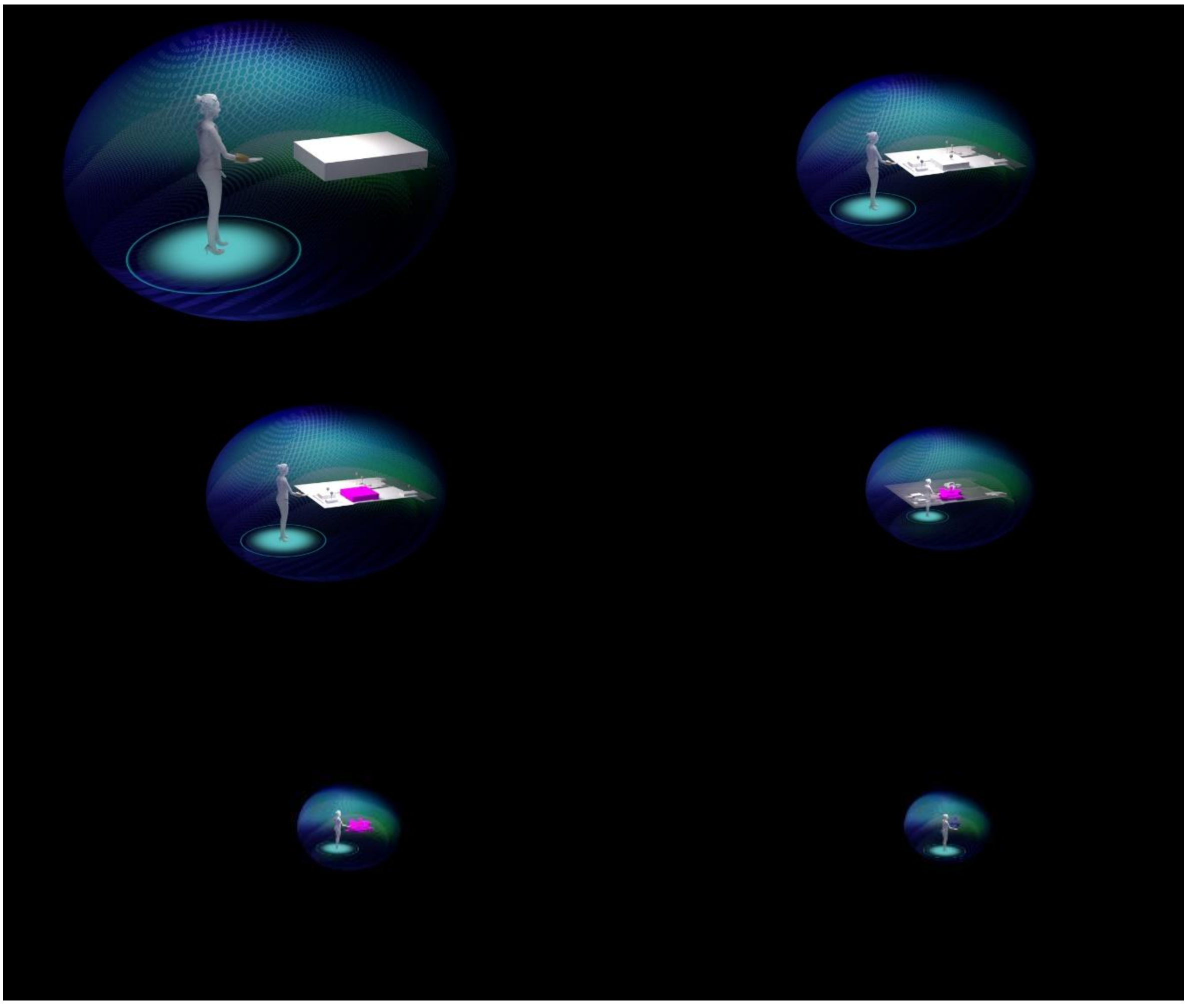


Figure 3.6.1g - Example of the Immersed Human interacting with PrismArch Design Objects

3.6.1.c User Interface Design for Meeting Sphere

Individuals need to be recognised by the collective in multipresence, co-authoring, and cross-disciplinary situations. This could be done through personalised avatars, name tags, organisation name, color coded by discipline. Content created by different disciplines has to be demarcated and authorship retained in any circumstances within the PrismArch platform. Further distinction is required to indicate who among meeting Participants is the meeting Host and/or the Presenter. During the course of the meeting, the Host remains persistent, however the Presenter can be passed on to the other team members, enabling them to show their content, when required. Please see a more detailed usage scenario for a meeting sphere arrangement for a cross disciplinary situation in D6.1, sct:1.2.

User scenario extract from D6.1, sct 2.1 can be found from below:
"*PD_A opens the dashboard on his* ***Personal User Interface*** *and accesses* ***the contact list widget****. He selects all the current project members in the list, then filters the 'internal kick-off' tag to clipboard* ***the spatial hyperlink*** *(to the world coordinate of the arranged meeting sphere). He* ***confirms the invite after filling in meeting information such as the meeting name, time, number of attendees/size****.*

Each attendee ***receives a meeting sphere link and visits the arranged location and rotation by entering the link from individual PWS at the arranged time. The event is automatically added to their calendars in the PUI dashboard."***

Types of User Interfaces accessed within the Meeting Sphere can be:

Personalisable and Customisable User Interface

- to view contact list
- to invite attendees
- to accept the invitation
- to fill in meeting information
- to change display modes
- to execute some of the shortcuts (e.g. Commenting and Markup, Clipping Plane, Toggle Camera functionality etc …)

Query User Interface

- to demarcate and tag to curate meeting and presentation assets
- to visit the arranged Meeting Sphere (in a similar way to the interaction explained in …)
- to record, play and pause meeting content

As mentioned thoroughly in the document, we would like to highlight that the PrismArch Core Functionalities should work in any condition inside the PrismArch VR platform. Functionalities below would be expected to be used often by the users inside the Meeting Sphere.

PrismArch Core Functionalities

- Commenting and Markup functionality
- Clipping Plane functionality

- Toggle Camera functionality
- Toggle View Mode functionality

Specialist and Proprietary Tools

- Zoom, Skype and/or alternative Steam or other Voice Chat functionalities
- Simulation tools

Based on frequently encountered user scenarios and sphereing levels we classified meetings as follows:

1. **Internal team meeting SL2 (design review, collaboration)**
   Locations and scales are carefully selected by the host to review specific aspects of the design. During the meeting IHs sketch, redmark, 3D model, call up references, switch between various display modes, and make notes.
2. **Multidisciplinary coordination meeting SL3 (to review and discuss solutions to design issues)**
   Locations and scales are carefully selected by the host to review and coordinate aspect of the design;
   During the meeting members of the design team review content produced by different disciplines. This content has to be clearly demarcated and recogniseable. Assets such as structure and MEP are hidden behind architectural envelopes; there is therefore a requirement to be able to display the “hidden” PADOs.
3. **Client, project stakeholders meeting SL4 (a curated tour of a project)**
   Locations and scales are carefully selected to curate the presentation narrative
   Polished photorealistic display mode is most likely to be used in this scenario (with a possibility to change it to review aspects of the project if required)
4. **Content accessible by the wider public SL5 (large audience events such as lectures, exhibitions, conferences)**
   Limited editing capabilities, Protected IP of authors

Regardless of the purpose for the meeting, there is a persistent functionality required for meetings at all levels:

- Changing display modes (“clay”,photorealistic, xray, etc..)
- Reviewing design issues in different locations and at different scales (flexibility to adjust scale and location when required)
- Redmarking
- Modeling
- Recording notes
- Immersive meeting capture
- **Team member authorship demarcation, disciplines content demarcation**
- **Automatic tagging of a content created during the meeting**

Regardless of the organisational association IHs should be able to prepare and curate the information to be discussed during meetings. From individual designer/engineer reviews, communicating with and coordinating multidisciplinary input, through to polished client presentations.

Each meeting has a purpose and should be conducted in an organised manner.
Users should be able to prepare the meeting including a preset of locations in 3D space at an appropriate scale to be able to present and discuss particular aspects of the design.

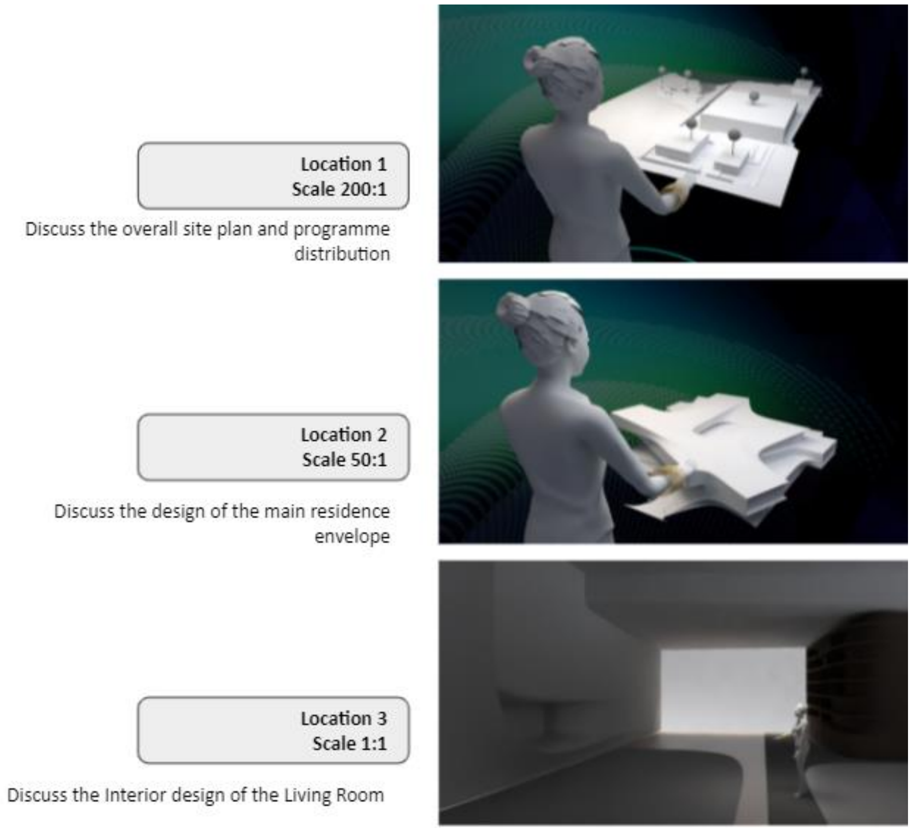

Location 1
Scale 200:1
Discuss the overall site plan and programme distribution
Location 2
Scale 50:1
Discuss the design of the main residence envelope
Location 3
Scale 1:1
Discuss the Interior design of the Living Room

Figure 3.6.1h - Examples of Meeting Sphere locations and meeting purposes

Meetings should take place in a PrismArch Meeting Sphere. The UI should enable users to customise meeting spheres in order to accommodate different numbers of participants and accommodate the need to examine PADO at various scales. This should allow IH flexibility to host a range of different meetings.

Meeting spheres should include assets such as tables and chairs, these are necessary to mirror the IH's standing or sitting position in the physical world. Their presence and number could be a part of a meeting preset.

Objects such as plinths, presentation panels, and a screen are required to facilitate any needs of a host to curate the meeting content. Prior to and/or during the meeting the users should be able to prepare all necessary material and references to support his/her narrative.

Once the meeting is concluded, the Meeting Spheres should be saved and archived to form a record of decision making throughout the life cycle of the project (the Golden Thread of project information). All of the items produced during the meeting should be automatically tagged and later become accessible by the participants, as they work on a solution within their organisations and PWS.

The example of output items could be: Work in progress Instance of the 3D model, sketches, markup, meeting minutes, action points, agenda for the next review, Capture of the immersive scene - if agreed by all participants.
The recording of a meeting should be possible to revisit in VR or on the screen.

Below reference to examples from the gaming industry, such as a scene from Cyberpunk 2077 [Cyberpunk 2077] illustrates well the recording and re-visiting a meeting scenario. Player records the scene and then revisits the recording from the third person perspective. Design issues, itemised in the meeting minutes should be hyperlinked to physical locations in the scene. The recording camera functionalities can be associated with the Toggle Camera functionality and Toggle View Mode functionality (as part of the PrismArch Core Functionalities).

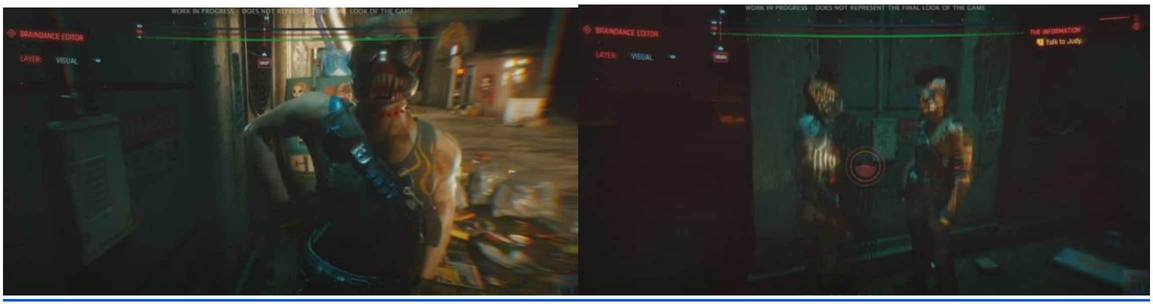

WORK IN PROGRESS - DOES NOT REPRESENT THE FINAL LOOK OF THE GAME
BRAINDANCE EDITOR
LAYER
VISUAL
Talk to Judy

Figure 3.6.1i - Reference to revisit a meeting record, from the computer game[Cyberpunk 2077].
Top shows a first person view and bottom shows a third Person View.

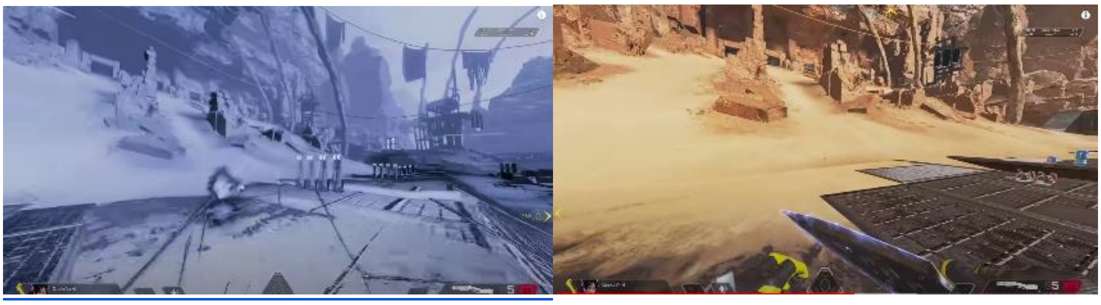


Figure 3.6.1j - Reference to X Ray d vs. Photorealistic display modes, illustrating the difference between revisiting a meeting and attending a meeting. Screenshots from computer game [Cyberpunk 2077].

3.6.1.d User Interface Design for Content Query and Demarcation

In section 3.4, the three types of User Interfaces in the PrismArch VR platform are proposed; Personalisable and Customisable UI, Billboard UI and Query UI. The terms of Personalisation and Customisation also is explored in the section. Section 3.6.1.a introduces a storyboard for the user to interact with a Billboard UI.

This section focuses on the user interaction with Query User Interface. Importantly, it should be noted that modifications of Design Objects (architectural geometries) are not included in the section - as these aspects will be covered by proprietary modelling software and applications such as Mindesk, and will be elaborated upon in other Deliverables (D4.2, T5.2, etc). The design experiment covered in this section is a demarcation tool for PrismArch contents that work inside the PrismArch World. Final tool should function to demarcate all four types of PrismArch root asset classes, and this should work for all project types and project scales. The tool should be intuitive, flexible and adaptable to different User Interface Modes.

Query User Interface

In the deliverable D6.1, Sphereing Level is proposed to organise and demarcate project metadata and information in order to maintain a safe, efficient and continuous collaborative VR experience. As the project grows, larger amounts of project metadata and information will be created, collected and stored inside the PrismArch Singular Data Space. This creates a complex data nest. The Query User Interface enables to filter user demanded inputs from the project data nest, and helps to visualise the filtered information with simple geometries (Metadata Nodes). The user interacts with the Metadata Nodes to preview and trace back the associated project information. The proposed tool enables the user to load queried Design Objects and its associated project metadata and information. In the deliverable D6.1, the Query User Interface is defined as '*an equivalent to a three dimensional project folder structure with a time scale and it is a dynamic navigation functionality with spatial hyperlinks to project design objects* …[and] each node would represent a project folder and/or file, for instance, a site image (jpeg), diagrammatic floor plan (pdf), technical drawing (dwg), commentary (txt), structural data (txt)".

"*High level manual sphereing is necessary for curating information specific to the needs of an IH. Whether it is to describe a certain aspect, to make a distinction, or to inform others, selected information needs to be placed in a specific context, and certain aspects brought to the foreground. Whenever the IH wishes to explain their reasoning, either as part of a dialogue with another IH or for a presentation to another discipline, they manually bring together and cast a net around a subject or several subjects and several core classes. This has the function of describing certain aspects or certain parts of the selected information, by giving them a name and a distinction, as well as recording the date, time, and location of this particular selection. This is part of the multidisciplinary aspect of looking at the same information, but forming independent inputs from unique points of view and areas of experience.*"
H.Kinzler, D.Zolotareva et al, 2021.

Figure 3.6.1k - Examples of BigData Visualisation Layouts, Sviatoslav Iguana, 2019

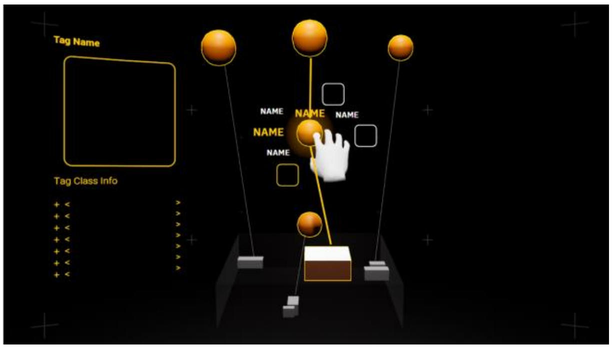


Figure 3.6.1l - The user hover/point at a Metadata Node and Billboard UIs appear near the selection

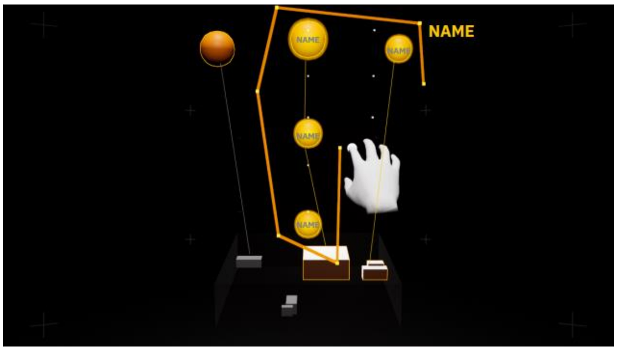


Figure 3.6.1m - The user drops multiple control points to demarcate multiple Metadata Nodes. Billboard UIs appear near the selection

Following aspects must be considered to interact with Metadata Nodes;

- Single as well as multiple Metadata Nodes can be interacted with the user
- The demarcation tool should work intuitively regardless of target amount, work mode or class type
- The user needs precise control within the selection action, meaning there should be several steps before the user confirming the selection
    - Preview: the user hovers each or multiple nodes to check the selected project information. The user optimises the selection by de-selecting, re-selecting the queried Design Objects or Metadata Nodes
    - Confirm the demarcation: to execute the Query function to load Design Objects and the Metadata Nodes

- Each Metadata Node can be pointable(hover-able) and/or selectable:
    - By hovering a node, the user can display its Billboard UI which includes the relative project information. The information LOD and the Billboard UI location depends on the User Interaction Mode, whether the user interacts with the avatar hand or with a ray attached to the hand. The Billboard UI can also be displayed as a Head Up Display (HUD), near a finger tip, or at the Node location (Figure 3.6.1l and Figure 3.6.1m).
    - By selecting single or multiple nodes, if there is any valid location information provided for them, the user can navigate from one node to the next, thus following 'the golden thread' of the project's development. As explained in On-Boarding storyboard in section 3.6.1.a User Interface Design for On-boarding (Figure 3.6.1f).
- Metadata Nodes can be minimised or maximised with the user's preferences. For instance, the Medata Nodes can be minimised or hidden when the user would like to focus on modifying Design Objects via a proprietary modelling software and application, or these can be maximised to review specific project information inside a Medata Node (Figure 3.6.1n and 3.6.1m)

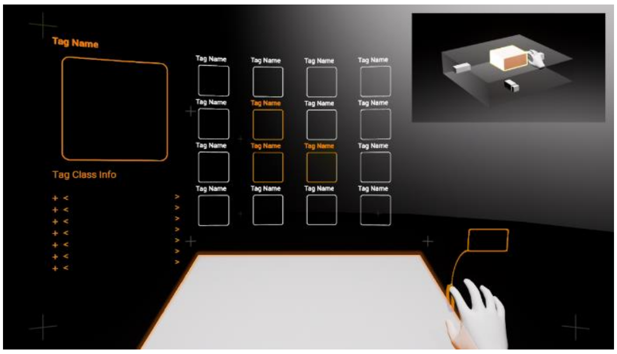


Figure 3.6.1n - The user hovers/points at a Design Object. The associated Metadata Nodes are closed and the user sees the **detailed** asset information on the Personal User Interface

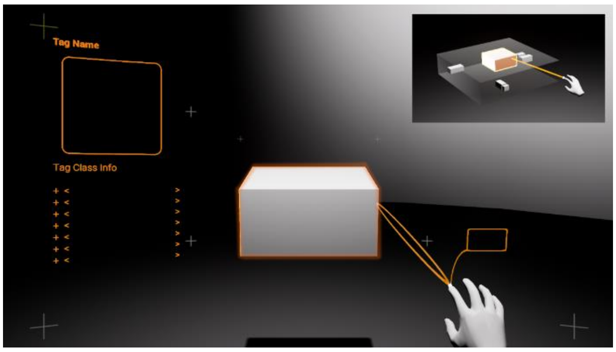


Figure 3.6.1o - The user hovers/points at a Design Object. The associated Metadata Nodes are closed and the user sees the **minimal** asset information on the Personal User Interface

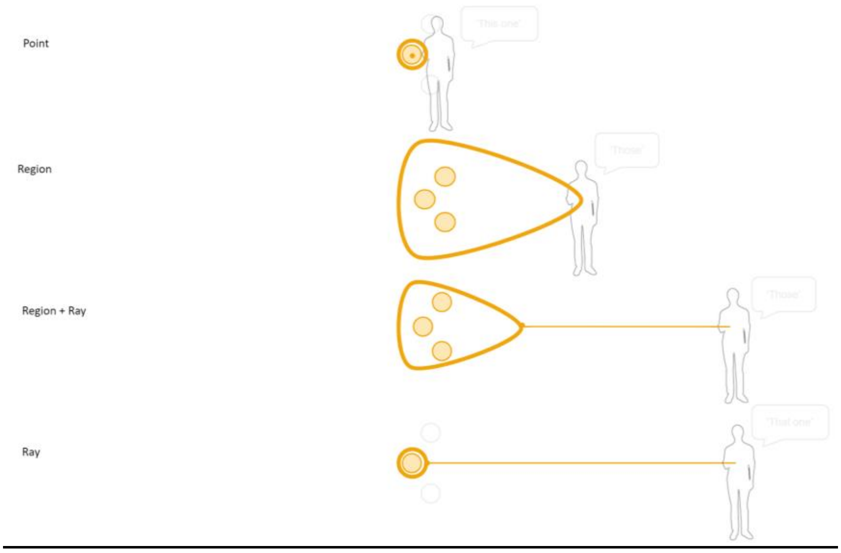


Figure 3.6.1p - Diagrams illustrating the transformation of the demarcation tool

Billboard UI and Work Space Occlusion

The query result might be large enough to clutter the loaded Metadata Nodes due to the complex nest of project information. This would cause the user to experience occlusion issues where some of the Metadata Nodes and the attached Billboard UI being hidden by other nodes rendered in front of them. Traditionally, in gaming, these problems are solved by post processing rendering to outline and/or filling the target in front of everything (Figure 3.6.1q). This method would work well, for example for avatars, however, it would not work with cluttered Metadata Nodes as it only renders the outline of the cluttered region.

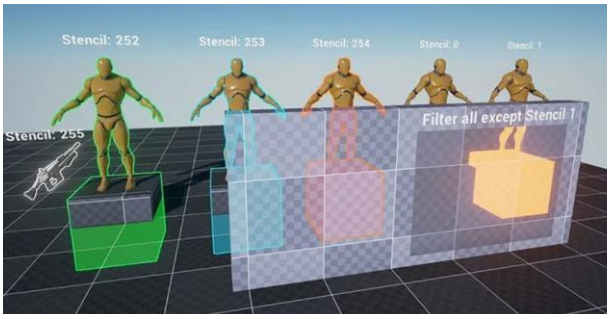


Figure 3.6.1q - Traditional occlusion solutions in gaming, Tom Looman, 2019

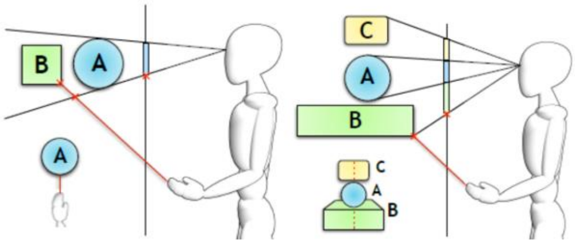
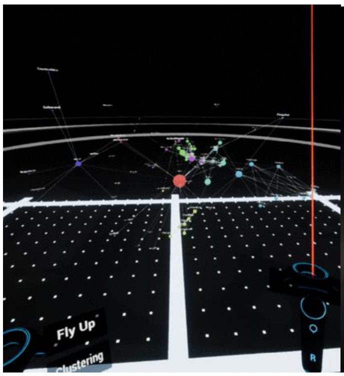
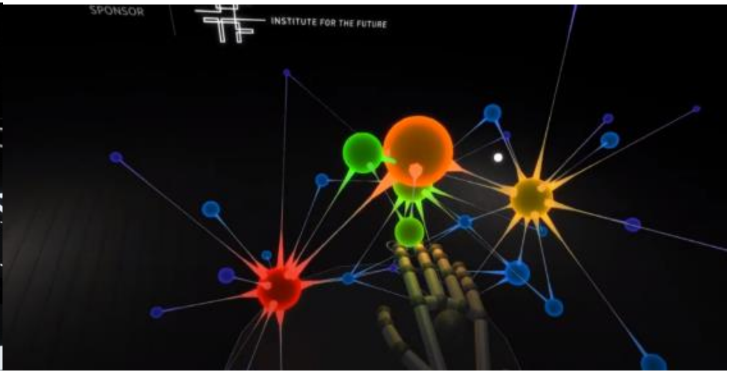


Figure 3.6.1r - Diagram illustrating a user occlusion condition, F.Argelaguet and C.Andujar, 2013

One of the suitable solutions to our case would be rotating the selected Metadata Nodes like a turntable to research the content behind (Figure 3.6.1s and Figure 3.6.1u). Another solution can be the user selecting specific Metadata Nodes and bringing them near their viewpoint, any nodes connected to the interacting source could follow the interacting node (Figure 3.6.1t). In addition to the above, text scale and rotation also should be updated. This can be in ratio to the source Metadata Node scale or can be the distance between the user and the Billboard UI (please find more information in the storyboard in section 3.6.1.b User Interface Design for Personal Work Sphere). Density and population of the nodes are vital to consider, too and this might be solved by scaling the selected Metadata Nodes (Figure 3.6.1v and Figure 3.6.1w). The information LOD can be high when the selected nodes are larger, it can be low when the selected nodes are smaller. The text size can be small for higher information LOD, and the opposite for lower information LOD.





Left: Figure 3.6.1s - Example of Metadata Node navigation using a turntable feature, GraphVR
Right: Figure 3.6.1t - Physics based interaction with nodes, Force Directed VR

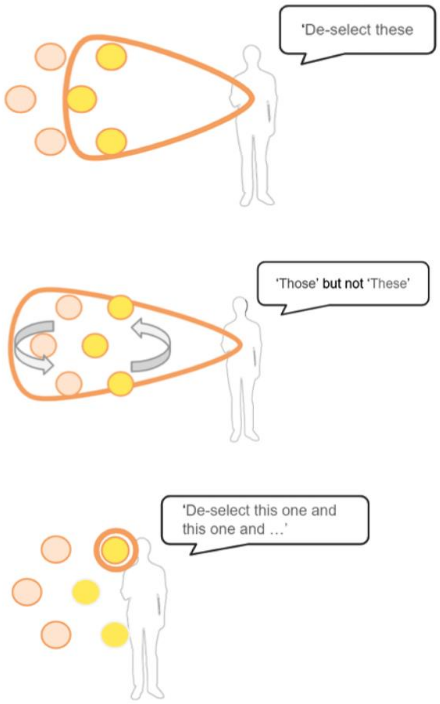
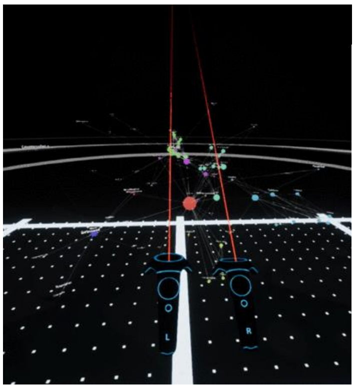


Figure 3.6.1u - Diagram illustrating precision and accuracy in the user selection of Metadata Nodes



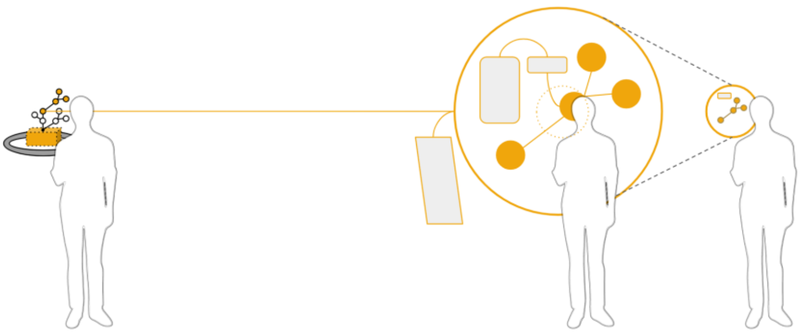


Figure 3.6.1v - Example of scaling a Metadata cluster, GraphVR
Figure 3.6.1w - Diagram illustrating a Metadata Node scale and the Metadata visualisation Level of Details

Another aspect to consider to interact with the Metadata Nodes is the maximum number of nodes to be able to visualise using game engines in the first place. In other words, there might be a limitation in the number of queryable information at once. This is because of performance issues, in fact, GraphVR (2019) explains that the number of loadable sphere actors is "... typically with a maximum value of around 300 nodes. This limit is due to the management of the number of Actors, in fact there is one for each node. It is also necessary to consider all actors that represent the arcs, the lights, the avatar, the billboard and everything that is part of the scene. Being the compute complexity very high, it has become necessary to find a compromise between rendering quality and system fluidity by reducing the Level of Details of the actors".

Minimising the Query User Interface would be helpful for the same reason for the Personalisable and Customisable User Interface, this will also help the user to have more free space to work on other tasks, or to focus to work on the Design Object modification using other proprietary modeling software and applications by completely hiding the Query User Interface (Figure 3.6.1n and Figure 3.6.1x).

There are a few examples that attempt to visualise Metadata inside VR, such as GraphVR and Force Direct, both developed using Unreal Engine 4 (Figure 3.6.1s and Figure 3.6.1t). These examples use a two-dimensional layout similar to *Gephi's 2D layout* and a three-dimensional based layout called Force-Directed layout. For PrismArch Metadata Node interaction, we would not require such extreme physics based animations, however, it is ideal to implement minimal haptic feedback for the demarcation tool in order to design an intuitive user interaction.

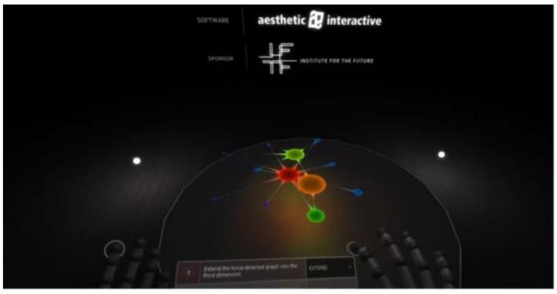


Figure 3.6.1x - Example of minimising a Metadata Node cluster, Force Directed VR

PrismArch Medata Nodes can be visualised by:

- Spawning blueprint actors inside the VR space with math calculations
- Particle simulations such as Unreal Engine Niagara Simulation System (e.g. Flocking simulation and Position Based Dynamic simulation etc)

However, particle simulation is not a VR suitable solution because it is performance expensive (Figure 3.6.1y) and considering the fact that each Metadata Node would include programmed functionalities, it would be ideal to spawning actor class objects inside the VR world.

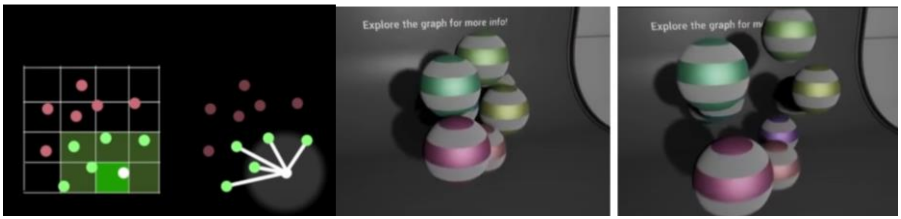
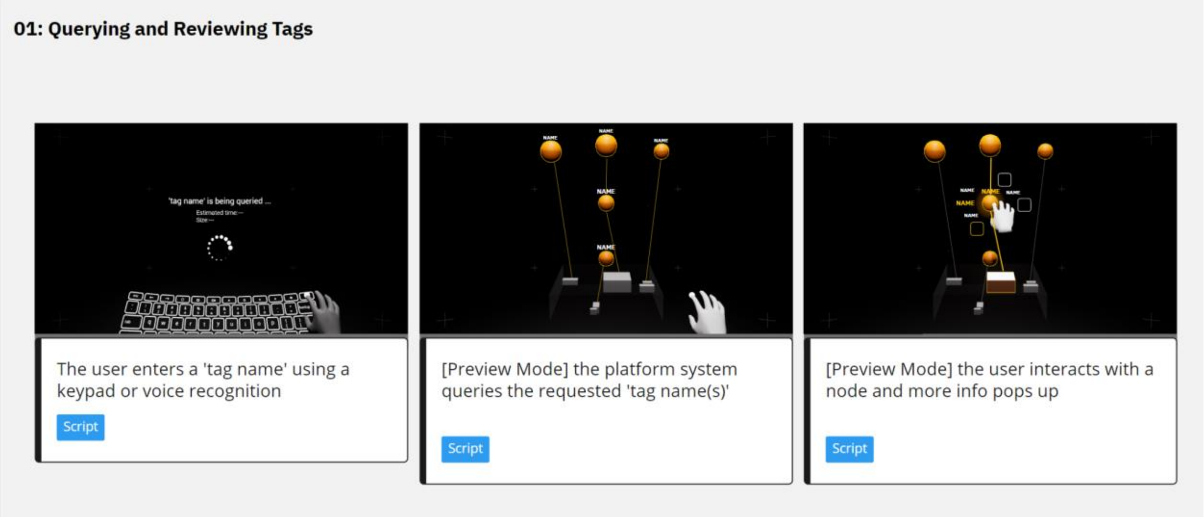


Figure 3.6.1y - Flocking Simulation (left) and Position Based Dynamic Simulation (right), Weiner 'Niagara Effects Inside Unreal'

STORYBOARD: Querying and Interacting with Metadata Nodes and Design Objects



01: Querying and Reviewing Tags

'tag name' is being queried ...
Estimated time:--
Size:--

The user enters a 'tag name' using a keypad or voice recognition

Script

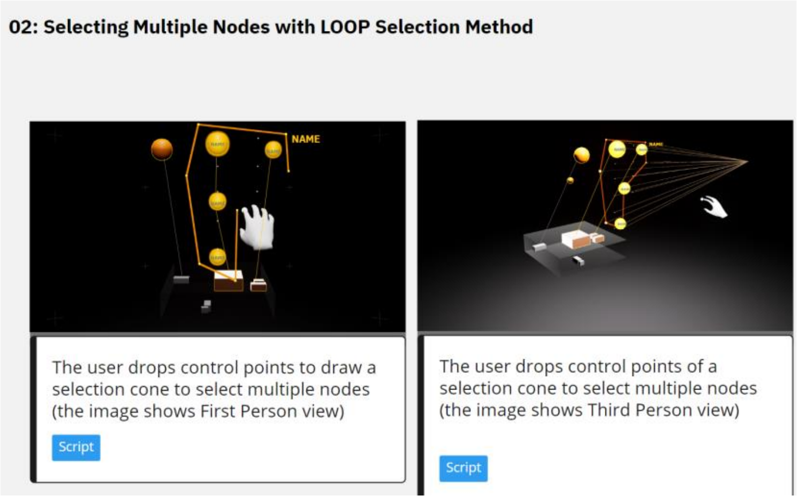
[Preview Mode] the platform system queries the requested 'tag name(s)'

Script

[Preview Mode] the user interacts with a node and more info pops up

Script

Figure 3.6.1z1 - Query Storyboard Part 1/3

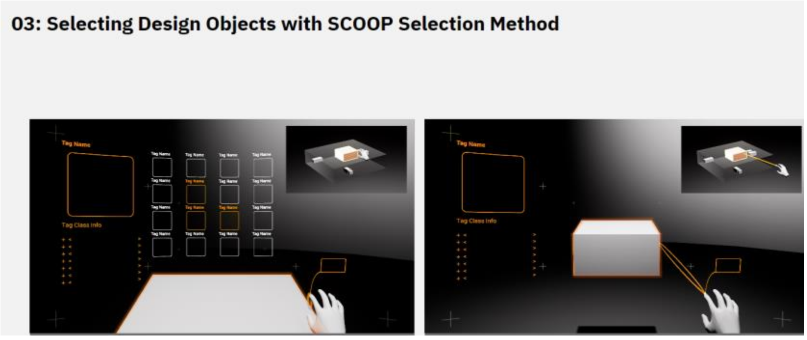


02: Selecting Multiple Nodes with LOOP Selection Method

The user drops control points to draw a selection cone to select multiple nodes (the image shows First Person view)

The user drops control points of a selection cone to select multiple nodes (the image shows Third Person view)

Figure 3.6.1z2 - Query Storyboard Part 2/3



03: Selecting Design Objects with SCOOP Selection Method

Figure 3.6.1z3 - Query Storyboard Part 3/3

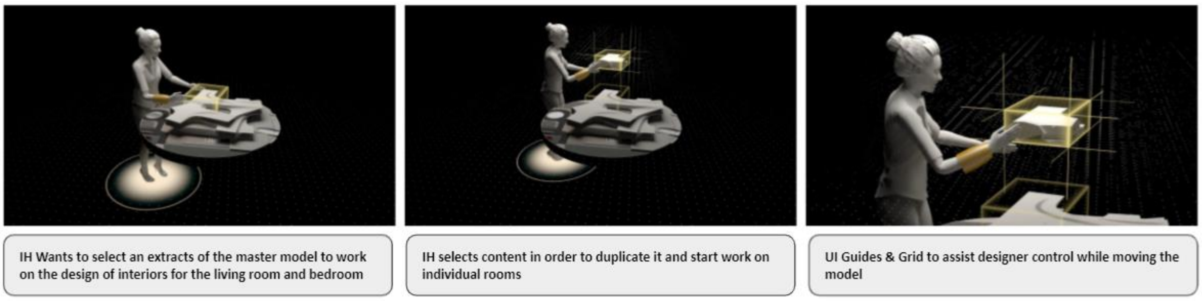
■ 3.6.2 Architecture-Specific UI Elements

3.6.2.a User Interface Design for Scope and Task Demarcations

This section proposes architecture-specific UI elements. It should be highlighted that the proposal below should also work across disciplines to achieve a continuous experience throughout the platform. The section content is placed because the functionalities are expected to be used by architectural disciplines in the early project stage, which is a rare case for the other two disciplines.

IH may wish to create an instance of the PrismArch Central Model (PCM) in order to develop a design proposal. This would mean that the instance of the model would be placed in their PWS on Sphereing Level 1, visible only to individual IH at this stage. IH can then manipulate the instanced DO using 3D modeling tools such as Mindesk, and create a number of design options.

The example below illustrates IH intending to work on a particular aspect/space of a project. IH wants to extract a portion of the model and isolate it in order to develop a design of interior space, IH has to define a clipping container. The content within this container is then trimmed out of the PrismArch Central Model (PCM) and extracted as an instance in PWS.



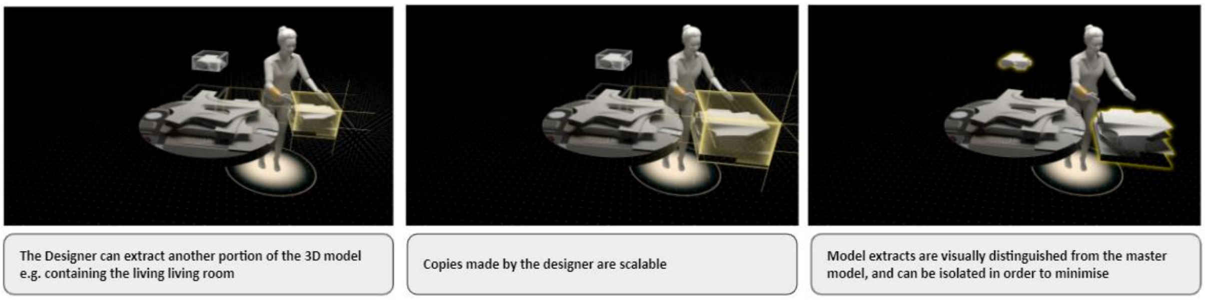
IH Wants to select an extracts of the master model to work on the design of interiors for the living room and bedroom

IH selects content in order to duplicate it and start work on individual rooms

UI Guides & Grid to assist designer control while moving the model

Figure 3.6.2a - Extracting a portion from the PrismArch Central Model

The extracting container leaves a trace in the original location, where the portion was taken from, to allow for its insertion back to the original location. IH can extract a number of instances, which should be visually distinguished from the PCM.



The Designer can extract another portion of the 3D model e.g. containing the living living room

Copies made by the designer are scalable

Model extracts are visually distinguished from the master model, and can be isolated in order to minimise

Figure 3.6.2b - Creating instances of the PrismArch Central Model extracts

3.6.2b User Interface Design for Archiving and Retrieving Design Options

This may include review of previous proposals, alternative design options, etc.
In this scenario the designer recalls previous versions of the massing proposed for the Villa.

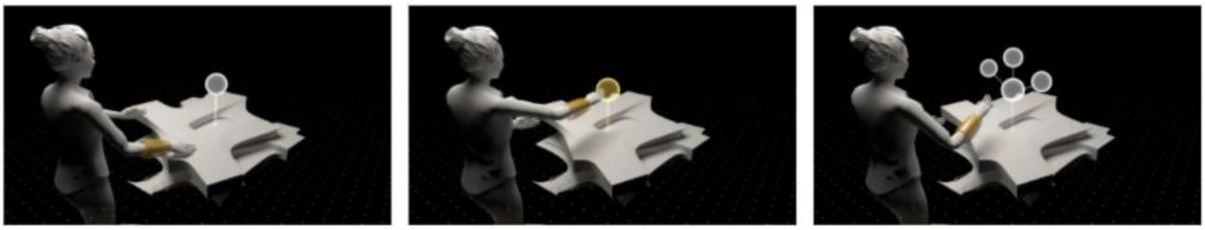


Figure 3.6.2c - Retrieving the archived Design Objects using the Query User Interface and Medata Nodes 1/2

In order to do that IH queries historic 3D models of the villa. The result is being displayed in a form of metadata nodes.

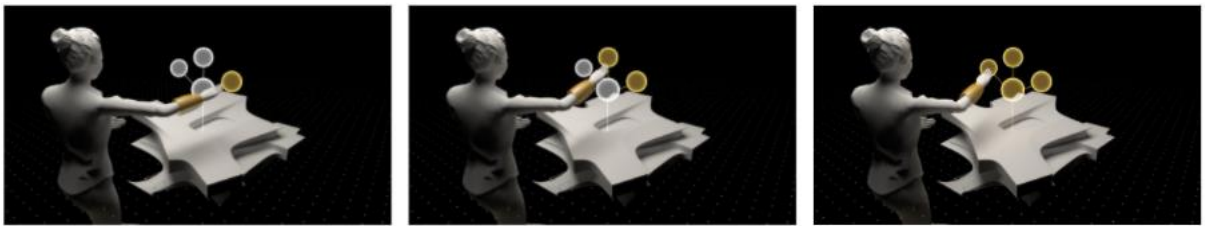


Figure 3.6.2d - Retrieving the archived Design Objects using the Query User Interface and Medata Nodes 2/2

IH can select a number of models he/she would like to recall by selecting corresponding metadata nodes. After the selection is confirmed. The models are Instanced into their PWS at Shereing level 1.

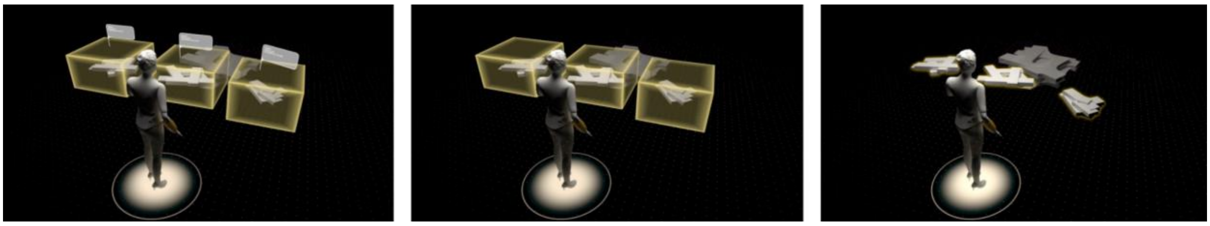


Figure 3.6.2e - Loading the archived Queried Design Objects

The Instances with their tags are editable in PWS

3.6.2.c User Interface Design for Creative Boards: Mood Board and White Board

During the design process, the user has access to external references. These references can be mixed-media, such as hand sketches, site photos, 3D sketches, material image references, clients' cultural references. For more information. Please visit the deliverable D1.1.

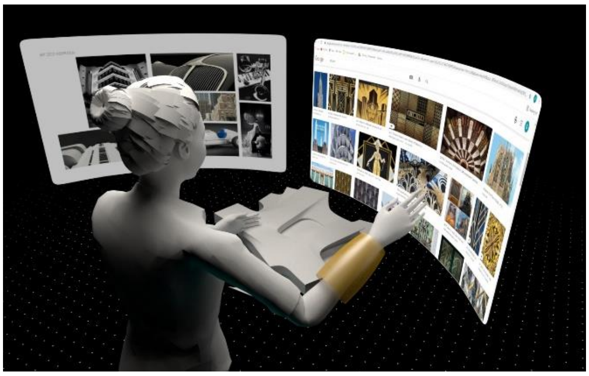


Figure 3.6.2f- White Board and Mood Board

3.6.2.d User Interface Design for Analytics and Simulations

Proprietary simulation tools and applications
Environmental (Sun, Wind, Thermal modeling), CFD, acoustics(sound simulation), structural etc..any other custom analysis play a vital role in the evaluation process. Architects also include the simulation results, calculated with their prefered proprietary tool, in their submission documents. These are also often presented in front of potential manufacturers to deliver their ideas and also helpful for feasibility studies in early design stages.
Results Inform decision making during the design process, proof of concept and are helpful during the client presentations. The Results are visualised through heat maps and can be loaded with a PrismArch tag via a user's preferable proprietary tool.

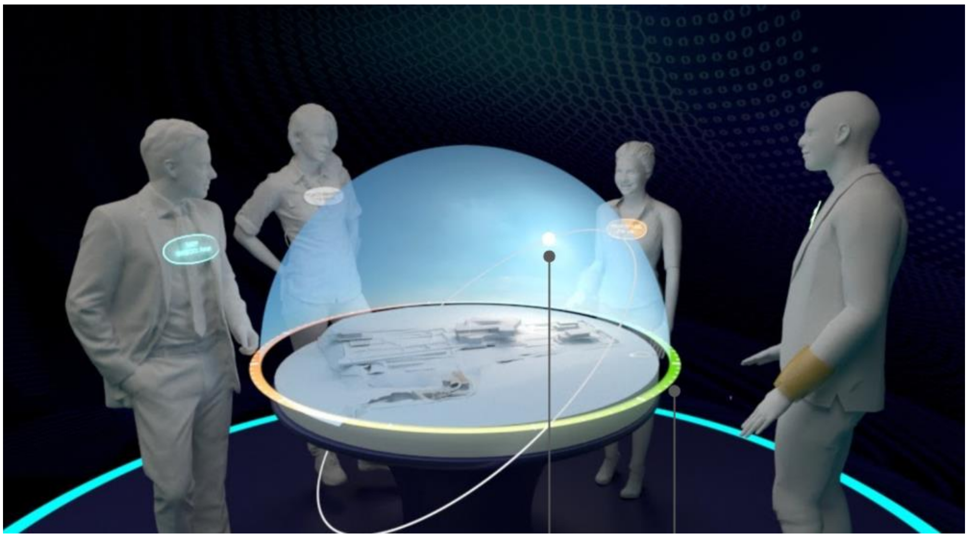


Figure 3.6.2g - Loading Sun Simulations inside an arranged Meeting Sphere

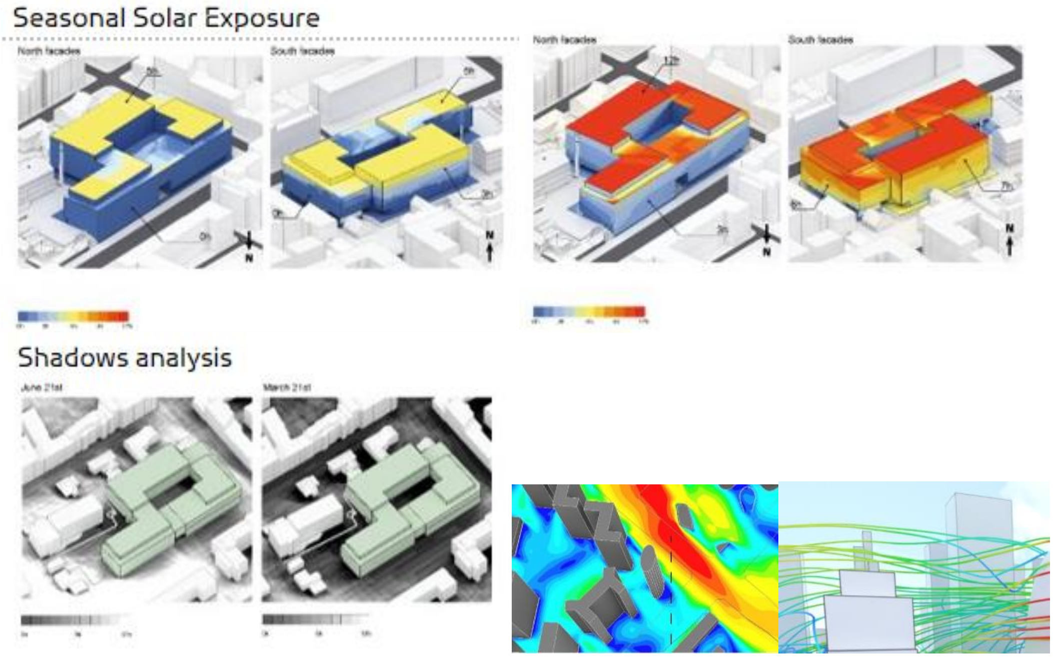


Figure 3.6.2h -Examples of Environmental analysis Image Copyright: ZHA

■ 3.6.3 Structural Engineering-Specific UI Elements

Conceptual Visualization of Structural Elements

The vast majority of structural elements can be defined using a small number of simple abstract representations - lines to represent columns and beams, meshes to represent walls and floors, and so on. These basic elements are assembled into the schematic 'stick' or 'mesh' models used throughout contemporary structural engineering design. These abstract conventions have several advantages:

- The engineer can construct simple yet accurate 'mental maps' of a structural design.
- Analysis algorithms can operate on lightweight memory objects, which are easily convertible into graph structures (nodes, edges) and easily processable by linear matrix solvers.
- Graphics algorithms can visualize the required information with a minimal number of rasterization primitives.

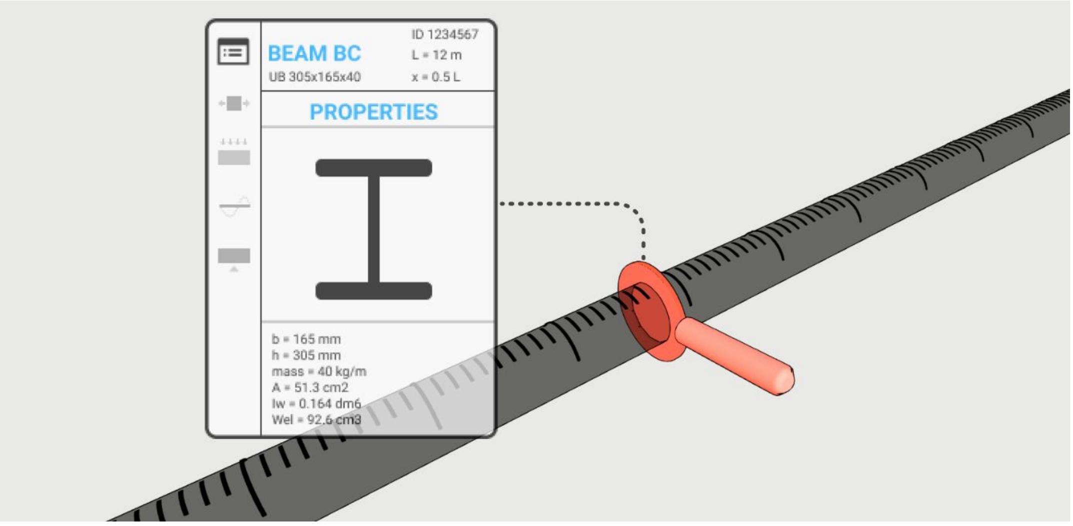
We therefore propose a UI for structural engineering elements in VR that builds upon these established conventions, expanding them to offer visually engaging and interactive elements for the user, that can be queried in real-time. By using 3-dimensional geometries for all elements, we believe the PrismArch UI will always properly convey a sense of depth in VR, thus avoiding the problematic flattening and overlapping data visualisation issues that currently occur in existing 2D structural engineering interfaces. (For examples, see Deliverable 6.1, Section 3.2, and also Section 5.2 of this document: *PrismArch Structural Engineering Interviews*).

Linear Elements as Thickened Line

In order for linear elements (beams or columns) to maintain an engaging and legible appearance in VR, we propose they are displayed as thick pipes instead of dimension-less wire frames. This allows the user to perceive depth more easily in VR and avoid moire-link overlap illusions, and it also allows the abstract elements to shade each other, further enhancing the user's spatial perception. Lastly, it provides an opportunity to display information on the elements themselves - such as ruler marks, or important annotations/ notes - that are attached at specific points along the beam or column.

The Query Handle

Any linear element should be able to be queried about its local properties. We propose the *Query Handle*, a 3-dimensional UI element that can be dragged along the length of a structural element using common VR gestures (for example *fist clench*, or *trigger press* followed by dragging horizontally / vertically). As the user drags the query handle along the length of the element, the underlying data is sampled locally and presented as options to the user via a floating, 3D window.



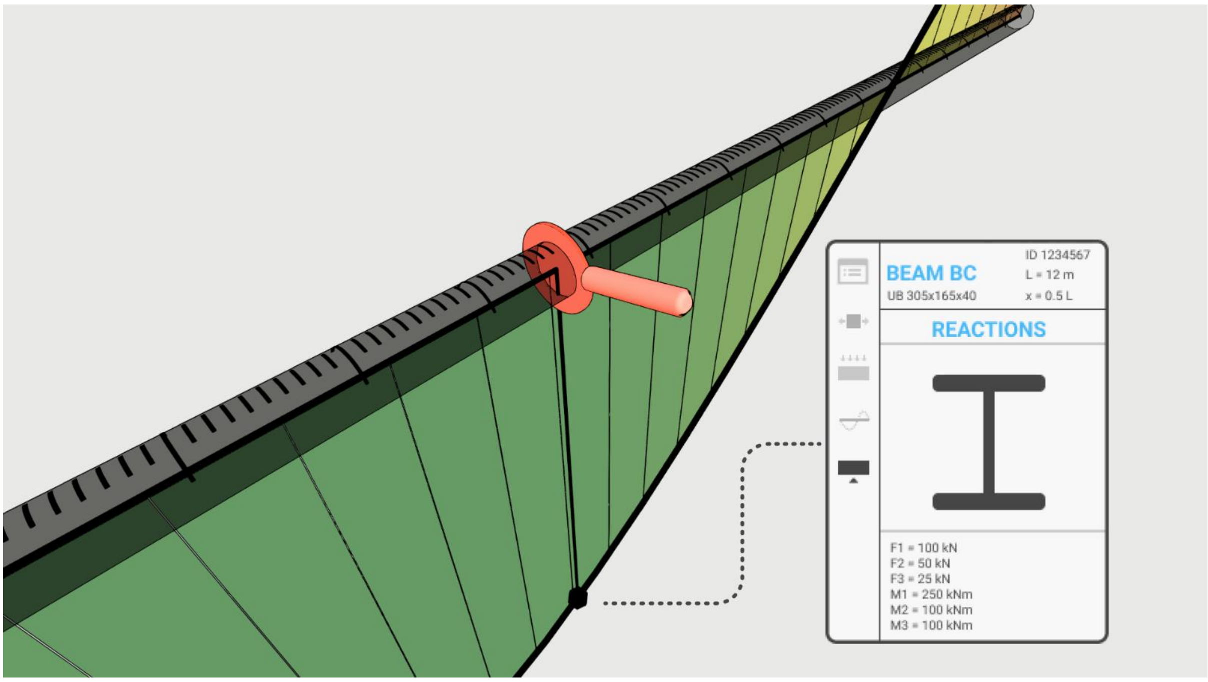
L = 12 m
x = 0.5 L
b = 165 mm
h = 305 mm
mass = 40 kg/m
A = 51.3 cm2
Iw = 0.164 dm6
Wel = 92.6 cm3

Figure 3.6.3a -Sampling a structural element at a location and viewing its sectional properties.

At a sampled location, users can choose to display a number of data. For example:

- General sectional properties of the element
- Continuous Stresses (forces) applied to element
- Strains (deformations) of the element in response to stresses
- Reaction diagrams (moment, shear)

In the example below, we present an example of a user displaying the *Bending Moment diagram* of a beam, and querying it locally.



L = 12 m
x = 0.5 L
F1 = 100 kN
F2 = 50 kN
F3 = 25 kN
M1 = 250 kNm
M2 = 100 kNm
M3 = 100 kNm

Figure 3.6.3b - Sampling a structural element at a location and inspecting its bending moment diagram at a specific location.

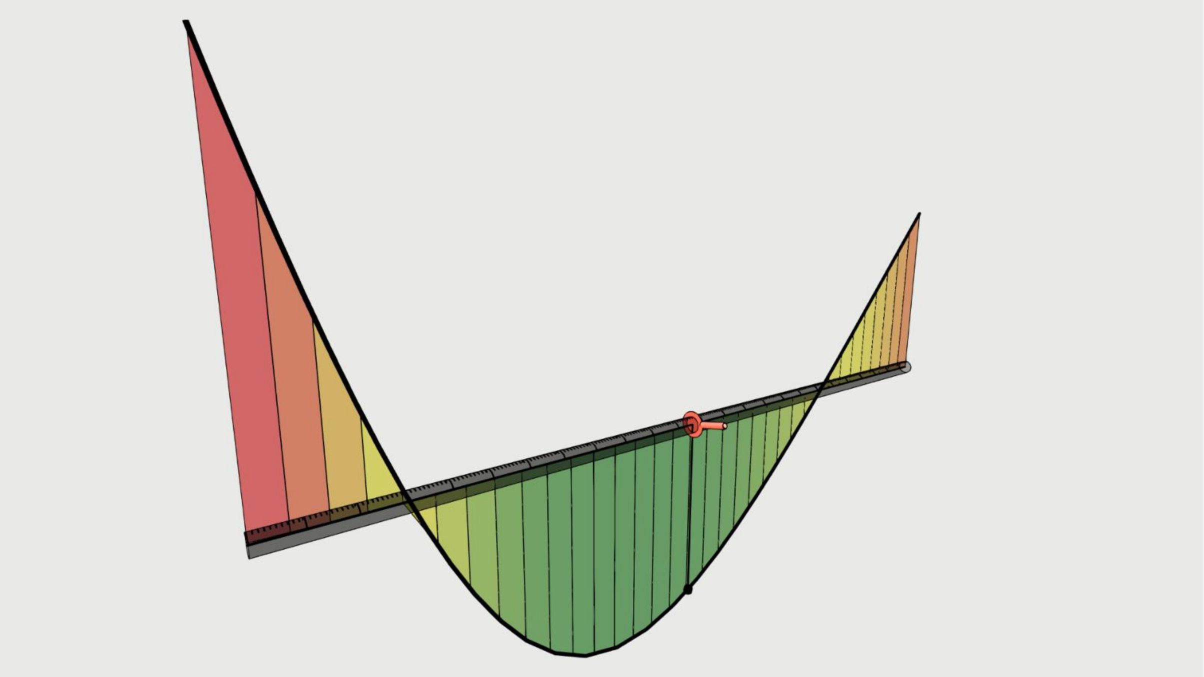


Figure 3.6.3c - Visualization of structural member's bending moment diagram, and sampling at a specific point along the length with the aid of a "grabbable" 3D handle.

The Force Arrow

In the case of non-continuous stresses - such as *concentrated point loads* - the continuous sampling along the length of an element cannot be reasonably defined. Instead, such types of forces should be represented as cylindrical 3D arrows, that are legible from all angles, and each with a paired disk element that indicates their exact application point along the linear element. Textual descriptions of their magnitude floati next to them in 3D space. The scale of these arrows has to be either dynamic or user-aware, to avoid situations in which a users zooms in, and these arrows become enlarged and obscure the underlying model.

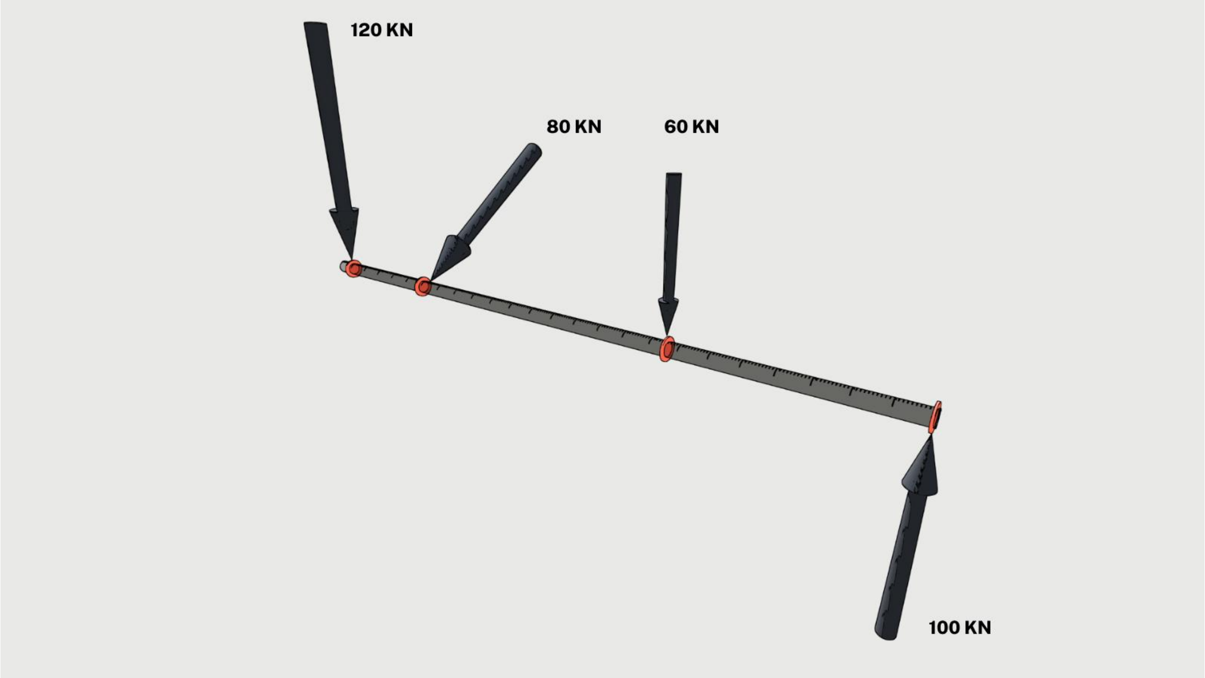


Figure 3.6.3d - 3-Dimensional point loads.

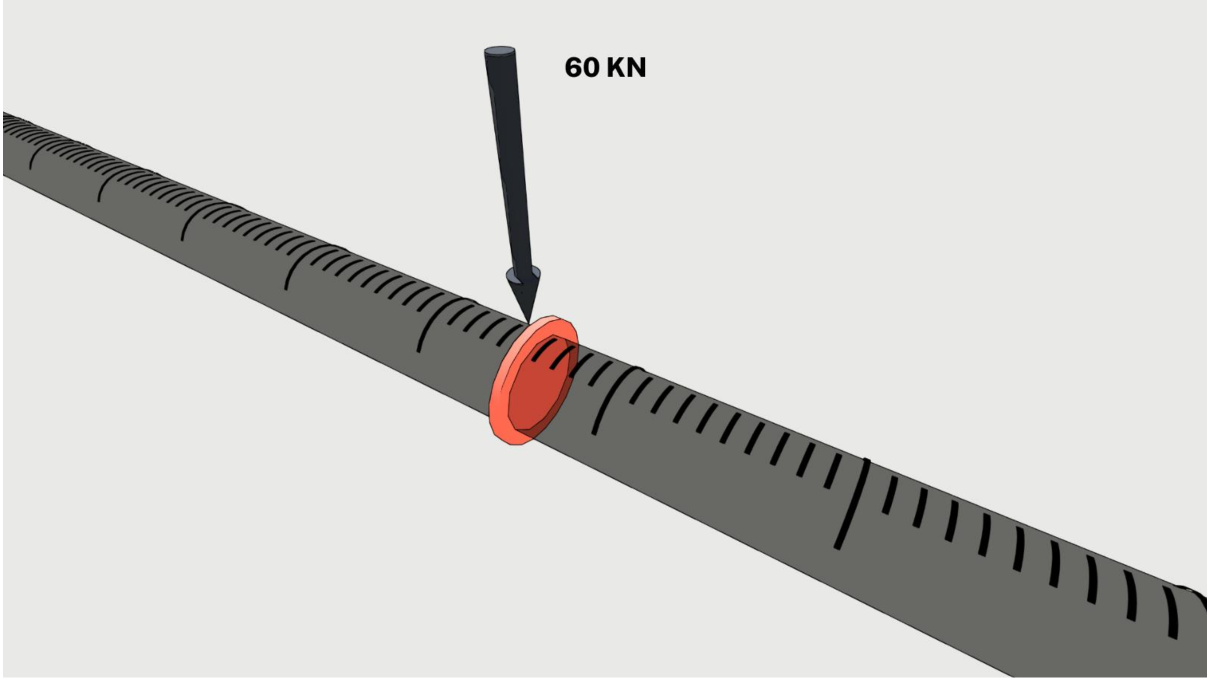


Figure 3.6.3e - 3-Dimensional point loads. Zooming-in changes the scale of the arrows and disk indicators.

The Planar Arrow Group

In the case of linear distributed loads, the requirement is for a UI element that can demonstrate the directionality of the load, can be clearly different visually from point loads, can have the ability to be queried locally using the query handle, and finally can be visible even from very oblique angles. For these reasons we propose the *Planar Arrow group*, essentially a combination of a planar diagram corresponding to the distribution of Load along the length of the beam and an array of 3D arrows, indicating the direction of the forces and allowing the Load to be partially seen even from angles parallel to the beam's direction.

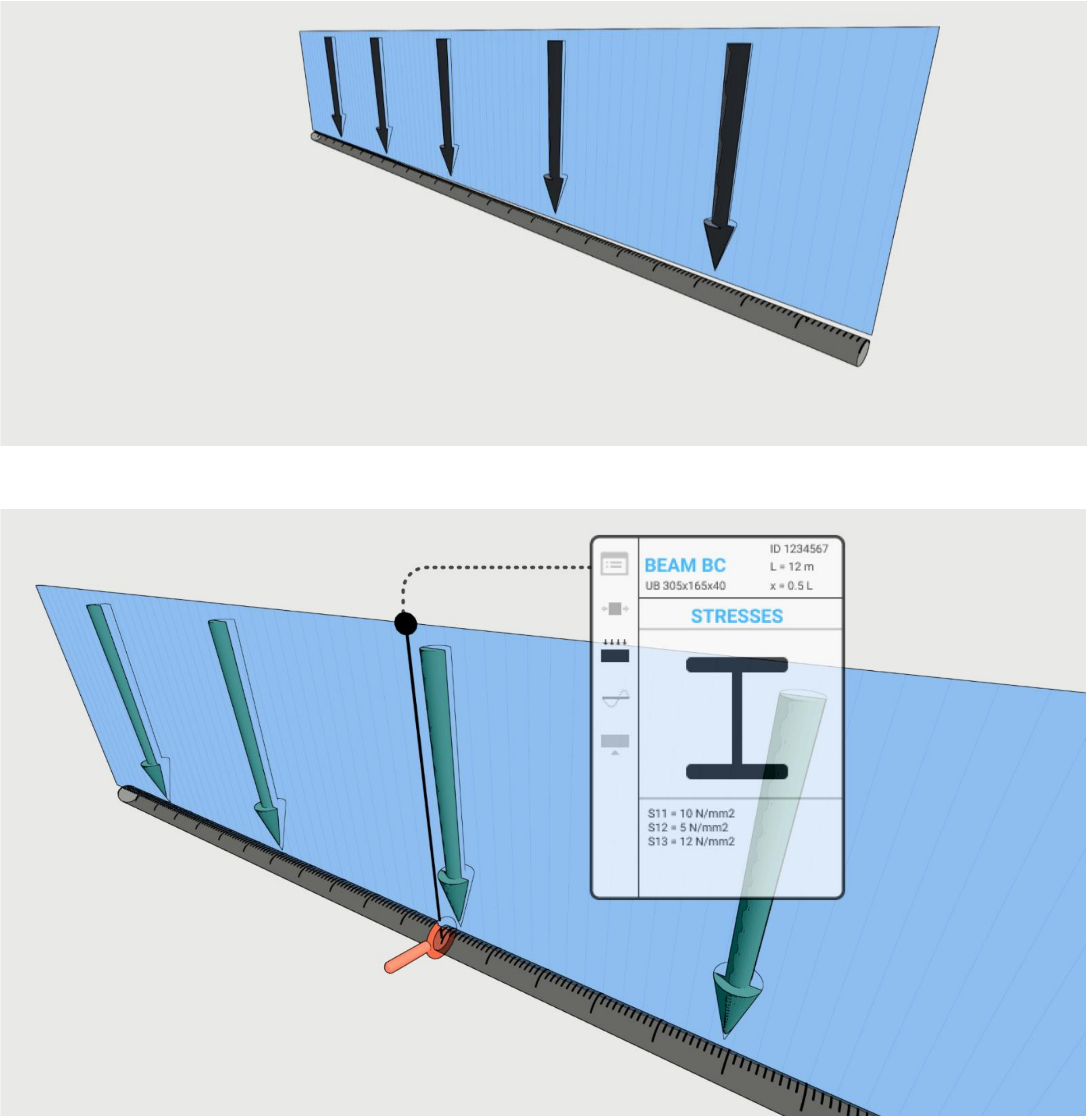


Figure 3.6.3f - 3-Dimensional Line Loads (above) and local sampling of the Line Load (below)

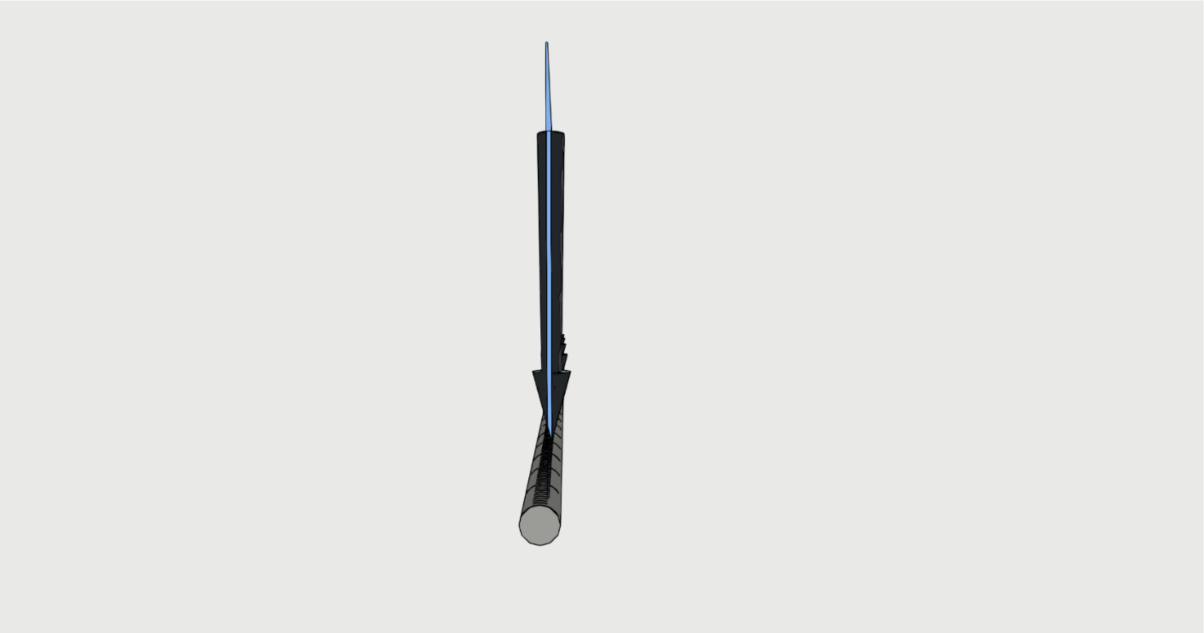


Figure 3.6.3g - 3-Dimensional Line Loads. 3-D arrows attempt to solve the issue of looking at the element from the side.

Support Conditions

No structural engineering element can be defined without support conditions at its end points. By support conditions we mean the degrees of positional and rotational freedom that an element has at connection points with other elements. These support conditions are clearly defined in existing structural engineering literature and practice, and use a standardized international set of 2D symbols.

We therefore propose that the PrismArch VR representation of these conditions be simply a set of 3D symbols that directly inherit the main properties of their 2D counterparts, in order to avoid any ambiguities as to what condition each refers to.

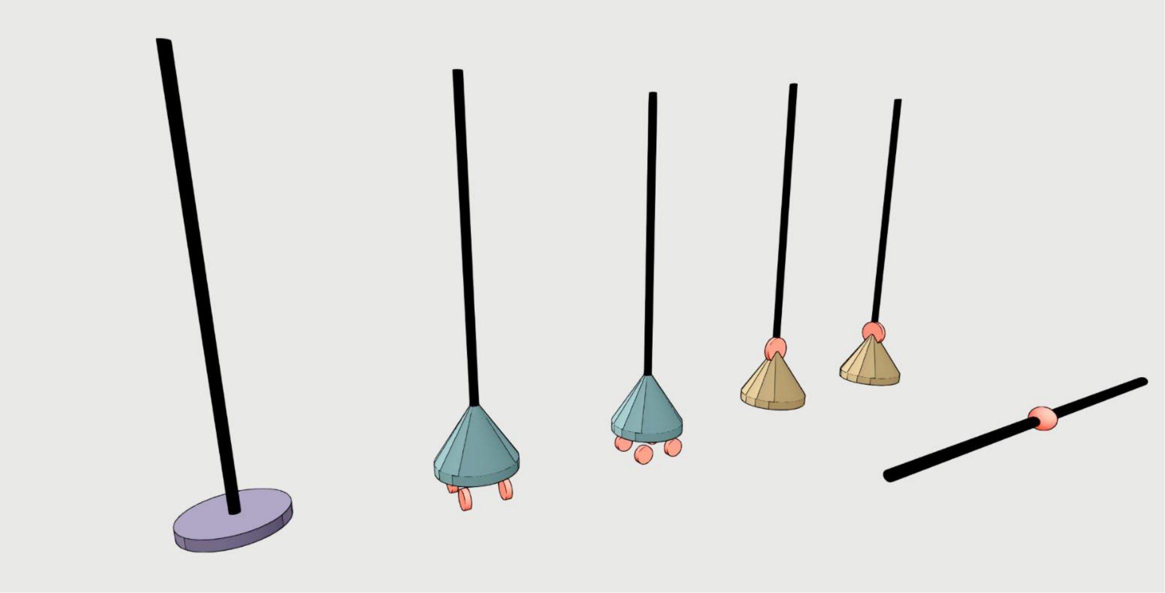


Figure 3.6.3h - 3D Representation of most common support types. From left to right: Fixed, Roller X, Roller Y, Pinned X, Pinned Y, Beam Hinge.

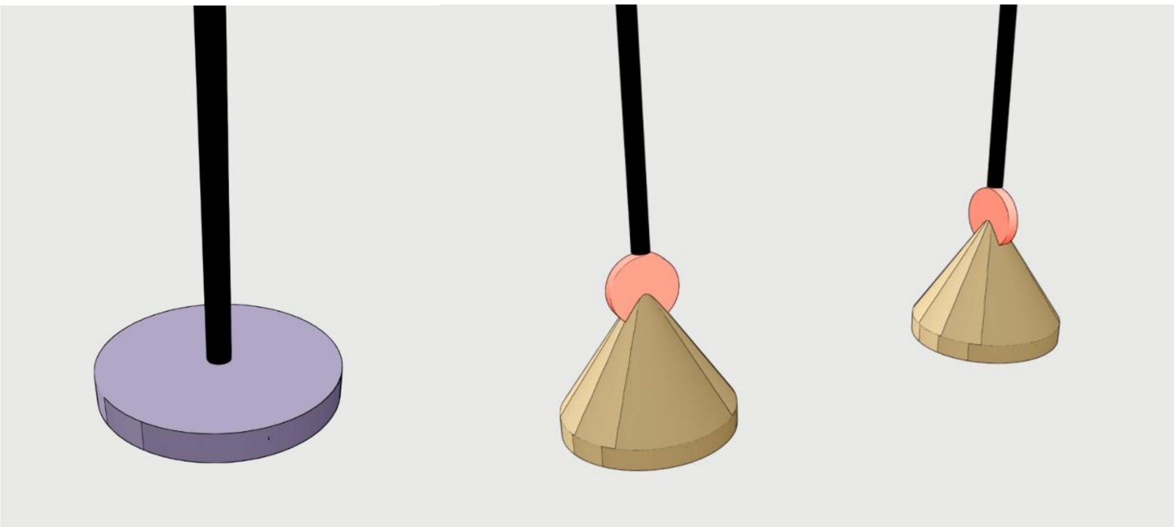


Figure 3.6.3i - 3D Representation of Fixed support (left) and Pinned supports in 2 orthogonal degrees of freedom (right)

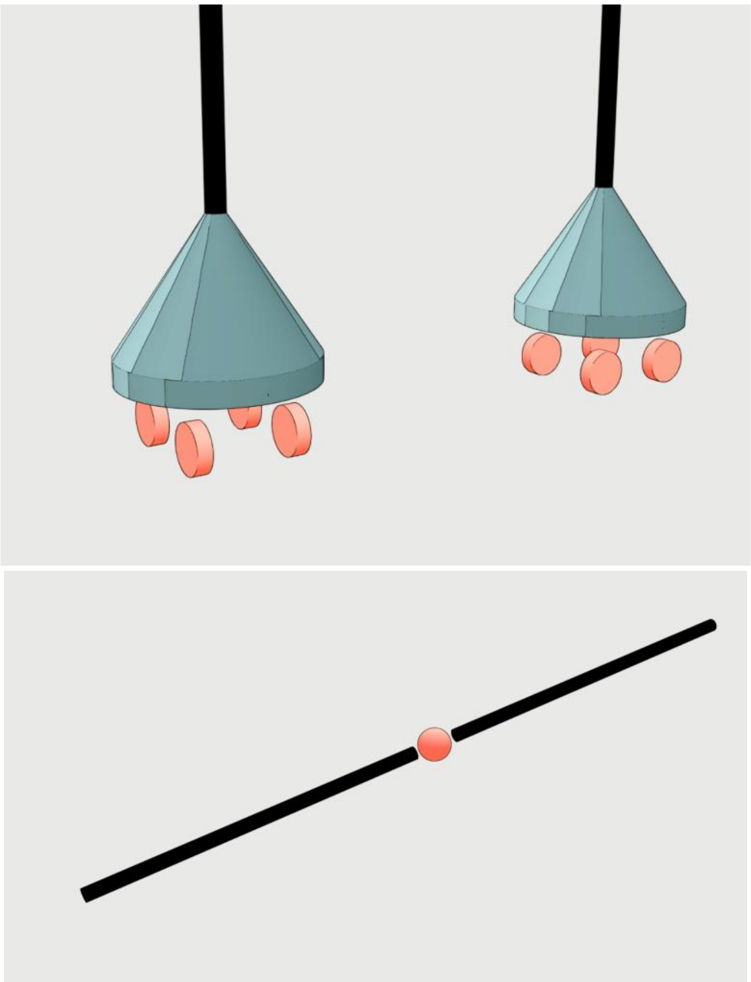


Figure 3.6.3j - 3D Representation of Roller supports in 2 orthogonal degrees of freedom (above) and Beam

Hinge (below)

Mesh Visualization

Mesh elements are displayed with false-colours to represent ‘embedded’ analysis data. This colouring is most easily achieved via vertex colours, however texture mapping is also possible, depending on the format of the input analysis data. The types and categories of analysis data that must be achievable through this process are defined in the previous *Displaying-Interacting-Exporting* table (Section 3.5.2.).

To aid legibility of the data, it is greatly advantageous if the false-colours assigned to meshes are not allowed to blindly linearly interpolate between the underlying data values present at the mesh vertices. Instead, a far more coherent and legible visual effect is achieved if the output colours are ‘bucketed’ or ‘banded’ to produce gradated contour plots (the effect is akin to isosurfacing techniques). See the image below for a good example of this effect:

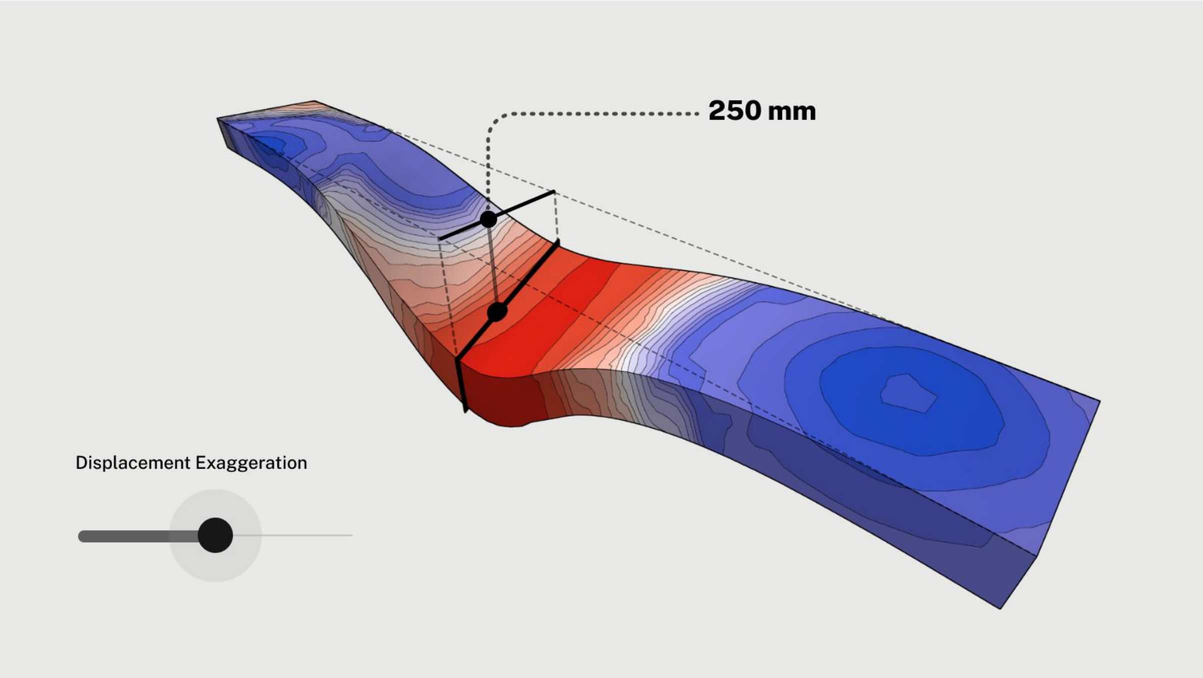
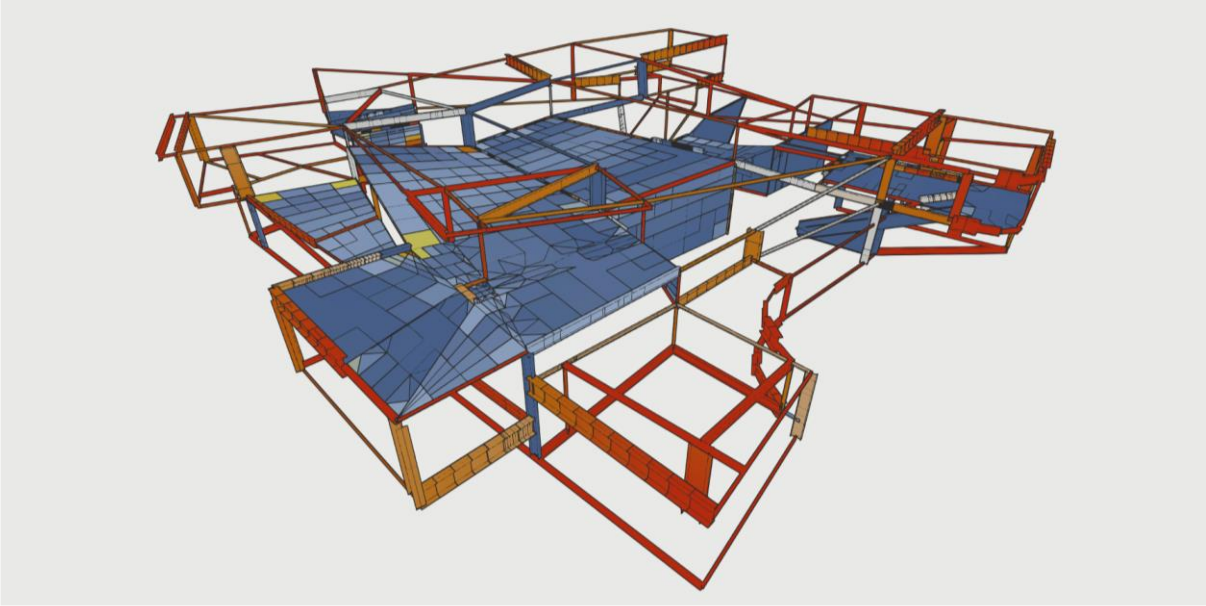


Figure 3.6.3k - Interactive visualization of a beam’s deformation.

The deformation shown above is based on displacement data stored on the structural mesh’s vertices as normalized values (0-1 range). The underlying raw data is used to colorize and contour the mesh interactively in the HLSL shader’s Fragment stage, while the exaggerated deformation is being calculated in render time in the shader’s Vertex stage.

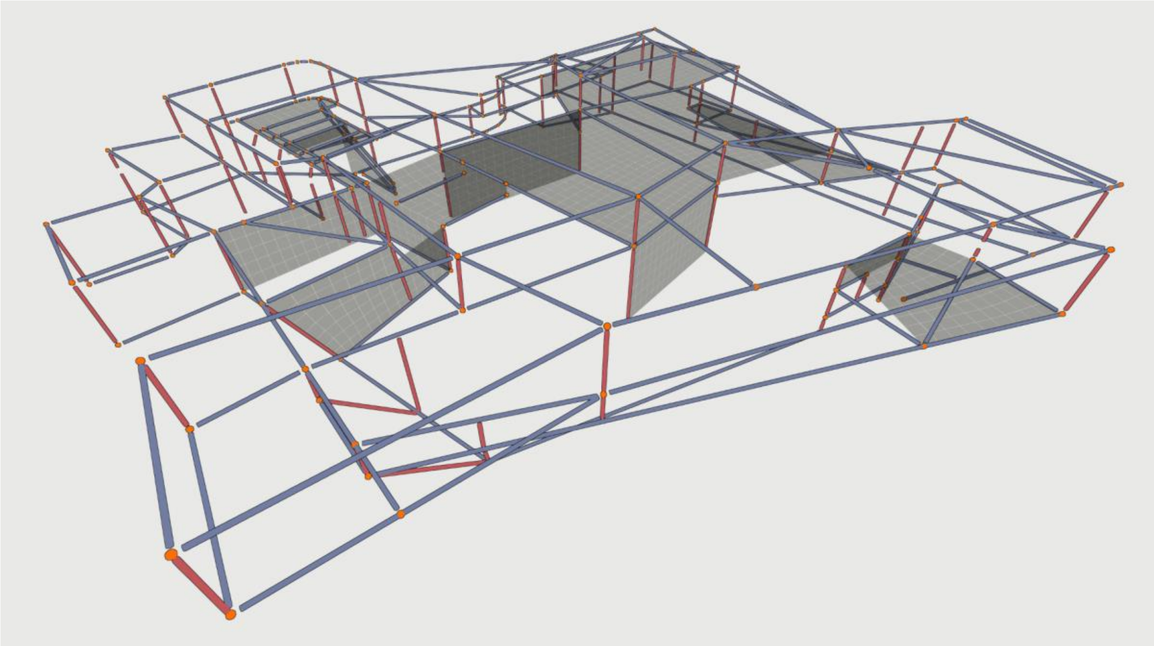
Full Building Visualization

Both linear and mesh visualization modes are applicable to models of entire buildings. In this case however, the individual per-element data (described above) should be hidden from view in order to avoid visual clutter, and only basic color information should be retained, to aid the user in understanding the assembly as a whole. It is important that the user retains the ability to still operate global filters: hiding elements based on their type and/ or their attached properties.



■

Figure 3.6.3l - Representation of a full building's structural meshes, colorized interactively by their loads, stored in each mesh's vertices as normalized values.



■

Figure 3.6.3m - Simplified, schematic view of the building’s structure. Columns, beams, nodes and surface elements have different visualization style.

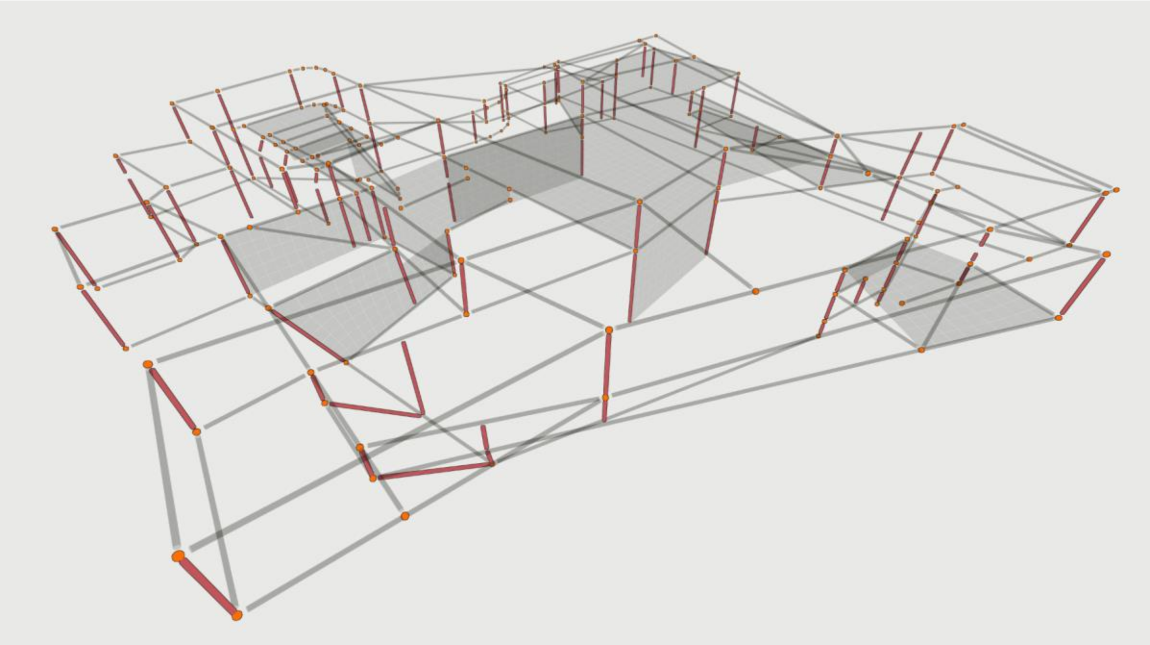


Figure 3.6.3n - Highlighting of the building’s columns.

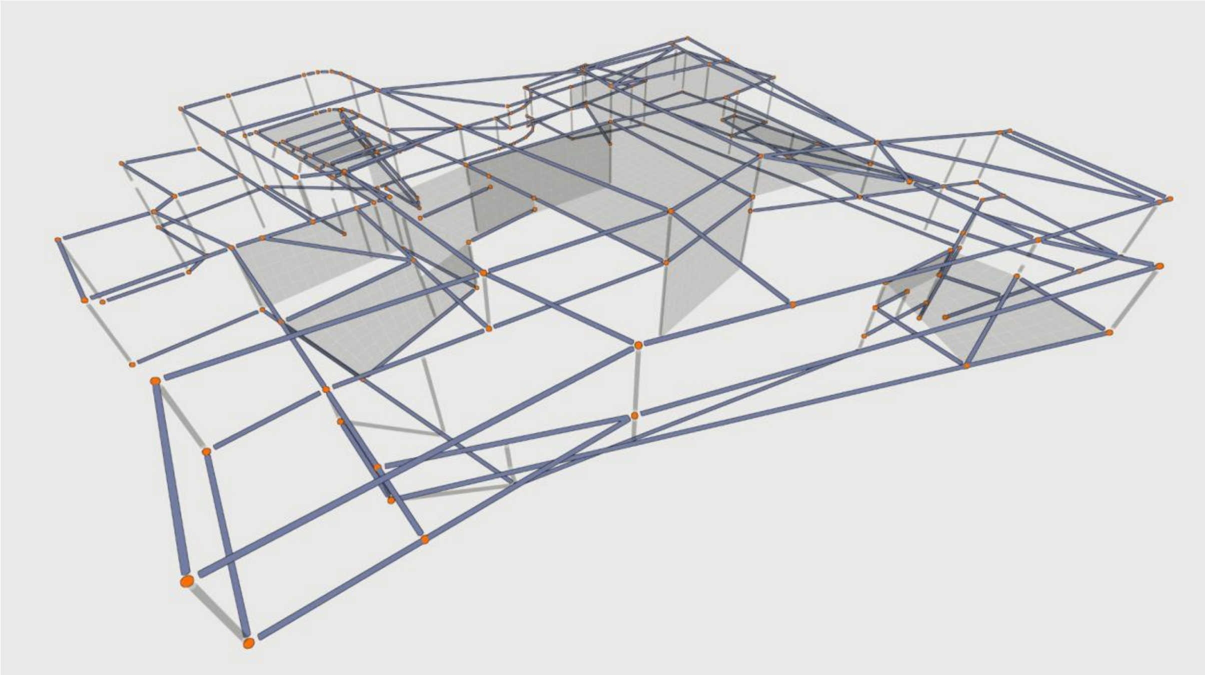


Figure 3.6.3n: Highlighting of the building's beams.

Element Selection

It is rare in structural engineering for a single isolated element to be of interest. More often, that element is of interest in relation to other elements surrounding it. Therefore, we propose an element selection behaviour in which the selection of an element triggers the selection of its neighboring elements too, while all other element in the model switch to a "ghosted" mode (no colour, low material alpha) - allowing the user to focus on the selected element and its immediate relationships.

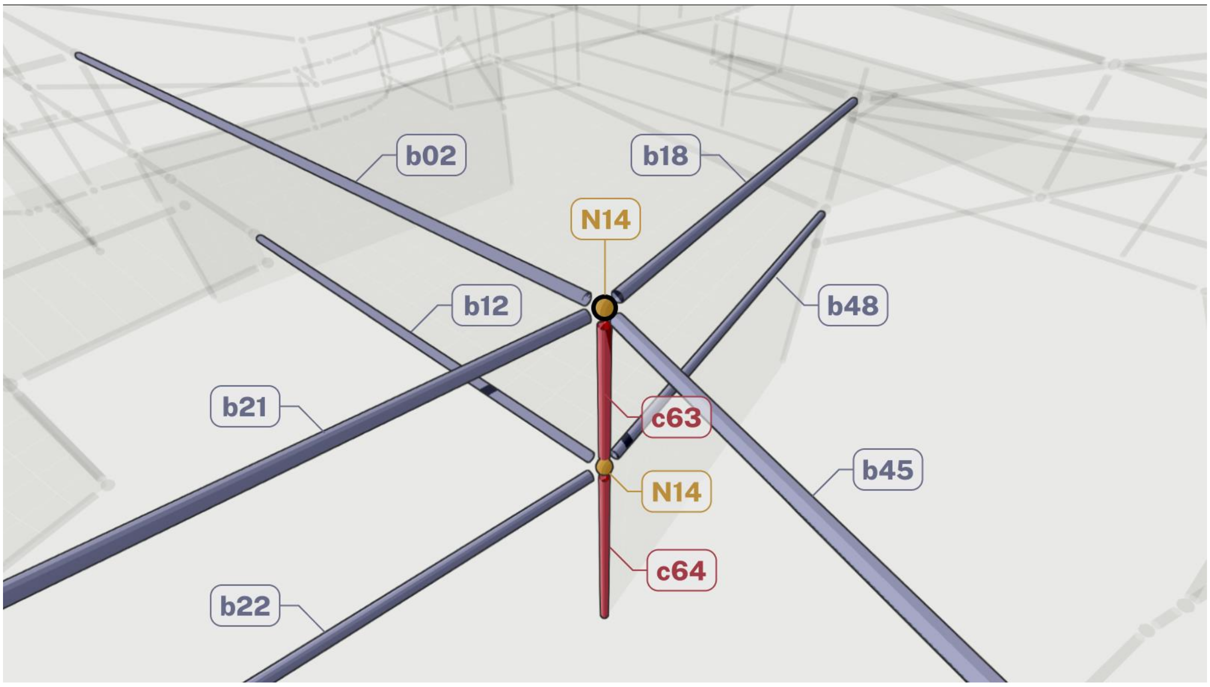
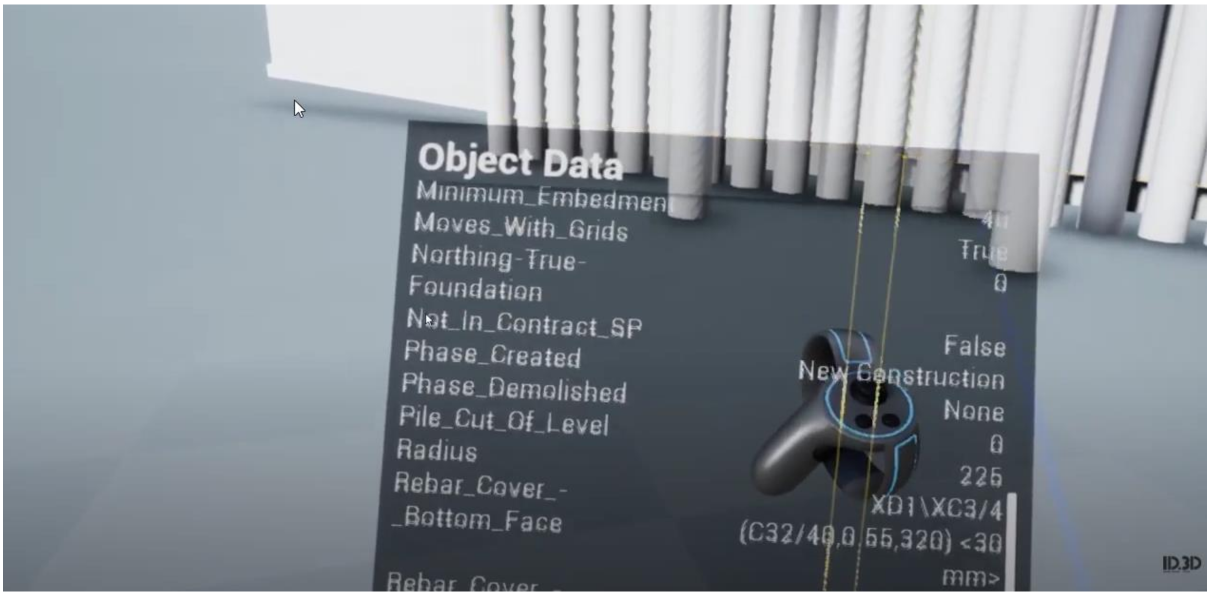


Figure 3.6.3o: Graphically selecting elements, and automatically isolating their topological neighbors

■ 3.6.4 MEP Engineering-Specific UI Elements

The MEP disciplines can have a broad number of UI Elements. Each discipline can take advantage of the innovative tools PrismArch may offer. What would be required is the ability to review, inspect and highlight issues similar to what existing software does in the traditional PC environment. However, the information added as parameters within the project files should be available in the IH's headsets.

We would propose that the best way to visualise the data of an element would be a widget which the user has attached to their wrist. The user can then point at an element and select it and the relevant data appears on the wrist display.



Object Data

Minimum_Embedment	4
Moves_With_Grids	True
Northing-True-Foundation	0
Not_In_Contract_SP	False
Phase_Created	New Construction
Phase_Demolished	None
Pile_Cut_Of_Level	0
Radius	225
Rebar_Cover_-_Bottom_Face	XD1\XC3/4 (C32/40,0,55,320) <30 mm>
Rebar_Cover_-	

Figure 3.6.4a BIM widget in VR. Source: https://youtu.be/mhCxFudzFX4

This would aid the process of identifying and reviewing elements and their properties. This could also be enhanced with search or filter tools to aid in finding the desired parameter instead of endless scrolling. This scrolling can be achieved with the motion controller's thumbsticks. The trigger buttons should be used for selection and there should also be an ability to increase/decrease the selection ray curve.

Additionally, the UI should be able to demonstrate clash detection results with the elements highlighted. A good example can be driven from Navisworks software.

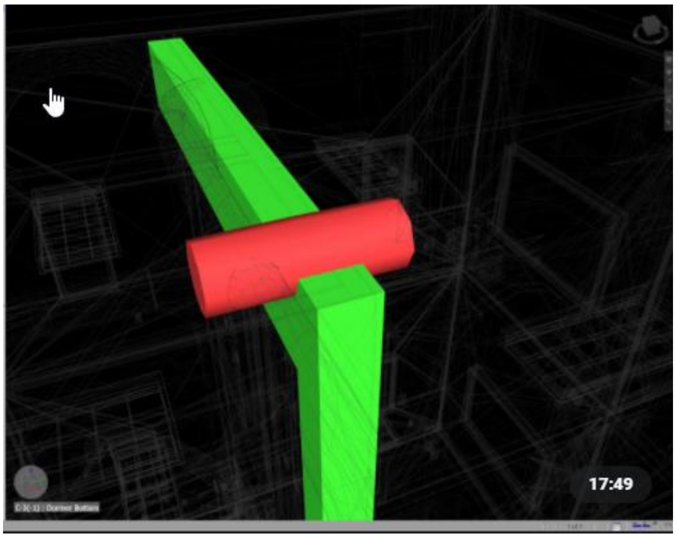
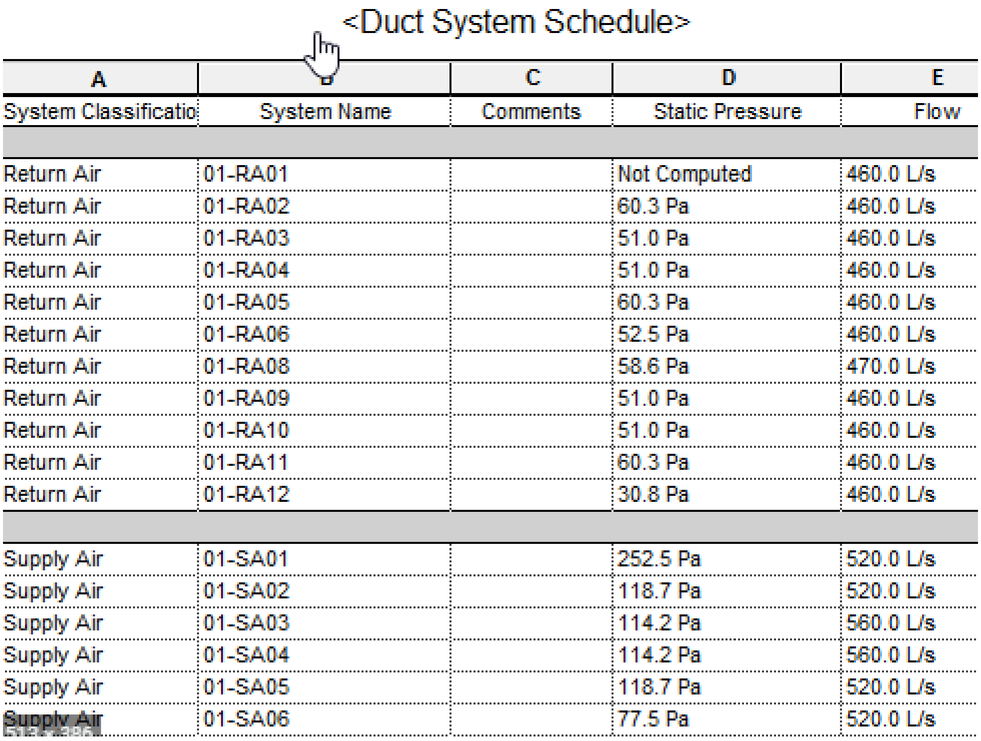


Figure 3.6.4b Example of a clash in Navisworks. Source: https://youtu.be/v6HtMKme3Ck

The ability to inspect and review the clashes should be aided from the clash detection report. The report can be produced in a variety of formats and inserted in PrismArch. Excel, .HTML or .XML files are compatible formats to perform this task. The report should contain all elements ID's, and be searchable within PrismArch.

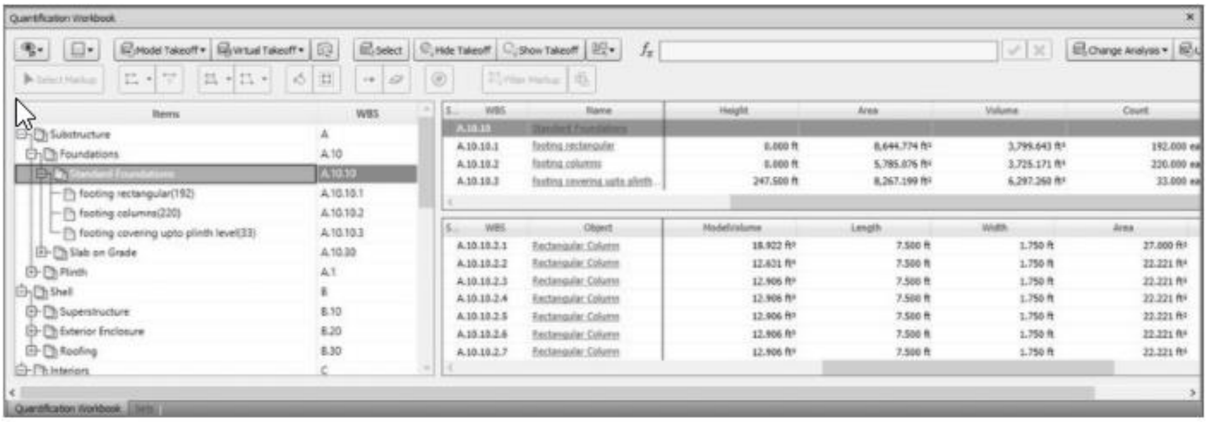
The IH should be able to inspect the schedules and quantifications within PrismArch and these should be updated in case a user makes a change (e.g. deleting an element within PrismArch). This data should be transferable back to the software the information was initially produced within.



<Duct System Schedule>

A	B	C	D	E
System Classificatio	System Name	Comments	Static Pressure	Flow
Return Air	01-RA01		Not Computed	460.0 L/s
Return Air	01-RA02		60.3 Pa	460.0 L/s
Return Air	01-RA03		51.0 Pa	460.0 L/s
Return Air	01-RA04		51.0 Pa	460.0 L/s
Return Air	01-RA05		60.3 Pa	460.0 L/s
Return Air	01-RA06		52.5 Pa	460.0 L/s
Return Air	01-RA08		58.6 Pa	470.0 L/s
Return Air	01-RA09		51.0 Pa	460.0 L/s
Return Air	01-RA10		51.0 Pa	460.0 L/s
Return Air	01-RA11		60.3 Pa	460.0 L/s
Return Air	01-RA12		30.8 Pa	460.0 L/s
Supply Air	01-SA01		252.5 Pa	520.0 L/s
Supply Air	01-SA02		118.7 Pa	520.0 L/s
Supply Air	01-SA03		114.2 Pa	560.0 L/s
Supply Air	01-SA04		114.2 Pa	560.0 L/s
Supply Air	01-SA05		118.7 Pa	520.0 L/s
Supply Air	01-SA06		77.5 Pa	520.0 L/s

Figure 3.6.4c Screenshot of a schedule within Revit. Source: https://www.cadlinecommunity.co.uk/hc/en-us/articles/115001913009-Revit-Copying-a-Standard-or-Custom-Schedule-to-other-Projects



Items	WBS
Substructure	A
Foundations	A.10
Standard Foundations	A.10.10
footing rectangular(192)	A.10.10.1
footing columns(220)	A.10.10.2
footing covering upto plinth level(33)	A.10.10.3
Slab on Grade	A.10.20
Plinth	A.1
Shell	B
Superstructure	B.10
Exterior Enclosure	B.20
Roofing	B.30
Interiors	C

S...	WBS	Name	Height	Area	Volume	Count
	A.10.10	Standard Foundations				
	A.10.10.1	footing rectangular	0.000 ft	8,644.774 ft²	3,799.643 ft³	192.000 ea
	A.10.10.2	footing columns	0.000 ft	5,785.076 ft²	3,725.171 ft³	220.000 ea
	A.10.10.3	footing covering upto plinth...	247.500 ft	8,267.199 ft²	6,297.260 ft³	33.000 ea

S...	WBS	Object	ModelVolume	Length	Width	Area
	A.10.10.2.1	Rectangular Column	18.922 ft³	7.500 ft	1.750 ft	27.000 ft²
	A.10.10.2.2	Rectangular Column	12.631 ft³	7.500 ft	1.750 ft	22.221 ft²
	A.10.10.2.3	Rectangular Column	12.906 ft³	7.500 ft	1.750 ft	22.221 ft²
	A.10.10.2.4	Rectangular Column	12.906 ft³	7.500 ft	1.750 ft	22.221 ft²
	A.10.10.2.5	Rectangular Column	12.906 ft³	7.500 ft	1.750 ft	22.221 ft²
	A.10.10.2.6	Rectangular Column	12.906 ft³	7.500 ft	1.750 ft	22.221 ft²
	A.10.10.2.7	Rectangular Column	12.906 ft³	7.500 ft	1.750 ft	22.221 ft²

Figure 3.6.4d Screenshot of a quantification workbook in Navisworks. Source: https://www.researchgate.net/figure/Quantification-workbook-of-Navisworks-Validation-by-stakeholders-The-advantages-of-BIM_fig6_326904461

These tools and approach can be applied in other disciplines beyond MEP. The MEP user should be able to use the same tools as mentioned in 3.6.2 for navigation, selection etc. For energy analysis, etc, the tools mentioned for shadow and sunlight simulation should also be available to the MEP user. HVAC and electrical calculations can be pushed to the MEP user widget.

-

● 4.0 SUMMARY

The ontological and interface requirements defined in this Deliverable will now be integrated into the timelines and technical delivery phases within the scope of Work Packages 2 to 6.

As these requirements are discussed and evaluated further, they must be prioritised to achieve the critical functionality necessary for the Deployment phase (Months 13-14), followed by first Pilot Studies and user testing due to occur in Months 15-16. See [PrismArch Grant Proposal] P.50 for further details. These tasks are overseen by Work Package 6, T6.2/ T6.3.

In the process of prioritising functionality, some may be determined to be beyond the scope of first Pilot Studies, in which case they could be held back for the second Phase of testing (Months 23-24). Other functionalities might even be determined to be beyond the scope of the entire PrismArch Horizon2020 research. In such a situation, those aspects will be diligently documented, and implementation proposals will be provided, such that these functionalities could still be developed by external parties after the conclusion of this research grant.

5.0 APPENDIX

5.1 References

PrismArch Documents

Deliverable D1.1 (2021, March). “*Report on current limitations of AEC software tools, leading to user and functional requirements of PrismArch*”.
URL: https://prismarch-h2020.eu/download/472

Deliverable D3.1 (2021, May). “*Report on cognitive issues in VR-aided design environments*”. URL: https://prismarch-h2020.eu/download/480

Deliverable D6.1 (2021, July). “*Define the architectural projects and usage scenarios for demonstration and evaluation*”.
URL: https://prismarch-h2020.eu/download/483

Other References

[AECDeltas] URL: https://github.com/aecdeltas

[AECDeltas, Spec] URL: https://aecdeltas.github.io//aec-deltas-spec/#description

[Akselsen 2019] Akselsen, M. V., Stenvold, S. T. “*Virtual Reality – Only a Hype or Real Improvement for Structural Design?”* Master’s thesis, Norwegian University of Science and Technology, 2019. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2623347/no.ntnu:inspera:2507506.pdf

[Alvarez and Font, 2020] A. Alvarez and J. Font, 2020. “*Learning the designer’s preferences to drive evolution*”. In Proceedings of the International Conference on the Applications of Evolutionary Computation.

[Alvarez et al. 2020] Alvarez A., Font J., Togelius J., “*Towards Designer Modeling through Design Style Clustering*”, 2020. ArXiv, abs/2004.01697.

[Amoor 1997] Amoor, R. “*A generalised framework for the design and construction of integrated design systems”,* (PhD Thesis) University of Auckland, 1997

[Argelaguet et al. 2013] Argelaguet F., Andujar C., "*A Survey of 3D Object Selection Techniques for Virtual Environments”*, 2013, URL:https://hal.archives-ouvertes.fr/hal-00907787/document

[Berlo et al 2012] Berlo, L. A. H. M, Beetz, J. , Bos, P. , Hendriks, H. , Tongerer, R. C. J “*Collaborative engineering with IFC: New insights and technology”,* Conference: Proc. 9th European Conference on Product and Process Modelling

[Bowman 2001] Bowman, D. A. “*Basic 3D interaction techniques”.* Dept. of Computer Science (0106), Virginia Tech, 2001, URL: https://people.cs.vt.edu/~bowman/3dui.org/course_notes/siggraph2001/basic_techniques.pdf

[Bowman 2012] Bowman, D. A., McMahan, R. P., Ragan, E. D. “*Questioning naturalism in 3D user interfaces”*. In Communications of the ACM, 55 (9) pp. 78-88, 2012, URL: https://dl.acm.org/doi/abs/10.1145/2330667.2330687

[Bowman 2014] Bowman, D. A., “*The Encyclopedia of Human-Computer Interaction*”, 2nd Ed. Chapter 32, 2014, URL: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/3d-user-interfaces [buildingSMART] URL: https://www.buildingsmart.org/

[Cyberpunk 2077] URL: https://www.cyberpunk.net/gb/en/

[De Vries 1991] De Vries H., "*The Minimal Approach",* Proceedings of the 1991 CIB W078 Conference, 1991

[Deliot and Heitz, 2018] Deliot, T., Heitz, E., "*Procedural Stochastic Textures by Tiling and Blending*", 2018. [ebook] URL: https://drive.google.com/file/d/1QecekuuyWgw68HU9tg6ENfrCTCVIjm6l/view

[Design Council] Design Council. "*What is the framework for innovation? Design Council's evolved Double Diamond".* Design Council, URL: https://www.designcouncil.org.uk/news-opinion/what-framework-innovation-design-councils-evolved-double-diamond

[Digital Blue Foam] "*Digital Blue Foam - Super building design for better cities*", URL: https://www.digitalbluefoam.com/

[Ekholm 2005] Ekholm A., "*Prerequisites for coordination of standards for classification and interoperability",* ITCon vol. 100, pg 275-289, 2005

[El-Diraby 2015] El-Diraby *T., "From Deep Blue to Watson: The Nature and Role of Semantic Systems in Civil Informatics"*, Chapter in [Issa. et. al 2015]

[Epic Games inc] Twinmotion, Real-time immersive 3D AEC visualization. URL: https://www.unrealengine.com/en-US/twinmotion

[EulerVR] Virtual reality software for structural engineering. URL: https://www.eulervr.com/

[Froese et. al 2015] Froese, T. M., Zeb, J., "*Transaction formalization in the infrastructure management using an ontological approach"*, Chapter in [Issa. et. al 2015]

[Gaier et al. 2018] Gaier A., Asteroth A., and Mouret J. B., "*Data-efficient design exploration through surrogate-assisted illumination*". Evolutionary Computation, 26:381–410, 2018.

[Galanos et al. 2021] Galanos T., Liapis A., Yannakakis G. N., and Koenig R., "*Arch-elites: Quality-diversity for urban design*," in Proceedings of the Genetic and Evolutionary Computation Conference, 2021.

[Gielingh 2008] Gieling, W. "*An assessment of the current state of product data technologies",* Computer Aided Design Vol. 40, p750 - 759, 2008
[GitLab] PrismArch GitLab account. Available at www.gitlab.com/prismarch

[GraphVR] GraphVR, "*A Virtual Reality Tool for the Exploration of Graphs with HTC Vive System: Graph exploring using Virtual Reality and UE4*", URL:http://graphics.unibas.it/www/UE4-Graph3D/index.md.html#abstract

[Grasshopper] Grasshopper - visual programming interface for Rhino, URL: https://www.grasshopper3d.com/

[Gravity Sketch] Gravity Sketch, "*Think in 3d, create in 3d*", URL: https://www.gravitysketch.com/

[Hamil 1994] Hamil. S *The end of babel - IFC promotional Video, 1994*
URL: https://www.youtube.com/watch?v=g_jmGQvr6dQ

[IAI 1999] "*An Introduction to the International Alliance for Interoperability and the Industry Foundation Classes*". Ed. Wix J.

[IFC Architecture Guide 1999] "*IFC Object Model Architecture Guide",* published by the International Alliance for Interoperability*, 1999*

[IFC Implementation Agreement] URL:
https://standards.buildingsmart.org/documents/Implementation/IFC_Implementation_Agreements/

[Issa et. al 2015] Issa. R., Mutis I., *"Ontology in the AEC Industry: A Decade of Research and Development in Architecture, Engineering, and Construction",* American Society of Civil Engineers, June 2015

[Kaley 2021] Kaley A., "*Mapping User Stories in Agile".* Nielsen Norman Group, URL:
https://www.nngroup.com/articles/user-story-mapping/

[Kinzler et al, 2021] H.Kinzler, D.Zolotareva, R.Tadauchi and A.Mnich-Spraiter , "*Sphereing: A Novel Framework for Real-time Collaboration and Co-presence in VR*", 2021, p,4.

[Laakso et. al 2012] Laakso, M., Kiviniemi, A., "*The IFC standard - A review of history, development and standardization",* ITcon Vol. 17, pg. 134-161, 2012

[Laubheimer 2017] Laubheimer, P. "*Personas vs. Jobs-to-Be-Done".* Nielsen Norman Group, URL:
https://www.nngroup.com/articles/personas-jobs-be-done/

[Liapis et al. 2013] A. Liapis, G. N. Yannakakis, and J. Togelius, "*Designer modeling for personalized game content creation tools*," in Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics, 2013.

[Liapis et al. 2014] A. Liapis, G. N. Yannakakis, and J. Togelius, "*Designer modeling for Sentient Sketchbook,*" in Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2014.

[Liu, 2009] Liu L., Tamer Ozsu, M., *Encyclopedia of Database Systems*, 2009. Retrieved from:
https://www.springer.com/gp/book/9780387355443

[McCabe 2015] McCabe A., Mcpolin D. O., "*Virtual reality: Immersed in the structural world*". The Structural Engineer. 93, 20-23, 2015, URL: https://www.researchgate.net/profile/Aimee-Mccabe/publication/285612355_Virtual_reality_Immersed_in_the_structural_world/links/5aa9480a0f7e9b88266e8ccb/Virtual-reality-Immersed-in-the-structural-world.pdf

[Miller, 2016] Miller N., "*The wicked problem of interoperability", 2016.*
Retrieved from: https://www.bdcnetwork.com/blog/wicked-problem-interoperability

[Mouret and Clune, 2015] J.-B. Mouret and J. Clune, *"Illuminating search spaces by mapping elites*," ArXiv, vol. abs/1504.04909, 2015.

[Nissila, 2014] Nissali, J., Heikkila, R., "*BIM-based schedule control for precast concrete supply chain*". URL:
https://www.researchgate.net/figure/An-example-of-XML-language-in-Status-exchange-Structural-designers-and-contractors_fig1_282727750

[Nystrom 2014] "*Game programming patterns"*. Genever Benning. URL :
https://www.gameprogrammingpatterns.com/flyweight.html

[Poinet, 2020] Poinet, P., Stefanescu, D., Papadonikolaki, E., "*Collaborative Workflows and Version Control Through Open-Source and Distributed Common Data Environment*", 2020.
URL: https://www.researchgate.net/figure/The-stream-revision-history-available-via-the-Speckle-admin-interface_fig2_342419999

[PrismArch Grant Proposal] "*PrismArch: Virtual reality aided design blending cross-disciplinary aspects of architecture in a multi-simulation environment*". Proposal number: 952002.
URL: https://cordis.europa.eu/project/id/952002

[Rittel and Webber, 1973] Rittel, H. W. J., Webber, M. M. "*Dilemmas in a General Theory of Planning" (1973)* https://web.archive.org/web/20070930021510/http://www.uctc.net/mwebber/Rittel+Webber+Dilemmas+General_Theory_of_Planning.pdf.
See also https://en.wikipedia.org/wiki/Wicked_problem#cite_note-Rittel_and_Webber_1973-4

[Rumbaugh et al 1991] Rumbaugh, J. , Blaha, M. , *"Object Oriented Modeling and Design",* Prentice Hall (January 1, 1991)

[Sherwin 2018] Sherwin, K. "*Card Sorting: Uncover Users' Mental Models for Better Information Architecture"*. Nielsen Norman Group, URL: https://www.nngroup.com/articles/card-sorting-definition/

[Sfikas et al. 2021] K. Sfikas, A. Liapis, and G. N. Yannakakis, "*Monte carlo elites: Quality-diversity selection as a multi-armed bandit problem*," in Proceedings of the Genetic and Evolutionary Computation Conference, 2021.

[SOFiSTiK] SOFiSTiK - FEM, BIM and CAD Software for Structural Engineers, URL: https://www.sofistik.com/

[Speckle] https://speckle.systems/careers/growth-marketing-manager/

[Speckle GitHub] URL: https://github.com/speckleworks

[Speckle Diffing] URL: (https://v1.speckle.systems/docs/developers/aec-delta/)

[Steed 2017] Steed, A. "*How virtual reality is changing engineering"*. Ingenia. 70, 2017, URL: https://www.ingenia.org.uk/ingenia/issue-70/how-virtual-reality-is-changing-engineering

[Stefanescu 2020] Stefanescu, D. , "*Alternate means of digital design communication"* (PhD thesis) UCL, London, 2020.

[Takagi 2001] H. Takagi, "*Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation*," Proceedings of the IEEE, vol. 89, no. 9, pp. 1275–1296, 2001.

[Tarandi 1998] Tarandi, V. , "*Neutral intelligent CAD communication: information exchange in construction based upon a minimal schema",* (PhD Thesis) KTH, 1998

[Tibuzzi 2016] Tibuzzi, E. *"Interweaving Practise",* in Design Engineering (Kara. H, Bosia. D) 2016.

[Unreal Engine] Unreal Engine Documentation, "Unreal Engine 4 Terminology", URL:https://docs.unrealengine.com/4.26/en-US/Basics/UnrealEngineTerminology/

[Unreal Lumen] URL: https://docs.unrealengine.com/5.0/en-US/RenderingFeatures/Lumen/

[Unreal Lightmass] URL: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/GPULightmass/

[Unreal TextureVariation] URL: https://forums.unrealengine.com/t/new-texturevariation-node-in-4-26-simple-non-tiling-materials/152671

[Wei et. al 2010] Wei, G. , Zhou, Z. , Zhao, X. , Ying, Y. "*Design of building component library based on IFC and PLIB standard", 2nd International Conference on Computer Engineering and Technology, 2010*

[Zaker 2018] Zaker, R., Coloma, E. "*Virtual reality-integrated workflow in BIM-enabled projects collaboration and design review: a case study".* Vis. in Eng. 6, 4, 2018, URL: https://viejournal.springeropen.com/track/pdf/10.1186/s40327-018-0065-6.pdf

[Delany, S. "Uniclass 2015"] URL: https://toolkit.thenbs.com/articles/classification#classificationtables

CSI Building Knowledge Improving Project Delivery "OmniClass Construction Classification System", [Online] URL: http:// www.omniclass.org

https://www.researchgate.net/publication/332352148_Method_for_Evaluating_a_Building_Information_Model

5.2 Extracts from Structural Engineering User Interviews

Available in the PrismArch project repository. Available on request to the PrismArch consortium.

5.3 Images

Figure	Reference
2.1.1a - 2.2.2.2a	Images by AKT II.
2.2.2.3a	Image by AKT II. Reproduction based on source [IAI 1999].
2.2.2.3b - 2.2.2.3c	Image by AKT II.
2.2.2.3d	Specification for ifcBeam. Retrieved from: https://standards.buildingsmart.org/IFC/RELEASE/IFC2x3/TC1/HTML/ifcsharedbldgelements/lexical/ifcbeam.htm
2.2.2.4a	From [Laakso et. al 2012]
2.2.2.4b	From [Nissila, 2014]
2.2.2.4c	Retrieved from: https://github.com/buildingSMART/ifcJSON
2.2.3.1a	Screenshot of Speckle interface, taken from *www.speckle.xyz*.
2.2.3.2a	Image by AKT II.
2.2.3.2b	Screenshot of Speckle interface, www.speckle.xyz
2.2.3.2c	Image by AKT II.
2.2.3.3a	Screenshot of Speckle interface, www.speckle.xyz
2.2.3.3b	Speckle documentation. Retrieved from: https://speckle.guide/dev/base.html
2.2.3.3.c	Speckle documentation. Retrieved from: https://speckle.guide/dev/base.html
2.2.3.3d	Speckle documentation.https://speckle.guide/dev/objects.html
2.2.3.3e	Speckle documentation https://github.com/specklesystems/speckle-sharp/blob/main/Objects/Objects/BuiltElements/Beam.cs
2.2.4.2a 2.2.4.2b	Images by AKT II.
2.3.3a	Georgios Adamopoulos, University College London (2021). Mapping from 3D Euclidean space to 2D Texture space. Retrieved from https://github.com/UCL/dfpi/wiki/Mesh-Basics

2.3.3b.	UVPackmaster. Comparison between 2 different UV Packing algorithms Retrieved from https://uvpackmaster.com/uvpackmaster-for-blender/
2.3.3c	Ben Golus (2017). Tri-Planar mapping example and concept. Retrieved from: https://bgolus.medium.com/normal-mapping-for-a-triplanar-shader-10bf39dca05a
2.3.3d.	Deliot and Heitz (2018). Procedural stochastic texturing. Hexagonal “stamp” selection (left) and blending (right). Retrieved from: https://drive.google.com/file/d/1QecekuuyWgw68HU9tg6ENfrCTCVIjm6l/view
2.3.4.2a	Nystrom (2014). Instancing diagrams (Flyweight pattern) - www.gameprogrammingpatterns.com/flyweight.html
2.3.4.2b	Nystrom (2014). Instancing diagrams (Flyweight pattern) - www.gameprogrammingpatterns.com/flyweight.html
2.4.1a - 2.4.4a	Images by AKT II.
2.4.5a - 2.4.5e	Images by AKT II.
2.6.1a	Speckle web interface. Retrieved from: www.speckle.xyz
2.6.1b - 2.6.1g	Images by AKT II.
3.2a	Design Council. What is the framework for innovation? Design Council's evolved Double Diamond. Retrieved from: www.designcouncil.org.uk/news-opinion/what-framework-innovation-design-councils-evolved-double-diamond
3.3.1a - 3.3.1d	Screenshots from [Gravity Sketch] interface. Retrieved from: www.gravitysketch.com
3.3.2a - 3.3.2b	Screenshots from [Grasshopper] interface. Retrieved from: www.rhino3d.com/6/new/grasshopper
3.4.1a - b, f, j-l	Images by ZHVR.
3.4.1c	Examples of the User Interface to configure avatar height, the existing plugin MetaHuman inside Unreal Engine 4 https://docs.metahuman.unrealengine.com/en-US/UserGuide/Body/index.html
3.4.1d	Examples of the User Interface to configure avatar LOD, the existing plugin MetaHuman inside Unreal Engine 4 https://docs.metahuman.unrealengine.com/en-US/UserGuide/Body/index.html
3.4.1e	Example of the existing application to calibrate an avatar inside VR, Holodeck User Guide https://docs.nvidia.com/holodeck/pdf/Holodeck-User-Guide.pdf
3.5.1a-b	Images by ZHVR.
3.5.2a - 3.5.2c	Images by AKT II.
3.5.2d	Screenshot from [EulerVR] interface. Retrieved from: www.eulervr.com

3.5.2e	Screenshot from [SolidVR] interface
3.5.2f	Image by AKT II.
3.5.2g - 3.5.2j	Screenshots from [SOFiSTiK] interface
	Images by AKT II.
3.5.2k	Screenshot from [Digital Blue Foam] interface
3.5.2l - 3.5.2q	Images by AKT II.
3.5.3a - 3.5.3i	
3.6.1a - h	Images by ZHVR.
3.6.1i	Reference to revisit a meeting record, from the game CyberPunk 2077 https://www.youtube.com/watch?v=m6L-7n5KtpA
3.6.1j	Reference to X Ray d vs. Photorealistic display modes, illustrating the difference between revisiting a meeting and attending a meeting https://youtu.be/tC47SIQkfMA
3.6.1k	Examples of BigData Visualisation Layouts, Sviatoslav Iguana, 2019 https://towardsdatascience.com/large-graph-visualization-tools-and-approaches-2b8758a1cd59
3.6.1 l - p, u - z3	Images by ZHVR.
3.6.1q	Traditional occlusion solutions in gaming, Tom Looman https://www.tomlooman.com/the-many-uses-of-custom-depth-in-unreal-4/
3.6.1r	Diagram illustrating a user occlusion condition, F.Argelaguet and C.Andujar, https://hal.archives-ouvertes.fr/hal-00907787/document
3.6.1s	Example of Metadata Node navigation using a turntable feature, GraphVR http://graphics.unibas.it/www/UE4-Graph3D/index.md.html#abstract
3.6.1t	Physics based interaction with nodes, Force Directed VR https://www.youtube.com/watch?t=33&v=O-AwY0gYLlQ&feature=youtu.be
3.6.2a - h	Images by ZHVR.
3.6.3a - 3.6.3n	Images by AKT II.
3.6.4a	BIM widget in VR. Source: https://youtu.be/mhCxFudzFX4
3.6.4b	Example of a clash in Navisworks. Source: https://youtu.be/v6HtMKme3Ck

3.6.4c	Screenshot of a schedule within Revit. Source: https://www.cadlinecommunity.co.uk/hc/en-us/articles/115001913009-Revit-Copying-a-Standard-or-Custom-Schedule-to-other-Projects
3.6.4d	Screenshot of a quantification workbook in Navisworks. Source: https://www.researchgate.net/figure/Quantification-workbook-of-Navisworks-Validation-by-stakeholders-The-advantages-of-BIM_fig6_326904461